PROOFS/NOTES

NIKHIL VASAN

1. Overview

2. Implementation

The implementation of the burning of the baseFee will be done through introducing a dependency between the Distribution Keeper, and feeMarket Keeper as such, in x/Distribution/types/interfaces.go

```
//code example
type FeeMarketKeeper interface{
  // fields entered here ..
}
```

3. GFS

Architecture. Master is defined as follows

```
type Master {
   // chunkData stored in memory
   chunkHandles map[FID]*chunkData
}
// identifier for files stored in GFS
type FID = string
// definition of chunkData, maintains the data required for
    master when granting
// access to chunkServer for chunk requested
type chunkData {
   // essentially IP address of chunk server
   servers []*chunkServer
   // primary, server with current lease for sequencing record
       appends / writes
   primary *chunkServer
}
```

GFS handles writes / reads to chunks as follows.

(1) Client sends request to chunk server, with FID / offset
(2) Client identifies chunk for offset, sends primary for lease to client
(3) Client caches primary address, and transmits data as well as operation to be sequenced by primary

Date: September 2021.

In writes / appends, the primary is tasked with sequencing the operation to the file for all replicas. In each case, it must be required that the data is atomically appended (not necessarily at end of chunk), to file, primary sequences concurrent writes, and transmits sequence to replicas. All replicas gossip data to each replica in sequence $(rep1 => rep2 => ...)$, where $repi$ is an ordering determined by latency, etc.
The Master maintains a log of all writes / reads. Readers of chunks obtain, read locks, ensure that version number for file is stored. Files with read-lock, ensure that name-space changes / leases to primary for writes cannot be executed until read-locks released. How to handle?

4. Distributed Algorithms

fail-stop - Any *process* can fail, but failures can be reliably deteceted by other processes.
fail-silent - Process crashes never can be detected
fail-noisy - Identification of Process failure is only eventual
fail-recovery - *processes* can fail / recover and participate in alg.
fail-arbitrary - *processes* can fail in perhaps arbitrar ways
randomized - Proceses may make probablistic choices given a common source of randomness

Denote set of processes in algorithm as Π, define *rank*, a mapping $rank : \Pi \to \{1, \cdots, N\}$
Processes send messages denoted $m_{p,i}$, where i is a *sequence number*, notice, combined w/ rank, a total ordering is defined (*eg. lexicographic ordering on rank, sequence num.*) over hte messages