Riak

Daniel Reverri
dan@basho.com
What’s in store?

- At a High Level
- For Developers
- When and Why
- In Production
- Etc.
At a High Level
Riak

- Dynamo-inspired key/value store
- Extras: search, MapReduce, 2i, links, pre- and post-commit hooks, pluggable backends, HTTP and binary interfaces
- Written in Erlang with C/C++
- Open source under Apache 2 License
Riak History

- Started internally at Basho in 2007
- Deployed in production the same year
- Used as data store for Basho’s SaaS
- Open sourced in August 2009; Basho pivots
- Hit v1.0 in September 2011
- Now being used by 1000s in production
Riak’s Design Goals

- High-availability
- Low-latency
- Horizontal Scalability
- Fault Tolerance
- Ops Friendliness
- Predictability
Masterless Cluster of Nodes
For Developers
Buckets, Keys, Values

- Buckets contain many Keys
- Keys have Values
- Values can be of any type (content agnostic)
bucket

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>value</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>key</td>
<td>value</td>
</tr>
<tr>
<td>key</td>
<td>value</td>
</tr>
<tr>
<td>key</td>
<td>value</td>
</tr>
</tbody>
</table>
APIs

- HTTP (just like the web)
- Protocol Buffers (thank you, Google)
Querying

- Get, Put, Delete
- Map Reduce
- Full Text Search
- Secondary Indexes
Client Libraries

Ruby, Node.js, Java, Python, Perl, OCaml, Erlang, PHP, C, Squeak, Smalltalk, Pharoah, Clojure, Scala, Haskell, Lisp, Go, .NET, Play, and more (supported by either Basho or the community).
Modular

Client Libraries

HTTP
Protocol Buffers

Riak KV
Riak Search
Riak Pipe

Riak Core

Bitcask
LevelDB
Merge Index
Riak: when and why
When to Use Riak

- When you have enough data to require > 1 physical machine (preferably > 4)
- When availability is more important than consistency
- When your data can be modeled as keys and values
User/MetaData Store

- User profile storage for xfinityTV Mobile app
- Storage of metadata on content providers and licensing
- Strict Latency requirements
Notifications

Yammer notification module powered by Riak
Session Storage

- First Basho customer in 2009
- Every hit to a Mochi web property results in at least one read, maybe write to Riak
- Unavailability or high latency = lost ad revenue
Ad Serving

- OpenX will serve ~4T ad in 2012
- Started with CouchDB and Cassandra for various parts of infrastructure
- Now consolidating on Riak and Riak Core for real-time data serving
Asset Storage

- Voxer
- Users
- Media
- Timelines
Voxer: Initial Stats

- 11 Riak nodes (switched from CouchDB)
- 500GB Data set
- ~20k Peak Concurrent Users
- ~4MM Daily Request
Walkie Talkie App Voxer Is Going Viral On iPhones And Androids, Trending On Twitter

sodmg.com
@souljaboy

Voxer. Soulja Boy.

50+ RETWEETS  8 FAVORITES

5:02 AM - 6 Dec 11 via web - Embed this Tweet

Reply  Retweet  Favorite
Voxer: Post Growth

- ~60 Nodes
- 100s of TBs of data (>1TB daily)
- ~400k Concurrent Users
- >2B Daily Requests
Etc
New in 1.2

• LevelDB Improvements
• FreeBSD Support
• New Cluster Admin Tools
• Folsom for Stats
• Much much more
Future Work

- Active Anti Entropy
- Bonafide Data Types
- Solr Integration
- Dynamic Ring Sizing
- Consistency
- Lots of other hotness
In Production
Riak will run on pretty much anything except for Windows (right now)

Basho builds packages for various environments: FreeBSD, RHEL, Debs, CentOS, Ubuntu, OpenSolaris, and a few more.

If you build from source, you’ll need Erlang. We package it with our builds.
Command Line Tools

- riak – start, stop, ping
- riak-admin – status, cluster, etc.
Cluster/Node Admin

# Start a node
$ riak start

# Stop a node
$ riak stop

# Add a node to a cluster
$ riak-admin cluster join <node>

# Remove a node from cluster
$ riak-admin cluster leave
Upgrades

- Basho tests and verifies upgrades two releases back (i.e. 1.0 and 1.1 are verified for 1.2)
- Rolling upgrade: stop, upgrade, start
- WIP: Automating rolling upgrades with Chef
Backups

• Bitcask and LevelDB are both log-structured stores; cp, rsync, tar, custom backup tools will work

• FS-level snapshots of directory; can be done while node is running

• Consistent snapshots can be difficult; Point-in-time is easier to accomplish
Config Files

- app.config – controls all the Riak and Erlang application setting; ports, search, core, pipe, backends, etc.
- vm.args – handles embedded Erlang node settings; node IPS, cookies, heart, etc.
- Config changes require a node restart
Configuration Management

- Chef cookbook – Basho maintained; more than 10 community contributors
- Puppet module – community maintained
Benchmarking

- Riak ships with sane defaults; we favor safety over speed
- N=3, R=W=2
- Bad for micro-bencharks, good for production and durability
- Basho develops Basho Bench, an open source k/v benchmarking tool. Uses R for graphing.
Security

- Riak has *no* built-in security (neither authentication or authorization); this will be the case for the foreseeable future.

- Exposing your DB to the Internet is a bad idea, Riak or otherwise.

- We prefer to let you use your existing tools and methods (because everyone has their preference). Put Riak behind a firewall.
When and where?
Wednesday, October 10 through Thursday, October 11
at the W Hotel in downtown San Francisco.

http://ricon2012.com
Questions?

Daniel Reverri
dan@basho.com
Under the Hood
Consistent Hashing and Replicas
Virtual Nodes
Vector Clocks
Handoff and Rebalancing
Gossiping
Append-only stores
Erlang/OTP
Consistent Hashing

- 160-bit integer keyspace
Consistent Hashing

- 160-bit integer keyspace
- divided into fixed number of evenly-sized partitions

32 partitions $2^{160}/4$

$2^{160}/2$
Consistent Hashing

- 160-bit integer keyspace
- divided into fixed number of evenly-sized partitions
- partitions are claimed by nodes in the cluster

32 partitions $2^{160}/4$

$2^{160}/2$
Consistent Hashing

- 160-bit integer keyspace
- divided into fixed number of evenly-sized partitions
- partitions are claimed by nodes in the cluster
- replicas go to the N partitions following the key
Consistent Hashing

- 160-bit integer keyspace
- divided into fixed number of evenly-sized partitions
- partitions are claimed by nodes in the cluster
- replicas go to the $N$ partitions following the key

$\text{hash("meetups/nycdevops")}$
Disaster Scenario
Disaster Scenario

- node fails
Disaster Scenario

- node fails
- requests go to fallback

hash(“meetups/nycdevops”)
Disaster Scenario

- node fails
- requests go to fallback
- node comes back

hash("meetups/nycdevops")
Disaster Scenario

- node fails
- requests go to fallback
- node comes back
- "Handoff" - data returns to recovered node

hash("meetups/nycdevops")
Disaster Scenario

- node fails
- requests go to fallback
- node comes back
- “Handoff” - data returns to recovered node
- normal operations resume

hash(“meetups/nycdevops”)
Virtual Nodes

- Each physical machine runs a certain number of Vnodes
- Unit of addressing, concurrency in Riak
- Storage not tied to physical assets
- Enables dynamic rebalancing of data when cluster topology changes
Vector Clocks

- Data structure used to reason about causality at the object level
- Provides happened-before relationship between events
- Each object in Riak has a vector clock*
- Trade off space, speed, complexity for safety
Handoff and Rebalancing

- When cluster topology changes, data must be rebalanced
- Handoff and rebalancing happen in the background; no manual intervention required*
- Trade off speed of convergence vs. effects on cluster perfo
Gossip Protocol

- Nodes “gossip” their view of cluster state
- Enables nodes to store minimal cluster state
- Can lead to network chatiness; in OTP, all nodes are fully-connected
Append-only Stores

- Riak has a pluggable backend architecture
- Bitcask, LevelDB are used the most in production depending on use-case
- All writes are appends to a file
- This provide crash safety and fast writes
- Periodic, background compaction is required
Erlang/OTP

- Shared-nothing, immutable, message-passing, functional, concurrent
- Distributed systems primitives in core language
- OTP (Open Telecom Platform)
- Ericsson AXD–301: 99.9999999% uptime (31ms/year)