6.033 FINAL EXAM REVIEW SHEET

A case for RAID

RAID-1: Mirrored Disks
Mirrored disks are the most expensive option. Every write to a data disk is also a write to a check disk. Since these writes happen in parallel, it should be at least half as fast.
Main drawback: duplicating all disks means doubling the cost of the disks or using only halving storage capacity.

RAID-2: Hamming Code for ECC
If all data bits in a group are read or written together, there is no impact on performance. Reads of less than the group size require reading the whole group for error checking, and writes to a portion of the group require reading the whole group, modifying the data, and writing the full group, including check information.
1 parity disk can detect 1 error, but to correct an error we need enough check disks to identify the disk with the error. For 10 data disks, we need 4 check disks.
Good for supercomputers, bad for TPS.

RAID-3: Single Check Disk Per Group
Only 1 parity bit is needed to detect an error. Information on a failed disk can be reconstructed by calculating the parity of the remaining good disks and comparing bit by bit to the parity calculated for the original full group. If the check disk is the failure, read all data disks and store group parity in the replacement disk.
Drawback vs RAID-2: Soft errors can be corrected without rereading a sector. Also, the extra check information for each sector is not needed, increasing disk space.
Good for supercomputers but bad for TPS

RAID-4: Independent Reads/Writes
No longer spreads individual transfer information across several disks, keep each individual unit in a single disk. Allows parallelism for reads - the ability to do more than one IO per group at a time.
Parity calculation is much simpler: new parity = (old data xor new data) xor old parity. The check disk in a group must be read and written with every small write in that group, making it the bottleneck.

RAID-5: No single check disk
Distributes data and checks across all the disks - including the check disks. Because of this, RAID 5 can support multiple individual writes per group.
Best of both worlds: small read/writes perform close to the speed of a level 1 RAID and it keeps the large transfer performance and high storage capacity of RAID 3/4. RAID-5 is good for supercomputers or transaction processing.

The Google File System

GFS is a scalable distributed file system for large distributed data-intensive application. It has fault tolerance and delivers high aggregate performance to a large number of clients.

Assumptions
Component failures are common
Modest number of very large file
Reads are large streams or small & random
Large sequential writes that append data
Multiple clients concurrently append to the same file
High bandwidth is more important than low latency

New Operations
Snapshot creates a copy of a file or directory at low cost
Record append allows multiple clients to append to the same file and guarantees the atomicity of each append.

Architecture
A GFS cluster has one master and many chunkservers, and is accessed by many clients
Files are divided into 64MB chunks
Each chunk is replicated on multiple chunkservers
The master maintains all system metadata, chunk lease management, garbage collection, and chunk migration.
File data is not cached by the client or the chunkserver
Metadata is cached by the client
GFS does not have a per-directory data structure that lists all the files in that directory.
It does not support aliases for the same file (symbolic links)
GFS represents its namespace as a lookup table mapping full pathnames to metadata, stored in memory using prefix compression.

Master
Clients do not read and write files through the master, they ask the master which chunkservers it should contact, and interact with this chunkserver for subsequent operations
The master executes all namespace operations, manages chunk replicas, makes placement decisions, creates new chunks and replicas, coordinates system-wide activities to keep chunks fully replicated, to balance load, and to reclaim unused storage.

Metadata
Master stores 3 types of metadata: File/chunk namespaces, mapping from files to chunks, and locations of each chunk's replicas.
All metadata is kept in the master's memory, so master operations are fast.
Master does not keep a record of which chunkservers have a replica of a given chunk. It polls chunkservers for that information at startup. A chunkserver has the final word over what chunks it does or does not have on its own disks.
The operation log contains a historical record of critical metadata changes.
The operation log is the only persistent record of metadata and serves as a logical time line that defines the order of concurrent operations.
Files, chunks, and their versions are all uniquely identified by the logical times at which they were created.
The master recovers its state by replaying the operation log.
The master's state is checkpointed whenever the log grows too large, so it can recover by loading the latest checkpoint and replaying the log records after that.

Leases
The lease mechanism is designed to minimize management overhead at the master. Each mutation happens at all of the chunk's replicas.
Leases are used to maintain a consistent mutation order across replicas.
The global mutation order is defined first by the lease grant order chosen by the master, and then by a serial order that the machine granted the lease chooses.

Consistency model

GFS has a relaxed consistency model
File namespace mutations are atomic.
A file region is consistent if all clients will see the same data.
A file region is defined if it is consistent and all clients will see the last mutation in its entirety.
When a mutation succeeds without interference from other writers, the affected region is defined.
Concurrent successful mutations leave the region undefined but consistent.
A failed mutation makes the region inconsistent.
Mutations may be writes or record appends
A record append causes data to be appended atomically at least once even in the presence of concurrent mutations, but at an offset of GFS's choosing.
GFS may insert padding or record duplicates in between. These occupy inconsistent regions and are usually small.
After a sequence of successful mutations the file region is guaranteed to be defined and contain the data written by the last mutation
A chunk is lost irreversibly only if all of its replicas are lost before GFS can react, typically within minutes. Even in this case, it becomes unavailable rather than corrupted.

Atomic Record Appends

In a record append, the client specifies the data, and GFS appends it to the file at least once atomically at an offset of GFS's choosing and returns that offset to the client.
If a record append fails at any replica, the client retries the operation.
As a result, replicas of the same chunk may contain different data possibly including duplicates of the same record in whole or in part.
GFS does not guarantee that all replicas are identical, just that the data is written at least once as an atomic unit.

Snapshot

The snapshot operation makes a copy of a file or directory tree almost instantaneously, using standard copy-on-write techniques.

Stale Replica detection

Chunk replicas become stale if a chunkserver fails and misses mutations to the chunk while it is down.
For each chunk, the master maintains a chunk version number to distinguish up-to-date and stale replicas
Whenever the master grants a new lease on a chunk, it increases the chunk version number and informs the up-to-date replicas.
The master removes stale replicas in its garbage collection. Before that, it considers stale replicas to not exist when it replies to client requests for chunk information.

Fault Tolerance

We cannot trust the machines, or the disks
Two strategies keep the system highly available: fast recovery and replication
Fast recovery: Both the master and the chunkserver are designed to restore their state and start in seconds no matter how they were terminated.
Chunk replication: Each chunk is replicated on multiple chunkservers on different racks
Master replication: The master state is replicated for reliability. Its operation log and checkpoints are replicated on multiple machines.
Each chunkserver independently verifies the integrity of its own copy by maintaining checksums