15-721 Database Management Systems

Benchmarking in Database Systems

Instructor: Anastassia Ailamaki
http://www.cs.cmu.edu/~natassa

Questions (already)

- What's **your** view of performance?
- How would **you** measure/compare performance of database systems?
- What would you do if it was **your** database system under test?

Why Benchmark DB Systems?

- Provide buying guide for customer on cost, performance
- Stake in the sand for vendors
- Target for developers

Why Not Benchmark DB Systems?

- Vendors cheat like mad.
  - (how can you cheat?)
- Benchmark specials
- Customers never achieve same level of performance as vendors
- "Single number" benchmarks are mainly marketing tools

Benchmarketing

- Benchmark "wars"
- Small "representative" application
- …run
- 1 winner, n losers
- …run "corrected" benchmark in "tuned" system
- (gurus get involved)
- Another winner, other losers
- … more of the same…
- "the new system will include this beta feature"

More Reasons Not to Benchmark

- Design is really hard
- Even if people agree on benchmark, they don't agree on how to compare performance
- Noone thinks benchmark is good (not even the winner)
- And cheating, cheating, cheating
  - Auditing benchmark results did not fix the problem

Anon. et. al.

- (Really Jim Gray)

- A Measure of Transaction Processing Plan Original version of paper published in DataMation on April 1, 1984

Anon. et. al. Benchmark

Benchmark consists of three tests:

- Debit/Credit Transaction
  - Simulates customer doing a banking transaction
  - Measures throughput and cost

- Scan
  - Series of batch updates of 1000 records
  - Measures performance available to application programmer (time+cost)

- Sort
  - Sort 1 million records
  - Illustrates raw performance of system (time+cost)

Observations

- Only debit/credit component survived – evolved into TPC/A and then TPC/B and then TPC-C

- Sort benchmark evolved into:
  1) “Datamation” sort
  2) Minute sort – how much can you sort in a minute
  3) Penny sort – how much you can sort for a penny

Performance Metrics

- For sort and scan: **elapsed time**
- For debit credit:

  Peak transactions per second with 95% of transactions having less than one second response time
- Speed of communication line is factored out.

  Response = interval between the arrival of the last bit from the communications line and the sending of the first bit of the response
- As we will see cost is factored in too

Calculating Cost

- Complex to calculate
- Ideally would capture entire "cost of ownership"
- Adopted the "vendors view" for its simplicity

  Cost = the 5-year capital cost of vendor supplied software & hardware in the machine room

  (Probably 1/5 the total cost)
- **Not included**:
  - Cost of money
  - Terminal cost
  - Communication line cost
  - Application development cost
  - Cost of running the operation

How is Cost Used?

- Benchmark is charged for resource used

  2^{-5} of the 5-year cost of the s/w and h/w
- Example for a sort that runs 1 hour:

Package	Package Cost	Per Hour Cost	Benchmark Cost
Processor	$80K	$1.8	$1.8
Memory	$15K	$ .3	$ .3
Disk	$50K	$1.1	$1.1
Software	$50K	$1.1	$1.1
Total			$4.3

Why Include Price in Benchmark?

- Cross-vendor comparisons for h/w and s/w
- Flexibility in system configuration used

- Sort example (drawn from Tandem):

1) A one CPU, 2 disk sort takes 30 minutes @ cost of $1.5

2) A 16 CPU, 2 disk, 8MB memory sort takes 10 minutes and cost $15

- Parallel sort is 3X faster but has 10X cost

Definition of Sort Benchmark

- Goal: measure what a wizard can get out of the system
- Excellent measure of input/output architecture (software and hardware) as well as overhead imposed by OS
- Definition:
  - Input file: 1M, 100 byte records stored sequentially on disk
  - first 10 bytes of each record constitutes a key
  - keys of the input file are in random order
  - Sort creates an output file on disk and fills it with the input records in key order
  - No restrictions – sort may use as much memory and as many scratch files as desired
  - Relevant metrics: elapsed time and cost

Scan Benchmark

- Typical of “end-of-day” processing in on-line transaction processing systems
  - E.g., each night the credit card company generates 1/30th of the months bills
- Based on a Cobol program that sequentially scans a sequential file, reading and updating each record
- Input file is 1M, 100-byte records
- Scan is broken into minibatch transactions each of which processes 1000 records
- Restrictions:
  - Must use fine grain locking so debit/credit transactions can run concurrently
  - Updates must be protected with a log
  - Application must be written using some end-user application interface in high-level programming language

Transaction Flow

Open file shared, record locking
Perform Scan 1000 times
  Begin
  BeginTransaction
    Perform 1000 times
      Read next record from input file with locking
    Rewrite record
  CommitTransaction
  End
Close File

Evaluation

Relevant measures:

- Elapsed time: average time between BeginTransaction steps
- Cost: the time-weighted system cost of Scan

Results:

- Theory: Elapsed time of 0.1 second.
- Practice: 1 to 100 seconds (10 second average) with costs ranging from $0.001 to $0.1

DebitCredit Benchmark

Background

- 1973, a large retail bank wanted to put its 1,000 branches, 10,000 tellers, and 10,000,000 accounts on-line. Goal: a peak load of 100 TPS + 99.5% availability
- Two bids were submitted:

1) $5M from a minicomputer vendor - $50K/TPS cost
2) $25M from a mainframe vendor $250K/TPS costs

- TP1 => TPC/A (w/ terminals) => TPC/B (w/o terminals)
- For a long time 1000 TPS was unachievable. Eventually vendors produced systems capable of 10,000 TPS!!
- Eventually TPC-C by Transaction Processing Council

DebitCredit Database

- Record types: branches, tellers, accounts, history
- Sizing for 100 TPS:
  - 1,000 branches (0.1MB, random access, 100B records)
  - 10,000 tellers (1 MB, random access, 100B records)
  - 10,000,000 accounts (1 GB, random access, 100B recs)
  - 90 day history (10 GB sequential, 50 byte records)
- Transaction flow:

  BeginTransaction
  Read message from terminal (100 bytes)
  Read and update account
  Write history
  Read and update teller
  Read and update branch
  Write message to terminal (200 bytes)
  CommitTransaction

Carnegie Mellon

DebitCredit Details

- Branch keys are generated randomly
- Teller within branch is picked at random
- Random account picked
  - 85% of the time same branch
  - 15% of the time different branch
- Account keys = 10B, other keys can be short (i.e. ints)

Carnegie Mellon

Other Restrictions

- All data files must be protected by
  - fine granularity locking
  - logging (duplexed)
- History file must be on stable storage
- 95% of the transactions must have a response time of 1 second or less
- Message handling and terminal must be incorporated
- Tellers have a 100 second think time

Carnegie Mellon

Benchmark Scaling

- For a 10 TPS system store only 1/10 of the DB (100 branches, 1,000 tellers, 1M accounts)
- For a 1000 TPS system scale up DB 100x (10K branches, 100K tellers, 100M accounts)
- For a 10,000 TPS system scale up DB 1000x (100K branches, 1M tellers, 1G accounts)

Carnegie Mellon

Criticism

- Doesn't match real business transactions (too simple)
- Performance benchmark only. Evaluates very little of the functionality offered by relational database systems
- Allowed vendors to ignore decision support issues for almost 10 years (84-94)
- Comments about functional benchmarks being non-portable are bogus. E.g., Wisconsin benchmark is certainly portable
- Cost structure is too simple
  - Ignores communications/terminal costs
  - Ignores cost of development and maintenance
  - Ignores cost of outages (lost labor)

Carnegie Mellon

Current TPC benchmarks

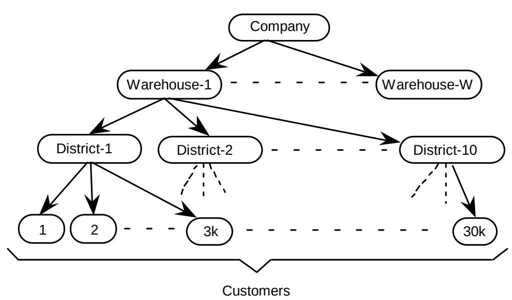
- TPC-C (OLTP)
- TPC-D: Decision-support =>TPC-H, TPC-R
- TPC-W: web
- http://www.tpc.org

Carnegie Mellon

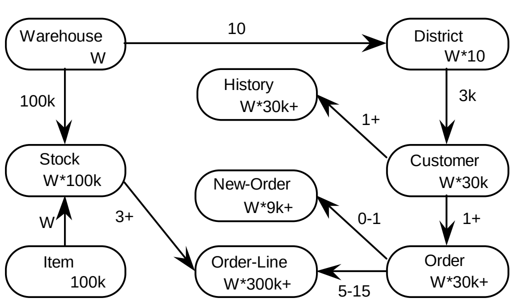
TPC-C

- Complete wholesale supplier computing environment
- Population of users executes transactions
- Activities of order-entry environment
  - entering and delivering orders
  - recording payments
  - checking the status of orders
  - monitoring the level of stock at the warehouses
- Transactions executed on-line or queued. Tests:
  - The simultaneous execution of complex transaction types
  - Multiple on-line terminal sessions, elapsed time, I/O, ACID
  - Non-uniform data distribution, type variety, contention
- Measure: new-order tpmC, $/tpmC, and availability date of the priced configuration.

TPC-C World



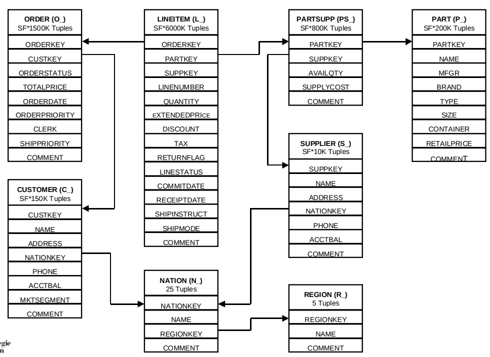
TPC-C Schema



TPC-H/R

- Suite of business oriented ad-hoc queries and concurrent data modifications
- Queries and the data relevant to industry
- Illustrates decision support systems that
  - examine large volumes of data
  - execute queries with a high degree of complexity
  - give answers to critical business questions
- TPC-H Composite Query-per-Hour Performance Metric (QphH@Size) reflects
  - selected database size
  - query processing power with single execution stream
  - query throughput with multiple concurrent users.
- Price/Performance metric: $/QphH@Size.

TPC-H Schema



TPC-H Example Query

(Shipping Priority (Query 3))

```
SELECT
      L_ORDERKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS
      REVENUE, O_ORDERDATE, O_SHIPPRIORITY
FROM CUSTOMER, ORDER, LINEITEM
WHERE
      C_MKTSEGMENT = 'FOOD' AND
      C_CUSTKEY = O_CUSTKEY AND
      L_ORDERKEY = O_ORDERKEY AND
      O_ORDERDATE < DATE 1.5.98 AND
      L_SHIPDATE > DATE 1.6.98
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE
```

TPC-W

- Transactional web benchmark
- Controlled internet commerce environment
- Business-oriented transactional web server
- Similar tests with TPC-C
- Metric: Web Interactions Per Second (WIPS)
- Profiles:
  - Primary shopping
  - Browsing
  - Web-based ordering

Carnegie Mellon

Other benchmarks

- Wisconsin benchmark
- 007 (OODBMSs)
- Sequoia2000 (GIS)
- Data mining benchmarks (similar to DSS)
- *Microbenchmarks*

- Excellent (and fun) reading: Gray's handbook

(on the web from www.benchmarkresources.com)

Carnegie Mellon