CS372H: Spring 2009 – Final Exam

Instructions

- This exam is closed book and notes with one exception: you may bring and refer to a 1-sided 8.5x11-inch piece of paper printed with a 10-point or larger font. If you hand-write your review sheet, the text density should not be greater than a 10-point font would afford. For reference, a typical page in 10-point font has about 55-60 lines of text.
- If a question is unclear, write down the point you find ambiguous, make a reasonable interpretation, write down that interpretation, and proceed.
- State your assumptions and show your work. **Write brief, precise, and legible answers.** Rambling brain-dumps are unlikely to be effective. **Think before you start writing** so that you can crisply describe a simple approach rather than muddle your way through a complex description that "works around" each issue as you come to it. **Perhaps jot down an outline** to organize your thoughts. And remember, a **picture can be worth 1000 words.**
- For full credit, show your work and explain your reasoning.
- To discourage guessing and rambling brain dumps, I will give approximately 25% credit for any problem left *completely blank* (e.g., about 1.5 for a 5 or 6 point question, 2.0 for a 7 point question, 3.5 for a 14 point question)

  If you attempt a problem, you will get between 0% and full credit for it, based on the merits of your answer.

  Note that for the purposes of this rule, a *problem* is defined to be any item for which a point total is listed. *sub-problems* are not eligible for this treatment – if you attempt to answer any subproblem for a problem, you may as well attempt to answer all subproblems.
- Write your name on this exam

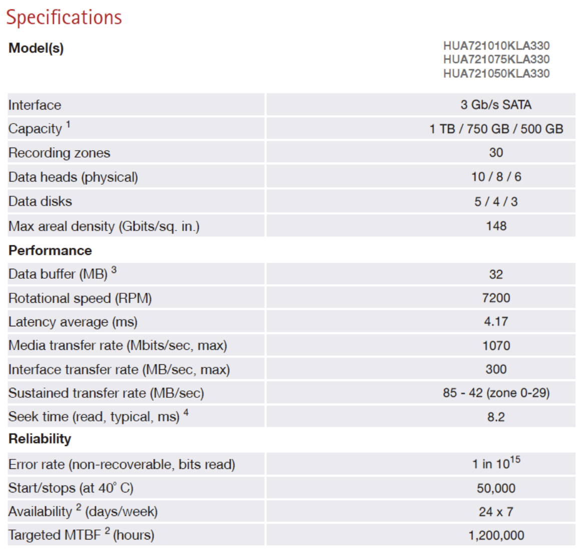
1 IO performance

Suppose a machine has 8GB of DRAM that it can use as a cache. Suppose the machine also has a 1TB disk of the kind specified to the right of this text and that this disk stores 128 GB (2^{37} bytes) of data. In particular there are 2^{30} data records, each record containing an 8-byte key and a 120 byte value ($128 = 2^7$ bytes per record).

The records are stored in a tree. Each *leaf* is a 128KB (2^{17}) block of data stored contiguously on disk. In order to accommodate new record inserts, initially each leaf block is half full, so each initially contains 512 (2^9) records. Thus, there are 2^{21} leaf blocks on disk. Assume that these leaf blocks are stored in key-sorted order contiguously on some 256 GB (2^{38} byte) region of the disk.

We'll call the leaf nodes level 0 (L0) of the tree. Each leaf's parent is a level 1 (L1) node, and so on.

The internal and root nodes (level l nodes for $l > 0$) of the tree are 4KB blocks, each stored contiguously on disk. Each internal node is an array of records: *key, addr* indicating the disk address *addr* where the 128KB leaf block whose earliest key is *key* is stored. Assume that all nodes for any given level l of the tree are stored contiguously on disk in sorted order.



Specifications

Model(s)	HUA721010KLA330 HUA721075KLA330 HUA721050KLA330
Interface	3 Gb/s SATA
Capacity [1]	1 TB / 750 GB / 500 GB
Recording zones	30
Data heads (physical)	10 / 8 / 6
Data disks	5 / 4 / 3
Max areal density (Gbits/sq. in.)	148
Performance	
Data buffer (MB) [3]	32
Rotational speed (RPM)	7200
Latency average (ms)	4.17
Media transfer rate (Mbits/sec, max)	1070
Interface transfer rate (MB/sec, max)	300
Sustained transfer rate (MB/sec)	85 - 42 (zone 0-29)
Seek time (read, typical, ms) [4]	8.2
Reliability	
Error rate (non-recoverable, bits read)	1 in 10^{15}
Start/stops (at 40° C)	50,000
Availability [2] (days/week)	24 x 7
Targeted MTBF [2] (hours)	1,200,000

- How many levels of tree do you need to index this full data set? (7 points)

- Assuming you want to provide high throughput for a workload that randomly accesses records in the leaves. Is it a reasonable engineering decision to cache all of the internal nodes of the tree in their entirety? For full credit, calculate the size of these nodes *and* make a quantitative engineering argument for whether caching this amount of data is a reasonable thing to do in this system. (7 points)

Suppose a workload generator generates a long series of 2^{30} random record insert operations *INSERT key value*. Notice that each insert operation requires a read-modify-write of a leaf block.

- Estimate the steady state throughput for this workoad assuming a *synchronous update in place* strategy in which update i is completed before update $i+1$ is begun, and assuming that none of the leaf blocks become full during the run. (7 points)

- Design a different update strategy that maximizes throughput (7 points)

- Estimate the throughput your solution achieves (7 points)

2 Short answer

- In the context of encryption protocols, what is the *nonce verification* rule? (5 points)
  - What conditions must hold to allow a node to apply the rule?
  - What conditions hold after the rule is applied?
  - Explain what the rule means and why such a rule is needed.
- Two-factor authentication is an alternative to passwords. Give an example of two-factor authorization and explain its advantages compared to passwords. (5 points)
- In the context of file systems, what is a *hard link*? (5 points)
- Why is it harder to implement hard links in the DOS/Windows FAT file system than in the Unix Fast File System (FFS)? (5 points)

- Define the ACID properties of a transaction (give the name and definition of each) (5 points)

- I am running a web service on a single web server machine. My users are observing slow response time to their requests. I split my users so that half send their requests to the original machine and half send their requests to an identical second machine. My users' response time gets better by *much more* than a factor of two. What is a likely reason for such a *superlinear speedup* (5 points)

3 Distributed commit

A commit protocol attempts to get all nodes to agree on a value. We are most interested in solving this problem assuming an *asynchronous fair network*.

An asynchronous fair network assumes that nodes may be arbitrarily slow, that the network may drop messages, reorder messages, or arbitrarily delay messages. However, it assumes that if a node periodically resends a message until it knows that the destination has received the message, then the message will eventually be received by the destination.

- Engineers have to make reasonable trade-offs, and it may sometimes be reasonable to ignore risks if they are really rare. For example, although assuming that the network is asynchronous might make sense for the Internet, perhaps in a machine room it is overly conservative. In particular, suppose I have a cluster of 10 machines, each machine has a 1 Gbit/s full-duplex Ethernet link, all Ethernet links go directly to a switch, the switch is extremely fast and has plenty of buffer space to queue packets instead of dropping them, and no machine ever sends at a rate higher than 1 Mbit/s. In such a system, packets should never be dropped and should arrive within a milliseconds of being sent, so rather than assuming no bound on message delivery, perhaps I could assume (conservatively?) that with an appropriate retransmission protocol, a packet is always received within 10 seconds of being sent unless the receiver has crashed. I could then design my protocol to work assuming a synchronous network rather than an asynchronous one.

  Why might it be dangerous to make such an assumption? (Be specific) (7 points)

- Suppose that two nodes, A and B, each store a local value ($value_A$ and $value_B$). Design a protocol that, under the asynchronous fair network model, guarantees that they first agree on the value max = MAX($value_A$, $value_B$) and then simultaneously print the value max to their screens *assuming that neither node crashes.* (7 points)

Not only do we want to solve agreement assuming an asynchronous fair network, we also want some fault tolerance. For example, we want to guarantee that the protocol works even if f nodes permanently crash.

Recall the 7-line nonblocking fault-tolerant consensus algorithm that required n = 3f+1 nodes (e.g., 4 nodes to tolerate f = 1 crash.) Initially each node gets to unilaterally vote for RED or BLUE, but in subsequent rounds a node's vote depends on what other nodes vote for so that eventually all nonfaulty nodes settle on the same value.

Here is the pseudo-code for a node

```
$e$ = 0;                    // election number
$c$ = RED or BLUE;          // vote this election
while (1){
    $e = e + 1$;
    send (VOTE, $e$, $c$) to all  // In background, keep periodically
                                  // sending to node i until I receive i's
                                  // vote for round $e$+1
    $VOTES$ = receive (VOTE, $e$, RED or BLUE) from 2f+1 nodes (including self) for election e
    $c$ = MAJORITY($VOTES$)
}
```

To determine the consensus value, I ask the nodes to tell me their current c values for a particular round and wait until I receive n-f = 3 replies. If all three replies match, then the system has reached consensus on the value in that reply, and any subsequent read would return the same value. If the replies don't match, consensus has not been reached, and I can check again later.

Modify above algorithm to work with a COMMIT/ABORT voting rule instead of majority RED/BLUE voting. In particular, your solution should guarantee five properties

1. Nontriviality – either COMMIT or ABORT may be decided
2. Safety – the system decides COMMIT only if all nodes initially vote COMMIT (as in two-phase commit, the system may decide ABORT in a range of situations.)
3. Consensus – eventually all non-crashed nodes decide the same thing and once the system has reached a decision the decision does not change
4. Visibility – I can learn the final decision (or determine that no decision has yet been reached) by querying the state of any $n - f$ live nodes
5. Eventual liveness – assuming the network is an asynchronous fair network and that nodes automatically resend messages until they are known to be received, then eventually all non-crashed nodes agree on some value

- State your protocol for nonblocking COMMIT/ABORT agreement (7 points)

- Prove that your protocol works by proving that it guarantees each of the 5 required properties (14 points)

(contd)