Scalable Persistent Storage for Erlang: Theory and Practice

Amir Ghaffari, Natalia Chechina, Phil Trinder

April 22, 2013
Outline

• What does this research aim to do as a part of the RELEASE Project?
• General principles of scalable DBMSs
• NoSQL DBMSs for Erlang
• Measuring the Reliability of Riak
• Scalability of Riak in Practice
• Investigating the scalability of distributed Erlang
• Conclusion & Future work
RELEASE project

- RELEASE is an European project aiming to scale Erlang onto commodity architectures with 100000 cores.
Scaling Erlang in three levels:

- Distributed Erlang (SD Erlang)
- In-memory Data Structure (ETS table)
- Scalable Persistent Storage for Erlang
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 on node acknowledges it)

*e.g. 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

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 a hash function</td>
<td>• Lounge is not part of each CouchDB node</td>
<td>• Automatic by using consistent hash technique</td>
<td>• Automatic by using consistent hash 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</strong></td>
<td>• Strong consistency</td>
<td>• Eventual consistency</td>
<td>• Eventual consistency</td>
<td>• Eventual consistency</td>
</tr>
<tr>
<td>tolerance</td>
<td></td>
<td>• Multi-Version Concurrency Control for reconciliation</td>
<td>• Vector clocks for reconciliation</td>
<td>• Use timestamp to reconcile</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>• The largest possible Mnesia table is 4Gb</td>
<td>• No limitation</td>
<td>• Bitcask has memory limitation</td>
<td>• No limitation</td>
</tr>
<tr>
<td>limitation</td>
<td></td>
<td></td>
<td>• LevelDB has no limitation</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 Riak (or Cassandra)
Availability and Scalability of Riak in Practice

- Basho Bench, a benchmarking tool for Riak
- We use Basho Bench on 348-node Kalkyl cluster
- How does Riak cope with node failure? (Availability)
- How adding more Riak nodes affect the throughput? (Scalability)
- There are two kinds of nodes in a cluster:
  - Traffic generators
  - Riak nodes
Node Organisation

We use one traffic generator per 3 Riak nodes
Traffic Generator

Basho Bench (An Erlang Application)

- Random Key/Value Generator
- Uniform Random Generator
- Config file:
  - List of IP address
  - List of ports
  - List of database operations

Update: Worker1
Insert: Worker89
Read: Worker90

HTTP/Protocol Buffers Interface

PUT: Riak node
POST: Riak node
GET: Riak node
Riak Availability

Time-line shows Riak cluster losing nodes
Riak Availability

How Riak deals with failures
Observation

• Number of failures (37)
• Number of successful operations (approximately 3.41 million)

• When failed nodes come back up, the throughput has grown which means Riak has a good elasticity.
Riak Scalability

Benchmark on 348-node Kalkyl cluster at Uppsala University
Failure
What is the Bottleneck?

CPU Usage

CPU usage

Riak nodes
Traffic generators

Number of nodes

CPU usage (Percentage)
Profiling DISK

DISK Usage

Disk usage

Traffic generators

Riak nodes

Number of nodes

Disk usage (Percentage)
Profiling RAM

Memory Usage

Number of nodes

Memory usage (Percentage)
Profiling-Network (Generator)

Network Traffic of Generator Nodes
Profiling-Network (Riak)

Network Traffic of Riak Nodes

Sent packets
Received packets
Retransmission packets
Bottleneck for Riak Scalability

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

Is Riak scalability limits due to limits in distributed Erlang? To find it, we need to measure the scalability of distributed Erlang.
DEbench

- We design Debench for measuring the scalability of distributed Erlang
- Based on Basho Bench
- Measures the Throughput and Latency of Distributed Erlang commands
Distributed Erlang Commands

• **Spawn**: a peer to peer command

• **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
DEbench P2P Nodes
Scalability of Distributed Erlang

0.5% Global operation

- Throughput peaks at 50 nodes
- Little improvement beyond 40 nodes
Frequency of Global Operation

<table>
<thead>
<tr>
<th></th>
<th>Max Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1% Global Operation</td>
<td>30 nodes</td>
</tr>
<tr>
<td>0.5% Global Operation</td>
<td>50 nodes</td>
</tr>
<tr>
<td>0.33% Global Operation</td>
<td>70 nodes</td>
</tr>
</tbody>
</table>
What is the Bottleneck?

Latency for register and unregister for 2% global update
What is the Bottleneck?

Latency of spawn

Latency (Microseconds) vs. Number of nodes for Spawn and whereis_name commands.
What is the Bottleneck?

Latency of `whereis_name`
Conclusion and Future works

• Our benchmark confirms that Riak is highly available and fault-tolerant.

• We have discovered the scalability limits of Riak is ~60 nodes

• Global operation limits the scalability of distributed Erlang.

• In the RELEASE, we are working to scale up Distributed Erlang by designing and implementing Scalable Distributed Erlang (SD Erlang)
Thank you!