Redis to the Rescue?

O’Reilly MySQL Conference
2011-04-13
Who?

- Tim Lossen / @tlossen
- Berlin, Germany
- backend developer at wooga
Diamond Dash
60 seconds gem rush

wooga is now no. 5 worldwide
over 18 m active users every month

Happy Hospital!
Cure cute pets from funny diseases

Based in Berlin, wooga is the leading European social games developer.

We are hiring!
Currently, we are searching for:

- Internship Social Media and PR (m/f)
- Game Designer – Bubble Island (m/f) (m/f)
- Software Engineer – Graduate Position (m/f)
- Internship IT Management (m/f)
Redis Intro
Case 1: Monster World
Case 2: Happy Hospital
Discussion
Redis Intro

Case 1: Monster World

Case 2: Happy Hospital

Discussion
What?

- key-value-store
- in-memory database
- “data structure server”
Data Types

• strings (integers)
Data Types

• strings (integers)
• lists
• hashes
Data Types

- strings (integers)
- lists
- hashes
- sets
- sorted sets
Performance

• same for reads / writes
Performance

- same for reads / writes
- 50 K ops/second
  - regular notebook, EC2 instance
Performance

- same for reads / writes
- 50 K ops/second
  - regular notebook, EC2 instance
- 200 K ops/second
  - intel core i7 X980 (3.33 GHz)
Durability

- snapshots
- append-only log
Other Features

- master-slave replication
- virtual memory
- ...

Redis Intro

Case 1: Monster World

Case 2: Happy Hospital

Discussion
Daily Active Users

April  May  June  July  Aug  Sept  Oct  2010
Challenge

- traffic growing rapidly
Challenge

- traffic growing rapidly
- bottleneck: write throughput
  - EBS volumes pretty slow
Challenge

- traffic growing rapidly
- bottleneck: write throughput
  - EBS volumes pretty slow
- MySQL already sharded
  - 4 x 2 = 8 shards
Idea

• move write-intensive data to Redis
Idea

• move write-intensive data to Redis
• first candidate: inventory
  - integer fields
  - frequently changing
Solution

- inventory = Redis hash
  - atomic increment / decrement!
Solution

- inventory = Redis hash
  - atomic increment / decrement!

- on-demand migration of users
  - with batch roll-up
Results

• quick win
  - implemented in 2 weeks
  - 10% less load on MySQL servers
Results

• quick win
  - implemented in 2 weeks
  - 10% less load on MySQL servers
• decision: move over more data
But ...

- “honeymoon soon over”
But ...

- “honeymoon soon over”
- growing memory usage (fragmentation)
  - servers need periodic “refresh”
  - replication dance
Current Status

- hybrid setup
  - 4 MySQL master-slave pairs
  - 4 Redis master-slave pairs
Current Status

• hybrid setup
  - 4 MySQL master-slave pairs
  - 4 Redis master-slave pairs
• evaluating other alternatives
  - Riak, Membase
Redis Intro
Case 1: Monster World
Case 2: Happy Hospital
Discussion
Challenge

- expected peak load:
  - 16000 concurrent users
  - 4000 requests/second
  - mostly writes
“Memory is the new Disk, Disk is the new Tape.”
— Jim Gray
Idea

• use Redis as main database
  - excellent (write) performance
  - virtual memory for capacity
Idea

- use Redis as main database
  - excellent (write) performance
  - virtual memory for capacity
- no sharding = simple operations
Data Model

- user = single Redis hash
  - each entity stored in hash field (serialized to JSON)
Data Model

- user = single Redis hash
  - each entity stored in hash field (serialized to JSON)
- custom Ruby mapping layer ("Remodel")
class User < Remodel::Entity
  has_many :pets, :class => Pet
  property :level, :class => Integer, :default => 1
  property :xp, :class => Integer, :default => 0
end

class Pet < Remodel::Entity
  property :pet_type, :class => String
end
<table>
<thead>
<tr>
<th></th>
<th>ul</th>
<th>ul_pets</th>
<th>p7</th>
<th>p8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1220032045</td>
<td></td>
<td>ul_pets</td>
<td>p7</td>
<td>p8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ul</td>
<td>ul_pets</td>
<td>ul_pets</td>
</tr>
<tr>
<td></td>
<td>1234599660</td>
<td>ul</td>
<td>ul_pets</td>
<td>ul_pets</td>
</tr>
</tbody>
</table>

- **ul**: `{"level": 4, "xp": 241}`
- **ul_pets**: `["p7", "p8"]`
- **p7**: `{"pet_type": "Cat"}`
- **p8**: `{"pet_type": "Dog"}`
- **ul**: `{"level": 1, "xp": 22}`
- **ul_pets**: `["p3"]`
  ```json```
  ```json```

  ...
<table>
<thead>
<tr>
<th>1220032045</th>
<th>ul</th>
<th>{“level”: 4, “xp”: 241}</th>
</tr>
</thead>
<tbody>
<tr>
<td>ul_pets</td>
<td>[“p7”, “p8”]</td>
<td></td>
</tr>
<tr>
<td>p7</td>
<td>{“pet_type”: “Cat”}</td>
<td></td>
</tr>
<tr>
<td>p8</td>
<td>{“pet_type”: “Dog”}</td>
<td></td>
</tr>
<tr>
<td>1234599660</td>
<td>ul</td>
<td>{“level”: 1, “xp”: 22}</td>
</tr>
<tr>
<td>ul_pets</td>
<td>[“p3”]</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ID</td>
<td>ul</td>
<td>ul_pets</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>1220032045</td>
<td>ul</td>
<td>ul_pets</td>
</tr>
<tr>
<td></td>
<td>{&quot;level&quot;: 4, &quot;xp&quot;: 241}</td>
<td>[&quot;p7&quot;, &quot;p8&quot;]</td>
</tr>
<tr>
<td>1234599660</td>
<td>ul</td>
<td>ul_pets</td>
</tr>
<tr>
<td></td>
<td>{&quot;level&quot;: 1, &quot;xp&quot;: 22}</td>
<td>[&quot;p3&quot;]</td>
</tr>
<tr>
<td>ID</td>
<td>ul</td>
<td>ul_pets</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>1220032045</td>
<td>ul</td>
<td>ul_pets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[&quot;p7&quot;, &quot;p8&quot;]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p8</td>
</tr>
<tr>
<td>1234599660</td>
<td>ul</td>
<td>ul_pets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[&quot;p3&quot;]</td>
</tr>
</tbody>
</table>
|               |                         | ...              | ...
<table>
<thead>
<tr>
<th>1220032045</th>
<th>ul</th>
<th>{&quot;level&quot;: 4, &quot;xp&quot;: 241}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ul_pets</td>
<td>[&quot;p7&quot;, &quot;p8&quot;]</td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>{&quot;pet_type&quot;: &quot;Cat&quot;}</td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>{&quot;pet_type&quot;: &quot;Dog&quot;}</td>
</tr>
<tr>
<td>1234599660</td>
<td>ul</td>
<td>{&quot;level&quot;: 1, &quot;xp&quot;: 22}</td>
</tr>
<tr>
<td></td>
<td>ul_pets</td>
<td>[&quot;p3&quot;]</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ID</td>
<td>ul</td>
<td>Level</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>1220032045</td>
<td>ul</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>ul_pets</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>Cat</td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>Dog</td>
</tr>
<tr>
<td>1234599660</td>
<td>ul</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ul_pets</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Virtual Memory

- server: 24 GB RAM, 500 GB disk
  - can only keep “hot data” in RAM
Virtual Memory

- server: 24 GB RAM, 500 GB disk
  - can only keep “hot data” in RAM
- 380 GB swap file
  - 50 mio. pages, 8 KB each
Dec 2010: Crisis

- memory usage growing fast
Dec 2010: Crisis

- memory usage growing fast
- cannot take snapshots any more
  - cannot start new slaves
Dec 2010: Crisis

- memory usage growing fast
- cannot take snapshots any more
  - cannot start new slaves
- random crashes
Analysis

- Redis virtual memory not compatible with:
  - persistence
  - replication
Analysis

• Redis virtual memory not compatible with:
  - persistence
  - replication

• need to implement our own!
Workaround

• “dumper” process
  - tracks active users
  - every 10 minutes, writes them into YAML file
ruby → redis → disk
ruby  redis  disk
ruby (redis)  disk
Workaround

- in case of Redis crash
  - start with empty database
  - restore users on demand from YAML files
Real Solution

- Redis “diskstore”
  - keeps all data on disk
  - swaps data into memory as needed
Real Solution

• Redis “diskstore”
  - keeps all data on disk
  - swaps data into memory as needed

• under development (expected Q2)
uptime_in_days: 102
total_commands_processed: 54428739517
vm_stats_swappin_count: 73139647
vm_stats_swappout_count: 77343129
db2:keys=4296618,expires=0
Results

- average response time: 10 ms
Results

• average response time: 10 ms
• peak traffic:
  - 1500 requests/second
  - 15000 Redis ops/second
Current Status

• very happy with setup
  - simple, robust, fast
  - easy to operate
• still lots of spare capacity
Redis Intro
Case 1: Monster World
Case 2: Happy Hospital
Discussion
Advantages

• order-of-magnitude performance improvement
  - removes main bottleneck
  - enables simple architecture
Disadvantages

• main challenge: durability
  - diskstore very promising
Disadvantages

• main challenge: durability
  - diskstore very promising
• no ad-hoc queries
  - think hard about data model
  - hybrid approach?
Conclusion

• Ruby + Redis = killer combo
Q & A