RethinkDB - Jepsen Test

Evaluating consistency semantics of an upcoming real-time open-source database

What is RethinkDB?

- An open source database for the “real-time” web
- Started by two CS students of the Stony Brook University in 2009
- Design Goal: Make building realtime applications easier
- Features
  - *Document Oriented*
  - *ReQL as a query language*
  - *Dynamic Schemas*
  - *Indices + Joins!*
  - *Clustered. Easy-to-scale. Use MapReduce under the hood*
- Built from scratch in C++ and completely open-source on Github

ReQL

```
SELECT * FROM users
WHERE name = "Peter"
AND age = 30
```

```
r.table("users").filter({
    "name": "Peter",
    "age": 30
})
```

```
SELECT * FROM users
WHERE name LIKE "P%"
```

```
r.table("users").filter(
    r.row['name'].match("^P")
)
```

```
SELECT * FROM users
ORDER BY name ASC
```

```
r.table("users").order_by("name")
```

```
SELECT * FROM users
ORDER BY name DESC
```

```
r.table("users").order_by(
    r.desc("name")
)
```

Pull Architecture

“Instead of polling for changes, the developer can tell RethinkDB to continuously push updated query results to applications in real-time”

Use-cases

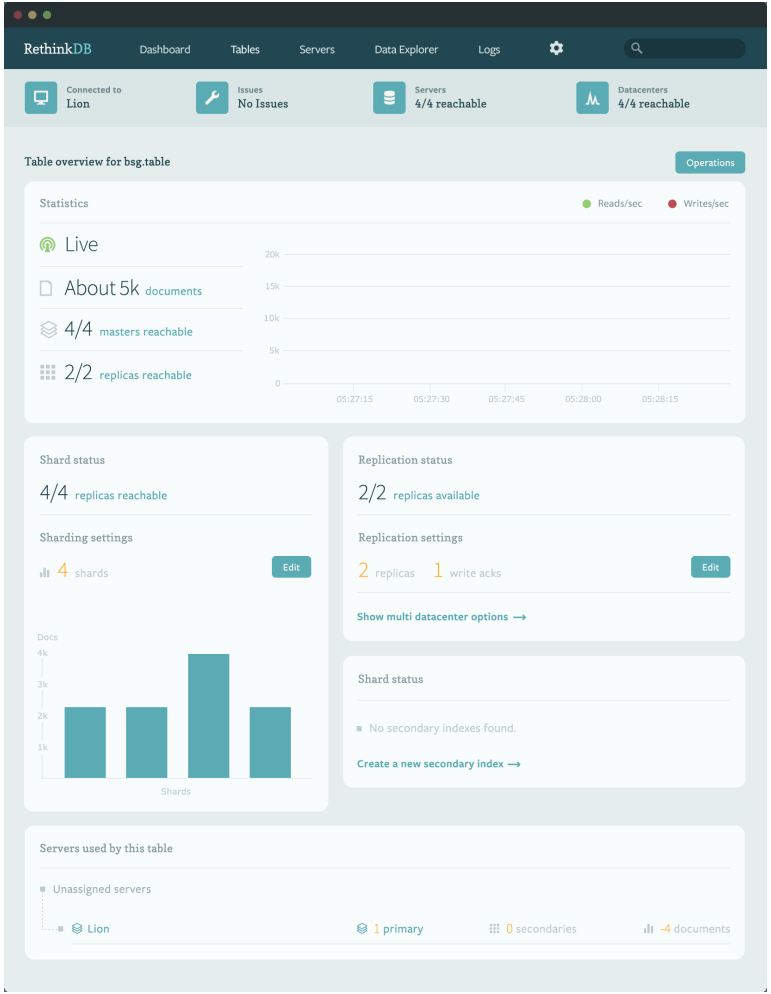
- ➔ Multiplayer games
- ➔ Streaming Analytics
- ➔ Interactive Marketplaces
- ➔ Collaborative web and mobile apps

1-command scaling

```
// set 5 shards each with 3 replicas
// for the ‘games’ table
r.table(‘games’).reconfigure(shards=5,
replicas=3)
```

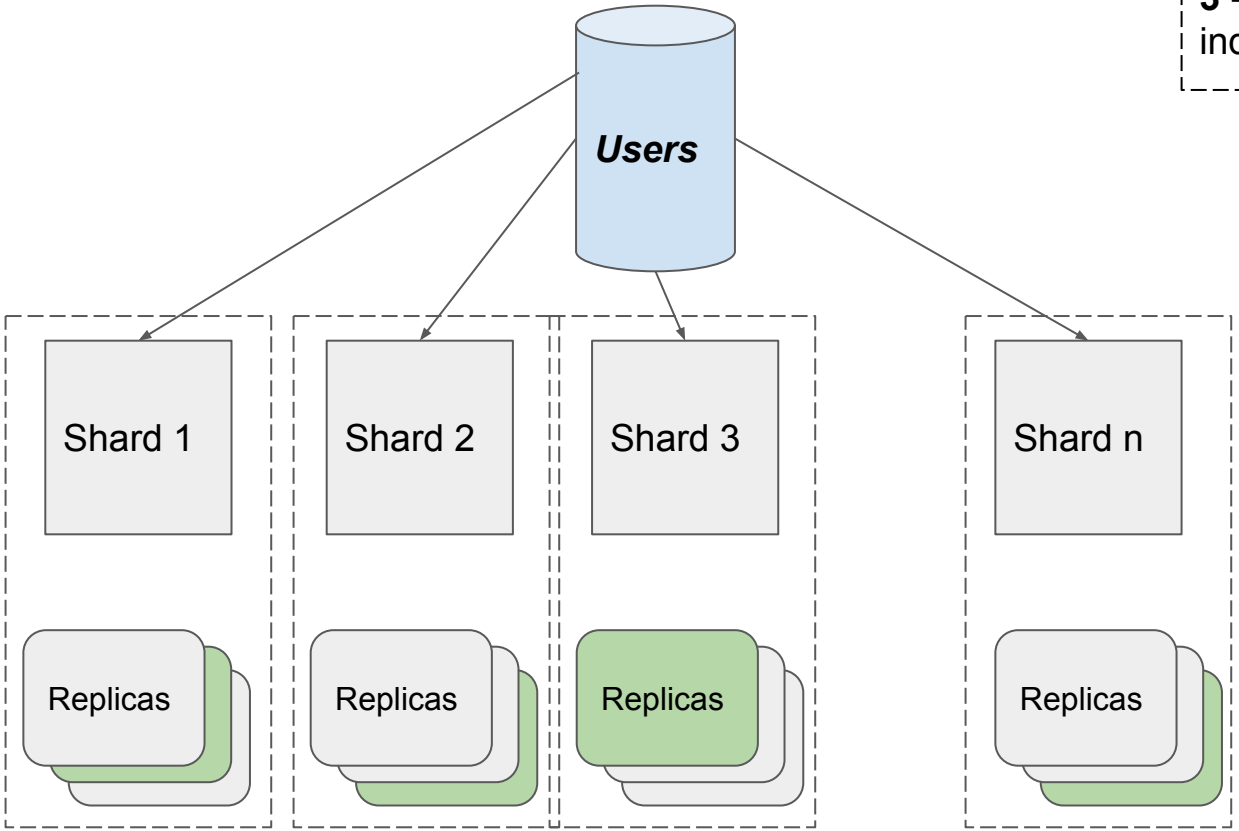
1-command pub-sub

```
// listen to the jobs table for changes
r.db(‘rethinkdb’).table(‘jobs’).changes()
```



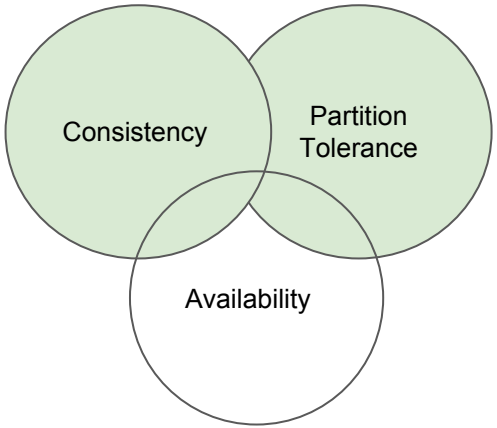
Architecture

n - shards
3 - replicas (green indicates primary)



Consistency Guarantees

- Architecture is **immediately consistent**
- In essence, it is a **CP (consistent - partition tolerant)** system.



RethinkDB Semantics

- Write acknowledgements
  - Majority
  - Single
- Durability
  - Hard
  - Soft
- Read mode
  - Single
  - Majority
  - Outdated

Project Proposal

Goals

- Analyze the consistency properties of RethinkDB
- Evaluate API semantics around consistency and availability guarantees

Methodology

- Setup a RethinkDB cluster with one shards, multiple replicas
- Design test-cases for validating each semantic
- Use the Jepsen clojure library to run test-cases, mimic network partitions and model failures etc.
- Validate results of operations and derive conclusions.