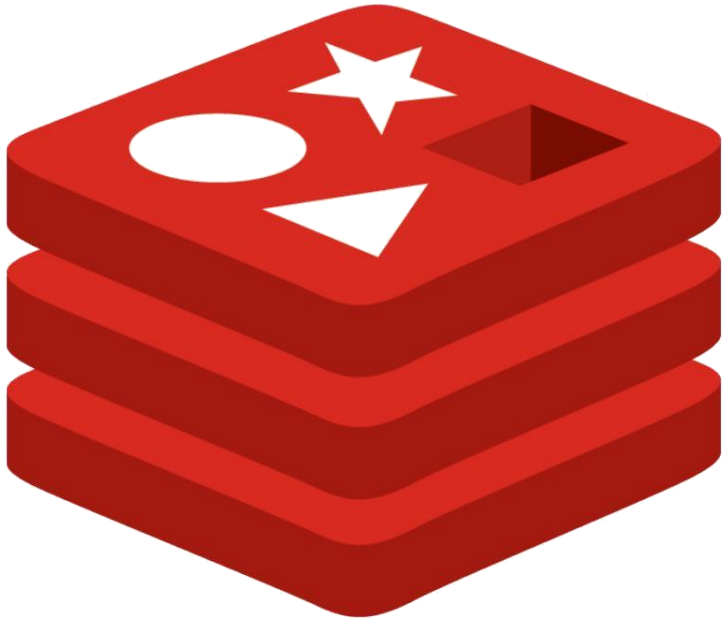

redis

"Pequeno Servidor de impressionante"

2011 Dvir Volk

Software Architect, Do @
dvir@doat.com http://doat.com

O que é redis

- Memcache-is de armazenamento de chaves / valores
- Mas também é persistente!
- E tem também os tipos de dados muito úteis:
  - listas
  - conjuntos
  - conjuntos ordenados
  - tabelas hash
  - buffer incremental (streaming)
- Código aberto; muito útil e amigável comunidade. Desenvolvimento é muito ativo e receptivo aos pedidos.
- Patrocinado pela VMWare
- Utilizado no mundo real: github, craigslist, EngineYard, ...
- Muito usado para banco de dados front-end, de busca, resolução geográfica

Principais Características e Coisas legais

- Todos os dados estão na memória (quase)
- Todos os dados são, eventualmente, persistente (Mas pode ser imediatamente)
- Lida com enormes cargas de trabalho facilmente
- Pesquisa em O(1)
- Ideal para cargas de trabalho de escrita-pesado
- Suporte para operações atômicas
- Suportes para transações
- Tem publish–subscribe funcionalidade /
- Toneladas de bibliotecas do cliente para todos os principais idiomas
- Único segmento, utiliza aync. IO
- scripting para LUA em breve

Um pouco de demo de performance

Realizado em um (Core i7 @ 2.2GHz)

- SET: 187265.92 solicitações por segundo
- GET: 185185.17 solicitações por segundo
- INCR: 190114.06 solicitações por segundo

- LPUSH: 190114.06 solicitações por segundo
- LPOP: 187090.73 solicitações por segundo

-
- SADD: 186567.16 solicitações por segundo
- POCA: 185873.61 solicitações por segundo

Escalonamento vertical

- replicação mestre-escravo fora da caixa
- Escravos podem ser feitas mestres online
- Atualmente não suporta o modo de cluster "real" ....
- ... Mas Redis-cluster para ser lançado em breve
- Você pode usa-lo do lado do cliente
- Único thread - num_cores / 2 instâncias na mesma máquina

Persistência

- Todos os dados são sincronizados para o disco - eventualmente, ou imediatamente
- Escolha o seu Vs. nível de risco desempenho
- Os dados são ou despejados em um processo bifurcada (RDB), ou escrito como um acréscimo apenas de Change-logs (AOF)
- Acrescente-only modo suporta disco transacional escreve para que você possa perder nenhum dado (custo: 99% de perda de velocidade :))
- arquivos AOF ficar enorme, mas Redis pode minimizá-los.
- Você pode salvar o estado explicitamente, fundo ou bloqueando

- configuração padrão:
  - Guardar após 900 segundos (15 min) se, pelo menos, uma chave alterada
  - Guardar após 300 segundos (cinco minutos) se pelo menos 10 teclas alterado
  - Salve após 60 segundos se pelo menos 10000 chaves mudou

Memória virtual

- Se seu banco de dados é muito grande - Redis pode lidar com a troca por conta própria.
- Chaves permanecem na memória e os valores menos utilizados são trocados para o disco.
- Trocando IO acontece em segmentos separados
- Mas se você precisar isto - não use Redis, ou acrescente mais máquinas

Mostre-me os recursos!

Agora vamos ver os recursos-chave:

- Get / Set / Incr - string / números
- Lists
- Sets
- Sorted Sets
- Hash Tables
- PubSub
- SORT
- Transações

Usaremos Redis-cli para os exemplos.

Alguns dos a saída foi modificado para facilitar a leitura.

O básico...

Obter / Sets - nada extravagante. Chaves são strings, vale tudo - basta citar espaços.

Redis> SET foo "bar" OK

Redis> GET foo
"bar"

Você pode atomicamente incrementar números

Redis> SET bar 337 OK

Redis> bar INCRBY 1000
(inteiro) 1337

Obtendo valores múltiplos de uma só vez

Redis> MGET foo bar
1. "bar"
2. "1337"

As chaves pode ter expiração automática

Redis> EXPIRE foo 1 (número

inteiro) 1

Redis> Get foo
(nil)

Tenha cuidado com EXPIRE - re-estabelecer um valor sem re-expirar ele irá remover a expiração!

Operações atômicas

GetSet coloca um valor diferente dentro de uma chave, retriving o antigo

Redis> bar SET foo
OK

Redis> GetSet foo baz "bar"

Redis> GET foo
"baz"

SETNX define um valor apenas se ele não existir

Redis> SETNX foo bar
* ESTÁ BEM*

Redis> SETNX foo baz
* FALHA *

SETNX + Timestamp => Locks nomeados! w00t!

Redis> SETNX myLock <current_time> OK

Redis> SETNX myLock <new_time>
* FALHA *

Observe que, se o cliente bloqueio trava que pode causar alguns problemas, mas pode ser resolvido facilmente.

operações de lista

- Listas são as suas listas ligadas comuns.
- Você pode empurrar e pop em ambos os lados, extrair faixa, redimensionar, etc.
- acesso aleatório e faixas em O (N)!

Redis> LPUSH foo bar
(inteiro) 1

Redis> LPUSH foo baz
(inteiro) 2

Redis> LRANGE foo 0 2
1. "baz"
2. "bar"

Redis> LPOP foo
"baz"

- BLPOP: Bloqueio POP - esperar até que a lista tem elementos e retira-os. Útil para o material em tempo real.

Redis> BLPOP Baz 10 [segundos]
..... Nós esperamos!

operações de conjunto (SET)

- Conjuntos são ... bem, conjuntos de valores únicos com push, pop, etc.
- Os conjuntos podem realizar operações de união, intersseção e diferença.
- Pode ser útil como chaves na construção de esquemas complexos.

```
Redis> SADD foo bar
(inteiro) 1
Redis> SADD foo baz
(inteiro) 1
Redis> SMEMBERS foo
["baz", "bar"]

Redis> SADD foo2 baz // << outro conjunto
(Inteiro) 1
Redis> SADD foo2 Raz
(inteiro) 1

Redis> SINTER foo foo2 // << apenas um elemento comum
1. "baz"
Redis> SUNION foo foo2 // << UNIÃO
[ "Raz", "bar", "baz"]
```

Conjuntos ordenados

- Mesmo como conjuntos, mas com pontuação por elemento
- Faixas classificadas, agregação de contagens em INTERSECT
- Pode ser usado como chaves ordenados em esquemas complexos
- Pense registros de tempo índice invertido, geohashing, intervalos de IP

```
Redis> Zadd foo 1337 hax0r
(inteiro) 1
Redis> Zadd foo 100 n00b
(inteiro) 1
Redis> Zadd foo 500 luser
(inteiro) 1

Redis> ZSCORE foo n00b "100"


Redis> ZINCRBY foo 2000 n00b "2100"


Redis> ZRANK foo n00b
(inteiro) 2
```

```
Redis> ZRANGE foo 0 10
1. "luser"
2. "hax0r"
3. "n00b"

Redis> ZREVRANGE foo 0 10
1. "n00b"
2. "hax0r"
3. "luser"
```

hashes

- As tabelas de hash como valores
- Pense em um armazenamento de objetos com acesso atômico se opor membros

```
Redis> HSEt barra foo 1 (número
inteiro) 1
Redis> HSEt foo baz 2 (número
inteiro) 1
Redis> HSEt foo foo foo
(inteiro) 1

Redis> HGETALL foo {
 "bar": "1", "baz": "2", "foo":
 "foo"}
```

```
Redis> bar HINCRBY foo 1 (número

inteiro) 2 redis> HGET foo bar "2"

redis> HKEYS foo
```

1. "bar"
2. "baz"
3. "foo"

PubSub - Publish / Subscribe

- Os clientes podem subscrever canais ou padrões e receber notificações quando mensagens são enviadas para os canais.
- Inscrição é O (1), postar mensagens é O (n)
- Pense chats, aplicações Rápidas: análises em tempo real, o Twitter

Redis> subscribe feed:joe feed:moe feed:
boe

// agora vamos esperar

....

<<<<< ----------

1. "mensagem"
2. "feed: joe"
3. "toda sua base é pertence-me"

Redis> publish feed:joe "all your base are belong to me"
(inteiro) 1 // *recebido por um*

SORT FTW!

Extensão

- Classificar conjuntos ou listas usando valores externos, e valores se juntam em uma só vez:

Extensão

- Extensão do Redis Key
- Classificar conjuntos ou listas usando valores externos, e valores se juntam em uma só vez:

```
SORT key
SORT key BY pattern (e.g. sort userIds BY user:*->age)
SORT key BY pattern GET othervalue
SORT userIds BY user:*->age GET user:*->name
```

Sintaxe

ASC | DESC, limite disponível, os resultados podem ser armazenados, a triagem pode ser numéricas ou alfabéticas

Tenha em mente que ele está bloqueando e Redis é único segmento. Talvez colocar um escravo de lado se você tem grandes tipos

Redo

Re

transações

- MULTI, ..., EXEC: Easy por causa do único segmento.
- Todos os comandos são executados depois de EXEC, valores de blocos e de retorno para os comandos como uma lista.
- Exemplo:

```
Redis> MULTI OK

Redis> SET "foo" "bar"
QUEUED
Redis> INCRBY "num" 1 redis
EM FILA> EXEC

1) OK
2) (número inteiro) 1
```

- As transações podem ser descartados com DISCARD.

- RELÓGIO permite bloquear as teclas enquanto você está na fila a sua transação, e evitar condições de corrida.

Lições Aprendidas

- Fragmentação de memória pode ser um problema com alguns padrões de uso. allocators alternativos (jemalloc, tcmalloc) facilitar isso.
- bug horrível com servidores 10.x Ubuntu e máquinas Amazon EC2 [resultou em longas e longas noites no escritório ...]
- 64 casos bit consumir muito mais memória RAM.
- Master / Slave sincronização longe de ser perfeito.
- NÃO use o comando KEYS !!!
- vm.overcommit_memory = 1
- Use MONITOR para ver o que está acontecendo

Exemplo: * Muito * Social Feed Simples

```
# vamos adicionar um par de seguidores
> > > client.rpush ( 'usuário: 1: seguidores', 2)
> > > numFollowers = client.rpush ( 'usuário: 1: seguidores', 3)
> > > MsgID = client.incr ( 'mensagens: id') OPERAÇÃO #ATOMIC

# Adiciona uma mensagem
> > > client.hmset ( 'mensagens:% s' % MsgID, { 'texto': 'Olá mundo', 'usuário': 1})

# distribuir aos seguidores
> > > seguidores = client.lrange ( 'usuário: 1: seguidores', 0, numFollowers)

> > > tubo client.pipeline = ()
> > > para f no seguidores:
    pipe.rpush ( 'utilizador:% s: alimentação' % f, MsgID)
> > > pipe.execute ()

> > > MsgID = client.incr ( 'mensagens: id') ID #increment
# ....repetir ... repeat..repeat..repeat ..
# agora obter Feed 2 de
> > > client.sort (name = 'user: 2: alimentação', obter = 'mensagens: * -> text') [ 'Olá mundo',
'bar foo']
```

Outras idéias caso de uso

- Geo Resolvendo com geohashing
  - Implementado e abriu por sinceramente https://github.com/doat/geodis
- análise em tempo real
  - utilização ZSET, SORT, œINCR de valores
- Chave API e gestão da taxa de
  - pesquisa, contadores de controle de taxa de chave muito rápidas usando INCR
- dados do jogo em tempo real
  - ZSETs para contagens elevadas, hashes para os usuários on-line, etc
- Banco de Dados Índice Shard
  - mapa key => id banco de dados. Contagem tamanho com SETS
- Comet - sem ajax polling
  - usar BLPOP ou pub / sub
- Queue Server
  - resque - uma grande porção da base do utilizador dos redis

Derreter - Meu mestre-plano maligno pouco

- Queríamos acesso freakin rápido aos dados sobre o front-end.
- mas a nossa capacidade para armazenar em cache dados vinculados personalizado e de consulta é limitado.
- Redis para o resgate!
- Mas nós ainda queremos que os dados estejam em um RDBMS.
- Por isso, fizemos uma estrutura para "derreter as fronteiras" entre eles ...

Apresentando fusão

- TODOS dados extremidade frontal está na RAM, denormalized e optimizado para a velocidade. negociações finais dianteiros apenas para Redis.
- Usamos definir recursos Redis' como chaves e vetores de pontuação.
- Todos os dados de back-end está em mysql, com um esquema normalizado administrável. As negociações de administração única para MySQL.
- A fila de sincronização no meio mantém ambas as extremidades até à data.
- A ORM simples é usado para gerenciar e sincronizar os dados.
- Automatiza a indexação em Redis, gera modelos de MySQL.
- Use o mesmo modelo em ambas as extremidades, ou criar conversões.
- gerador Id Central.

Derreter - um exemplo:

```
#sincronizar objetos:
com MySqlStore :
    Comercial = Comercial. obter ({Users.id: Int (1,2,3,4)})
    com RedisStore : para do
        utilizador dentro Comercial:
            Comercial. Salve • ( do utilizador)


#empurrando um novo item de alimentação da frente para trás:
com RedisStore :
    #criar um objeto - qualquer objeto!
    FeedItem = FeedItem ( ID do usuário , título , Tempo. Tempo ())
    #usar o modelo para salvá-lo
    Alimentação. Salve • ( FeedItem)
    #agora basta dizer a fila para colocá-lo no outro lado
    SyncQueue. pushItem ( açao = 'Update', modelo = FeedItem,
      fonte = '', redis dest = 'Mysql,
       identidade = FeedItem. identidade)
```

Em breve a um github perto de você! :)

Mais recursos

website Redis':

http://redis.io

Excelente e mais detalhada apresentação por Simon Willison:

http://simonwillison.net/static/2010/redis-tutorial/

Muito mais complexa clone twitter:

http://code.google.com/p/redis/wiki/TwitterAlikeExample

referência de comando completo:

http://code.google.com/p/redis/wiki/CommandReference