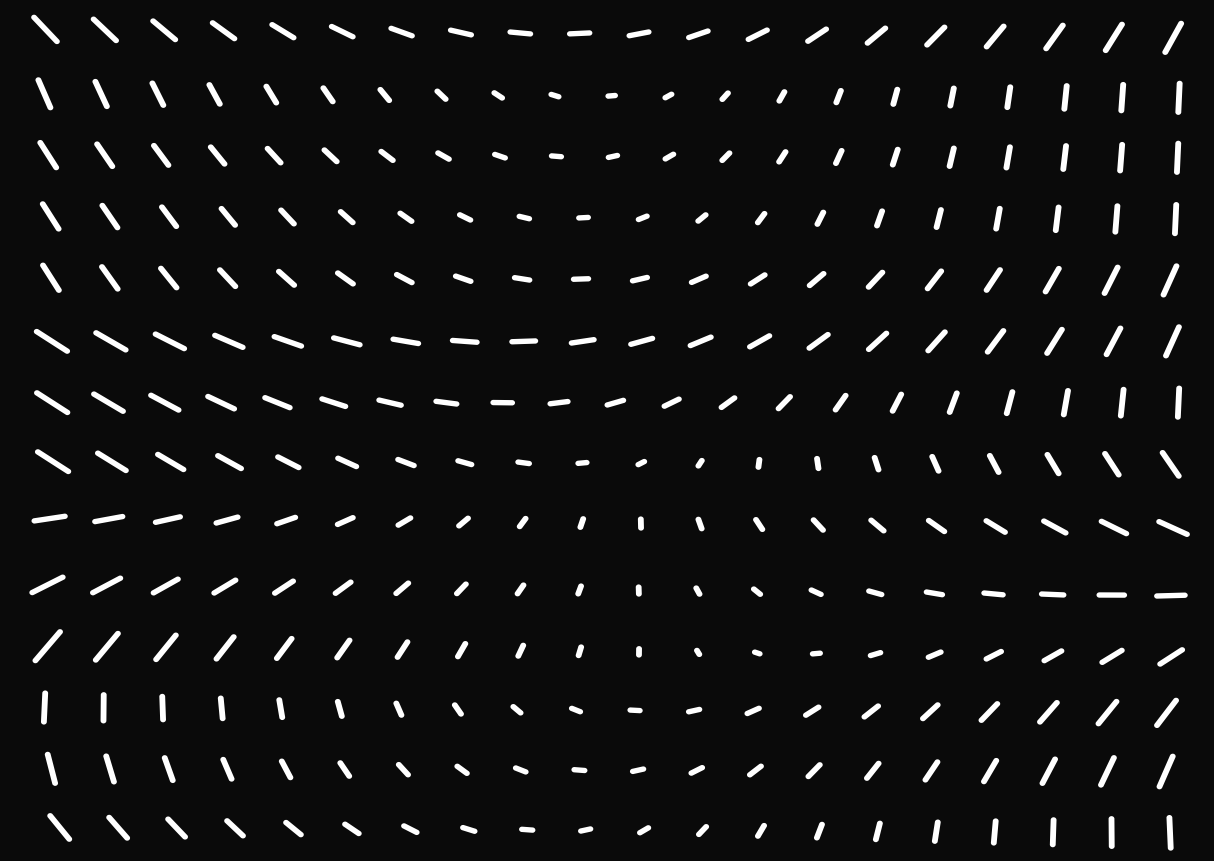


Scalar Time

- Example. Use d = 1 and assign time stamps for the events.

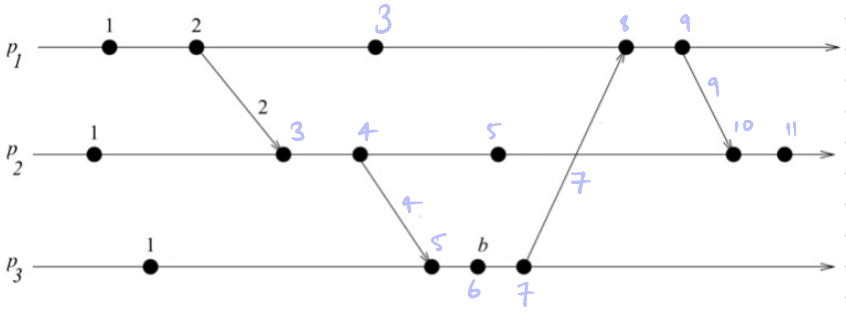
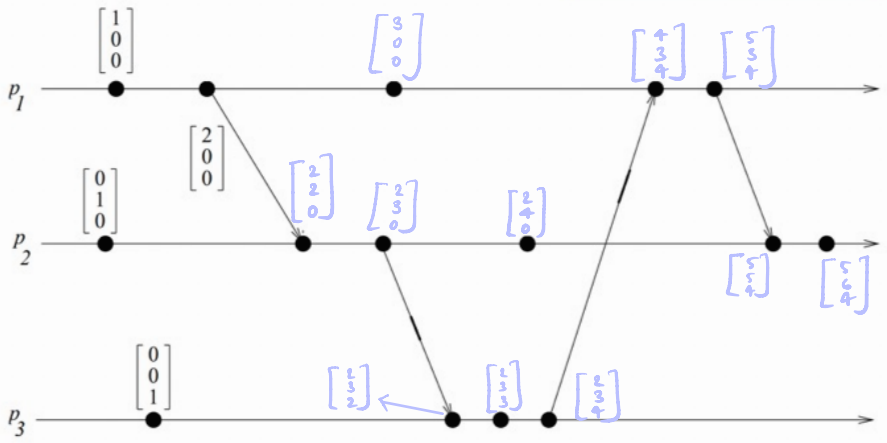


Figure 3.1: The space-time diagram of a distributed execution.

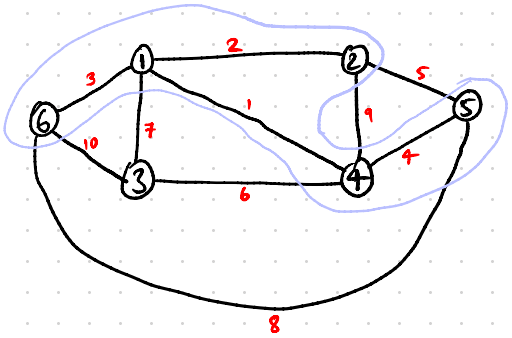
Vector Time – Example

- Double-click to edit

P2's second event:
max([2,0,0], [0,1,0])=[2,1,0]
Then Rule1: [2,2,0]



$2x + 2y \equiv 1 \mod 2$



e_{14} = Branch

e_{21} = Branch

e_{54} = Branch

e_{34} = Branch

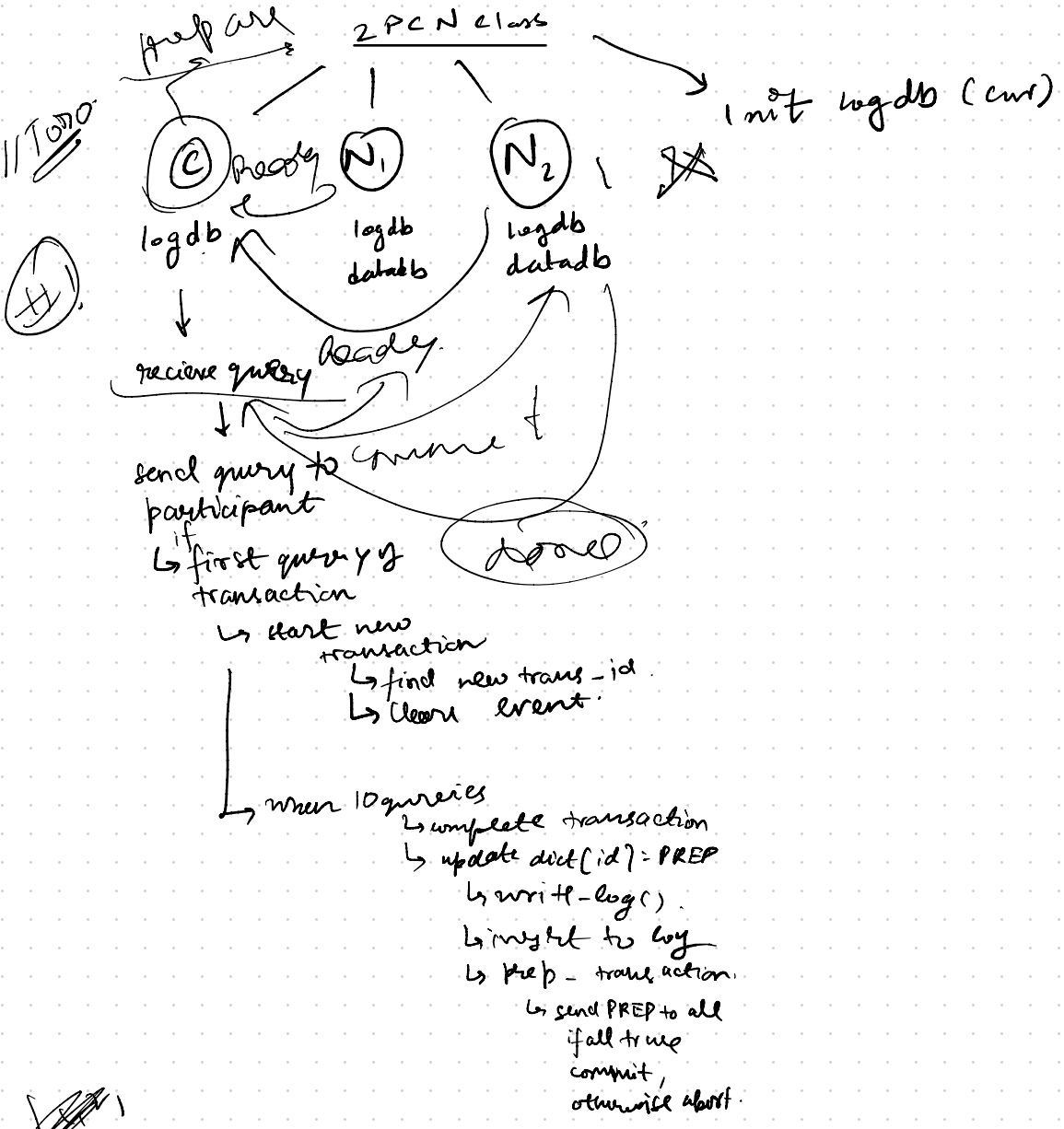
e_{41} = Branch

e_{61} = Branch

e_{12} = Branch

e_{16} = Branch

e_{45} = Branch



Two Phase Commit Participant

Google file System (GFS)

GFS → Big & Fast

To get Big & Fast, we need to split the files across many servers (sharding)

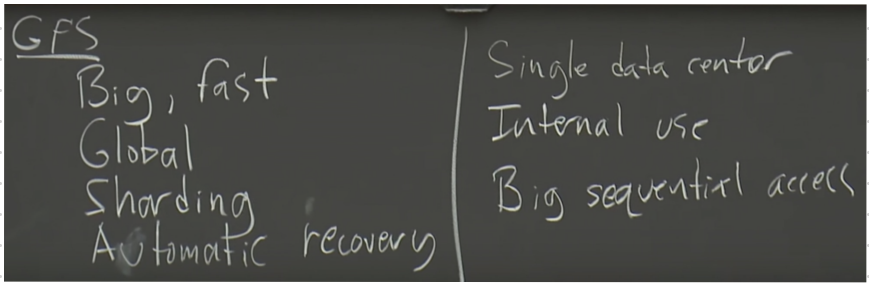
∴ GFS will automatically split the files across many servers we will get high aggregate throughput and store files bigger than any single disk.

Since we are we using 100s of servers, we want automatic failure recovery

GFS was designed to run in a single data center.
∴ We are not talking about placing replicas all over the world.
Not designed to serve customers, only for internal use by google engineers.

Only designed for Big sequential access (not Random)

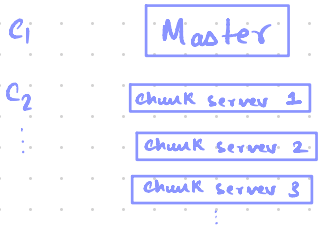
No optimization on latency, but on throughput of big megabyte operations



Single Master

Applications could compensate for odd data sent by GFS by using checksums sent along the data

We have many clients (100s of clients)



Master keeps the mapping of file names to chunk servers (where to find the data from)

The Master is all about naming and knowing where all the chunks, whereas the chunk servers are where all the actual data is stored. For every file, the master keeps track of a list of chunk identifiers. Each chunk is 64 MB. The list is sequential.

Master Data

Master has two main tables:

Table 1:

file name ⟶ Array of chunk ids/chunk handles (NV)

Table 2:

chunk handle ⟶ Meta data about the chunk handle

Meta data ⟶ 1) list of chunk servers (V) → on rebooting ask each cs what chunks they have (chunks are replicated across servers)

2) Version number of the chunk (NV)

3) Primary (which chunk) (V)

4) Lease time (Primary only allowed to primary for a certain lease time) (V)

All writes for a chunk have to be sequenced through the chunk's primary ⟵ 3)

To reboot the Master it needs to store its data onto a disk as well as in memory. Though not all of its data needs to stored on disk, and therefore, we need to identify which of the data Volatile (V) and which of the data is Non-Volatile (NV)

It also uses a LOG (rather than a database or B-tree) which makes it efficient to

When master crashes it also checkpoints its entire state onto a disk so that it doesn't have to read the entire

history from when it was installed to reboot.

when rebooted, it plays only the log starting from where the checkpoint was created

READ

Read → The client running application knows a file name and the offset in the file from where it wants to read data from.

① C_i —(file name + offset)→ Master (M)

② M —(CH + list of servers; chunk handle)→ C_i

C_i will then choose a server most appropiate

C_i also caches this result, so it doesn't have to ask M again and again.

③ C_i —(CH + offset)→ Chunk Server (CS)

CS stores each chunk in linux file system with file name = chunk handle / identifier (CH)

∴ CS just finds file with right file name (CH)

④ C_i ←(data)— CS

C_i has a GFS library that figures what chunks are required and to put them back together when received from the CS.

C_i only thinks in terms of file names and offsets

Library and M conspire to do everything chunk related.

Paper says C_i reads from cs that are in the same rack or switch (perhaps the one that are closest to them)

WRITES

C_i will have a file name, ~~a range bytes specified to write~~ and a buffer data to ~~write~~ append to the ~~location~~ file.
C_i will make an API call to the GFS library with all of this information

Since for writing there needs to be a primary chunk, we need to consider the case when there is no primary chunk.

Case I: No Primary

M needs to find the set of cs that has the most upto date copy of the chunk.

On M:

1) Find CS containing upto date chunk replicas

Upto date: Replica version # = latest version # issued by M.

2) Among the set of most upto date Replicas, pick one to be Primary and rest to be secondaries.

3) M tells Primary and secondaries V # and they also write the V # to disk to

not forget + lease to Primary

4) M writes V# to disk

Case II: Primary found