Scalable Persistent Storage for Erlang
Theory and Practice

Amir Ghaffari
Natalia Chechina, Phil Trinder

Jon Meredith

London Riak Meetup - October 22, 2013
Outline

• RELEASE Project
• General principles of scalable DBMSs
• NoSQL DBMSs for Erlang
• Riak 1.1.1 Scalability in Practice
• Investigating the scalability of distributed Erlang
• Riak Elasticity
• Conclusion & Future work
RELEASE project

• RELEASE is an European project aiming to scale Erlang onto commodity architectures with 100,000 cores.
The RELEASE consortium work at following levels:

- Virtual machine
- Language
  - scalable Computation model
  - Scalable In-memory data structures
    - Scalable Persistent data structures
- Infrastructure levels
- Profiling and refactoring tools
General principles of scalable DBMSs

**Data Fragmentation**

1. **Decentralized model** (e.g. P2P model)
2. **Systematic load balancing** (make life easier for developer)
3. **Location transparency**

e.g. 20k data is fragmented among 10 nodes
General principles of scalable DBMSs

**Replication**

1. **Decentralized model** (e.g. P2P model)
2. **Location transparency**
3. **Asynchronous replication** (write is considered complete as soon as the node acknowledges it)

*Example: Key X is replicated on three nodes.*
General principles of scalable DBMSs

CAP theorem: cannot simultaneously guarantee:

- **Partition tolerance**: system continues to operate despite nodes can't talk to each other
- **Availability**: guarantee that every request receives a response
- **Consistency**: all nodes see the same data at the same time

Not achievable because network failures are inevitable

Solution: Eventual consistency and reconciling conflicts via data versioning

ACID=Atomicity, Consistency, Isolation, Durability
# NoSQL DBMSs for Erlang

<table>
<thead>
<tr>
<th></th>
<th>Mnesia</th>
<th>CouchDB</th>
<th>Riak</th>
<th>Cassandra</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fragmentation</strong></td>
<td>• Explicit placement</td>
<td>• Explicit placement</td>
<td>• Implicit placement</td>
<td>• Implicit placement</td>
</tr>
<tr>
<td></td>
<td>• Client-server</td>
<td>• Multi-server</td>
<td>• Peer to peer</td>
<td>• Peer to peer</td>
</tr>
<tr>
<td></td>
<td>• Automatic by using</td>
<td>• Lounge is not part of each</td>
<td>• Automatic by using consistent hash</td>
<td>• Automatic by using consistent hash</td>
</tr>
<tr>
<td></td>
<td>a hash function</td>
<td>CouchDB node</td>
<td>technique</td>
<td>technique</td>
</tr>
<tr>
<td><strong>Replication</strong></td>
<td>• Explicit placement</td>
<td>• Explicit placement</td>
<td>• Implicit placement</td>
<td>• Implicit placement</td>
</tr>
<tr>
<td></td>
<td>• Client-server</td>
<td>• Multi-server</td>
<td>• Peer to peer</td>
<td>• Peer to peer</td>
</tr>
<tr>
<td></td>
<td>• Asynchronous (Dirty operation)</td>
<td>• Asynchronous</td>
<td>• Asynchronous</td>
<td>• Asynchronous</td>
</tr>
<tr>
<td><strong>Partition Tolerant</strong></td>
<td>• Strong consistency</td>
<td>• Eventual consistency</td>
<td>• Eventual consistency</td>
<td>• Eventual consistency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Multi-Version Concurrency</td>
<td>• Vector clocks for reconciliation</td>
<td>• Use timestamp to reconcile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Control for reconciliation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>**Query Processing &amp;</td>
<td>• The largest possible Mnesia</td>
<td>• No limitation</td>
<td>• Bitcask has memory limitation</td>
<td>• No limitation</td>
</tr>
<tr>
<td>Backend Storage**</td>
<td>table is 4Gb</td>
<td>• Supports Map/Reduce Queries</td>
<td>• LevelDB has no limitation</td>
<td>• Supports Map/Reduce queries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Supports Map/Reduce queries</td>
<td></td>
</tr>
</tbody>
</table>
Initial Evaluation Results

**General Principles**

**Initial Evaluation**

• Mnesia
• CouchDB
• Riak
• Cassandra

Scalable persistent storage for SD Erlang can be provided by Dynamo-style DBMSs such as Riak, Cassandra
Riak Scalability in Practice

- Basho Bench: a benchmarking tool for Riak
- We measure Basho Bench on 348-node Kalkyl cluster

- **Scalability**: How does adding more Riak nodes affect the throughput?

- There are two kinds of nodes in a cluster:
  - *Traffic generators*
  - *Riak nodes*
Node Organisation

Heuristic: one traffic generator per 3 Riak nodes
Riak 1.1.1 Scalability

Scalability benchmark

Benchmark on 100-node cluster (800 cores)
Failures
Profiling Resource Usage

CPU Usage

Graph showing CPU usage vs. number of nodes for Riak nodes and Traffic generators.
Profiling Resource Usage

Disk usage

Disk Usage (Percentage)

Number of nodes

Disk Usage

Risk nodes
Traffic generators

DISK Usage
Profiling Resource Usage

Memory Usage

Number of nodes

Memory usage (Percentage)
Profiling Resource Usage

Network Traffic of Generator Nodes
Profiling Resource Usage

Network Traffic of Riak Nodes

Network Traffic of Riak Nodes

- Sent packets
- Received packets
- Retransmission packets

Number of nodes

Network traffic (Number of packets)
**Bottleneck for Riak Scalability**

*CPU, RAM, Disk, and Network* profiling reveal that they can't be bottleneck for Riak scalability.

Is the Riak scalability limits due to limits in distributed Erlang?

To find out, let's measure the scalability of distributed Erlang.
DE-Bench

• DE-Bench: a benchmarking tool for distributed Erlang

• It is based on Basho Bench

• Measures the throughput of a cluster of Erlang nodes

• Records the latency of distributed Erlang commands individually
Distributed Erlang Commands

- **Spawn/RPC**: peer to peer commands
- **register_name**: global name tables located on every node
- **unregister_name**: global name tables located on every node
- **whereis_name**: a lookup in the local table
DE-Bench’s P2P Design
Frequency of Global Operation

Global Operations limit the scalability of distributed Erlang

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Max Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>30 nodes</td>
</tr>
<tr>
<td>0.5%</td>
<td>50 nodes</td>
</tr>
<tr>
<td>0.33%</td>
<td>70 nodes</td>
</tr>
<tr>
<td>0%</td>
<td>1600 nodes</td>
</tr>
</tbody>
</table>
Riak Software Scalability

• Monitoring `global.erl` module from OTP library shows that Riak does NOT use any global operation.

• Instrumenting `gen_server.erl` module reveals that:
  ➢ Of the 15 most time-consuming operations, only the time of `rpc:call` grows with cluster size.
  ➢ Moreover, of the five Riak RPC calls, only `start_put_fsm` function from module `riak_kv_put_fsm_sup` grows with cluster size.
Eliminating the Bottlenecks

• Independently, Basho identified that two supervisor processes, i.e. *riak_kv_get/put_fsm_sup*, become bottleneck under heavy load, exhibiting build up in message queue length.

• To improve the Riak scalability in version 1.3 and 1.4 Basho applied a number of techniques and introduced new library *sidejob* ([https://github.com/basho/sidejob](https://github.com/basho/sidejob)).
Riak 1.1.1 Elasticity

Time-line shows Riak cluster losing and gaining nodes
Riak1.1.1 Elasticity

How Riak cluster deals with nodes leaving and joining
Observation

• Number of failures (37)

• Number of successful operations (approximately 3.41 million)

• When failed nodes come back up, the throughput has grown that shows Riak1.1.1 has a good elasticity.
Conclusion and Future work

✓ Our benchmark confirms that Riak has a good elasticity.

✓ We establish for the first time scientifically the scalability limit of Riak 1.1.1 as 60 nodes.

✓ We have shown how global operations limits the scalability of distributed Erlang.

✓ Riak scalability bottlenecks are eliminated in Riak versions 1.3 and upcoming versions.

✓ In RELEASE, we are working to scale up distributed Erlang by grouping nodes in smaller partitions.
References

- DE-Bench [https://github.com/amirghaffari/DEbench](https://github.com/amirghaffari/DEbench)


- Clusters at UPPMAX [http://www.uppmax.uu.se/hardware](http://www.uppmax.uu.se/hardware)

- Sidejob [https://github.com/basho/sidejob](https://github.com/basho/sidejob)