Section 11: File Systems

CS 162

Nov 12, 2021

Contents

1 Vocabulary

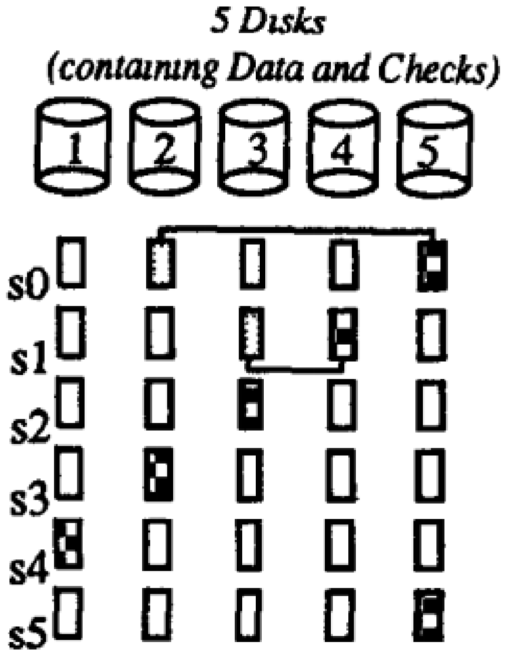
- **Fault Tolerance** The ability to preserve certain properties of a system in the face of failure of a component, machine, or data center. Typical properties include consistencies, availability, and persistence.
- **Transaction** - A transaction is a unit of work within a database management system. Each transaction is treated as an indivisible unit which executes independently from other transactions. The ACID properties are usually used to describe reliable transactions.
- **ACID** - An acronym standing for the four key properties of a reliable transaction.
  - *Atomicity* - The transaction must either occur in its entirety, or not at all.
  - *Consistency* - Transactions must take data from one consistent state to another, and cannot compromise data integrity or leave data in an intermediate state.
  - *Isolation* - Concurrent transactions should not interfere with each other; it should appear as if all transactions are serialized.
  - *Durability* - The effect of a committed transaction should persist despite crashes.
- **Idempotent** - An idempotent operation can be repeated without an effect after the first iteration.
- **Log** - An append only, sequential data structure.
- **Checkpoint** - Aka a snapshot. An operation which involves marshaling the system's state. A checkpoint should encapsulate all information about the state of the system without looking at previous updates.
- **Write Ahead Logging (WAL)** - A common design pattern for fault tolerance involves writing updates to a system's state to a log, followed by a commit message. When the system is started it loads an initial state (or snapshot), then applies the updates in the log which are followed by a commit message.
- **Serializable** - A property of transactions which requires that there exists an order in which multiple transactions can be run sequentially to produce the same result. Serializability implies isolation.
- **ARIES** - A logging/recovery algorithm which stands for: Algorithms for Recovery and Isolation Exploiting Semantics. ARIES is characterized by a 3 step algorithm: Analysis, Redo, then Undo. Upon recovery from failure, ARIES guarantees a system will remain in a consistent state.
- **Logging File System** - A logging file system (or journaling file system) is a file system in which all updates are performed via a transaction log ("journal") to ensure consistency in case the system crashes or loses power. Each file system transaction is written to an append-only redo log. Then, the transaction can be committed to disk. In the event of a crash, a file system recovery program can scan the journal and re-apply any transactions that may not have completed successfully. Each transaction must be idempotent, so the recovery program can safely re-apply them.
- **Metadata Logging** - A technique in which only metadata is written to the log rather than writing the entire update to the log. Modern file systems use this technique to avoid duplicating all file system updates.
- **EXT4** - A modern file system primarily used with Linux. It features an FFS style inode structure and metadata journaling.
- **Log Structured File System** - A file system backed entirely by a log.

- **RAID** - A system consisting of a Redundant Array of Inexpensive Disks invented by Patterson, Gibson, and Katz.

  The fundamental thesis of RAID is that in most common use cases, it is cheaper and more effective to redundantly store data on cheap disks, than to use/engineer high performance/durable disks.

- **RAID I** - Full disk replication. With RAID I two identical copies of all data is stored. If disk heads are not fully synchronized, this can decrease write performance, but increase read performance.

- **RAID V+** - Striping with error correction. In RAID V, 4 sequential block writes are placed on separate disks, then a 5th parity block is written by XORing the data blocks on the same stripe. RAID VI uses the EVENODD scheme to encode error correction. In general, Reed Solomon coding can be used for an arbitrary number of error correcting disks.



  Note: Due to the large size of disks in practice, RAID V is no longer used in practice, because it is too likely that a second disk will fail while the first is recovering. RAID VI is usually combined with other error recovery techniques in practice.

- **Eventual Consistency** - A weaker form of a consistency guarantee. If a system is eventually consistent, it will converge to a consistent state over time.

2 Comparison of File Allocation Strategies

In lecture three file allocation strategies were discussed: (a) Indexed files. (b) Linked files. (c) Contiguous (extent-based) allocation.

Each of these strategies has advantages and disadvantages, which depend on the goals of the file system and the expected file access patterns. For each of the following situations, rank the three designs in order of best to worst. Give a reason for your ranking.

1. You have a file system where the most important criteria is the performance of sequential access to very large files.

1. c (extent-based)
2. b (linked)
3. a (indexed)

It is easy to see that (c) is the best structure for sequential access to very large files, since in (c) files are contiguously allocated and the next block to read is physically the next on the disk. No seek time to find the next block, and each block will be read sequentially as the disk head moves.

Both (b) and (a) require some look up operation in order to know where the next block is. However, (a) may be slightly more expensive, since for "very large files", multiple disk accesses are required to read the indirect blocks.

2. You have a file system where the most important criteria is the performance of random access to very large files.

1. c (extent-based)
2. a (indexed)
3. b (linked)

(c) is still the best structure here: just need to use an offset.

(a) will probably need to look at some levels of indirect blocks in order to find the right block to access (we are dealing with very large files).

(b) is absolutely the worst structure. In fact, in order to find a random block, we will need to traverse a linked list of blocks, which will take a time linear in the offset size.

3. You have a file system where the most important criteria is the utilization of the disk capacity (i.e. getting the most file bytes on the disk).

1. b (linked)
2. a (indexed)
3. c (extent-based)

(c) can suffer heavily of external fragmentation, especially for large files. So it is not the best structure for getting the most bytes on the disk, since lots of space will be wasted. However, for small files and large block size, (c) might prove to be better than (a) and (b).

(a) and (b) stuctures are generally more suitable for this question. The metadata overhead for (b) is likely to be smaller than the one for (a), since it only needs pointers to the next allocated block rather than an entire block (or blocks) which may or may not be totally used.