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 two-gigabyte 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.
Please give me data

Clients

It’s 10 past the hour
Here is data.

You’re out of date
Here is new data.

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:
  \( \text{concat}(\text{hash(process id)}, \text{sequential key}). \)

- 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.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Many days

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.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Single day

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 \( S_{old} \), read the parity subsector \( P_{old} \),
  \[ P_{new} := (S_{old} \ XOR \ S_{new}) \ XOR \ P_{old} \]

- 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


Simple, Rational Guidance for Chopping Up Transactions

or

How to Get Serializability Without Paying For It

Dennis Shasha
Eric Simon
Patrick Valduriez
Outline

- Motivation
- Critical Assumption
- Example
- Sufficient Conditions
- Chopping Optimization
- Algorithm and Analysis
- Applications
Motivation

- Many proposals for concurrency control methods.
  Aimed at designers.

- Practitioners are stuck with two phase locking. Their only tuning knobs are
  - chop transactions into smaller pieces
  - choose degrees 1 or degree 2 isolation.
Critical Assumption

Environment in which we know the transaction mix (e.g., real-time or on-line transaction processing)

That is, no unpredictable, ad hoc queries.
Purchase Transaction — 1

Purchase:
add value of item to inventory;
subtract money from cash.

Constraint: cash should never be negative.
Purchase Transaction — 2

Application programmers chop as follows:

1. First transaction checks to see whether there is enough cash.
   If so, add value of item to inventory.
   Otherwise, abort the purchase.

2. The second transaction subtracts the value of the item from cash.

Cash sometimes becomes negative. Why?
Purchase Application — 3

By contrast, if each numbered statement is a transaction, then following bad execution can occur. Cash is $100 initially.

1. P1 checks that cash > 50. It is.

2. P2 checks that cash > 75. It is.


4. P2 completes. Cash = −25
Purchase Transaction — 3

No surprise to your university professor, who says something like:

You Idiot! You should never have chopped this transaction up!

Why did I pass you from my course anyway?
Purchase Transaction — 4

Surprise: Simple variant guarantees that cash will never become negative.

1. First transaction checks to see whether there is enough cash.
   If so, subtract cash.
   Otherwise, abort the purchase.

2. The second transaction adds value of item to inventory.

Goal of research: Find out why this works!
Special Recovery Hacks

Must keep track of which transaction piece has completed in case of a failure.

Suppose each user X has a table UserX.

- As part of first piece, perform insert into UserX (i, p, 'piece 1'), where i is the inventory item and p is the price.

- As part of second piece, perform insert into UserX(i, p, 'piece 2').

Recovery includes reexecuting the second pieces of inventory transactions whose first pieces have finished.
Assumptions

• Possible to characterize all transactions during some interval.

• Want serializability for original transactions.

• On failure, possible to determine which transactions completed and which did not.
Chopping

For simplicity, assume sequential transactions.

A chopping is a partition of the transaction into pieces such that the first piece has all rollback statements.

Each piece will execute using two phase locking (if it aborts, execute again).
Graphical Characterization

Chopping graph — Undirected graph whose nodes are pieces. Two kinds of labeled edges.

1. Conflicts: C edge between p and p’ if the two pieces come from different transactions and issue conflicting instructions.

2. Siblings: S edge between p and p’ if they come from the same original transaction.

Note: no edge can have both an S and a C label.
Correctness

A chopping of a set of transactions is *correct* if any execution of the chopping is equivalent to some serial execution of the original transactions.

Equivalent = every read returns same value in two executions and writes write same value.
Sufficient Conditions for Correctness

SC-cycle — a simple cycle that includes at least one S edge and at least one C edge.

Theorem 1: A chopping is correct if its chopping graph contains no SC-cycle.
Proof of theorem 1

Suppose there were a cycle in the serialization graph of original transactions.
T1 → T2 → ... → Tn → T1.
Ti → Tj means Ti issues op that conflicts with and precedes an op in Tj.

Identify pieces associated with each transaction that are involved in this cycle:
p → p’ → ... → p”
Both p and p” belong to transaction T1.

If every arrow corresponded to a C edge, then p ≠ p” since each piece uses two phase locking so serialization graph on pieces is acyclic. Otherwise p = p” is possible, but then every other edge would be either a C edge of an S edge. So, cycle among original transactions implies SC-cycle. Contradiction.
Purchase Example

Original transactions:

\[ P(i,p): \]
1. if cash > p then invent(i) += p;
2. cash -= p;

If we chop \( P(i,p) \) into two transactions, we’ll get an SC-cycle.
Purchase Variant

Original transactions:

P(i,p):
1. if cash > p then cash -= p;
2. invent(i) += p;

Chopping P(i,p) does not introduce an SC-cycle.
CHOPPING EXAMPLE

Purchase transaction (price, item)

1. read(cash); If cash $>$ price then
   read(inventoty[item]); inventory[item] += price;
   write(inventory[item]);

2. read(cash); cash -= price;
   write(cash);

The chopping graph has SC-cycles

Fig.C.1
CHOPPING EXAMPLE

MODIFIED PURCHASE TRANSACTION (price, item)

1. read (cash); if cash > price then
   read (cash); cash -= price;

2. read(inventory[item]; inventory[item] += price;
   write(inventory[item]);

There is no SC-cycle in the chopping graph

Fig. C.2
Example 1

Original three transactions:

T1: R(x) W(x) R(y) W(y)
T2: R(x) W(x)
T3: R(y) W(y)
Example 1 — Chop T1

T11: R(x) W(x)
T12: R(y) W(y)

No cycle.
Example 1 — too much chopping

Break up T11 further into

T111: R(x)
T112: W(x)

will result in an SC-cycle.
Optimization

Question: Does a finest chopping exist?

Answer: yes.

Key Observation: If T is chopped and is in an SC-cycle with respect to T’, then chopping T’ further or gluing the pieces of T’ together will not eliminate that cycle.

Moral: if chopping T causes a cycle, then nothing you do to other transactions can help.
Reasons

Suppose we break p of T’ into p1 and p2. If p is in a cycle, then p1 will have an S edge to p2, so at most the cycle will be lengthened.

Suppose we combine two pieces p1 and p2 of T’ into piece p. If p1 alone had been in a cycle, then so will p. If cycle went through S edge between p1 and p2, then cycle will just be shorter.
Systematic Method to Obtain Finest Chopping

Original set of transactions: T1, T2, ..., Tn.

- For each transaction Ti, Fi = chop Ti as finely as possible with respect to the other (unchopped) transactions.

- Finest chopping is F1, F2, ..., Fn.

Algorithm is connected components algorithm in C graph for each Ti. Complexity is $O(n \times (e + m))$, where e is the number of C edges in the transaction graph and m is max number of database accesses.
Putting the three pieces of T3 into one will not make the chopping of T1 OK. Nor will chopping T3 further.

Fig. C.3
Application to Typical Database Systems

SQL system with bind variables, e.g. update salary of employee :x.

Determining conflicts is difficult. Can use predicate locking,
AND name LIKE 'T\%'
vs.
AND name LIKE 'S\%'

Our new idea: number of conflicts is significant.
Example Application

Suppose a single query of form:

```
SELECT ...  
FROM account
```

is concurrent with updates of the form:

```
Update ...  
FROM account
WHERE acctnum = :x
```

If acctnum is a key, then conflict on only one record.

Can run at degree 2 isolation. (Or could chop if all updates in first query were modifies.)
Degree 2 isolation has the effect of chopping the scan so each record access is a single transaction. In this case, degree 2 isolation is as good as degree 3.

Reason: no SC-cycle because Replace is on a key.

Fig. C.4
Related Work

A lot of work that seeks to reduce concurrency control constraints, mostly through new algorithms or weakening of isolation guarantees.

Aimed at implementors of DBMS’s.

Our work aims at users.

Farrag and Ozsu — application knowledge of application, e.g. hotel reservations.

Garcia-Molina — partition transactions into classes. In same class run concurrently, whereas synchronize among classes. Weaker notions of consistency by using counterstep transactions if something bad happens.

Lynch — nested classes. Ensure a specific order among conflict steps.
Related Work — 2

Bayer — new concurrency control and recovery mechanism to allow a single batch transaction to run among many short transactions.

Oracle and Gemstone use similar scheme for long readers.

Hsu and Chan — special concurrency control algorithms for situations in which data is divided into raw data and derived data. Consistency of the raw data is not important.

O’Neil — exploits commutativity of increments to release write locks early.
Related Work — 3

Wolfson — looks at early release of locks when user has complete control over acquisition and release of locks.

Bernstein, Shipman and Rothnie — introduced conflict graph in SDD-1 context.

Casanova — generalized that notion to include program flow.

Shasha and Snir — further generalized to include atomicity constraints in access of parallel shared memory. SC-graphs are a special case.
Future Work

Have:
simple, efficient algorithm to partition transactions into the smallest pieces possible when transaction mix is known.

Open:

- How to extend to sagas (undo transactions), tree-locking, multi-level concurrency control?

- Suppose a given set of transactions do not chop well. Can one partition the set into several subsets, execute each subset in its own subinterval, and thereby achieve a good chopping?