Knowledge Index

Q&A

Stateless

The server processes requests based only on information relayed with each request and doesn't rely on information from earlier requests – this means that the server doesn't need to hold onto state information between requests (or the state can be held into an external service, like a database)

Different requests can be processed by different servers

The fact that any service instance can retrieve al service state necessary to execute a behavior from elsewhere enables resiliency, elasticity, and the ability for any available service instance to execute any task at all

Stateful

Stateful services are either a database or based on an internet protocol that needs a tight state handling on a single host

The server processes requests based on the information relayed with each request and information stored from earlier requests

The same server must be used to process all requests linked to the same state information, or the state information needs to be shared with all servers that need it

PCM Phase Changed Memory Evaluation

PCM may replace or complement SSD in the furute.

Benefits: Very good access time, good endurance (key benefit over SSD)

Drawbacks: Poor data density (too low)

Trends:

Not currently possible to tell what the trend in endurance will be.

Both access time and data density are likely to inrease which is good.

SSD Random Write Issue

Writes to SSD require a block to be read, copied (buffered in memory), erased and then rewritten. This is slower for random writes as we must do this from scratch, whereas with sequential write we can prefetch blocks into memory (so only need to perform the erase and rewrite steps.)

The block is the smallest unit that can be erased on an SSD because it requires a larger voltage to erase than to read. It's difficult to erase a small number of pages without erasing the surrounding ones.

SSD & HDD Performance Comparison

Random access is much faster on flash than on disk, as the disk R/W arm must constantly move around the disk during random access, adding seek time + rotational delay. Sequential access will also be slightly faster on flash than on disk, since the disk will have to occasionally move the R/W head to a different track or a different sector, whereas flash doesn't need any moving parts to read data.
Random writes are slower on flash than random reads since random writes often involve copying data, and deleting old pages in flash requires quite high voltages.
An erase block is bigger than a read block because, due to the high voltages involved in erasing pages, it is hard to only erase a small number of pages without also erasing the pages around them.

Waste in Memory Bandwidth

17-18 Exam Question 2 b)
50B of bandwidth is wasted. File size = 10 000 * 100 = 1 000 000B. 1 000 000 / 75 = 13333.3 and 13333 * 75 = 999 975 so there is 25B left to transfer, and thus 50B is wasted as 75 – 25 = 50.

Embedding & Linking

Embedding is essentially a pre-computed join. It is convenient as the nested components can be accessed easily by the server and this doesn't require any query optimisation.

Linking is a way of representing two joinable documents by keeping an identifier of one stored in the other. This allows for more flexibility but is more computationally expensive to process.

XML Schema Free Approaches

Yes, a similar schema-free approach can be implemented for XML (It's known as a model-mapping approach) on top of a RDB. Examples of this being done are Edge, Monet XParent and XRel.
Advantages of this are that it' capable of supporting a range of XML applications that are either static (DTD does not change) or dynamic (DTD can change). Additionally, it doesn't require extending the expressive power of DB models to support XML.

Some disadvantages of this are that query optimisation is made more difficult and some of the models don't perform optimally for certain types of query / workload.

XML Evaluation Techniques

c) XRel is a node-oriented model-mapping approach, which creates tables for Path, Element Text and Attribute fields of an XML document.
XParent is edge oriented, using tables for LabelPath, DataPath, Element and Data.
Edge is also edge oriented, keeping a single table with information about all edges in an XML graph.
The performance of each is highly dependent upon the query workload. For example, Edge performs well for simple queries, whereas XRel and XParent outperform edge for more complex queries.

XML & Graph Conversion

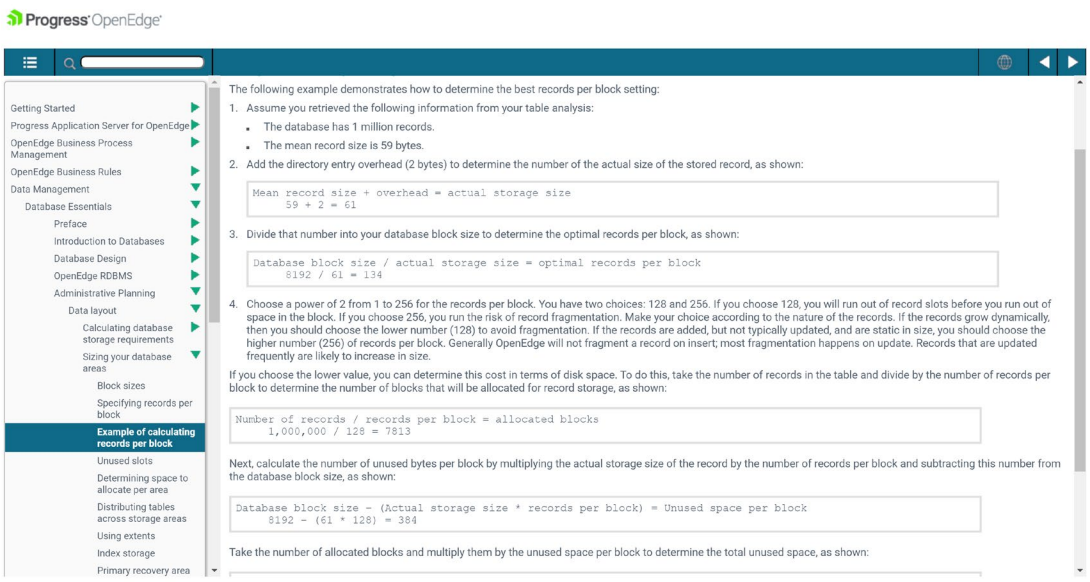
- We can think of the nested tags of an XML document as a tree structure
- We can represent the tree in a graph databases with edges for parent/child relationships
- The edge can include the ordinal number so the file can be exactly reconstructed

- It is possible to convert a graph database into a XML document if the graph is non-cyclic. However, cyclic graphs cannot be represented in a XML document

JOINS in Document Database MongoDB and Neo4J Graph Database

- Graph and document databases both avoid needing an intermediate table, and extra joins, required for many to many relationships.
- For some relationships document databases designers may choose to embed documents within each other to join them. In some cases this might require duplicating some data, which would require extra space but may make querying more convenient. In the case of one-to-many relationships however this is quite compact
- Alternative:
    - Graph databases don't need joins. The relationship between specific nodes is already represented and otherwise unintuitive joins over separate tables are represented as paths in the graph database.

Disk Calculation Formula



CAP Theorem

Consistency means that data is the same across the cluster, so you can read or write from/to any node and get the same data.

Availability means the ability to access the cluster even if a node in the cluster goes down.

Partition tolerance means that the cluster continues to function even if there is a "partition" (communication break) between two nodes (both nodes are up, but can't communicate).

In order to get both availability and partition tolerance, you have to give up consistency. Consider if you have two nodes, X and Y, in a master-master setup. Now, there is a break between network communication between X and Y, so they can't sync updates. At this point you can either:

A) Allow the nodes to get out of sync (giving up consistency), or

B) Consider the cluster to be "down" (giving up availability)

All the combinations available are:

- **CA** - data is consistent between all nodes - as long as all nodes are online - and you can read/write from any node and be sure that the data is the same, but if you ever develop a partition between nodes, the data will be out of sync (and won't re-sync once the partition is resolved).

- **CP** - data is consistent between all nodes, and maintains partition tolerance (preventing data desync) by becoming unavailable when a node goes down.
- **AP** - nodes remain online even if they can't communicate with each other and will resync data once the partition is resolved, but you aren't guaranteed that all nodes will have the same data (either during or after the partition)

Eventual Consistency

Eventual consistency makes sure that data of each node of the database gets consistent eventually. Time taken by the nodes of the database to get consistent may or may not be defined.

Strong Consistency offers up-to-date data but at the cost of high latency.

While Eventual consistency offers low latency but may reply to read requests with stale data since all nodes of the database may not have the updated data.

Strongly consistent reads may have higher latency than eventually consistent reads. Strongly consistent reads are not supported on global secondary indexes. Strongly consistent reads use more throughput capacity than eventually consistent reads.

External Consistency

What is the difference between strong consistency and external consistency?

A replication protocol exhibits "strong consistency" if the replicated objects are linearizable. Like linearizability, "strong consistency" is weaker than "external consistency", because it does not say anything about the behavior of transactions.

1. The **CAP theorem** is the idea that a scalable system can't have consistency, availability and partition tolerance at a high level. One must be compromised.
2. **Resource Disaggregation:** In datacenters racks are now considered the unit of deployment not individual servers. In future we would like to divorce resources from their physical layout and just think of allocating resources to an application regardless of how those resources are distributed.
3. **Hinted Handoff:** Used to deal with temporary failures in the Dynamo system. If a node is unresponsive a replica will be sent to the next node after the first N in the preference list with a hint in the metadata of where the replica was meant to go. The node will store this data in a separate database and attempt to send it when the original node is healthy again.

4. **Hybrid Cloud:** A public cloud is where a company provided cloud infrastructure over the internet to the public, typically anyone with a credit card. Alternatively a private cloud is where the cloud infrastructure is just used internally by the company who manages it. A hybrid cloud is where both types of infrastructure are used an channels are setup so data and even applications can be shared.

i) A **distributed file system** is a system for storing data across multiple machines, that is able to process requests for the data and serve it from the machine it is stored from in a distributed fashion. Two examples of this are GFS and HDFS, which are by google and hadoop respectively.

ii) A **nearline system** is one that incorporates aspects of both a batch and online system. Thus, it is required to support workloads that process large groups of data but also to be able to process individual transactions and behave at a low latency.

iii) A **coordinator** is the part of a system responsible for maintaining concurrency between processes of a system. It may also have other tasks and is usually a centralized component that's purpose is to keep the system running.

iv) A **session** is when a client connects to an application and exchanges multiple requests before disconnecting.

i. **Location transparency** means that a client is unable to tell the difference in system performance no matter which location the server sending the response is in. We see this in spanner with it's dynamic cross-DC load balancing.
ii. A **batch processing system** is one that applies operations on collections of data (as opposed to individual transactions on data items). MapReduce is an example of such a system.
iii. **Partition/Aggregation Pattern:** It's what it says on the tin. The pattern by which data is partitioned. Commonly used to determine which machines keep which data in a distributed system
iv. A **multi-version store** is a data store that maintains multiple versions of the same data. BigTable is a key example of this, allowing the user to specify how many versions are kept.

i. **Soft lease** – when lease expires, the provider holds onto the user's cache until and unless they need to hand out that address/space or they receive instructions on how to clear their cache
ii. **Coordination service** – Service which provides coordination for processes sent to it and ensures the distributed system acts accordingly

iii. **Data locality** – essentially how close the data is to the computation; moving computation closer to the data saves bandwidth => Tasks are executed on the node containing the data
iv. **Tail latency** – small percentage of response times that take the longest in comparison to the bulk of its response times
i. **Quorum** – A quorum is the minimum number of votes that a distributed transaction must obtain in order to be allowed to perform an operation in a distributed system. A quorum-based technique is implemented to enforce consistent operation in a distributed system.
ii. **Distributed hash table** – A distributed hash table is a distributed system that provides a lookup service like a hash table: key-value pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.
iii. **Vector Clocks** are used in a distributed system to determine whether pairs of events are causally related. Using Vector Clocks, timestamps are generated for each event in the system, and their causal relationship is determined by comparing those timestamps.
iv. A **distributed transaction** is a set of operations on data that is performed across two or more data repositories (especially databases). It is typically coordinated across separate nodes connected by a network, but may also span multiple databases on a single server.

HDD Questions

Track Capacity = sectors size * track size
Surface Capacity = Track Capacity * Surface Size
Disk Capacity = # of platters (*2 if doule-sided) * surface capacity
# of Blocks = Disk Capacity / Block Size

# of records per block = floor(Block Size / Size per Record)
Total blocks = Total Records / # of records per block

Wasted Space = wasted space per block * Total Blocks

Time to read with seek time 10ms, rotational delay 5ms, transfer time 1ms per block
To read 10,000 blocks sequentially

Blocks per track = Track Capacity / Block Size = 25600 / 1024 = 25 blocks/track
Blocks in a cylinder = # of records per block * blocks per track = 250 blocks

10,000 blocks / 250 blocks = 40 cylinders to read

Transfer time = 0.001 * 10000 blocks = 10 secs
Avg seek time = 0.1 secs. 40 cylinder * 0.1 secs = 4 secs
Rotational Delay = 0.005 (for sequential)
Read Time = 14.005 secs

IF READ IN PARALLEL
Transfer rate would be 0.01 secs per 10 blocks (for 10 platters)

Notes

1. **Flash Disk:**
    - **A.** Random Write
2. **FTL Flash Translation layer**
    - A. The FTL is a device driver that acts as a layer of abstraction between the OS and the flash storage device. It can tune performance of the device and help increase its lifetime.
    - B. 1. Balance writes across every flash page to improve flash overall lifetime
    - C. 2. Avoids making updates wait for an erase by writing to a new page and diverting all subsequent reads to that address
    - D. 3. Keep a list of all old invalid pages, and erasing them later
3. Comparison between Flash and Phase-Changed Memory (Non-Volatile RAM)
4. Conflict between DBMS design and SSD storage device
    - **A.** Focus more on optimal I/O performance
5. **Flash-only OLTP:** Good for random rad, but might not good for writes.
    - **A. Append/Pack:** For update request, firstly write sequentially as much as we can. If a in-place update is needed, not update the original space but append to the end and invalidate the original page.
    - **B. Result:** Smaller variance and stable performance
6. **Flash-aided Business Intelligence OLAP:** Incoming updates stored in SSD / Flash, and use merged data from disks and flash to answer query
    - **A. Materialized Sort-**Merge: Do all the merging and update in the Main Memory
        - **i. Coarse-**Grain Index: The precision for finding a particular update in the sorted runs (SSD)

7. **Logging: Defined by small sequential writes**
   A. For HDD, it will incur full rotational delays, but perfectly good for SSD

In-Memory Databases

1. **Definition of OLAP & OLTP**
2. **Performance is dominated by** Latching, Recovery, Locking, Buffer Pool.

	Locks	*Latches*
Separate …	User transactions	Threads
Protect …	Database contents	In-memory data structures
During …	Entire transactions	Critical sections
Modes …	Shared, exclusive, update, intention, escrow, schema, etc.	Read, writes, (perhaps) update
Deadlock …	Detection & resolution	Avoidance
… by …	Analysis of the waits-for graph, timeout, transaction abort, partial rollback, lock de-escalation	Coding discipline, "lock leveling"
Kept in …	Lock manager's hash table	Protected data structure

Fig. 2. Locks and latches.

3.
   A. With main memory deployment, buffer pool is eliminated
4. **Old SQL (Not suitable for distributed case)**
5. **NoSQL**
   A. SQL is translated at compile time into sequence of low-level operations, but missing query optimization, parser, abstraction
   B. No ACID such as the data written might not be available to all users immediately
   C. Appropriate for non-transactional systems
6. **New SQL**
   A. New record level locking (Latching), New buffer pool overhead (Buffer Pool), shared data structures (locking), New write-ahead logging (fault tolerance mechanism)
   B. VoltDB: Comparison with other solutions
      i. Built for horizontally scale
      ii. K-safety, in-memory operation, partitions operate autonomously and single-threaded
      iii. **Single / Multi-partitioned**
      iv. **Partitioned / Replicated:** Partitioned well for transactional data (high frequency of modification), Replicated good for relatively static data

(low frequency of modification)

v. **Ways to improve scalability, availability snapshots, spolling and disaster recovery**

C. VoltDB and OLTP

i. Asynchronous Communication (Subscribed)

ii. CANNOT call rollback on a SQL transaction

iii. No concurrency (single-threaded) => Good for OLTP, not OLAP

iv. BUILT FOR THROUGHPUT OVER LATENCY

D. Important Lesson

i. The source of overhead is totally different with a changing medium.

XML

1. **DTD: Document Type Definition**
2. **Storing XML in Database**
   - **A.** Structure- Mapping Approach
   - **B.** Model-Mapping Approach
3. **Conversion to Data**
   - **A. Ordinal, Label-Path, Data-Path**
   - **B. Edge**
     - i. **Only need 1 table in total. Ref / Val**
   - **C. Monet**
     - i. Partitioned based on label-paths.No. of Tables = No. of distinct label-paths
   - **D. XRel**
     - i. Path, Element, Text, Attribute
   - **E. XParent**
     - i. LabelPath, DataPath, Element, Data)
   - **F.** XRel and XParent outperform Edge, XRel and XParent outperform Edge in complex queries. Edge performs better when using simple queries. Label-paths help in reducing querying time.

Document Database

A. Index

甲、 Single Filed Indexes

乙、 Compound Field Indexes

丙、 Multikey Indexes

Multicores

1. The majority of processor stall time goes to L2 Data and L1 Instruction
2. Transistor counts doubles every 8 years, but the clock speeds and power hit the wall
3. Processor Trends
    A. Pipelining / ILP / Multithreading => Multicores => Multisocket Multicores
4. **BottleNeck for Workload Scalability**
    A. OLTP: Access Latency. When increasing the number of threads, the access latency (i.e., locking) will limit the maximum throughput
    B. OLAP: Memory Bandwidth. Read lots of data until exhaust memory bandwidth.
5. More than 50% of time goes to stalls on average (i.e., Memory Cycles), leaving only ~1 instruction per cycle
6. **Source of Memory Stalls**
    A. Lots of stalls goes to L1 Instruction level and L3 data.
    B. It is **mainly caused by** instruction fetch & long-latency data misses
        i. So Instructions need more capacity (Limited Improvement Possibility)
        ii. Data Misses are compulsory (Cannot be improved also)
    C. **Improvement Target:** L1-I locality & Cache Line utilization for data
7. **Pre-Fetching**
    A. Lite Solution
        i. Next-line: miss A -> Fetch A+1 (Favors sequential access & Spatial Locality)
        ii. Not useful for instructions / data
            1. Can be solved by branch prediction or pointer chasing.
    B. Temporal Streaming
        i. Exploit recurring control flow: Prefetch instruction more accurate and precise by predicting the same fetching pattern for each operation.
        ii. Example: ways to locate letter “A” in the database should be similar to the way to locate letter “B” in the database. The instruction will be roughly the same by calling function “A” -> “B” -> “C” …
    C. Software Guided
        i. Similar idea, by pre-fetching a child node based on guesses.
8. **Cache Conscious**
    A. Code Optimizations: Minimize jumps, Profile-guided optimizations, Ad-hoc optimizations, etc.
    B. Cache Conscious Data Layouts
        i. Maximize cache line utilization & Exploit next-line prefetcher
        ii. OLTP: Row-store. Try to access many columns (Select *)

iii. **OLAP:** Column-store. Try to access a few columns

iv. **Example:** For OLTP, the one cache line contains as many columns as possible, as the transaction might affect multiple related fields. For OLAP, the one cache line contains the maximum amount of data it can stored for a particular field, so to prefetch the possible used data later on.

v. **Lookup-heavy workload for trees:** Go down the tree frequently

vi. **Scan heavy workload for trees:** Level by level storing

C. **Vectorized Execution**

i. Pull the entire cache line back to the user. Whatever data is in it, pull it out.

ii. Promotes **SIMD:** Single Instruction, Multiple Data.

9. **Common Instruction Employment**

A. Significant overlapping between instruction used, but small reused data. **High Overlap in same-type transactions**

B. **Computation Spreading**

i. Move the transaction to where the data is. For example, for instruction on any thread, if the data already exists, move the instruction to that core.

10. **Conclusion**

A. For L1-Instruction misses, we cannot improve the capacity, so we can minimize footprint & maximizing re-use to get a “illusion of larger cache) like SLICC approach

B. For Last Level Cache (LLC) data misses, it is compulsory for first-time seen data, but we can maximize cache-line utilization through cache-conscious algorithms and layout (i.e., essentially to prefetch stuff)

11. **For Scale-up Option**

A. Signification locking & Latching time spent on locking (i.e., for shared data structures)

B. **Critical Section Types**

i. **Unbounded**: Lots of thread can/want access it at the same time => Major scalability bottleneck caused by locking and latching

ii. **Fixed**: Fixed number of threads that will enter (i.e., with more threads, it will not cause a bottleneck)

iii. **Cooperative**: Requests from different threads can be combined and done together

C. **Improvement Plans**

i. **Locking-related Issues**

1. We always release and request the same locks repeatedly throughout the agent thread execution timeline.
2. Use **Speculative Lock Inheritance scheme,** Transaction 1 commit without releasing a hot lock, and the next transaction 2 seed the lock list and re-use it directly.
    1. **Benefit:** Significantly reduce lock contention issue
3. **Centralized Locking => Thread-local Locking**
    1. For a particular thread, it will always access 1 subset of a data, leading to a predictable data accesses
4. **In-Memory Databases**
    1. Reduce the need for concurrency control (i.e., access disk is not needed anymore)
    2. It is optimized for better cache utilization (i.e., cacheline)
    3. No Buffer Management needed

ii. **Latching Related Issues**

1. Separate threads
2. **Physiological Partitioning (PLP)**
    1. Each worker will be responsible for a particular range of data. If received a request to access Range 1, use worker 1, if Range 2, use worker 2. No need to fight for a particular resources in Index and Heap pages.
    2. **Issue:** How to balance the workload of workers? For Range 1, it might only receive 10 % of requests, 90% for Range 2. Unbalanced => Bottleneck again.

iii. **Logging-related Issues**

1. Aether Holistic Logging

iv. **Non-Uniform Communication**

1. Access a different hardware island is time-consuming
2. Shared-everything: Not optimal because access a different island needs 500 cycles
3. Shared-nothing: Avoid communication, but sensitive to workload.
4. Want to find a middle ground.
5. **Adaptive Transaction Processing:** Each request will be sent to the Island that has most sufficient resources

v. **Conclusion**

1. Important to find the best cost-effective part to optimize
2. DO NOT ASSUME uniformity in communication. (Not the case for **multisocket architecture** as the communication between

different islands)

Cold Storage

1. For cold storage device: WORO Write Once – Read Occasionally.
2. **Pelican Rack**
   - A. **Right-Provisioned:** Only 8% of the disks are spinning concurrently, specially designed for **blobs which are infrequently accessed.**
   - B. **Domains:** Cooling, Power, Bandwidth
3. Comparison between Full-Provisioning, Pelican, Random Placement
   - A. The random placement needs to calculate resource conflict on the go, so its average throughput is limited
   - B. Pelican needs on average 14.2 seconds to activate the group for sending the first byte
4. **Special Design for using CSD**
   - A. User has no uniform access and control layout ability on CSD
   - B. **Goal:** Hard-driven data access to minimize group switches
   - C. **Solution 1:** Buff-Pack
     - i. **Flush it in group order**
     - ii. Flush into the current disk group to avoid switching if possible, or switch to disk group with the largest buffer
   - D. **Solution 2:** Off-Pack
     - i. **Flush entire buffer into disks (i.e., write data into wrong disk group)** then transfer later on
   - E. **Analysis**
     - i. Total Time = Switch Time (Dominant Factor) + Seek Time + Read Time + Write Time