Section 13: Networking, Distributed File Systems

CS 162

August 5, 2020

Contents

1 Vocabulary **2**

2 Networking **4**

3 Distributed File Systems **7**

1 Vocabulary

- **TCP** - Transmission Control Protocol (TCP) is a common L4 (transport layer) protocol that guarantees reliable in-order delivery. In-order delivery is accomplished through the use of sequence numbers attached to every data packet, and reliable delivery is accomplished through the use of ACKs (acknowledgements).
- **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.
- **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.
- **Checksum** - A mathematical function which maps a (typically large) input to a fixed size output. Checksums are meant to detect changes to the underlying data and should change if changes occur to the underlying data. Common checksum algorithms include CRC32, MD5, SHA-1, and SHA-256.
- **Replication** - Replication or duplication is a common technique for preserving data in the face of disk failure or corruption.

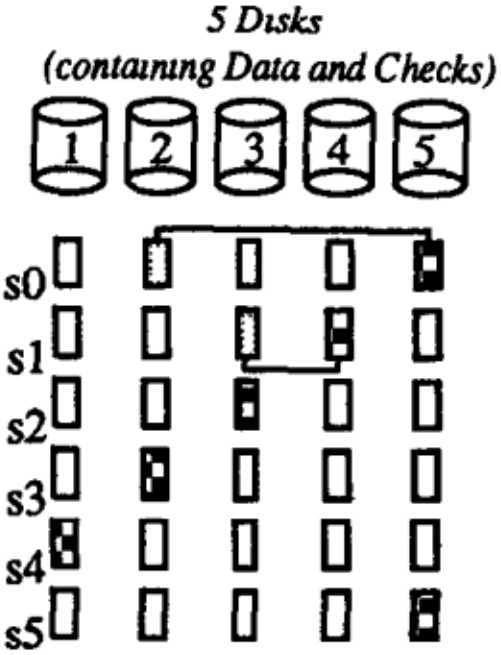
  If a disk fails, data can be read from the replica. If a sector is corrupted, it will be detected in the checksum. The data can then be read from another replica.

- **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.

- **Network File System (NFS)** - A distributed file system written by Sun. NFS is based on a stateless RPC protocol. Buffers are **write behind**. Few strong consistency guarantees on parallel writes. NFS is **eventually consistent**.

- **Andrew File System (AFS)** - A distributed file system written at CMU. Full files are buffered locally upon `open`. Buffers are **write back** and only flushed on `close`. File contents follow "last write wins" semantics.

2 Networking

1. (True/False) IPv4 can support up to 2^{64} different hosts.

   False, it has 32 bits per IP address

2. (True/False) Port numbers are in the IP header.

   False, they are in the transport layer. (TCP/UDP).

3. (True/False) UDP has a built in abstraction for sending packets in an in order fashion.

   False, this is a part of the TCP protocol. In UDP there is no notion of a sequence number.

4. (True/False) TCP provide a reliable and ordered byte stream abstraction to networking.

   True.

5. (True/False) TCP attempts to solve the congestion control problem by adjusting the sending window when packets are dropped.

   True.

6. In TCP, how do we achieve logically ordered packets despite the out of order delivery of the physical reality? What field of the TCP packet is used for this?

   The seqno field.

7. Describe how a client opens a TCP connection with the server. Elaborate on how the sequence number is initially chosen.

   3 way handshake. Client sends a random sequence number (x) in a syn packet. Server sees this and sends another random seqeuence number back (y) in addition to acknowledging the sequence number that it received from the client (sends back x+1) in a syn-ack packet. Client acknowledges this sequence number in an ack packet by sending back y+1.

   It is important to randomize the sequence number so an off path attacker cannot guess it and send spurious packets to you.

8. Describe the semantics of the acknowledgement field and also the window field in a TCP ack.

   The acknowledgement field says that the receiver has received all bytes up until that number (x). The window field says how many additional bytes past x the receiver is ready to receive.

9. List the 5 layers specified in the TCP/IP model. Layering adds modularity to the internet and allows innovation to happen at all layers largely in parallel. What is the function of each layer?

   (a) Physical Layer: the physical layer is responsible for the delivery of raw bits from one endpoint to another. It includes ethernet, fiber, and other mediums of data transmission.

   (b) Datalink Layer: this layer adds the packet abstraction and is responsible for the local delivery of packets between devices within the same network (usually a LAN but can also include WANs). Devices here include switches, bridges, and network interface cards (NICs).

(c) Network Layer: this layer is the skinny waist of the internet and routing algorithms here only speak IP. The network layer is responsible for global delivery of packets across one or more networks.

(d) Transport Layer: this layer provides host to host or end to end communication. Because processes operate in terms of streams of data rather than individual packets, the transport layer handles the multiplexing of packets into streams, end to end reliable delivery, and introduces the concept of flows. This layer lives in the operating system, and TCP and UDP are examples of popular transport layer protocols.

(e) Application Layer: the application layer provides network support for applications. The protocols that live here include: HTTP, FTP, DNS, SSH, TLS, etc.

10. The end to end principle is one of the most famed design principles in all of engineering. It argues that functionality should **only** be placed in the network if certain conditions are met. Otherwise, they should be implemented in the end hosts. These conditions are:

    – Only If Sufficient: Don't implement a function in the network unless it can be completely implemented at this level.
    – Only If Necessary: Don't implement anything in the network that can be implemented correctly by the hosts.
    – Only If Useful: If hosts can implement functionality correctly, implement it in the network only as a performance enhancement.

    Take for example the concept of reliability: making all efforts to ensure that a packet sent is not lost or corrupted and is indeed received by the other end. Using each of the three criteria, argue if reliability should be implemented in the network.

    (a) Only If Sufficient

    NO. It is not sufficient to implement reliability in the network. The argument here is that a network element can misbehave (i.e. forwards a packet and then forget about it, thus not making sure if the packet was received on the other side). Thus the end hosts still need to implement reliability, so it is not sufficient to just have it in the network.

    (b) Only If Necessary

    NO. Reliability can be implemented fully in the end hosts, so it is not necessary to have to implement it in the network.

    (c) Only If Useful

    Sometimes. Under circumstances like extremely lossy links, it may be beneficial to implement it in the network. Lets say a packet crosses 5 links and each link has a 50% chance of losing the packet. Each link takes 1 ms to cross and there is an magic oracle tells the sender the packet was lost. The probability that a packet will successfully cross all 5 links in one go is $(1/2)^5 = 3.125\%$. This means the end hosts need to try 32 times before it expects to see the packet make it through, taking up to # of tries $\times$ max # of links per try $= 32 \times 5 = 160$ ms. Likewise at each hop, if the router itself is responsible for making sure the packet made it to the next router, each router would know if the packet was dropped on the link to the next router. Thus each router only has to send the packet until it reaches the next router, which will be twice on average. So to send this packet, it will take on average # of tries per link # number of links $= 2 \times 5 = 10$ ms. This is a huge boost in performance, which makes it useful to implement reliability

in the network under some cases.

3 Distributed File Systems

Distributed file systems must provide the same access API as standard file systems. That access API quickly becomes non standard in the face of concurrent operations.

Compare the performance of AFS, NFS, EXT4, and EXT4 with the streaming api. Assume processes run on separate machines wherever it is applicable.

1. **open**

   In AFS, open is an expensive operation in terms of performance as it requires transferring the entire file over the network.

   In NFS, little data is transferred, but the client now begins to poll the server for changes to the file.

   Both open and fopen finish quickly for ext4.

2. **write**

   In AFS, this operation occurs faster than NFS, and requires no network activity. It operates entirely on the local copy of the file.

   In NFS, this operation is slow as it requires an RPC.

   With ext4 write is relatively slow compared to fwrite as it involves a syscall and i/o (fwrite might flush its buffer which involves syscall and i/o, but typically it will not).

   With the streaming api, this operation is relatively fast as it's likely userspace buffered.

3. **read**

   In AFS not network I/O will occur. It operates entirely on a local copy of the file.

   In NFS, and the streaming API expensive I/O is not likely to occur, but can occcur if the data being read is not cached.

   For ext4, this operation still requires a syscall and potentially an I/O operation.

4. **close**

   In AFS, this is an expensive operation. It involves transferring the entire file over the network.

   In NFS, little data is transferred since all buffers are already flushed, and the RPC is stateless.

   For ext4, close finishes relatively quickly. Very little I/O will occur.

   For the streaming api, the entire userspace buffer must be flushed to disk, which is potentially large.

Now compare the behavior of AFS, NFS, EXT4, and EXT4 with the streaming api (with a large buffer). Assume processes run on separate machines wherever it is applicable and that buffers are only flushed when filled and upon close.

1. Process A and Process B write to a file simultaneously, then Process A closes the file, then Process B closes the file.

   In AFS, only Process B's writes are reflected in the final file.

   In NFS, it is undefined what the contents of the file are.

   For ext4, it is undefined what the contents of the file are (writes aren't atomic).

   For the streaming api (assuming the buffer isn't flushed on write), only Process B's writes are

reflected in the final file.

2. Process A increments an integer (using read and write), 1 second later, Process B increments the integer too. Both processes close the file.

   In AFS, NFS, and the streaming api, the number is incremented once.

   In ext4, the number is incremented twice.

3. Process A increments an integer (using read and write), 1 minute later, Process B increments the integer too. Both processes close the file.

   In AFS and the streaming api, the number is incremented once.

   In ext4 and NFS the number is incremented twice.

   Note that these answers are different because NFS tends to poll on a timescale of seconds. Therefore it is likely that in the one second case NFS will not have pulled the updated, whereas after a minute, we expect the update to be reflected locally.

4. Process A writes a large amount of data, then Process B writes a large amount of data. Then Process B closes the file, then Process A closes the file.

   In AFS only Process A's data is reflected in the file.

   In NFS, only Process B's data is reflected in the file.

   In ext4, only Process B's data is reflected in the file.

   In the streaming api, the file is mostly Process B's data, but ends with Process A's data.