Performance Tuning

Wrap-up and Conclusions

Harvard Extension School

Cody Doucette, Ph.D.

Lecture designed by David G. Sullivan

Reference

Database Tuning: A Principled Approach, Dennis E. Shasha

Goals of Performance Tuning

- Increase *throughput* – work completed per time
  - in a DBMS, typically transactions per second (txns/sec)
  - other options: reads/sec, writes/sec, operations/sec
  - measure over some interval (time-based or work-based)
- Decrease *response time* or *latency*
  – the time spent waiting for an operation to complete
  - overall throughput may be good,
    but some txns may spend a long time waiting
- Secondary goals (ways of achieving the other two):
  - reduce lock contention
  - reduce disk I/Os
  - etc.

Challenges of Tuning

- Often need to balance conflicting goals
  - example: tuning the *checkpoint interval*
    – the amount of time between checkpoints of the log.
    - goals?
      -
      -
- It's typically difficult to:
  - determine what to tune
  - predict the impact of a potential tuning decision
- The optimal tuning is workload-dependent.
  - can vary over time

What Can Be Tuned?

- Three levels of tuning:
  1. low level: hardware
     - disks, memory, CPU, etc.
  2. middle level: DBMS parameters
     - page size, checkpoint interval, etc.
  3. high level
     - schema, indices, transactions, queries, etc.
- These levels interact with each other.
  - tuning on one level may change the tuning needs on another level
  - need to consider together

1. Hardware-Level Tuning (Low Level)

- Disk subsystem
  - limiting factor = rate at which data can be accessed
  - based on:
    - disk characteristics (seek time, transfer time, etc.)
    - number of disks
    - layout of data on the disk
  - adding disks increases parallelism
    - may thus increase throughput
  - adjusting on-disk layout may also improve performance
    - sequential accesses are more efficient than random ones
- Memory
  - adding memory allows more pages to fit in the cache
  - can thereby reduce the number of I/Os
  - however, memory is more expensive than disk

Other Details of Hardware Tuning

- Can also add:
  - processing power
  - network bandwidth (in the case of a distributed system)
- Rules of thumb for adding hardware (Shasha)
  - start by adding memory
    - based on some measure of your *working set*
  - then add disks if disks are still overloaded
  - then add processing power if CPU utilization >= 85%
  - then consider adding network bandwidth
- Consider other options before adding hardware!
  - tune software: e.g., add an index to facilitate a common query
  - use current hardware more effectively:
    - example: give the log its own disk

2. Parameter Tuning (Middle Level)

- DBMSs—like most complex software systems—include parameters ("knobs") that can be tuned by the user.
- Example knobs:
  - checkpoint interval
  - deadlock-detection interval
  - several more we'll look at in a moment
- Optimal knob settings depend on the workload.

Example: Tuning Lock Granularity

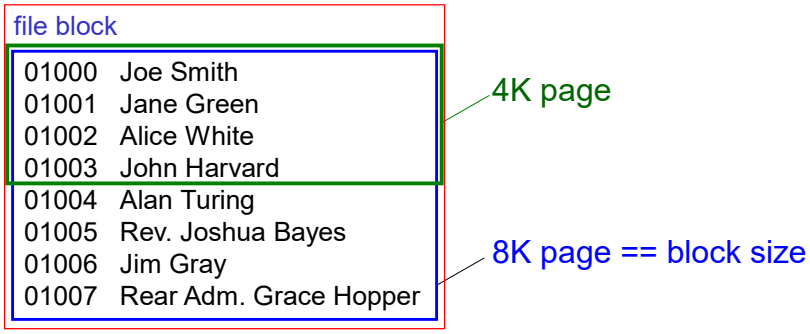
- possibilities include: page, record, entire table
- How could finer-grained locking improve performance?
  -
- How could finer-grained locking degrade performance?
  -
  -
- Rule of thumb (Shasha):
  - measure the "length" of a txn in terms of the percentage of the table that it accesses
  - "long" txns should use table-level locking
  - "medium" txns that are based on a clustered/internal index should use page-level locking
  - "short" txns should use record-level locking

Example: Tuning the MPL

- MPL = maximum number of txns that can operate concurrently
- How could increasing the MPL improve performance?
  -
- How could increasing the MPL degrade performance?
  -
- Shasha: no rule of thumb works in all cases.
  Instead, use an incremental approach:
  - start with a small MPL value
  - increase MPL by one and measure performance
  - keep increasing MPL until performance no longer improves

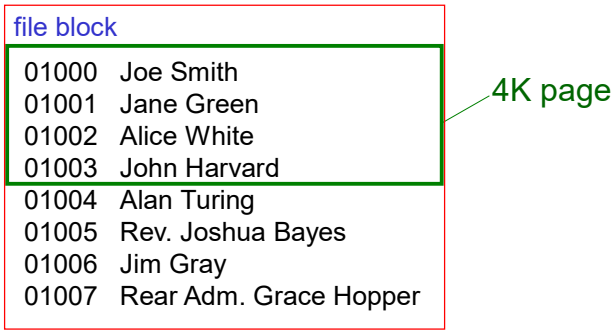
Example: Tuning Page Size

- Recall:
  - the filesystem transfers data in units called *blocks*
  - the DBMS groups data into *pages*
    - may or may not correspond to a block



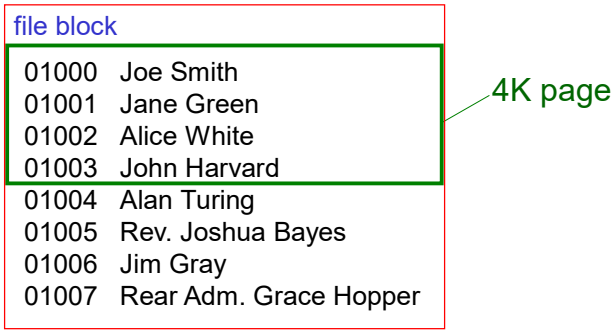
Tuning Page Size (cont.)

- How could a smaller page size improve performance?



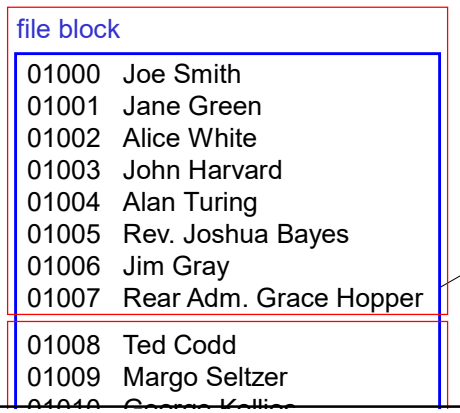
Tuning Page Size (cont.)

- How could a smaller page size *degrade* performance?



Tuning Page Size (cont.)

- What if we select a page size > block size?
  - can reduce your ability to keep useful data in the cache (when accesses are more or less random)
  - if page-level locking, can increase contention for locks
  - can lead to unnecessary I/O due to OS prefetching



+ can reduce the number of overflow pages
+ reduces I/O for workloads with locality (e.g., range searches)

16K page

Tuning Page Size (cont.)

- Rule of thumb?
  - page size = block size is usually best
  - if lots of lock contention, reduce the page size
  - if lots of large items, increase the page size

3. High-Level Tuning

- Tune aspects of the schema and workload:
  - relations
  - indices/views
  - transactions/queries
- Tuning at this level:
  - is more system-independent than tuning at the other levels
  - may eliminate the need for tuning at the lower levels

Tuning a Relational Schema

- Example schema: *account(account-num, branch, balance)*
  customer(customer-num, name, address)
  owner(account-num, customer-num)
  (One account may have multiple owners.)
- Vertical fragmentation: divide one relation into two or more
  - e.g., what if most queries involving account are only interested in the account-num and balance?
- Combining relations:
  - e.g., store the join of account and owner:
    account2(account-num, branch, balance, customer-num)
  - what's one drawback of this approach?

Recall: Primary vs. Secondary Indices

- Data records are stored inside a *clustered* index structure.
  - also known as the *primary* index
- We can also have *unclustered* indices based on other fields.
  - also known as *secondary indices*
- Example: *Customer(id, name, street, city, state, zip)*
  - primary index:
    (key, value) = (*id*, all of the remaining fields)
  - a secondary index to enable quick searches by name
    (key, value) = (*name*, *id*) *does not include the other fields!*

Tuning Indices

- If SELECTs are slow, add one or more secondary index.
- If modifications are slow, remove one or more index. Why?
- Other index-tuning decisions:
  - what type of index?
    - hash or B-tree; see lecture on storage structures
  - which index should be the clustered/primary?
- Complication: the optimal set of indices may depend on the query-evaluation plans selected by the query optimizer!

Tuning Transactions/Queries

- Banking database example:
  - lots of short transactions that update balances
  - long, read-only transactions that scan the entire account relation to compute summary statistics for each branch
  - what happens if these two types of transactions run concurrently? (assume rigorous 2PL)
- Possible options:
  - execute the long txns during a quiet period
  - multiversion concurrency control
    - make the long, read-only txns operate on an earlier version, so they don't conflict with the short update txns
  - use a weaker isolation level
    - ex: allow read-only txn to execute without acquiring locks

Deciding What to Tune

- Your system is slow. What should you do?
- Not a simple process
  - many factors may contribute to a given bottleneck
  - fixing one problem may not eliminate the bottleneck
  - eliminating one bottleneck may expose others

Deciding What to Tune (cont.)

- Iterative approach (Shasha):

```
repeat
    monitor the system
    tune important queries
    tune global parameters (includes DBMS params,
      OS params, relations, indices, views, etc.)
until satisfied or can do no more

if still unsatisfied
    add appropriate hardware (see rules of thumb from earlier)
    start over from the beginning!
```

Example Tuning Scenarios

- From Shasha's book
- All scenarios start with the complaint that an application is running too slowly.
- Scenario 1:
  - workload:
    - data-mining application for a chain of department stores
    - queries the following relation during the day:
      oldsales(cust-num, cust-city, item, quantity, date, price)
    - indices on cust-num, cust-city, item to speed up the queries
    - at night:
      - updates performed as a bulk load
      - bulk delete to eliminate records more than 3 weeks old
  - specific problems:
    - bulk load times are very slow
    - daytime queries are also degenerating

Example Tuning Scenarios (cont.)

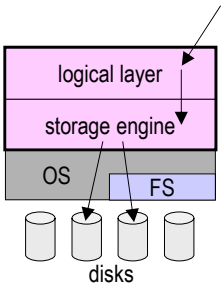
- Scenario 2:
  - workload:
    - an application that is essentially read-only
    - performs many scans of a relation
  - relevant info:
    - disks show high access utilization but low space utilization
    - the log is on a disk by itself
    - each scan currently requires many disk seeks
    - management refuses to buy more disks

Example Tuning Scenarios (cont.)

- Scenario 3:
  - workload:
    - an airline manages 100 flights per day
    - two tables:
      passenger(passenger-name, flight-num, seat-num)
      occupancy(flight-num, total-passengers)
    - every reservation txn updates both tables
  - relevant info:
    - there is a high degree of lock contention

Looking Back

- Recall our two-layer view of a DBMS:
- When choosing an approach to information management, choose an option for each layer.



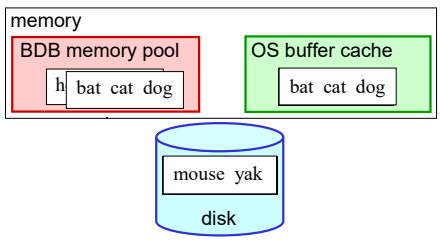
- We've seen several options for the storage layer:
  - transactional storage engine
  - plain-text files (e.g., for XML or JSON)
  - native XML DBMS
  - NoSQL DBMS (with support for sharding and replication)
- We've also looked at several options for the logical layer:
  - relational model
  - semistructured: XML, JSON
  - other NoSQL models: key/value pairs, column-families

One Size Does *Not* Fit All

- An RDBMS is an extremely powerful tool for managing data.
- However, it may not always be the best choice.
  - see the first lecture for a reminder of the reasons why!
- Need to learn to choose the right tool for a given job.
- In some cases, may need to develop new tools!

Implementing a Storage Engine

- We looked at ways that data is stored on disk.
- We considered *index structures.*
  - B-trees and hash tables
  - provide efficient search and insertion according to one or more key fields
- We also spoke briefly about the use of *caching* to reduce disk I/Os.



Implementing a *Transactional* Storage Engine

- We looked at how the "ACID" properties are guaranteed:

  Atomicity: either all of a txn's changes take effect or none do

  Consistency preservation: a txn's operations take the database from one consistent state to another

  Isolation: a txn is not affected by other concurrent txns

  Durability: once a txn completes, its changes survive failures

Distributed Databases and NoSQL Stores

- We looked at how databases can be:
  - fragmented/sharded
  - replicated
- We also looked at NoSQL data stores:
  - designed for use on clusters of machines
  - can handle massive amounts of data / queries

Logical-to-Physical Mapping

- The topics related to storage engines are potentially relevant to *any* database system.
  - not just RDBMSs
  - any logical layer can be built on top of any storage layer
- Regardless of the model, you need a *logical-to-physical mapping.*
- In PS 2, you implemented part of a logical-to-physical mapping for the relational model using Berkeley DB.
- We also looked at options for how to perform this mapping for XML.