Open source in-memory data structure store
Features – At a glance

• Atomicity

• Transactions

• Publish/Subscribe Messaging Paradigm
Atomic Operations

Atomicity stipulates that it should be possible to place every operation at a singular point (linearization point) between its invocation and response\(^1\).
Atomic Operations – Explained

Atomicity gives us a guarantee that only one of two things can happen before anything else happens:

1. The operation succeeds and we are left with the expected result.
2. The operation fails and no changes are made to the underlying data.

Every command in Redis is performed atomically!
Redis Transactions

The execution of a group of commands in a single step, with two important guarantees:

1. All the commands are serialized and executed sequentially.

2. Either all of the commands or none are processed.

Does this look familiar? **Redis Transactions are atomic!**
Redis Transactions – Specifics

**MULTI**: Enters “transaction mode” where we can now list operations

**EXEC**: Executes the transaction, which now contains multiple operations

**DISCARD**: Flushes the transaction queue and exits the transaction
PubSub – Redis

Redis allows you to easily implement the Publish/Subscribe Messaging Paradigm.

- SUBSCRIBE
- UNSUBSCRIBE
- PUBLISH
PubSub – Redis

In Redis, messages are organized into channels.

Very robust channel subscribing features:

```
SUBSCRIBE news.*
SUBSCRIBE news.art.figurative
SUBSCRIBE news.music.jazz
```
Data Model

Key-Value Store
Vs
In-Memory Data Structure Store
Data Model - Basics

- Key-Value pairs
- Value usually string (other datatypes possible)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS4930_171F</td>
<td>Advanced Database</td>
</tr>
<tr>
<td>CIS4301</td>
<td>Info and Database Sys 1</td>
</tr>
<tr>
<td>CEN3031</td>
<td>Intro Software Engr</td>
</tr>
</tbody>
</table>
## Data Model – RDBMS vs Key-Value

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Cred</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS4930_171F</td>
<td>Advanced Database</td>
<td>3</td>
</tr>
<tr>
<td>CIS4301</td>
<td>Info and Database Sys 1</td>
<td>3</td>
</tr>
<tr>
<td>CEN3031</td>
<td>Intro Software Engr</td>
<td>3</td>
</tr>
</tbody>
</table>

**Key**

- Course:CIS4930_171F:Title
- Course:CIS4301:Title
- Course:CEN3031:Title

**Value**

- Advanced Database
- Info and Database Sys 1
- Intro Software Engr

**Key**

- Course:CIS4930_171F:Cred
- Course:CIS4301:Cred
- Course:CEN3031:Cred

**Value**

- 3
- 3
- 3
Data Model – Reverse Index

• Easy way to model relationships
• Reverse index acts as foreign key

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course:CIS4930_171F:Students</td>
<td>[Student:0, Student:10, …]</td>
</tr>
<tr>
<td>Course:CEN3031:Students</td>
<td>[Student:10, Student:13, …]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student:0:Courses</td>
<td>[Course:CIS4930_171F, …]</td>
</tr>
<tr>
<td>Student:10:Courses</td>
<td>[Course:CIS4930_171F, Course:CEN3031, …]</td>
</tr>
</tbody>
</table>
Data Structures - Overview

• Strings
• Lists
• Sets
• Sorted Sets
• Hashes
• HyperLogLogs
Data structures - Strings

• Base type
• Maximum length is 512 MB
• Implemented using C-Strings (Append O(n) operation)
• Binary safe
• Can be used as:
  • Number (Increment, etc.)
  • Vector
  • Binary Data (Bitwise operations)
Data Structures - Lists

- Implemented as a doubly linked list
  - $O(1)$ push and pop operations
  - $O(n)$ accessing other nodes
- Can be used to build Queues or Stacks
- Maximum elements = $2^{32} - 1 = 4,294,967,295$
- Can block retrieval until value is available (BLPOP)
Data Structures - Sets

• Unordered collection of Strings
• Implemented using a hash table
  • $O(1)$ checking for existence
• Set Operations
  • Union, Intersection and Difference
• No duplicates allowed
• Maximum Number of Elements = 4,294,967,295
• Useful to represent relationships
• Randomized retrieval of members possible
Data Structures – Sorted Sets

• Ordered collection of Strings
• Implemented using a skip list
  • $O(\log(n))$ operations
• No duplicates allowed
• Each member has a rank
• Retrieval by range possible
• Can be used to index key-value pairs for range retrieval
Data Structures - Hashes

• Maps between String fields
• Implemented using a hash table
  • $O(1)$ retrieval/existence operations
• Used to represent objects
• Maximum Number of Elements = 4,294,967,295
Data Structures - HyperLogLogs

• Probabilistic data structure
  • HyperLogLog is a well known algorithm
• Used to estimate cardinality
• Trades memory for precision
Data Store Comparisons
Persistence

Two kinds of persistence:
• Append only files
• Redis database files
Append Only Files (AOF)

Pros:
Readability
No seeks or corruption problems if there’s an outage

Cons:
Larger than RDB files
Slower than RDB files
Redis Database File (RDB)

**Pros:**
- Great for disaster recovery
- Small and compact

**Cons:**
- Poor if you need to minimize the change of data loss in case Redis stops working
- Needs to fork() often to persist on disk using child process
Replication
Scaling

App

Partition-1

Partition-2

Partition...
Partitioning

Advantages

• Can handle larger data sets
• Can scale to multiple cores and computers
• Not limited to single CPU’s RAM

Disadvantages

• Can’t partition if operations involve multiple keys
• Increased data handling complexity
• Altering capacity becomes difficult
How to Partition

3 Methods:

1. Client Side Partitioning
2. Proxy Assisted Partitioning
3. Query Routing
Partitioning Implementations

• Redis Cluster

• Twemproxy

• Clients supporting consistent hashing
Partitioning Implementations

- **Redis Cluster**
  - Automatically splits dataset among multiple nodes
- **Twemproxy**
  - Will still operating when some of the nodes are failing
- **Clients supporting consistent hashing**
Partitioning Implementations

• Redis Cluster
  Is a proxy between the clients and Redis instances
  Can automatically shard data among instances
  Supports consistent hashing

• Twemproxy

• Clients supporting consistent hashing
Partitioning Implementations

• Redis Cluster
• Twemproxy
• Clients supporting consistent hashing

Simply use a client that implements client side partitioning (via consistent hashing)

Examples: Redis-rb, Predis
Redis

Common Operations and Query Language
Popular Storage Commands

- **SET**
- **APPEND**
- **PUSH**

<SET> <Key> <Value>

Sets the value to the corresponding key. If key already exists, it is overwritten.

Returns String “OK” if successful, NULL otherwise.
Popular Storage Commands

- SET
- APPEND
- PUSH

<APPEND> <Key> <Value>

Appends the value to the corresponding key. If key doesn’t exist, it is created.

Returns Integer corresponding to new size of Value.
Popular Storage Commands

• SET
• APPEND
• PUSH

<PUSH> <Key> <Value>

Appends the value to the corresponding key. If key doesn’t exist, it is created. Key must be a list.

Returns Integer corresponding to new size of Value.
Popular Retrieval Commands

- **GET**

- **MGET**

  \[
  \text{<GET> <Key>}
  \]

  Returns the value for the corresponding key.

  Returns value of key or NULL if key does not exist.
Popular Retrieval Commands

- **GET**
- **MGET**

\[ <\text{MGET}> <\text{Key}> [\text{Keys...}] \]

Returns all values for all the corresponding keys.

Returns values of keys in an array. Cell in array is NULL if corresponding key doesn’t exist.
Other Popular Commands

- **DEL**
- **RENAME**
- **EXISTS**
- **DBSIZE**

<DEL> <Key> [Keys...]

Deletes all keys and associated values.

Returns the number of keys deleted.
Other Popular Commands

• DEL
• RENAME
• EXISTS
• DBSIZE

<RENAME> <Key> [newKey]

Renames key to newKey.
Returns error if key does not exist. If newKey already exists it is overwritten.
Other Popular Commands

- **DEL**
- **RENAME**
- **EXISTS**
- **DBSIZE**

<EXISTS> <Key>

Determines whether or not key exists in database.

Returns True if key exists, False otherwise.
Other Popular Commands

- DEL
- RENAME
- EXISTS
- DBSIZE

<DBSIZE>
Returns number of keys in the database.
Redis supports different languages

<table>
<thead>
<tr>
<th>ActionScript</th>
<th>Bash</th>
<th>C</th>
<th>C#</th>
<th>C++</th>
<th>Clojure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Lisp</td>
<td>Crystal</td>
<td>D</td>
<td>Dart</td>
<td>Delphi</td>
<td>Elixir</td>
</tr>
<tr>
<td>emacs lisp</td>
<td>Erlang</td>
<td>Fancy</td>
<td>gawk</td>
<td>GNU Prolog</td>
<td>Go</td>
</tr>
<tr>
<td>Haskell</td>
<td>Haxe</td>
<td>Io</td>
<td>Java</td>
<td>Julia</td>
<td>Lasso</td>
</tr>
<tr>
<td>Lua</td>
<td>Matlab</td>
<td>mruby</td>
<td>Nim</td>
<td>Node.js</td>
<td>Objective-C</td>
</tr>
<tr>
<td>OCaml</td>
<td>Pascal</td>
<td>Perl</td>
<td>PHP</td>
<td>Pure Data</td>
<td>Python</td>
</tr>
<tr>
<td>R</td>
<td>Racket</td>
<td>Rebol</td>
<td>Ruby</td>
<td>Rust</td>
<td>Scala</td>
</tr>
<tr>
<td>Scheme</td>
<td>Smalltalk</td>
<td>Swift</td>
<td>Tcl</td>
<td>VB</td>
<td>VCL</td>
</tr>
</tbody>
</table>
Java-specific initialization

- Client to access Redis is called “Jedis”
- Jedis jedis = new Jedis(String host, int port)
Java-specific commands

- jedis.set("key1", "abc")
- jedis.get("key2")

String “abc” is now associated with String “key1”.
Java-specific commands

• jedis.set("key1", "abc")
• jedis.get("key2")

Returns value associated with key2 if it exists.
Companies that use Redis

- Github
- Craigslist
- Digg
- Amazon (AWS)

Uses Redis to find a user’s route, defined to be “the hostname of the file server on which that user’s repositories are kept.”
Companies that use Redis

- Github
- Craigslist
- Digg
- Amazon (AWS)

Uses Redis to map hostname to port numbers of different users.
Companies that use Redis

- Github
- Craigslist
- Digg
- Amazon (AWS)

Uses Redis to keep track of page views and clicks.

“Redis rocks”

-Digg
Companies that use Redis

- Github
- Craigslist
- Digg
- Amazon (AWS)

Amazon Web Services (AWS) ElastiCache uses Redis.

My experience.
Other Companies that use Redis

- Twitter
- Snapchat
- Uber
- Instagram
- Slack
- Imgur
- Grooveshark
- Airbnb
- Tumblr