Section 4: Week 7: Tree-Structures and Fault Tolerant Design

Nate Bachmeier

TIM-8120: Distributed Systems

November 3rd, 2019

North Central University

# Fault Tolerant Design

## Influence of Hierachy

Generally speaking, there are two mechanisms for modeling distributed systems, lists, and trees. A list can efficiently manage small groups of related nodes; however, it can become cumbersome with more massive sets. Trees allow for more expansive designs as the system can hierachially describe the problem, through multiple levels of control. Consider the difference between Domain Name Services (DNS, tree) and NetBIOS (list). NetBIOS can easily manage a small branch office, not the Internet, because its simple flat list structure is *globalized*. In contrast, DNS has multiple sub domains, with each subdomain owned by heterogeneous service providers. Since each sub domain holds a specific set of children, read and write operations can be *localized*.

## Influence of Partitioning

Localized designs are inheritently more performant and fault tolerant, because of the containment of both scale and blast radius. Imagine a scientific dataset that has grown to several petabytes in size. The storage network would need to decompose this logical file system into multiple blocks and then replicate it across multiple physical servers. These physical servers will run into mechanical failures, such as disk corruption or power outages. When these outages occur, other nodes need to Setup, Challenge, and Repair (SCR) the missing data in an efficient manner (Chen & Curtmola, 2017). The time necessary to perform that repair operation is proportional to the size of each block and the system’s ability to horizontally scale the reconstruction over multiple peers. Assume that 1TB of the dataset has entered a failed state and needs to recover across a 10GB/s network (see Table 1). If only one virtual peer has a copy of the data, then the system will heal in 102.4 seconds. Then constrast that with the smaller block size of 128GB and which can economically be sprawled across many servers, reaching an MTTR of under a second!

|  |  |  |  |
| --- | --- | --- | --- |
|  | Repair 1TB of Data | | |
| Block Size | **Virtual Peers** | **Num Blocks** | **MTTR (s)** |
| 1024 GB | 1 | 1024 | 102.4 |
| 8 | 1024 | 12.8 |
| 16 | 1024 | 6.4 |
| 512 | 2 | 2048 | 51.2 |
| 16 | 2048 | 6.4 |
| 32 | 2048 | 3.2 |
| 256 | 4 | 4096 | 25.6 |
| 32 | 4096 | 3.2 |
| 64 | 4096 | 1.6 |
| 128 | 8 | 8192 | 12.8 |
| 64 | 8192 | 1.6 |
| 128 | 8192 | 0.8 |

Table 1: Mean Time to Recover

## Influence of Fail-Over Groups

Proxy servers and similar brokers operate on ephemerial requests and need fault tolerance to come from a different source. One strategy is to maintain a target group of service instances and then monitor their availability (see Figure 1). The monitoring can come from at least three reference points: (1) the network operating system, (2) the observed traffic of the broker itself, and (3) a local health agent on the service instance. As new requests arrive, the broker can use the Observed Health State Store (OHSS) to select the most appropriate receiver.

A recovery policy could also exist to manage any Service Level Objectives (SLO) of the backend application. For instance, if the backend application needs to be highly available, then the broker could be augmented to trap specific exceptions and automatically route to another node. Other systems need to optimize across a different metric, such as more consistent response time, and would choose completely different behaviors.

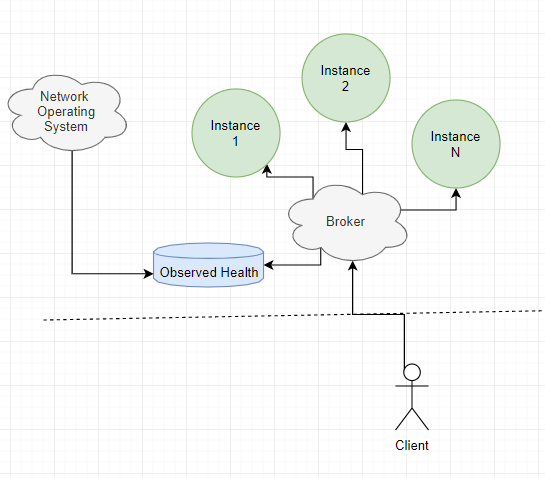


Figure 1: Broker Fail-Over

## Influence of Geo-Redundancy

Cloud Service Providers enable fault tolerance across multiple regions, so that entire data centers can fail without impacting applications uptime (see Figure 2). The scheme starts with the deployment of the service stack into two or more locations, such as Seattle and New York. Next, data store replication enables the sites to be kept in sync. Finally, the user can discover the most performant service stack instance from a location aware Canolical Naming Service (CNAME). That system can take into account latency and other metrics, similar to the proposed Fail-Over Group solution.

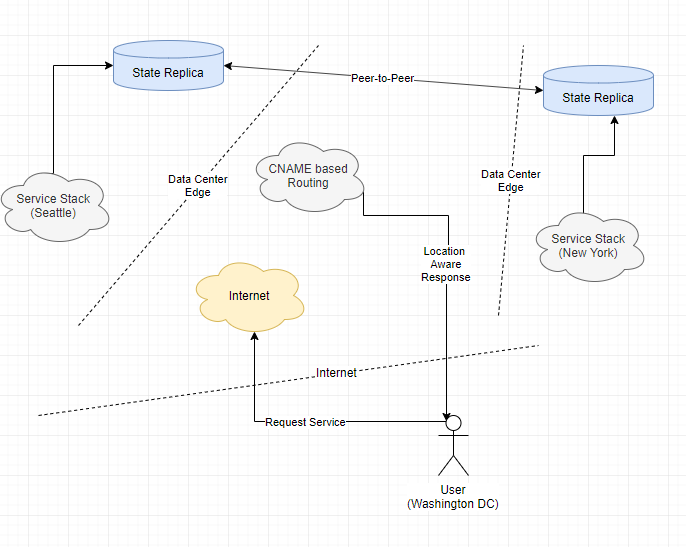


Figure 2: Multi-Region Deployment

## Influence of Concensus

The physical distance between the sites forces the need for eventual consistency protocols that range in complexity from (a) the latest time stamp wins, (b) Paxos algorithms, and (c) Byzantine General’s solutions (Zhao, 2014). The latest timestamp wins, is easy to understand but needs to rely on highly reliable distributed clock synchronization, an open research problem in itself (Ting, Chun-Yang, Di, Xiao-ming, & Heng, 2014). Under Paxos (see Figure 3), multiple rounds of prepare, accept, and learn phases take place to gain concensus. This elegant protocol can efficiently recognsile a single systems image, provided none of the nodes are malicious. If malicious or erroneous nodes exist, then 3f +1 cross validations need to take place (Zhao, 2014).

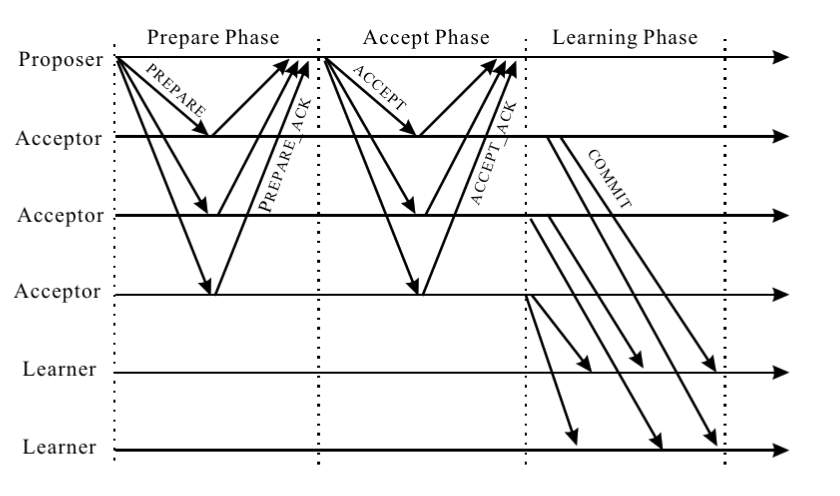


Figure 3: Paxos Concensus (Zhao, 2014, p. 196)

# Data Structures for File Server Organization

A Distributed File System (DFS) needs to leverage multiple data structures to persist and retrieve content (see Table 2). The Contoso Network File System (see Figure 4) forces users to request a security token from their *Security Token Service (STS)*. The STS uses a B-tree to scan through an index to find the associated authorization policy. Next, the client can send their storage operation and user context to the *Storage Controller*, which places the payload in a Durable Priority Queue (DPQ), before acknowledging the client’s message received. A controller thread pulls the next DQP message and uses the *Storage Directory* to identify the involved replicas. File objects can be arbitrarily large, with blocks spanning one or more modules. A tree structure maintains the replica information using linked lists at child levels, for dynamic expansion across physical resources.

|  |  |  |
| --- | --- | --- |
| Name | Description | Example Use Case |
| Trees | Parent-child relationship model | The hierarhcial topology of object replica information |
| Stacks | Last-In/First-Out (LIFO) list | An ancestry list derived from a recursive traversal |
| Queues | First-In/First-Out (FIFO) list | An in-order pending operation buffer |
| Linked List | Flexible reorderable list | Ordered file list within a folder |

Table 2: File System Data Structures

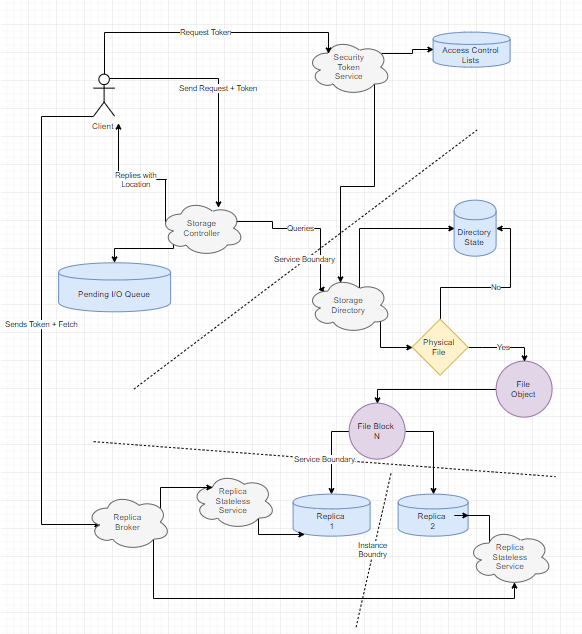


Figure 4: Contoso DFS Logical View

Contoso’s DFS is capable of supporting multiple concurrent application workloads, as there is a decoupling of incoming requests from event processing. This behavior is achived through the Pending I/O queue and allows the service to remain highly available even during burst traffic. The system must maintain multiple copies of the content to ensure high durability of stored content. User traffic can be load balanced across these different copies to increase read performance. Many scenarios, such as centralized service logs, can tolerate data within the file being slightly out of order. For these append only use cases, the DFS could support multiple concurrent writers to the same object, by creating new blocks for each producer, and recording them in the *Storage Directory* tree. These partial blocks are reconciled, just-in-time for the consumer.

# Shortest Path Algorithms