Riak Core
An Erlang Distributed Systems Toolkit
Andy Gross (@argv0)
Basho Technologies
SF Bay Erlang Factory 2011
Déjà vu

• 1999, Akamai: Large-scale log aggregation: consistent hashing, cluster membership, node monitoring

• 2005, Apple: Distributed filesystem: consistent hashing, cluster membership, node monitoring

• 2007, Mochi Media: Various apps: cluster membership, node monitoring
Riak Core

• Toolkit for writing highly-available distributed systems (based on Dynamo)
• Foundation of Riak KV and Riak Search
• ~8000 LOC
• Tested, production ready
consistent hashing

vector clocks

merkle trees

bloom filters

VNode Master

VNode

VNode

VNode

Storage Backend

Storage Backend

Storage Backend

node watcher

node event handler

handoff manager

handoff receiver

ring manager

ring event handler

monitoring

cluster config

debugging

utilities
Consistent Hashing

• Hashing technique that suffers minimal reshuffling when # of buckets changes
• Tolerant of divergent client views
• Coordinates both replica selection and replication
Consistent Hashing

A ring with 32 partitions

\[ 2^{160} \]

\[ 2^{160}/2 \]

\[ 2^{160}/4 \]

A single vnode/partition

hash(<<"artist">>, <<"REM">>)
N/R/W Values

- $N =$ number of replicas to store (on distinct nodes)
- $R =$ number of replica responses needed for a successful read (specified per-request)
- $W =$ number of replica responses needed for a successful write (specified per-request)
N/R/W Values

put("artist", "REM")

(N=3)
N/R/W Values

get/put("artist", "REM", R/W=2)

{ok, Object}
Vector Clocks

- Reasoning about time and causality in distributed systems is hard
- Integer timestamps don’t necessarily capture causality
- Vector clocks provide a happens-before relationship between two events
Vector Clocks

• Simple data structure: [(ActorID,Counter)]

• All data has an associated vector clock, actors update their entry when making changes

• ClockA happened-before ClockB if all actor-counters in A are less than or equal to those in B
Virtual Node Master

- Receives messages from coordinating FSMs
- Translates partition numbers to local PIDs and dispatches commands to individual vnodes
- One vnode_master per virtual node type (Riak KV, Riak Search)
Virtual Nodes

- One Erlang process per partition in the consistent hashing ring
- Receives work for its portion of the hash space
- Fundamental unit of replication, fault tolerance, concurrency
Virtual Node Behavior

```prolog
\-spec behaviour_info(atom()) -> 'undefined' | [{atom(), arity()}].
behaviour_info(callbacks) ->
  [{init,1},
   {handle_command,3},
   {handoff_starting,2},
   {handoff_cancelled,1},
   {handoff_finished,2},
   {handle_handoff_command,3},
   {handle_handoff_data,2},
   {encode_handoff_item,2},
   {is_empty,1},
   {terminate,2},
   {delete,1}];
behaviour_info(_Other) ->
  undefined.
```
Writing VNode Modules

• Define commands and handlers
• Define handoff behavior
• Start a riak_core_vnode_master for the vnode module
• riak_core:register_vnode_module(VNodeMod)
Node/Service Watcher

- gen_event process for monitoring nodes and local services
- Allows administrative removal of nodes
- Allows distributed applications to define services - service availability info is synchronized among nodes
- Used in the calculation of fallback nodes
Ring Manager

- Stores local copy of gossiped ring data
- Optimized for frequent reads, infrequent writes (using mochiglobal)
- Client applications manipulate ring data, Riak Core handles gossip/conflict resolution
Ring Event Handler

- gen_event that receives notifications on ring changes and broadcasts to subscribers
- Notifications of cluster membership changes
- Notifications of metadata changes
Handoff

• VNodes periodically check to see if they’re not on their “home” node and attempt handoff.

• Riak Core manages handoff connection management, your app handles encoding/decoding.

• Handoff is optional.
Other Utilities

• System monitoring
• Statistical data structures
• Utilities for
  • inter-node communication
  • tracing/debugging
  • vector clock/preference list manipulating
Riak KV

- **HTTP**
- **Protobufs**
- **KV Request FSMs**
- **VNode Master**
- **VNNode**
- **VNNode**
- **VNNode**
- **Storage Backend**
- **Storage Backend**
- **Storage Backend**
- **Bitcask**

- **node watcher**
- **node event handler**
- **handoff manager**
- **handoff receiver**
- **ring manager**
- **ring event handler**

- **consistent hashing**
- **vector clocks**
- **merkle trees**
- **bloom filters**

- **monitoring**
- **cluster config**
- **debugging**
- **utilities**
Riak Search

- VNode Master
  - VNode
  - Storage Backend
  - Bitcask
  - consistent hashing
  - vector clocks
  - merkle trees
  - bloom filters

- Protobufs
  -KV FSMs
    - Search FSMs

- node watcher
  - handoff manager
  - ring manager

- node event handler
  - handoff receiver
  - ring event handler

- monitoring
- cluster config
- debugging
- utilities

- SOLR
- Protobufs
- Riak Search
- Search FSMs
Future Directions

- Easier creation of new Riak Core based apps
- HTTP APIs for more functionality
- Stronger consistency support?
Greenspun’s Tenth Rule

“Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp”
Armstrong’s Corollary

“Any sufficiently complicated concurrent program in another language contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Erlang”
Basho’s Corollary

“Any sufficiently complicated Erlang distributed system contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Riak Core”
Thanks!