UNIVERSITAT ROVIRA i VIRGILI

Lab 3 – REDIS

Distributed Systems

What is REDIS

I see Redis definitely more as a flexible tool that as a solution specialized to solve a specific problem: his mixed soul of cache, store, and messaging server shows this very well

-Salvatore Sanfilippo

REDIS In-memory store

- Written in C (25K LOC)
- Uses memory as main storage
- Single Threaded
- Uses disk for persistence
- Screamingly fast performance
- 50K read/write operations per seconds
- 200K read/write ops per second on a regular EC2 instance

Data Structures

- Strings
- Lists
- Sets
- Sorted Sets
- Hashes

String API

- SET
- GET
- SETX
- MGET

List API

- LPUSH
- LPOP
- LLEN
- LMOVE
- LRANGE
- LTRIM
- BLPOP
- BLMOVE

Queue API

- LPUSH
- RPOP
- LRANGE
- BRPOP
- BLPOP
- RPOPLPUSH

Pubsub API

- PING
- PSUBSCRIBE
- PUNSUBSCRIBE
- QUIT
- RESET
- SSUBSCRIBE
- SUBSCRIBE
- SUNSUBSCRIBE
- UNSUBSCRIBE

Running redis

```
docker pull redis
docker run --name my-redis -d -p 6379:6379 redis
docker exec -it my-redis redis-cli

PING SET key "value" GET key

LPUSH mylist "first"
LPUSH mylist "second"
LRANGE mylist 0 1
LLEN mylist
```

Redis1.py keys

```
import redis
# Connect to Redis server
client = redis.Redis(host='localhost',
port=6379, db=0, decode_responses=True)

# Set a key-value pair
client.set("name", "Alice")

# Get the value of the key
name = client.get("name")
print(f"Retrieved value: {name}")

# Delete the key
client.delete("name")

# Check if key exists
exists = client.exists("name")
print(f"Does 'name' exist? {'Yes' if exists
else 'No'}")
```

Redis2.py (lists)

```python
import redis

# Connect to Redis server
client = redis.Redis(host='localhost', port=6379, db=0,
decode_responses=True)

# Push elements to a Redis list (left push)
client.lpush("fruits", "apple", "banana", "cherry")

# Get all elements from the list
fruits = client.lrange("fruits", 0, -1)
print(f"Fruits list: {fruits}")

# Pop an element from the list (right pop)
popped_fruit = client.rpop("fruits")
print(f"Popped fruit: {popped_fruit}")

# Get updated list
updated_fruits = client.lrange("fruits", 0, -1)
print(f"Updated fruits list: {updated_fruits}")

# Delete the list
client.delete("fruits")
```

Consumer.py

```python
import redis

# Connect to Redis
client = redis.Redis(host='localhost', port=6379, db=0,
decode_responses=True)

queue_name = "task_queue"

print("Consumer is waiting for tasks...")

while True:
    task = client.blpop(queue_name, timeout=0)  # Blocks
indefinitely until a task is available
    if task:
        print(f"Consumed: {task[1]}")
```

Producer.py

```
import redis
import time

# Connect to Redis
client = redis.Redis(host='localhost', port=6379, db=0,
decode_responses=True)

queue_name = "task_queue"

# Send multiple messages
tasks = ["Task 1", "Task 2", "Task 3"]

for task in tasks:
    client.rpush(queue_name, task)
    print(f"Produced: {task}")
    time.sleep(1)  # Simulating a delay in task
production
```

subcriber.py

```python
import redis

# Connect to Redis
client = redis.Redis(host='localhost', port=6379, db=0,
decode_responses=True)

channel_name = "news_channel"

# Subscribe to channel
pubsub = client.pubsub()
pubsub.subscribe(channel_name)

print(f"Subscribed to {channel_name}, waiting for
messages...")

# Continuously listen for messages
for message in pubsub.listen():
    if message["type"] == "message":
        print(f"Received: {message['data']}")
```

publisher.py

```
import redis
import time

# Connect to Redis
client = redis.Redis(host='localhost', port=6379, db=0,
decode_responses=True)

channel_name = "news_channel"

# Publish multiple messages
messages = ["Breaking News: Market hits new high!",
            "Weather Update: Heavy rain expected",
            "Sports: Local team wins championship"]

for message in messages:
    client.publish(channel_name, message)
    print(f"Published: {message}")
    time.sleep(2)  # Simulating delay between messages
```

Task 1 - InsultService

Use a REDIS blocking queue two create the following services:

- InsultProducer: A producer that sends new insults to the queue every five seconds.
- InsultConsumer: A consumer that reads new insults from the queue and them to a REDIS list (INSULTS) only if they are new.

Use REDIS pubsub to create the following services:

- InsultBroadcaster: A publisher that sends insults from INSULT list to a pubsubchannel.
- InsultReceiver: A subscriber that subscribes to events from the broadcaster. Try to launch several InsultReceivers to check that it works.

Task 2 - TextFilter with a Work Queue

TextProducer: A producer that sends a text without insults to a Work Queue every 5 seconds

AngryProducer: A producer that sends a text with insults to a Work Queue every 3 seconds

Insultfilter: A consumer that retrieves a text from the Work Queue, replaces insults from INSULT list in the text for "CENSORED" and store the clean text in a RESULTs list.

Launch several InsultFilters to check that the work is distributed among workers.