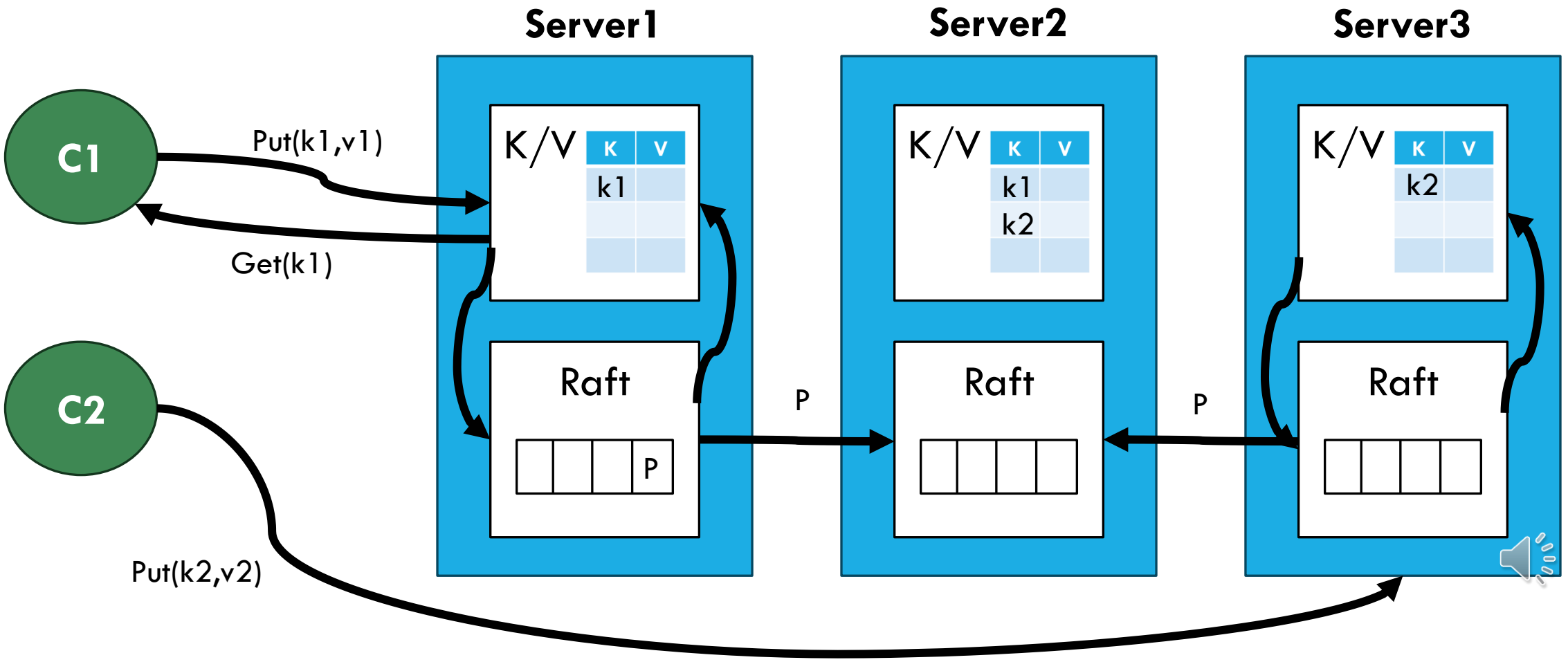
CMP636 Distributed Systems
Static Sharded KV Server Lab

Ayman AboElHassan, PhD
Assistant Professor
ayman.abo.elmaaty@eng.cu.edu.eg

Basic Idea


Server1
Server2
Server3
C1
C2
Put(k1,v1)
Get(k1)
Put(k2,v2)
K/V
K
V
k1
k2
Raft
P
P
P

Basic Idea

1. Client send request to KV server
2. Hash function determines which group holds the key-value pair
3. KV server shard send operation to Raft
4. Raft reaches value consensus ➔ Total Broadcast to followers
5. Followers reply to leader
6. Once quorum Ack received ➔ Commit KV change

Requirement

Change last lab key/value server

1. Create a shard manager server
2. Create 2 sets of “3 servers”
3. Each server maintains its own in-memory map of key/value pairs (Local KV Map)
4. Each group maintain a set of different key/value pairs (sharding)
5. Clients sends read/write/append to 1 of the 3*2 servers (according to the sharded group)
6. Servers of same group reach consensus on KV operation through Raft

Shard Manager Server

A logical server which knows which group of servers has which key

- Basic implementation “Static” ➔ Hash function map ➔ % nGroups

FNV-1 hash [edit]

The FNV-1 hash algorithm is as follows:[9][10]

```
algorithm fnv-1 is
    hash := FNV_offset_basis

    for each byte_of_data to be hashed do
        hash := hash × FNV_prime
        hash := hash XOR byte_of_data

    return hash
```

Python Implementation

Server

1. Create SyncObj
2. Create empty Replicated Dict
3. Create 1 RPC server stub
4. Wait for client requests
5. Perform client's request and print the operation input/result

Client

1. Create 5 client threads
2. Each thread
   1. Select a random request out of:
      - **Get** value of K@
      - **Put** value ClientNum in K@
      - **Append** value ClientNum to K@
   2. Get server group from shard manager
   3. Send a request to the target group
   4. Wait for response

Python Implementation

gRPC proto “kv server”

- Service1: **Get**
- Message: **key**
- Behavior:
  return map[key]
- Response: **value**

- Service2: **Put**
- Message: **key, value**
- Behavior:
  map[key] = value
- Response: **success**

- Service3: **Append**
- Message: **key, args**
- Behavior:
  old_value = map[key]
  map[key] += args
- Response: **old_value**

Python Implementation

gRPC proto “Shard Manager”

- Service1: **GetShardIndex**
- Message: **key**
- Behavior:
  return Shard’s Group gRPC port ID
- Response: **gRPC port ID**

Why Kubernetes?

1. Dynamic deployment “port assignment”
2. Dynamic scale-out “vertical scale-up”
3. Restart on failure
4. Roll-out updates

Thank you