Scalable Persistent Storage for Erlang

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

Amir Ghaffari, Natalia Chechina, Phil Trinder

University of Glasgow | School of Computing Science

Introduction

- RELEASE, an European project, aims to improve the scalability of Erlang.
- Erlang is an open-source functional programming language for building parallel and distributed system.
- A key requirement for a scalable language is scalable persistent storage.
- This research tries to find a scalable persistent storage for Erlang.

Step1

Challenge: Indentify the principles of scalable persistent storage

Achievement:

- Data Fragmentation:
  - Decentralized Model
  - Systematic Load Balancing
  - Location Transparency
- Replication:
  - Decentralized Model
  - Location Transparency
  - Asynchronous Replication
- Availability:
  - Eventual Consistency
  - Reconciling Conflicts via Data Versioning
- Query Processing:
  - Location Transparency
  - Local Execution
  - Parallelism

Step2

Challenge: Evaluate some popular DBMSs for Erlang, i.e. Mnesia, CouchDB, Riak, and Cassandra against the principles outlined in step1.

Achievement:

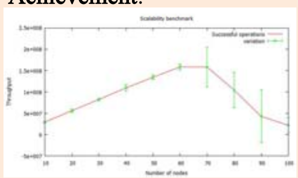
	Mnesia	CouchDB	Riak	Cassandra
Fragmentation	•Explicit placement •Client-server •Automatic by using a hash function	•Explicit placement •Multi-server •Lounge is not part of each CouchDB node	•Implicit placement •Peer to peer •Automatic by using consistent hash technique	•Implicit placement •Peer to peer •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

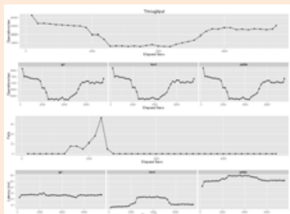
✓ Dynamo-style DBMSs like Riak and Cassandra can provide scalable persistent storage for Erlang

Step3

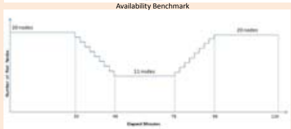
Challenge : Investigate the scalability and availability of Riak in practice.

Achievement:





Availability Benchmark



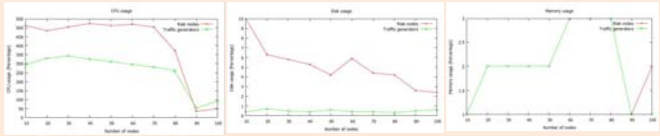
- Riak version 1.1.1 doesn't scale beyond ~60 nodes
- Riak provides a highly available and fault-tolerant service

Step4

Challenge : Investigate the reasons for the Riak 1.1.1 scalability limitation.

Achievement:

- Measuring the processor, RAM, disk, and network usage shows that they can't be a bottleneck for Riak scalability.



- By instrumenting the global and gen_server OTP libraries we identify a specific Riak remote procedure call (start_put_fsm function from module riak_kv_put_fsm_sup) that fails to scale.
- To avoid single process bottleneck, in Riak version 1.3 get/put FSM processes are created directly on the external API-handling processes that issue the requests, i.e. Riak_kv_pb object (protocol buffers interface) or riak_kv_wm_object (REST interface).

Conclusion and Future Work

✓ We identified the requirements for scalable persistent storage and we evaluate some popular NoSQL DBMSs for Erlang against these requirements. We concluded that Dynamo-style DBMSs like Riak and Cassandra meet the requirements.

✓ Scalability benchmark shows that Riak 1.1.1 doesn't scale beyond ~60 nodes.

✓ The availability benchmark shows that Riak provides a good elasticity and a highly available and fault-tolerant service.

✓ we identify a specific Riak remote procedure call that fails to scale. We discuss how that single process bottleneck has been removed in Riak versions 1.3 and 1.4.

✓ The RELEASE project aims to improve the scalability of Erlang. We hope that improvements can be leveraged into persistent storage engines implemented in Distributed Erlang.