Scalable Persistent Storage for Erlang: Theory and Practice

Amir Ghaffari[1], Natalia Chechina[1], Phil Trinder[1], Jon Meredith[2]

[1] The University of Glasgow, G12 8QQ, UK [2] Basho Technologies, Inc.

RELEASE
http://www.release-project.eu/

Introduction

The RELEASE project aims to improve the scalability of Erlang on emergent commodity architectures with 10^5 cores.

We anticipate that such architectures require scalable and available persistent storage on up to 100 hosts.

This research investigates the provision of persistent data structures by studying the ability of Erlang distributed DBMS to scale on our target 10^5 core architectures.

Theoretical Analysis

Mnesia and **CouchDB** have scalability limitations:
- Explicit placement of replicas and fragments
- A single point of failure due to the lack of a P2P model

Dynamo-style NoSQL DBMS like Riak and Cassandra:
- Do have a potential to provide scalable persistent storage for large distributed architectures

Scalability of Erlang DBMSs

The requirements for scalable and available persistent storage

Replication
- Decentralized model
- Location transparency
- Asynchronous replication

Data Fragmentation
- Decentralized model
- Systematic load balancing
- Location transparency

Query Processing
- Location Transparency
- Local Execution
- Parallelism

Mnesia, CouchDB, Cassandra, riak

Availability
- Eventual consistency
- Reconciling conflicts

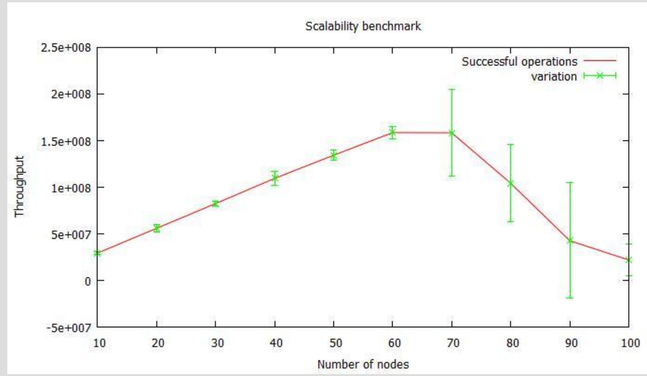
	Mnesia	CouchDB	Riak	Cassandra
Fragmentation	• Explicit placement • Client-server • Automatic by using a hash function	• Explicit placement • Multi-server • Lounge is not part of each CouchDB node	• Implicit placement • Peer to peer • Automatic by using consistent hash technique	• Implicit placement •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

Riak Scalability

We investigate the scalability of Riak version 1.1.1 using the Basho Bench benchmarking tool on the Kalkyl cluster at Uppsala University.

We measure throughput rises vs. the number of Riak nodes
- Every experiment is repeated 3 times
- The scalability diagram depicts the mean values
- The green line represents variation from the mean

Riak scales linearly up to 60 nodes, but it does not scale beyond 60 nodes.



Resource usage
- Maximum RAM usage -- 3%
- Maximum disc usage -- 10%
- Maximum core usage -- 5.5 of 8 cores

Network profiling
The number of *retransmitted* packets (200 packets) is negligible in comparison with the total number of successfully transmitted packets $5*10^8$ packets.

The scalability of Riak software.
- Riak makes no *global.erl* calls
- Of the 15 most time-consuming *gen_server.erl* operations only one, *rpc:call* grows with cluster size
- Of the 5 Riak RPC calls only *start_put_fsm* function from module *riak_kv_put_fsm_sup* grows with cluster size
- *riak kv_get/put_fsm_sup* and statistics reporting are supervisor processes bottleneck
- Riak version 1.3 and 1.4 employ the library *sidejob* to tackle the problem
- *sidejob* library is available at: https://github.com/basho/sidejob

Conclusion & Future Work

We have analysed Erlang DBMSs against the requirements for scalable persistent storage
- *Dynamo-style* NoSQL DBMS have a potential to provide scalable persistent storage for large distributed architectures
- We have shown that Riak 1.1.1 already provides scalable persistent storage on up to *60 nodes*
- We further show that resources like disc and network do not limit scalability, and identify two bottlenecks for improvement
- The Riak single-process bottleneck issues are addressed in versions 1.3 and 1.4.

We are investigating the scalability limitations of Distributed Erlang, and developing techniques to further improve the scalability of persistent storage engines implemented in distributed Erlang

References

- Ghaffari, N. Chechina, P. Trinder, and J. Meredith. Scalable Persistent Storage for Erlang: Theory and Practice. In Proceedings of the Twelfth ACM SIGPLAN Workshop on Erlang, pages 73-74, September 2013.
- N. Chechina, P. Trinder, A. Ghaffari, R. Green, K. Lundin, and R. Virding. The Design of Scalable Distributed Erlang. In Draft Proceedings of the Symposium on Implementation and Application of Functional Languages 2012, July 2012.