2022-02-08

Title to be decided

David Kleingeld

Email

Contents

1 Background

1.1 Distributed Computing

When state of the art hardware is no longer fast enough to run a system the only option is scaling out. Then there is a choice, do you buy expansive, reliable high performance supercomputer or commodity servers connected by Ip and ethernet? This is the choice between High Performance (HPC) and Distributed Computing. With HPC faults in the hardware are rare and can be handled by restarting, easing development. In a distributed context faults are the norm, restarting the entire system is not an option or you would be down all the time. Resiliance against faults comes at an, often significant, cost to performance. It may also limit scalability. As the scale of a system increases so does the frequancy with which one of the parts fails. Even the most robust part will fail and given enough of them the system will fail frequantly. Therefore at very large scales HPC is not even an option.

1.2 Faults and Delays

Before we can build a fault resistant system we need to know what we need to keep in mind. While hardware failures are, the norm in distributed computing, faults are not the only issue to keep in mind.

It is entirely normal for the clock of a computer to run slichty to fast or to slow. The drift will be tens of milliseconds [1] unless special measures are taken[1]. Worse a process can be paused and then resumed at any time. Such a pause could be because the process thread is pre-emted, because its virtual machine is paused or because the process was stopped and resumed after a while[2].

In a distributed system the computers that form the system or *the nodes*, are connected by IP over ethernet. Ethernet gives no guarentee a packet is deliverd on time or at all. A node can be unreachable before suddenly working fine again.

[1] One could synchronize the time within a datacenter or provide nodes with more accurate clocks

[2] On linux by sending SigStop then SigCont

A system model is an abstraction defining what an algorithm can assume. Regarding timing there are three models.

1. The Synchronous model allowes an algorithm to assumes that clocks are synchronized within some bound and network traffic will arrive within a fixed time.
2. The Partially synchronous model is a more realistic model. Most of the time clocks will be correct within a bound and network traffic will arrive within a fixed bound. However sometimes clocks will drift unbounded and some traffic might be delayed forever.
3. The Asynchronous model has no clock, it is very restrictive.

For most distributed systems we work with the Partially Synchronous model. We assume hardware faults cause a crash from which the node can be recoverd later. Either automatically as it restarts or after maintenance.

1.3 Consistancy Algorithms

1. quorums
2. consensus as a service (zookeeper)
3. paxos
4. raft

1.4 File System

A file system is split into two parts, the files and the directory structure. File properties, or metadata, such as its name, identifier, size etc are stored in the directory. Typically the directory entry itself only contains the file name and its unique identifier. Using the identifier the other metadata for the file can be fetched. The content of the file is split into blocks these blocks are stored on stable storage such as an hard drive or ssd. The file system defines an API to allow modifying the files system providing ways to *create*, *read*, *write*, *seek* and *trunctate* files.

Usually the system adds a distinction between open and closed files. The apis *read write* and *seek* are then only allowed on open files. This makes it possible to provide some concistancy guarentees in a concurrent envirement. For example allowing a file to be opend only if it was not already open. This can prevent a user from corrupting data by writing from multiple processes at the same place in the file. There is no risk to reading the same file from multiple process, even while appending to it from other processes[3]. To allow such use a file systems can define opening a file in read-only, append-only or read-write mode. On Linux this is opt in[4]. Even more semantics exist for example allowing opening multiple non overlapping ranges of a file for writing.

1.5 Existing distributed file systems

1.5.1 Network File System

Often we want to share filesystems over a network to share files using a *Network file system*. These integrate in the interface of the client. A widely supported system is NFS. In NFS a part of a local directory is exported/shared by a local NFS-server. Other machines can then connect and overlay part of their directory with the exported one. The NFS protocol forwards file operations from the client to the host over the network. When an operation has been applied on the host the result is traced back to the client. To increase performance the client (almost always) caches file blocks and metadata.

In a shared envirement it is commanplace for multiple users to simultaniously access the same files. In NFS this can be problamatic, as meta data is cached new files can appear to other users after 30 seconds. Further more simultaneous writes can become interleaved as each write gets split into multiple network packets [3, p. 527], writing corrupt data. Version 4 improves the semantics respecting unix advisory file locks [2]. Most applications do not take advisory locks into account still risking data corruption.

1. GFS, single node so easy
2. Hadoop FS HA, opt in consensus
3. Ceph, subtree partitioning

[3]The OS can ensure append writes are serialized, this is usefull for writing to al log file where each write call appends an entire log line to a file opend in append mode

[4]see flock or fcntl or mandatory locking

References

[1] M Caporaloni and R Ambrosini. “How closely can a personal computer clock track the UTC timescale via the internet?” In: *European Journal of Physics* 23.4 (June 2002), pp. L17–L21. doi: 10.1088/0143-0807/23/4/103. url: https://doi.org/10.1088/0143-0807/23/4/103.

[2] S Shepler et al. *Network File System (NFS) version 4 Protocol*. RFC 3530. IEFT, Apr. 2003. url: https://www.ietf.org/rfc/rfc3530.txt.

[3] Abraham Silberschatz, Peter Baer Galvin and Greg Gagne. *Operating system conceps*. John Wiley & Sons, 2014.