




Distributed Storage System

IDSS630E



Submitted by

Amit K S - IIT2016107
Varun Sonagra - IMM2016004

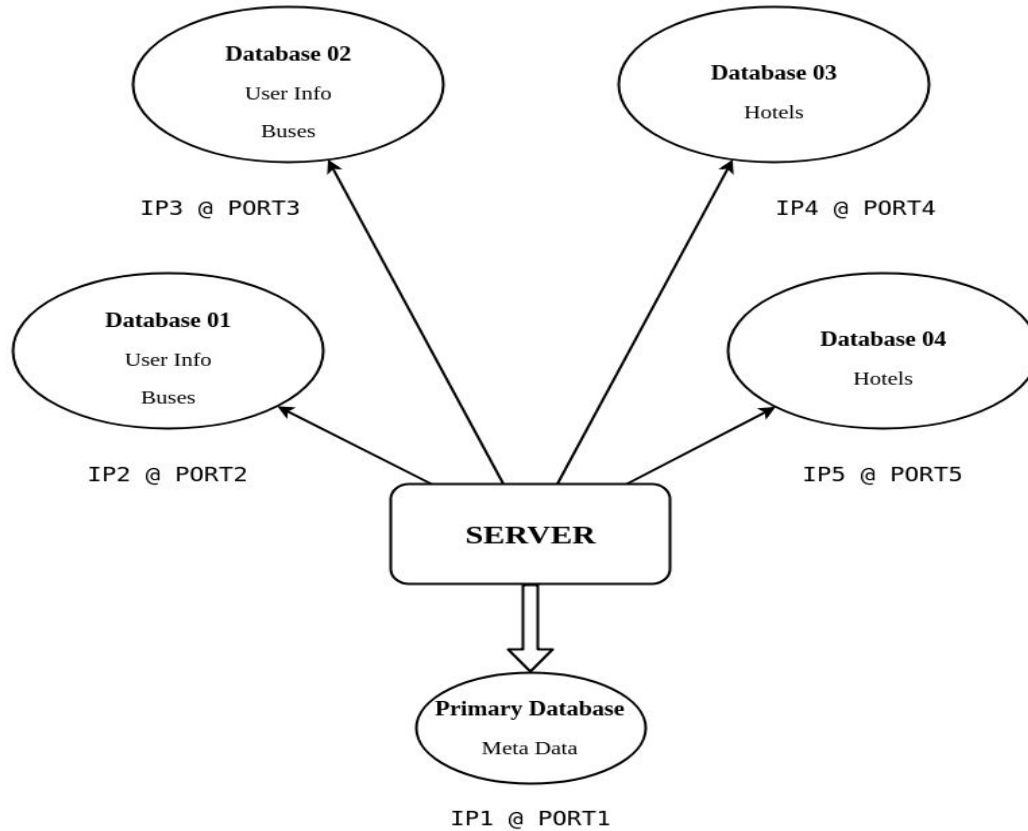
Overview

1. Introduction
2. Architecture
3. System Model
4. Handling Writes
5. Handling Updates
6. Handling Reads
7. Failures
8. HeartBeat

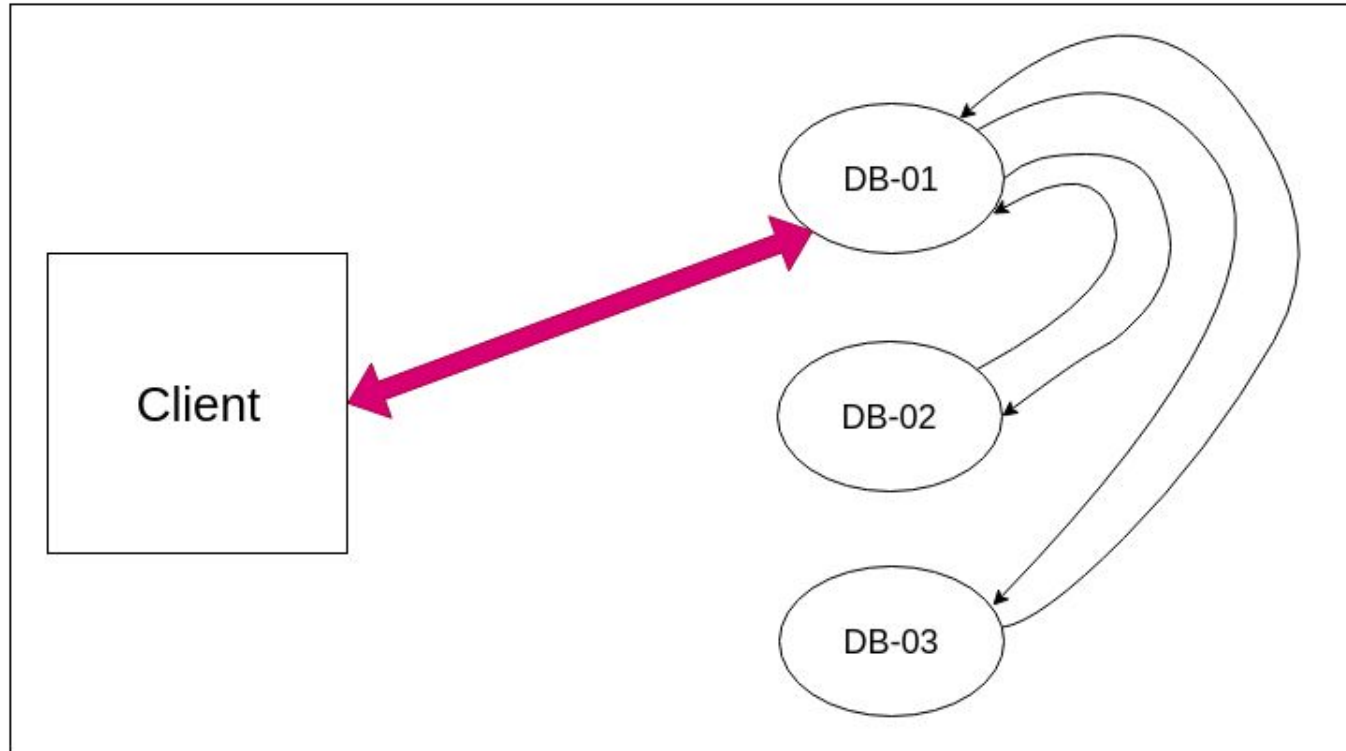
INTRODUCTION

- Distributed Storage
 - Do not store all data at one place
 - Reliable
 - Fault tolerance?
 - Transparency
- Architecture
 - Master Slave - Master contains metadata
 - Integrated with Google File System architecture
 - Primary and Secondary Data Servers
 - Replication (3)

ARCHITECTURE



ARCHITECTURE



ASSUMPTIONS and DETAILS

- The master server never crashes (single point failure)
 - More than 2 servers are never down at any point of time (replication = 3)
 - Transaction handling is out of scope for this project
 - There may be a finite delay in message requests
 - There are no permanent crashes
 - There are atleast 3 databases available (active) at any point of time
 - Primary server may not crash while waiting for response
-
- There is a single master server and 8 data servers
 - Heartbeat rate can be changed by the admin
 - New databases can be added only by the superuser
 - Replication factor is set to 3, i.e 3 copies of any data are stored.

Handling Writes

- Request received at master:
 - Master forwards the request until 3 data servers respond positive
 - Assign “primary” to one of the 3 servers and “secondary” to the other 2
 - Returns status = 200 OK (positive) to the client
- Request received at active data server:
 - Request forwarded by master
 - Write data
 - Respond status = 200 OK to the master, if successful
 - Respond status = 404 BAD REQUEST (or any other relevant status), if unsuccessful

Handling Updates

- Request received at master:
 - Figure out the primary database and secondary databases
 - If primary is not active, make any of the active secondary server primary
 - Forward update request to the primary and wait for response
- Request received primary server:
 - Write data
 - Forward the request to the secondary servers
 - Return relevant response
 - All OK
 - Any DB not updated?

Handling Updates

- Request received at secondary servers:
 - Write data
 - Return response to primary
- Response received at Master:
 - Check if any DB not updated
 - Add request to “Pending Requests” corresponding to the crashed DB
 - Return relevant response

Handling Reads

- Request received at master server:
 - Forward request one after another to all active databases (or just 3, depends on the situation)
 - Return as soon as any one responds
- Request received at Data Server:
 - Return status along with requested data

Failures

- Primary data server may not crash while waiting for response (Assumption)
- Consider the following situations:

MASTER

1. Make sure primary is active
2. Forward update request to primary

What if primary crashed in between 1 and 2?

MASTER

Heartbeat rate = 30 seconds

1. $T = 0$, heartbeat detects primary data server alive
2. $T = 5$, primary data server fails
3. $T = 10$, update request for a record where primary is the failed server
4. Next heartbeat at $t = 30$

IN BOTH CASES, USER RECEIVES A MESSAGE - "Internal Error"

HeartBeat

- A parallel thread manages heartbeat requests along with the running master server
- Sends request to all registered data servers and marks their status
- HeartBeat rate can be set by the admin
- Recommended to have heartbeat below 10 seconds, else user has to wait long to update data (receives Internal Error)

Tools Used

- 1) Front End : HTML, CSS (Bulma UI), Javascript
- 2) Back -End : Python Django
- 3) Database : Postgres SQL

Thank You