EVERYWHERE.

Transactions and Concurrency Control in Redis

RedisConf18 Introductory Training

redisconf18

#Redis #RedisConf

presented by:

redislabs
home of redis

Agenda

- What are transactions
- Redis Execution Model
- Transaction Commands: MULTI, EXEC, DISCARD
- Optimistic Concurrent Control: WATCH, UNWATCH
- Durability
- Pipelines

ACID Transactions

redislabs

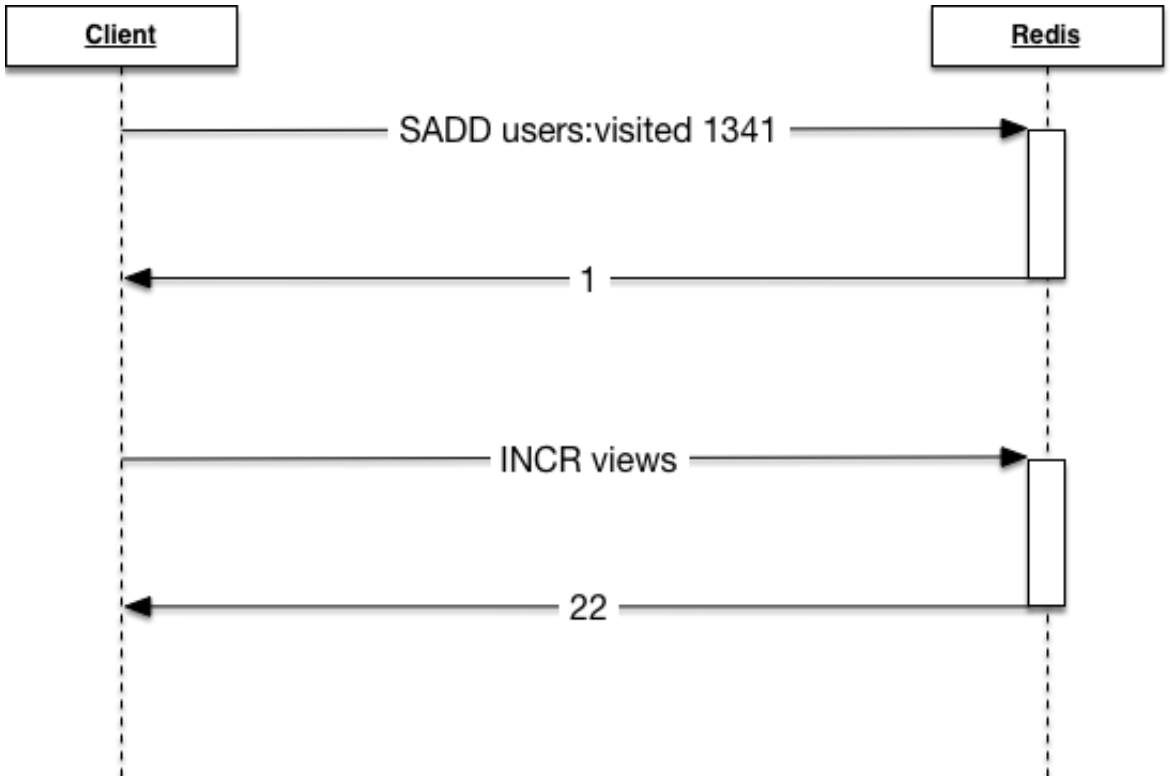
ACID Transactions

A - Atomicity
  -- Transaction executes as an indivisible unit
C - Consistency
  -- Transaction takes database from one valid state to another
I – Isolation
  -- Transactions result in a state as if they were executed sequentially
D – Durability
  -- Transaction changes are available event in the event of failure

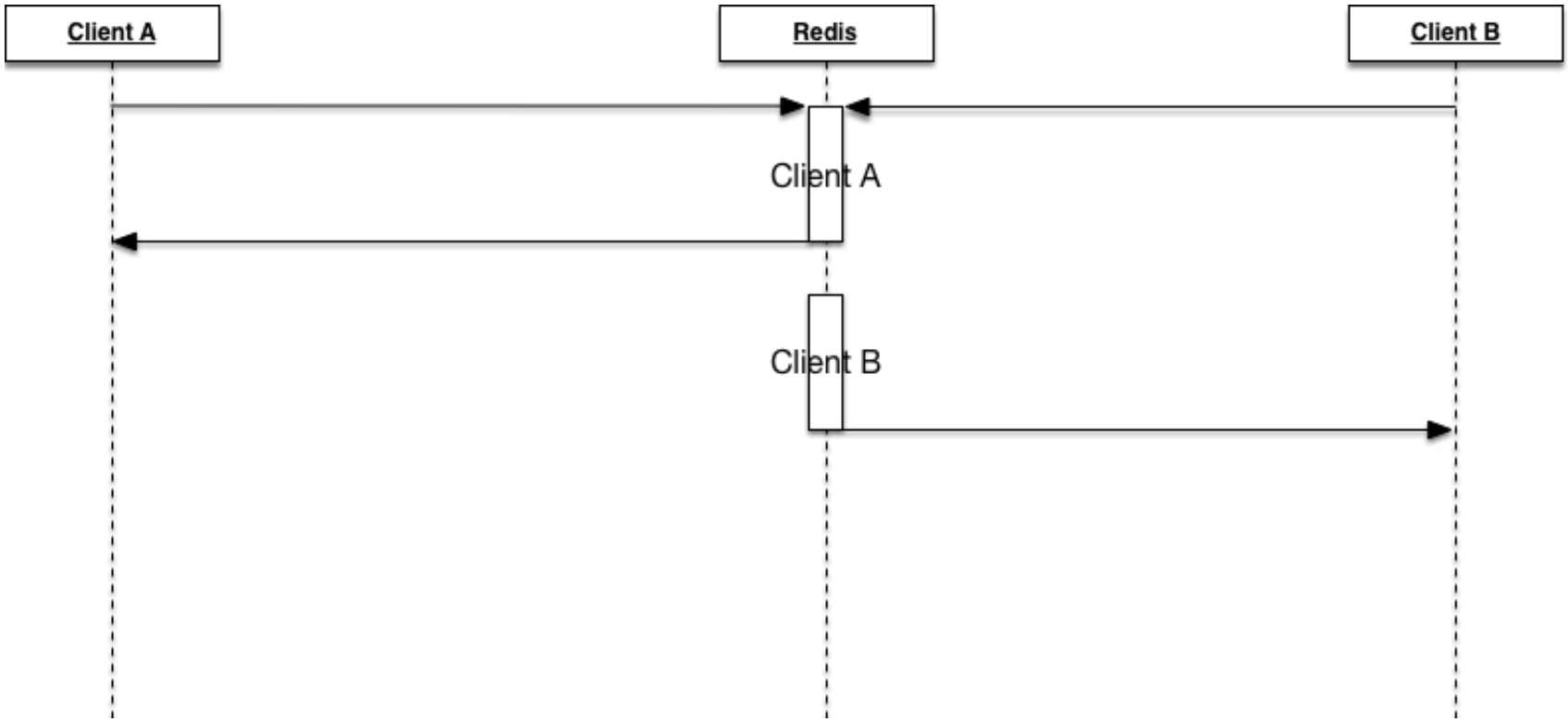
Redis Execution Model

redislabs

Single Client – Execution Flow



Two Client – Execution Flow



Blocking and Non-Blocking Commands

- Most Redis commands are synchronous

- Non-Blocking Commands
  - BGSAVE, BGREWRITEAOF
  - UNLINK (v4)
- Client Blocking Commands
  - SUBSCRIBE
  - BLPOP, BRPOP, BRPOPLPUSH
  - MONITOR
  - WAIT

Variadic (Dynamic Arity) Commands

- Variable number of arguments
- Examples (Non-exclusive)
  - MSET
  - MGET
  - HGET (v4)
  - HSET (v4)

- Executed as a single command

Multiple Command Transactions

redislabs

Multiple Command Transactions

- MULTI to start transaction block
- EXEC to close transaction block
- DISCARD to aboard transaction block

- Commands are queued until exec
- All commands or no commands are applied
- Transactions can have errors

MULTI Example

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> sadd site:visitors 124
QUEUED
127.0.0.1:6379> incr site:raw-count
QUEUED
127.0.0.1:6379> hset sessions:124 userid tague ip 127.0.0.1
QUEUED
127.0.0.1:6379> EXEC
1) (integer) 1
2) (integer) 1
3) (integer) 2
```

DISCARD Example

```
127.0.0.1:6379> sadd site:visitors 124
QUEUED
127.0.0.1:6379> incr site:raw-count
QUEUED
127.0.0.1:6379> DISCARD
OK
```

Transactions with Errors – Syntactic Error

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> set site:visitors 10
QUEUED
127.0.0.1:6379> ste site:raw-count 20
(error) ERR unknown command 'ste'
127.0.0.1:6379> EXEC
(error) EXECABORT Transaction discarded because of previous errors.
```

Transactions with Errors – Semantic Error

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> set messages:hello "Hello World!"
QUEUED
127.0.0.1:6379> incr messages:hello
QUEUED
127.0.0.1:6379> EXEC
1) OK
2) (error) ERR value is not an integer or out of range
```

Conditional Execution/Optimistic Concurrency Control

- WATCH to conditionally execute transaction if key unchanged
- UNWATCH clear c
- DISCARD to aboard transaction block (CLI)

- All commands or no commands are applied
- Transactions **can** have errors

Dependent Modifications– Incorrect Version

```
def incorrectCheckBalanceAndTransferAmount(debit, credit, amount):
      amount = float(amount)
      debitkey
      creditkey = 'account:{}'.format(credit)
      fname = 'balance'
      balance = r.hget(debitkey, fname)

      # Potential race condition - start
      if balance >= amount:
            tx = r.pipeline()
            tx.hincrbyfloat(debitkey, fname, -amount)
            tx.hincrbyfloat(creditkey, fname, amount)
            return tx.execute()
      # Potential race condition - end
      else:
            raise Exception('insufficient funds')
```

Dependent Modifications – Correct Example

```
def checkBalanceAndTransferAmount(debit, credit, amount):
        amount = float(amount)
        debitkey = 'account:{}'.format(debit)
        creditkey = 'account:{}'.format(credit)
        fname = 'balance'
        while True:
        try:
                tx = r.pipeline()
                tx.watch(debitkey)
                balance = float(tx.hget(debitkey, fname))
                tx.multi()
                if balance >= amount:
                        tx.hincrbyfloat(debitkey, fname, -amount)
                        tx.hincrbyfloat(creditkey, fname, amount)
                        return tx.execute()
                else:
                        raise Exception('insufficent funds - time to get a job')
        except WatchError:
                # Reaching here means that the watched 'balance' value had changed,
                # so we can just retry or use any other backoff logic
                continue
```

Durability

redislabs

Disk Based Persistence

- Redis continues to serve commands from main memory
- Multiple Persistence modes
  - Snapshot Based (RDB)
  - Changelog based (AOF)

- Provides durability of data across power loss
  - Look into replication to prevent data loss in case of node loss

RDB Persistence

- Persistence
  - Fork Redis process
  - Child process writes new RDB file
  - Atomic replace old RDB file with new

- Configuration
  - SAVE directive (Redis.conf): SAVE <seconds> <min-changes>
  - Runtime : CONFIG SET SAVE "60 1000 120 100 180 1"

- Trigger manually
  - SAVEcommand (synch)
  - BGSAVE (backgroud)

- All commands or no commands are applied
- Transactions **can** have errors

AOF Persistence

- Configuration
  - APPENDONLY directive (Redis.conf): APPENDONLY YES
  - Runtime : CONFIG SET APPENDONLY YES
- AOF File fsynch options
  - Trade off speed for data security
  - Options: None, everysecond, always
- BGREWRITEAOF
  - aof file grows indefinitely
  - BGREWRITEAOF – trigger compaction of AOF file

Transaction Review

redislabs

Transactions

- *Mostly* ACID Transactions
  - Atomic – through MULTI/EXEC
  - Isolation, Consistency – single threaded nature
  - Durability – persistence modes
- No Rollback – transaction commands are queued then sent to server
- Single threaded event-loop for serving commands
- WATCH for optimistic concurrency control
- Pipelines for asynchronous commands

Review Questions

- What transaction guarantees does Redis provide?
- Give an example of how a Redis transaction can fail?
- What is the watch command used for?
- How to Redis transactions differ from SQL transactions?