redis

Michael Parker
June 2, 2012

HACKER NEWS

What is Redis?

- Like the key-value store Memcache
- Optional persistence to disk
- Treats values not as opaque data, but as data structures
    - Calls itself a "data structure server"

Memcache

`key1`	`value1`
`key2`	`value2`
`key3`	`value3`

Redis

`key1`	`[1, 1, "foo"]`
`key2`	`{2, 3, "bar", "mgp"}`
`key3`	`"f1"→"v1", "f2"→"v2"`

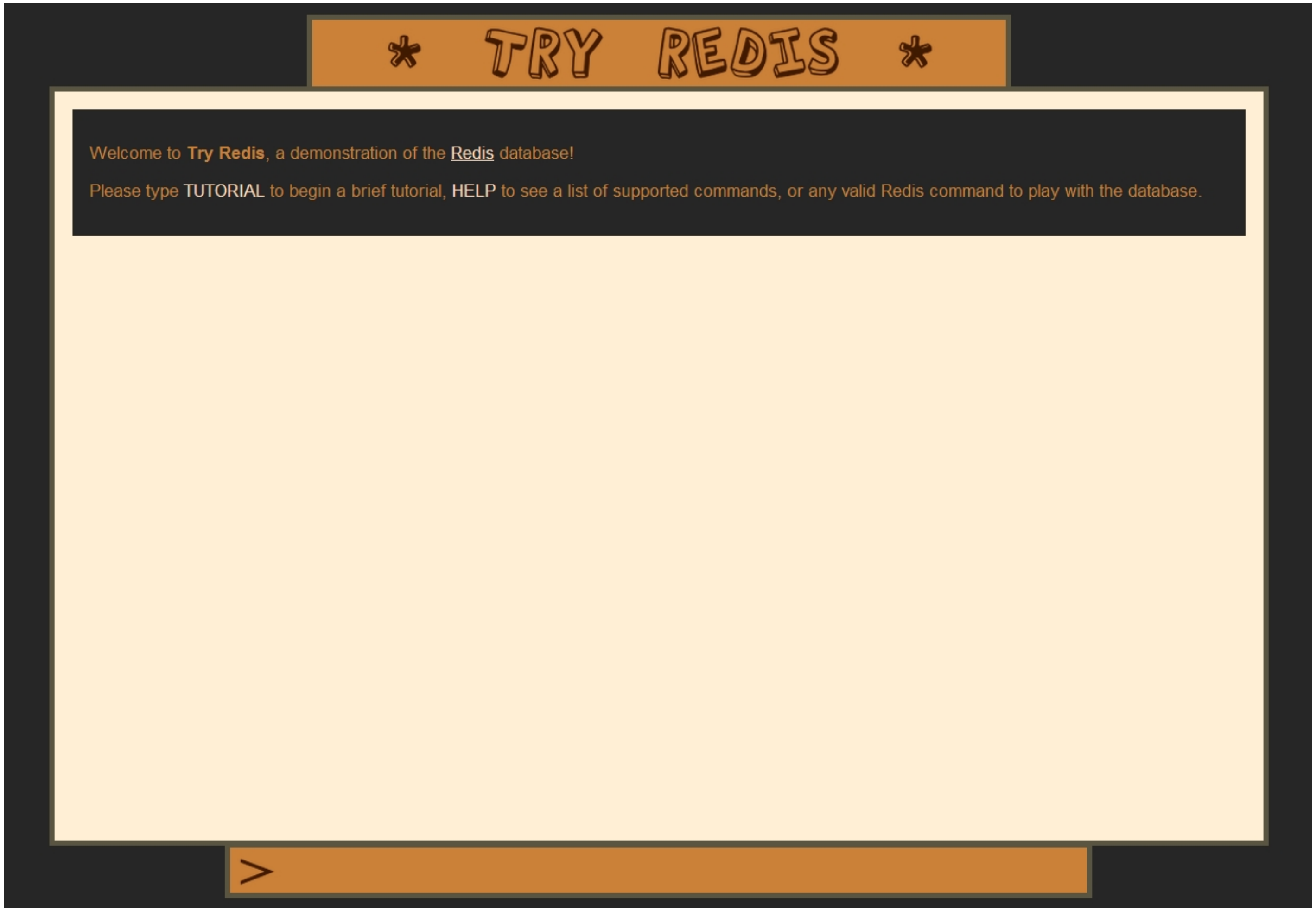
Types and abstractions in your code

- Numbers (integer, floating point, booleans)
- Strings (including characters)
- Hash tables (including objects)
    - get, set, contains, delete
- Lists
    - push, pop, get, set
- Sets
    - add, remove, union, intersection, difference
- Sorted sets
    - add, remove, first, last, range

Types and abstractions in Redis

- Numbers (integer, floating point, booleans)
- Strings (including characters)
- Hash tables (including objects)
  - get, set, contains, delete
- Lists
  - push, pop, get, set
- Sets
  - add, remove, union, intersection, difference
- Sorted sets
  - add, remove, first, last, range

Play along!
http://try.redis-db.com



Welcome to **Try Redis**, a demonstration of the Redis database!

Please type TUTORIAL to begin a brief tutorial, HELP to see a list of supported commands, or any valid Redis command to play with the database.

Numbers and strings

```
> SET "num1" 5
"OK"
> INCR "num1"
6
> SET "str1" "hackernews"
"OK"
> GET "str1"
"hackernews"
> MGET "num1" "str1" "unknown_key"
["6","hackernews",null]
```

Lists

```
> LPUSH "list1" 5
1
> LPUSH "list1" 4
2
> RPUSH "list1" 6
3
> LSET "list1" 1 "moo"
"OK"
> LRANGE "list1" 0 -1
["4","moo","6"]
```

Sets

```
> SADD "set1" "cats"
true
> SADD "set1" "dogs"
true
> SADD "set2" "dogs"
true
> SADD "set2" "monkeys"
true
> SUNION "set1" "set2"
["cats","dogs","monkeys"]
```

Maps (hashes)

```
> HSET "map1" "field1" "value1"
true
> HSET "map1" "field2" "value2"
true
> HEXISTS "map1" "field3"
false
> HGETALL "map1"
{"field1":"value1","field2":"value2"}
```

Transactions

- Commands to set/get values in maps, and add/remove values in lists and sets and sorted sets already take multiple arguments
- But transactions work across multiple keys
  - Atomicity
  - Fewer RPCs

Transactions - writing

```
> MULTI
"OK"
> HSET "map1" "field1" "value1"
"QUEUED"
> LPUSH "list1" 5
"QUEUED"
> SADD "set1" "cats"
"QUEUED"
> EXEC
[1,1,1]
```

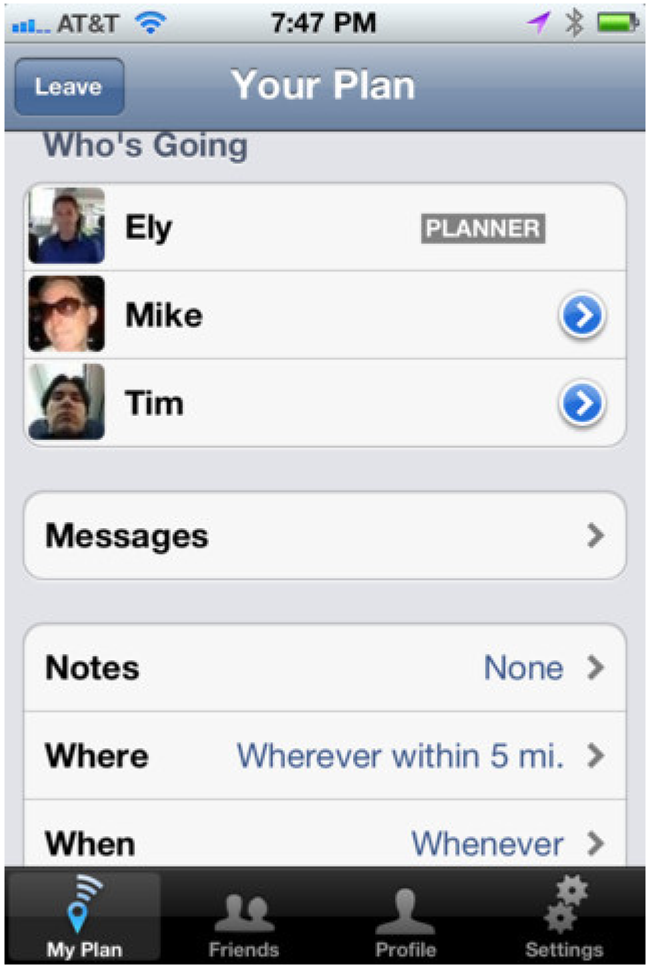
Transactions - reading

```
> MULTI
"OK"
> HGETALL "map1"
"QUEUED"
> LRANGE "list1" 0 -1
"QUEUED"
> SMEMBERS "set1"
"QUEUED"
> EXEC
[["field1","value1"],["5"],["cats"]]
```

Odds and ends

- Sorted sets
  - Allow you to implement a heap or priority queue
- Publish and subscribe
  - Like an event bus
  - Kind of out of place
  - Spun off from blocking pop on lists

ReadyUp!



AT&T 7:47 PM

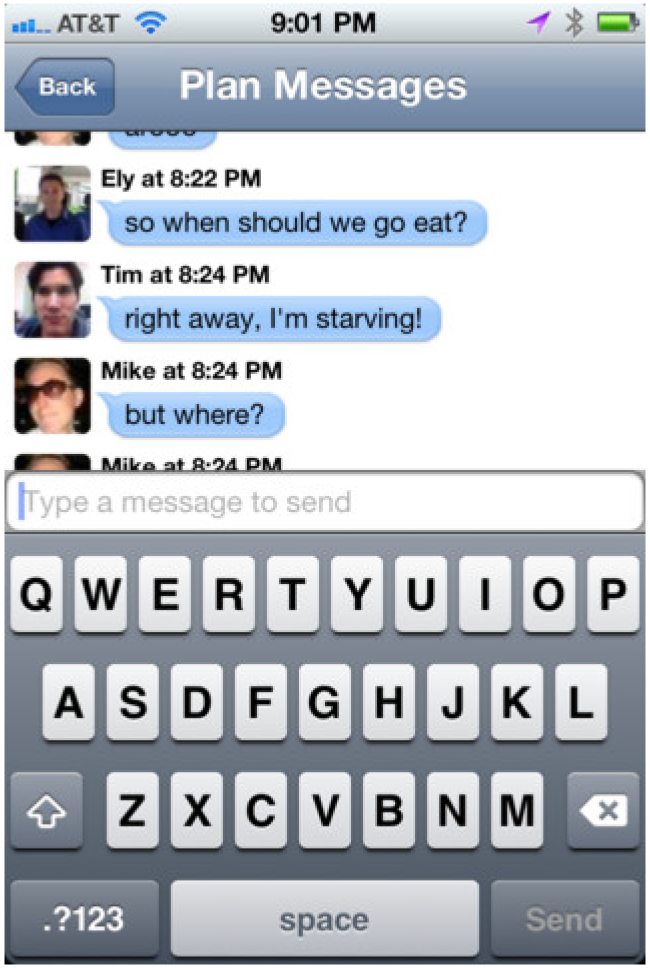
Leave **Your Plan**

Who's Going

Ely	PLANNER
Mike	
Tim	

Messages

Notes	None
Where	Wherever within 5 mi.
When	Whenever



AT&T 9:01 PM

Back **Plan Messages**

Ely at 8:22 PM
so when should we go eat?

Tim at 8:24 PM
right away, I'm starving!

Mike at 8:24 PM
but where?

Mike at 8:24 PM

Type a message to send

Creating plan identifiers

```
> INCR "next_plan_id"
1
> INCR "next_plan_id"
2
> INCR "next_plan_id"
3
```

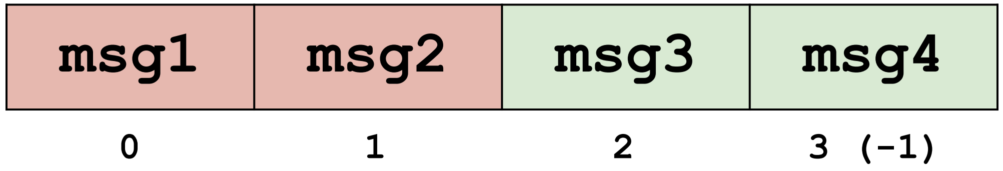
Do the same for user identifiers.

Reading new messages

Client sends **`plan_id`** and **`num_messages`**

```
messages_key = "%s_messages" % plan_id
new_messages = redis.lrange(
    messages_key, num_messages, -1)
```

So if **`num_messages = 2`**



Reading one plan

```
def get_plan(plan_id):
  pipeline = redis.pipeline()
  pipeline.hgetall("%s_hash" % plan_id)
  pipeline.smembers("%s_attendees" % plan_id)
  pipeline.lrange("%s_messages" % plan_id,
      0, -1)
  plan_data, attendees, messages =
      pipeline.execute()
 return Plan(
    plan_id, plan_data, attendees, messages)
```

Reading multiple plans - filling the pipeline

```
pipeline = redis.pipeline()
for plan_id in plan_ids:
  pipeline_get_plan(plan_id, pipeline)

def pipeline_get_plan(plan_id, pipeline)
  pipeline.hgetall("%s_hash" % plan_id)
  pipeline.smembers("%s_attendees" % plan_id)
  pipeline.lrange("%s_messages" % plan_id,
      0, -1)
```

hash$_1$	attendees$_1$	messages$_1$	hash$_2$	attendees$_2$	messages$_2$

Reading multiple plans - emptying the pipeline

```
results = pipeline.execute()
iterator = iter(results)
plans = [get_pipelined_plan(plan_id, iterator)
    for plan_id in plan_ids]

def get_pipelined_plan(plan_id, iterator):
  plan_data = next(iterator)
  attendees = next(iterator)
  messages = next(iterator)
  return Plan(
      plan_id, plan_data, attendees, messages)
```

In the works

- 2.6 (soon)
  - Lua scripting on the server-side
  - Performance and replication improvements
  - Redis ASCII art logo at startup
- 3.0
  - Clustering

Thanks!

http://redis.io

michael.g.parker@gmail.com
https://github.com/mgp

http://mgp.github.com/redis-la-hn.pdf