Scalable Persistent Storage for Erlang: Theory and Practice

Amir Ghaffari, Natalia Chechina, Phil Trinder

May 7, 2013
Outline

• Why Persistent Storage?
• General principles of scalable DBMSs
• NoSQL DBMSs for Erlang
• Reliability of Riak in Practice
• Scalability of Riak in Practice
• Investigating the scalability of distributed Erlang
• Conclusion & Future work
RELEASE is an European project aiming to scale Erlang onto commodity architectures with 100000 cores.
Why Persistent Storage?

- Erlang application need to store their data persistently.

- Scalability limits of persistent storage can limit the scalability of Erlang application.
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

*Example: 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

![Diagram of replicated keys on multiple nodes](image-url)
**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 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 Control for reconciliation</td>
<td>• Vector clocks for reconciliation</td>
<td>• Use timestamp to reconcile</td>
</tr>
<tr>
<td><strong>Query Processing &amp; Backend 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></td>
<td></td>
<td>• Support Map/Reduce Queries</td>
<td>• LevelDB has no limitation</td>
<td>Support Map/Reduce queries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Support 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-like DBMSs, e.g. Riak, 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
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

Number of failures

FAILURES

NUMBER OF NODES

Failure graph showing the number of failures increasing with the number of nodes.
What is the Bottleneck?
Profiling DISK

DISK Usage

Disk usage

Risk nodes
Traffic generators

Disk usage (Percentage)

Number of nodes

10  20  30  40  50  60  70  80  90  100

0  1  2  3  4  5  6  7  8  9  10
Profiling RAM

Memory Usage

- Risk nodes
- Traffic generators
Profiling-Network (Generator)

Network Traffic of Generator Nodes
Profiling-Network (Riak)

Network Traffic of Riak Nodes
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.
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

### Frequently

<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>

*Graph showing the scalability of Distributed Erlang with different frequencies of global operation.*
What is the Bottleneck?

Latency for register and unregister for 2% global update

Latency for global update commands

Number of nodes

Latency (Microseconds)
What is the Bottleneck?

Latency of Spawn and `whereis_name` commands

Latency (Microseconds)

Number of nodes

Latency of `spawn`
What is the Bottleneck?

Latency of *whereis_name*

![Graph showing Latency of whereis_name](chart.png)
Conclusion and Future work

• 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.

• We are trying to find the Riak global operations.

• In RELEASE, we are working to scale up Distributed Erlang by grouping nodes in smaller partitions.
Thank you!