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 more consistent response times and would choose completely different behaviors.

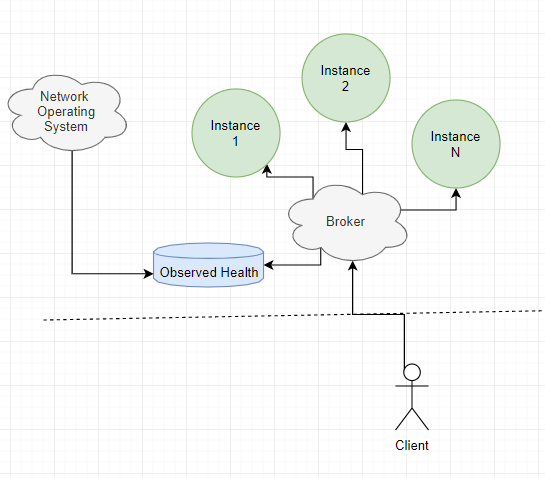


Figure 1: Broker Fail-Over