Distributed Systems Project - **Phase 1**

Cairo University - Faculty Of Engineering - Computer Engineering Dept.

March 2019

1 Introduction

With the ever-growing technological expansion of the world, distributed systems are becoming more and more widespread. They are a vast and complex field of study in computer science. A distributed system in its most simplest definition is a group of computers working together as to appear as a single computer to the end-user.

These machines have a shared state, operate concurrently and can fail independently without affecting the whole system's uptime.

This project is a vehicle **tracking** and **counting** platform. Each group of students will implement **distributed database**, **distributed file system** and **distributed processing** modules of this project.

2 User Registration - The Distributed DataBase

Traditional databases are stored on the file system of one single machine, whenever you want to fetch/insert information in it, you talk to that machine directly.

For us to **distribute this database system**, we'd need to have this database run on **multiple machines** at the same time. The user must be able to talk to whichever machine he chooses and should not be able to tell that he is not talking to a single machine. if he inserts a record into node1, node3 must be able to return that record.

2.1 Master slaves model

Imagine that our platform got insanely popular. Imagine also that our database started getting twice as much queries per second as it can handle. Your application would immediately start to decline in performance and this would get noticed by your users.

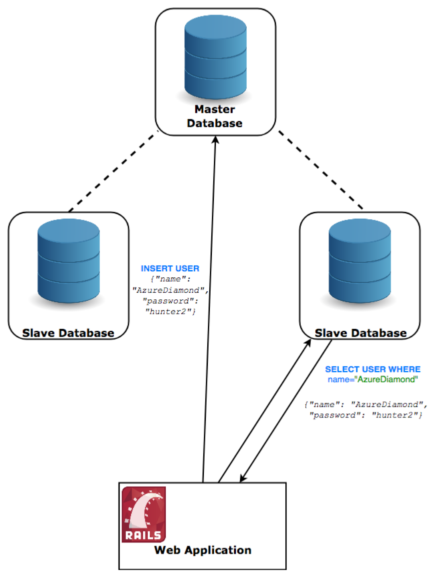
Master Database
Slave Database
INSERT USER
{"name": "AzureDiamond", "password": "hunter2"}
Slave Database
SELECT USER WHERE name="AzureDiamond"
{"name": "AzureDiamond", "password": "hunter2"}
RAILS
Web Application

Figure 1: Master Slaves Database model

In a typical application client normally **read information much more frequently than you insert new information or modify old one**.

There is a way to increase read performance and that is by the so-called **Master-Slave** Replication strategy. Here, you create two new database servers which sync up with the main one. The catch is that you can only read from these new instances.

Whenever you insert or modify information, you talk to the master database. It, in turn, asynchronously informs the slaves of the change and they save it as well.

2.2 Shards Model

With **sharding** you split your server into multiple smaller servers, called shards. These shards all hold different records, you create a rule as to what kind of records go into which shard. It is very important to create the rule such that the data gets spread in an **uniform way**.

A possible approach to this is to define ranges according to some information about a record (e.g users with name A-D).

This sharding key should be chosen **very carefully**, as the load is not always **equal** based on arbitrary columns. (e.g more people have a name starting

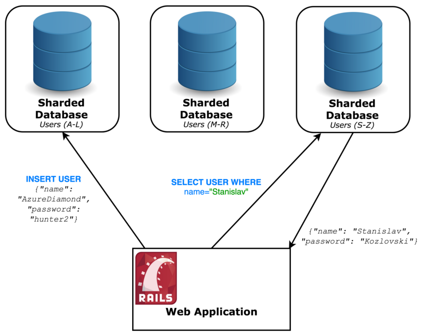
Sharded Database
Users (A-L)
Sharded Database
Users (M-R)
Sharded Database
Users (S-Z)
INSERT USER
{"name": "AzureDiamond", "password": "hunter2"}
SELECT USER WHERE name="Stanislav"
{"name": "Stanislav", "password": "Kozlovski"}
RAILS
Web Application

Figure 2: sharding model

with C rather than Z). A single shard that receives more requests than others is called a hot spot and must be avoided. Once split up, re-sharding data becomes incredibly expensive and can cause significant downtime.

2.3 User Registration Requirements

RQ01-000 - Client process may be at any machine.
RQ01-001 - Client processes has no limits.
RQ01-002 - A client process can sign-up a new user.
RQ01-003 - A client process MUST ask the new user for user name, email address and password.
RQ01-004 - Users database might be MySQL, SQL Server or MongoDB. choose the database server you prefer **from them**.
RQ01-005 - Users database MUST be at least on 3 machines.
RQ01-006 - Choose one of the models mentioned above or use your own logical design such that :.
RQ01-007 - Users database MUST be consistent at any point of time over all the machines.
RQ01-008 - Users database MUST be available all the time **except** all the database machines are down !!.
RQ01-009 - Processes that deal with the database SHALL wait at maximum 200msec for every DB transaction.

3 Third Year Distributed File System (TYDFS)

Our platform allow users to upload mp4 files to their account, and download them again. We want our platform data to be distributed over multiple machines. The following architecture is mixture of multiple typical distributed file system, so we called it (TYDFS) !.

3.1 The Architecture

TYDFS is a centralized distributed system. TYDFS has 2 types of machine nodes. First, the **Master Tracker** node. This node has a look-up table. The look-up table columns are (user_id, file name, data node number, file path on that data node, is data node alive). We will assume the all the files will be mp4 files. Second, the **Data Keeper** nodes. Data Keeper nodes are the actual nodes that have the data files.

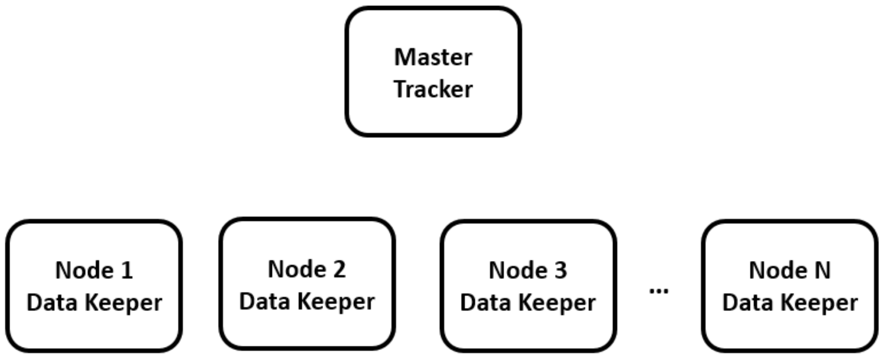

Figure 3: TYDFS Arch.

3.2 I am Alive messages

Students **MUST** use the **Publisher - Subscriber Model** to implement this function. Every **1 sec**, Every data keeper node send an 'Alive' message to the master tracker node. The master tracker node then update the look-up table mentioned above. If one of the data keeper nodes are down it will change the corresponding cell in the 'is data node alive' column.

Student can visit this link to learn more about PUB-SUB in pyzmq.

Note, Design your own message to inform the tracker the identity of the node and the necessary info.

3.3 Master Tracker is multi-process module

The master tracker node is on one machine. There are at least 3 processes to do the tracker job. All processes do the same thing, They are multiple, for the purpose of speeding up the requests. **Note, the look-up table MUST be shared across the N processes. Also, every process listen on a different port on the same machine (remember Clinet-Server of PyZmq).**

Note, Design your own cluster such that every data keeper node notify the look-up table.

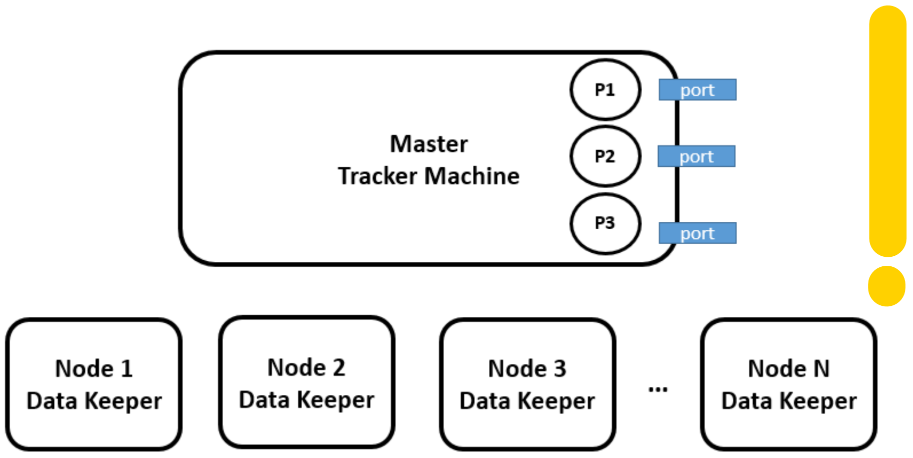

Figure 4: Master Node as a multi-process module. P1, P2, P3 are the processes mentioned.

3.4 Data Keeper Node is multi-process module

The data keeper node is on one machine. There are at least 3 processes to do the data keeper job. All processes do the same thing, They are multiple, for the purpose of speeding up the requests. Every process listen on a different port on the same machine (remember Clinet-Server of PyZmq).

3.5 Upload A file

For a client to upload a file to a cluster, the following protocol MUST be followed:

1) A client process **MUST** communicate with the master tracker node processes.

2) The master tracker responds with a port number of one of the data keeper nodes.

3) The client then constructs a communication with this port, and transfers the file to it.

4) When the transferring procedure is finished, The data keeper node is then notify the master tracker.

5) The master tracker then add the file record to the main look-up table. 6) The master will notify the client with a successful message.

7) The master chooses 2 other nodes to replicate the file transferred. The protocol of this statement will be explained in the next section.**N-Replicates Protocol**

The following figure wraps up all the steps of this protocol.

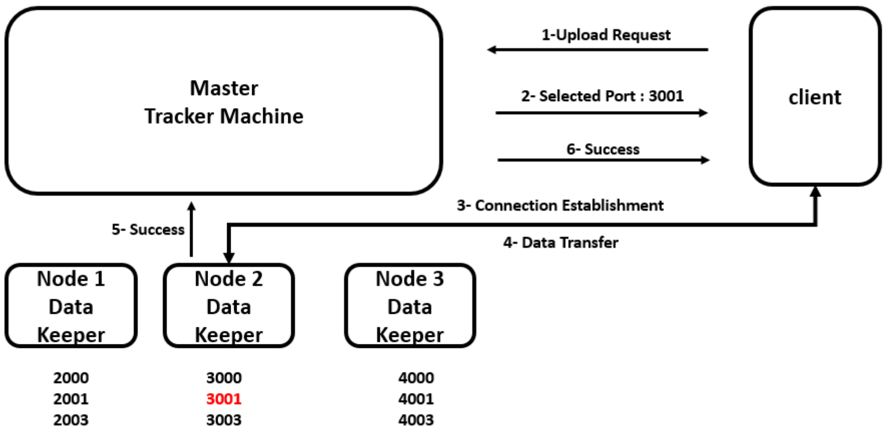

Figure 5: Upload a file protocol.

Note, Make you own design such that high performance, simple fault tolerance and consistency are achieved.

3.6 N-Replicates Protocol

This procedure is running on the master tracker machine. The procedure is periodic. This procedure wants every file to be at least on 3 machines. Clearly, file a.mp4 must have 3 version, every version on a different machine.

```
initialization;
while distinct file instances ¡ K do
    instance_count = getInstanceCount(file[k]);
    if instance_count is less than 3 then
        source_machine = getSourceMachine(file[k]);
        machine_to_copy_1 = selectMachineToCopyTo();
        machine_to_copy_2 = selectMachineToCopyTo();
        NotifyMachineDataTransfer(source_machine, machine_to_copy_1);
        NotifyMachineDataTransfer(source_machine, machine_to_copy_2);
    else
        //DoNothing
    end
end
```

Algorithm 1: Make N-Replicates algorithm

getSourceMachine() is a function that takes a file record,Then gets the source machine and the file path on that machine. **SelectMachineToCopyTo()** return a valid IP and a valid port of a machine to copy a file instance to. **NotifyMachineDataTransfer(a, b)**, this function must notify both source and destination machine to start coping the file.

Note make suitable designs for the functions such that the mentioned algorithm runs correctly.

3.7 Download A file

A client can download his mp4 files. The following protocol MUST be followed:
1) Client request from the Master Tracker to download a certain file name.

2) Master Tracker responds with list of machines IPs and ports to download a file from. **note:** the list length must be at least ~~6~~.3

3) Client process MUST request from every port **uniformly**. remember **Client-Server PyZmq**

Design your own algorithm such that the procedure be fault tolerant, connection oriented and efficient

4 Conclusion

This phase is the preparatory phase for the video processor cluster. **Good Luck!**