Distributed Erlang Systems In Operation

Andy Gross <andy@basho.com>, @argv0
VP, Engineering
Basho Technologies

Erlang Factory SF 2010

Architectural Goals

- Decentralized (no masters).
- Distributed (asynchronous, nodes use only local data).
- Homogeneous (all nodes can do anything).
- Fault tolerant (emergent goal).
- Observable

Anti-Goals

- Global state:
  - pg2/hot data in mnesia
  - globally registered names
- Distributed transactions
- Reliance on physical time

Compromise your Goals

- Decentralized (no masters).
- Distributed (nodes use only local data).
- Homogeneous (all nodes can do anything).
- No distributed transactions/global state.
- No reliance on physical time.

Systems Design

- Cluster Membership
- Load balancing/naming/resource allocation
- Liveness checking
- Soft Global State

Cluster Membership

- Option I: Use a configuration file:
  - Requires out-of-band sync of configuration file across machines.
  - Not "elastic" enough for some use-cases.
- Option II: Contact a seed node to join and use gossip protocol to propagate state.

Load Balancing and Resource Allocation

- Static assignment
- Round-robin/Random
- Static hashing: Nodes[hash(Item) mod length(Nodes)]
- Dynamo/Riak/Cassandra/Voldemort: Consistent Hashing

Liveness Checking

- nodes() and net_adm:ping() operations can be too low-level.
- Sometimes you'd like to divert traffic from a node at the application level while keeping distributed Erlang up.
- Use net_kernel:monitor_nodes() and an app-level mechanism for liveness.

Soft State/Gossip Protocols

- An eventually-consistent alternative to global state.
- Nodes make changes, gossip to another node.
- Nodes receive changes, merge with local state, gossip to another node.
- Requires up-front thought about data structures, dealing with slightly-stale data.

Running Your System

- Shipping code
- Upgrading code
- Debugging your own systems
- Living with other people's systems

Shipping Code

- Don’t rely on working Erlang on end-user machines (many Linux distros are broken or out of date).
- Ship code with an embedded runtime and libraries.
- Put version/build info in code.

Upgrading Code

- Hot code loading for small, emergency fixes.
- For new releases, reboot the node.
- Why not .appups?
  - Systems I’ve worked on have changed/ evolved too fast.
  - A reboot is a good test of resiliency.

Debugging Running Systems

- Remote Erlang shells are awesome, except when distributed Erlang dies (it happens).
- run_erl (or even screen(1)) give you a backdoor for when -remsh fails.
- rebar (http://hg.basho.com/rebar) makes this easy.
- What if you don’t have access to the box?

OPS - Other People's Systems

- Your Erlang, Enterprise firewalls.
- Erlang shell is powerful, but scary.
- Provide a debugging module.
- Get data out via HTTP/SMTP/SNMP
- Use disk_log/report_browser.

Questions?

“You know you have [a distributed system] when the crash of a computer you’ve never heard of stops you from getting any work done”

-Leslie Lamport

Resources

- unsplit: http://github.com/uwiger/unsplit
- gen_leader: http://github.com/KirinDave/gen_leader_revival
- Dynamo: http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html
- Hans Svensson: Distributed Erlang Application Pitfalls and Recipes: http://www.erlang.org/workshop/2007/proceedings/06svenss.ppt
- Consistent Hashing and Random Trees: Distributed Caching Protocols for relieving Hot Spots on the World Wide Web: http://bit.ly/LewinConsistentHashing