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 project

- RELEASE is an European project aiming to scale Erlang onto commodity architectures with 100000 cores.

Why Persistent Storage?

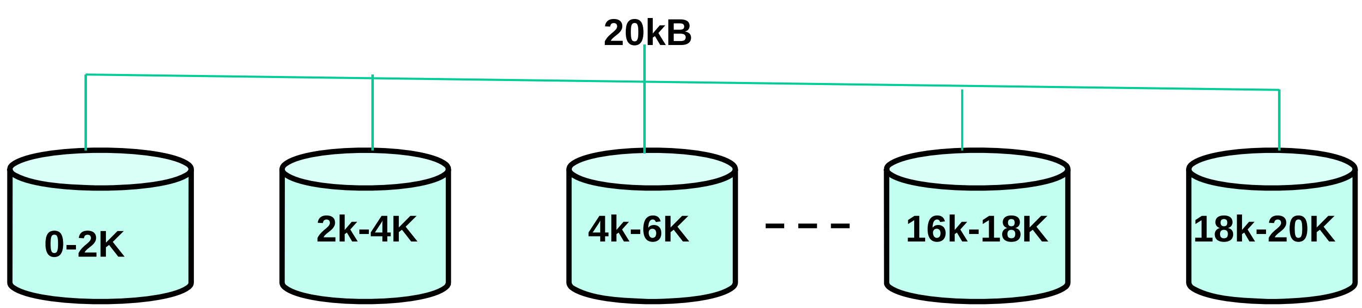
- Erlang application need to store their data persistently.
- Scalability limits of persistent storage can limit the scalability of Erlang application.

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

e.g. 20k data is fragmented among 10 nodes



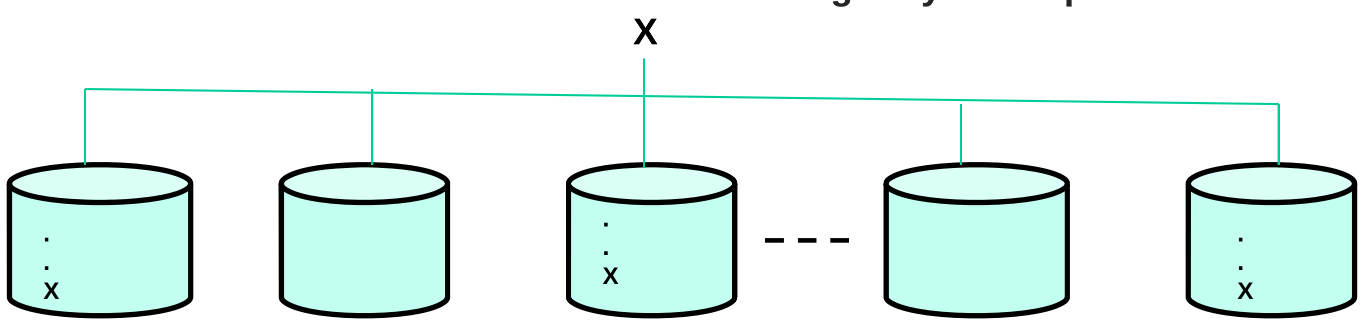
RELEASE

General principles of scalable DBMSs

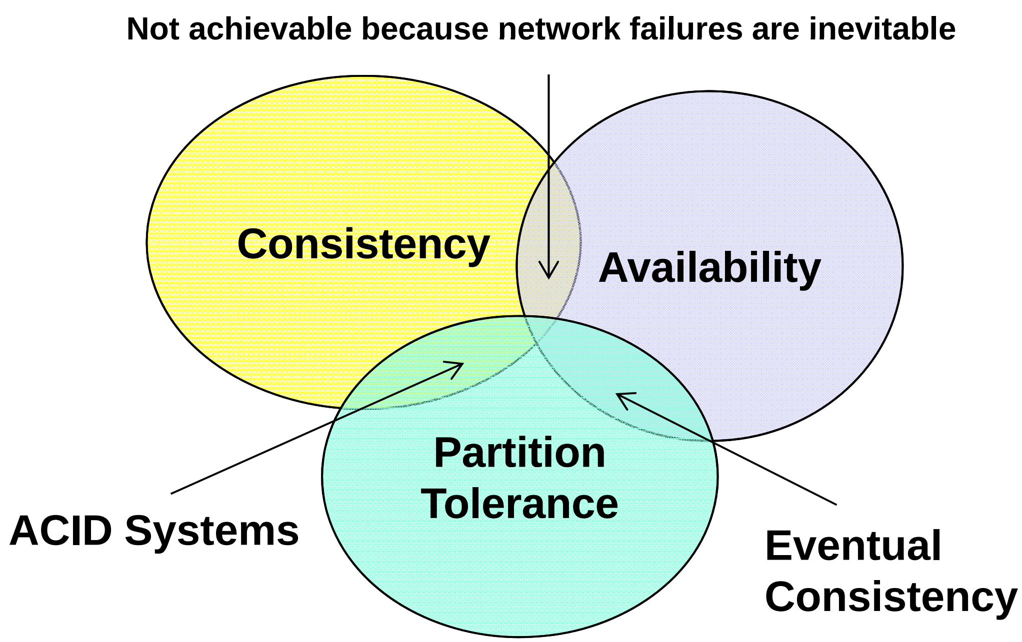
Replication

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

RELEASE

NoSQL DBMSs for Erlang

	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
Query Processing & Backend Storage	•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

RELEASE

Initial Evaluation Results

General Principles

+

Initial Evaluation

- Mnesia
- CouchDB
- Riak
- Cassendra

→

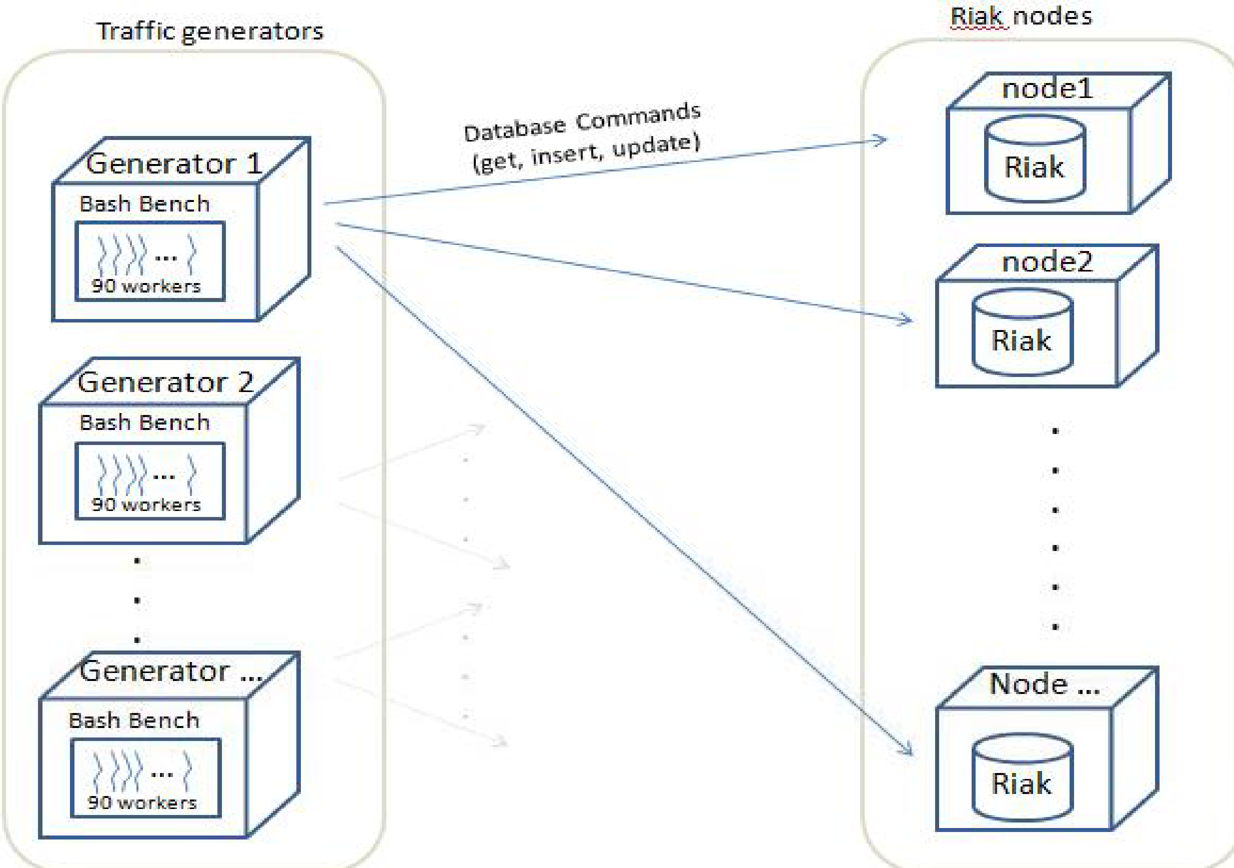
Scalable persistent storage for SD Erlang can be provided by Dynamo-like DBMSs, e.g. Riak,Cassandra

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*

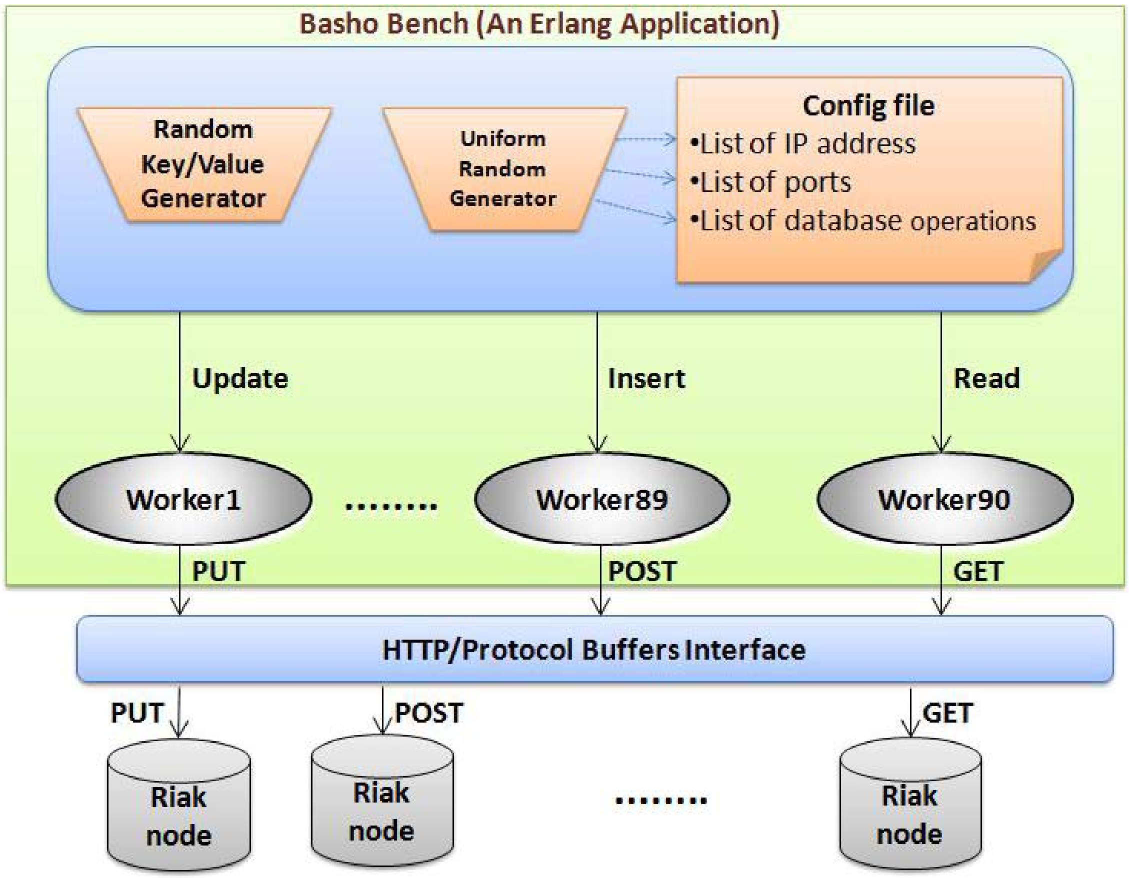
RELEASE

Node Organisation



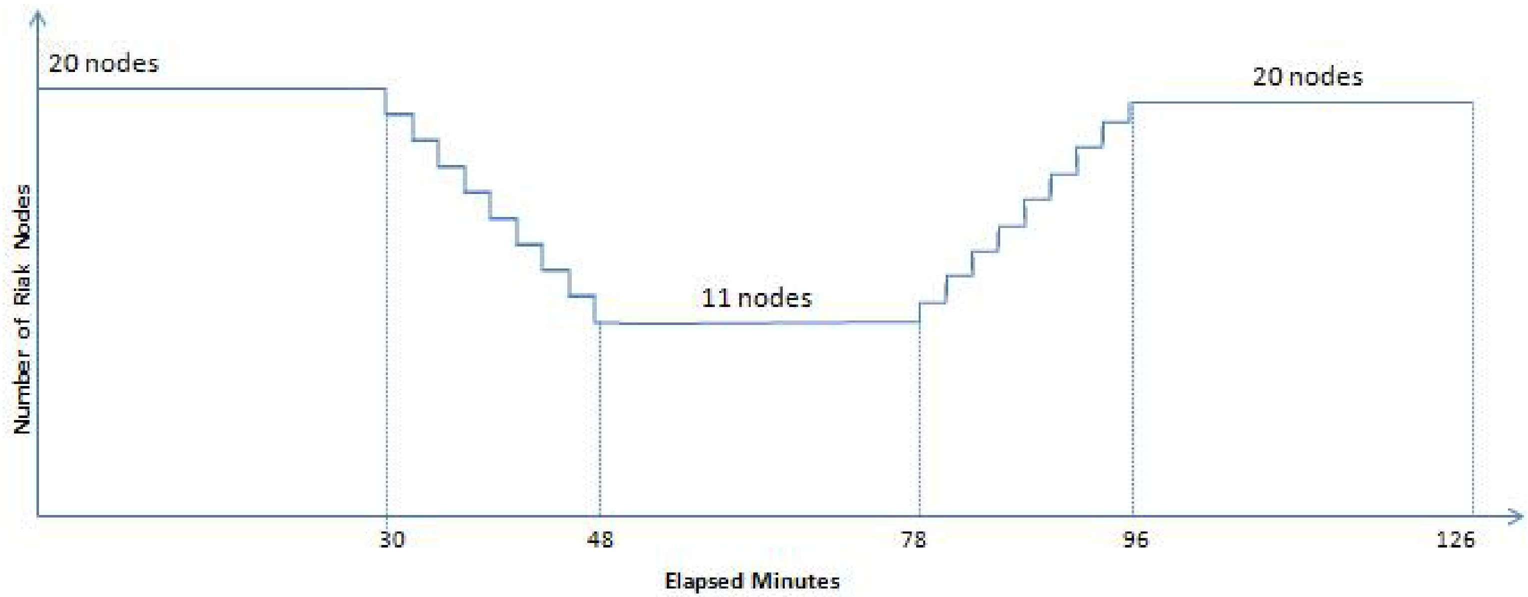
We use one traffic generator per 3 Riak nodes

Traffic Generator



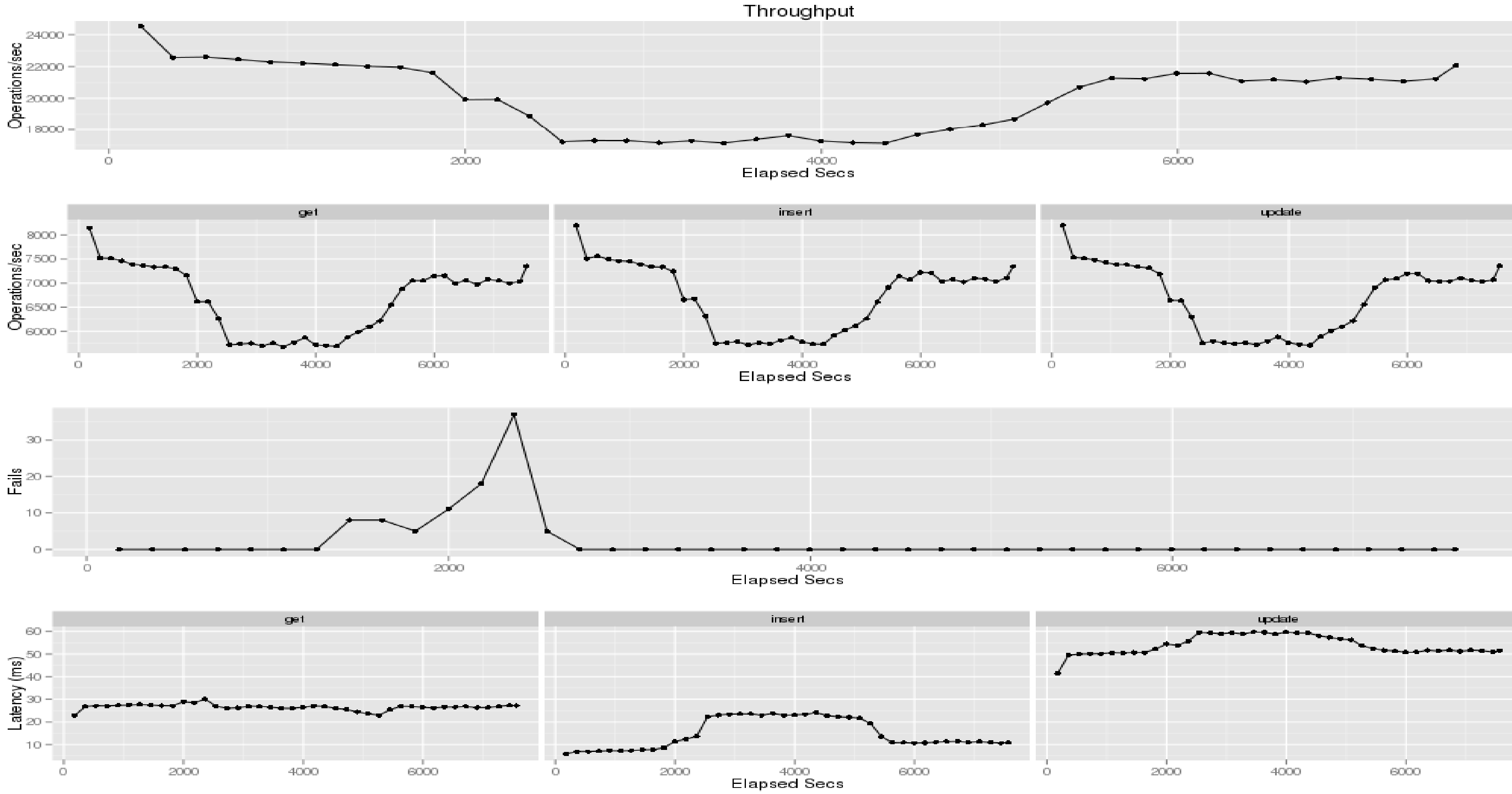
RELEASE

Riak Availability



Time-line shows Riak cluster losing nodes

Riak Availability



How Riak deals with failures

RELEASE

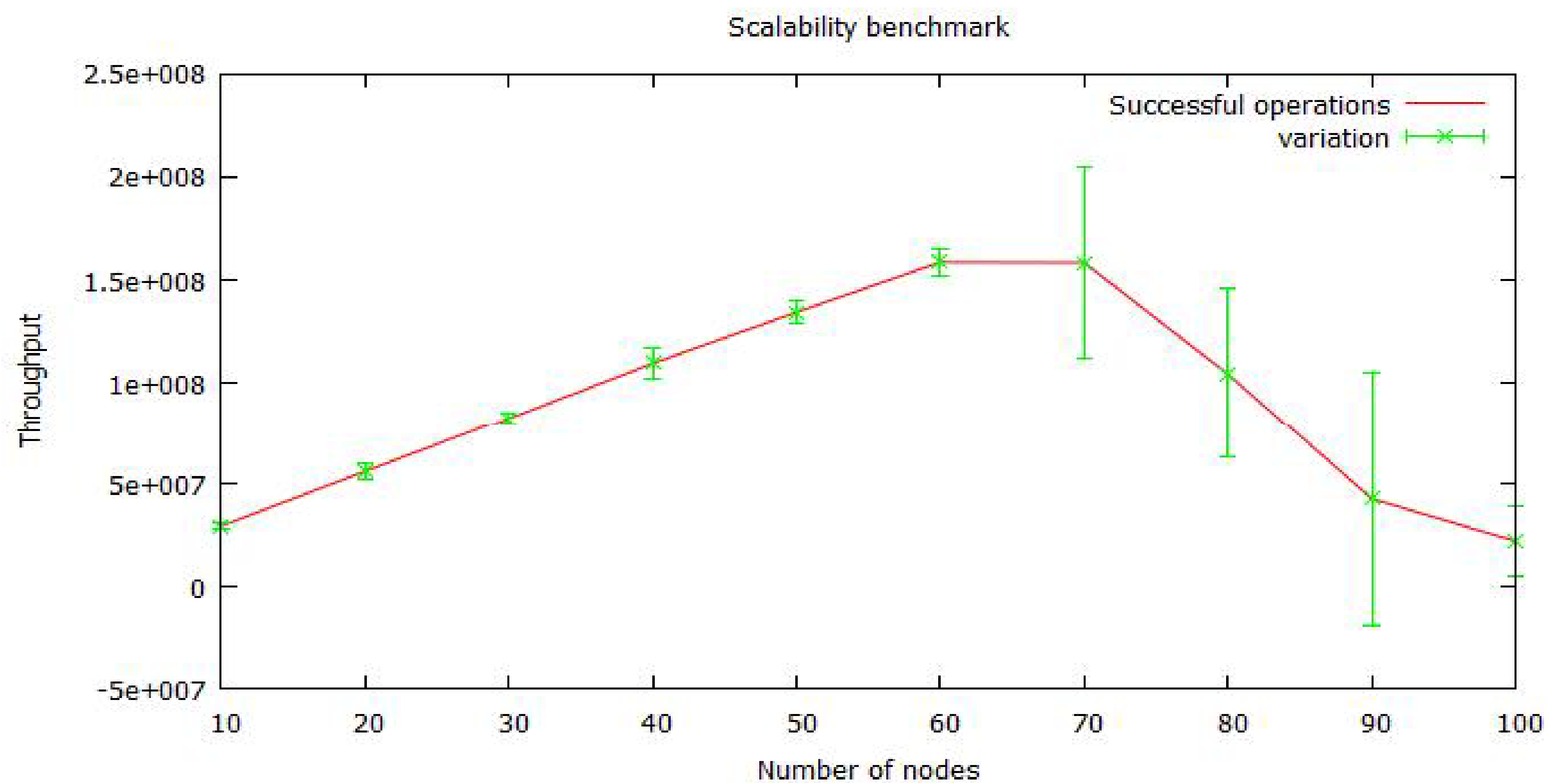
Observation

• 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**.

RELEASE

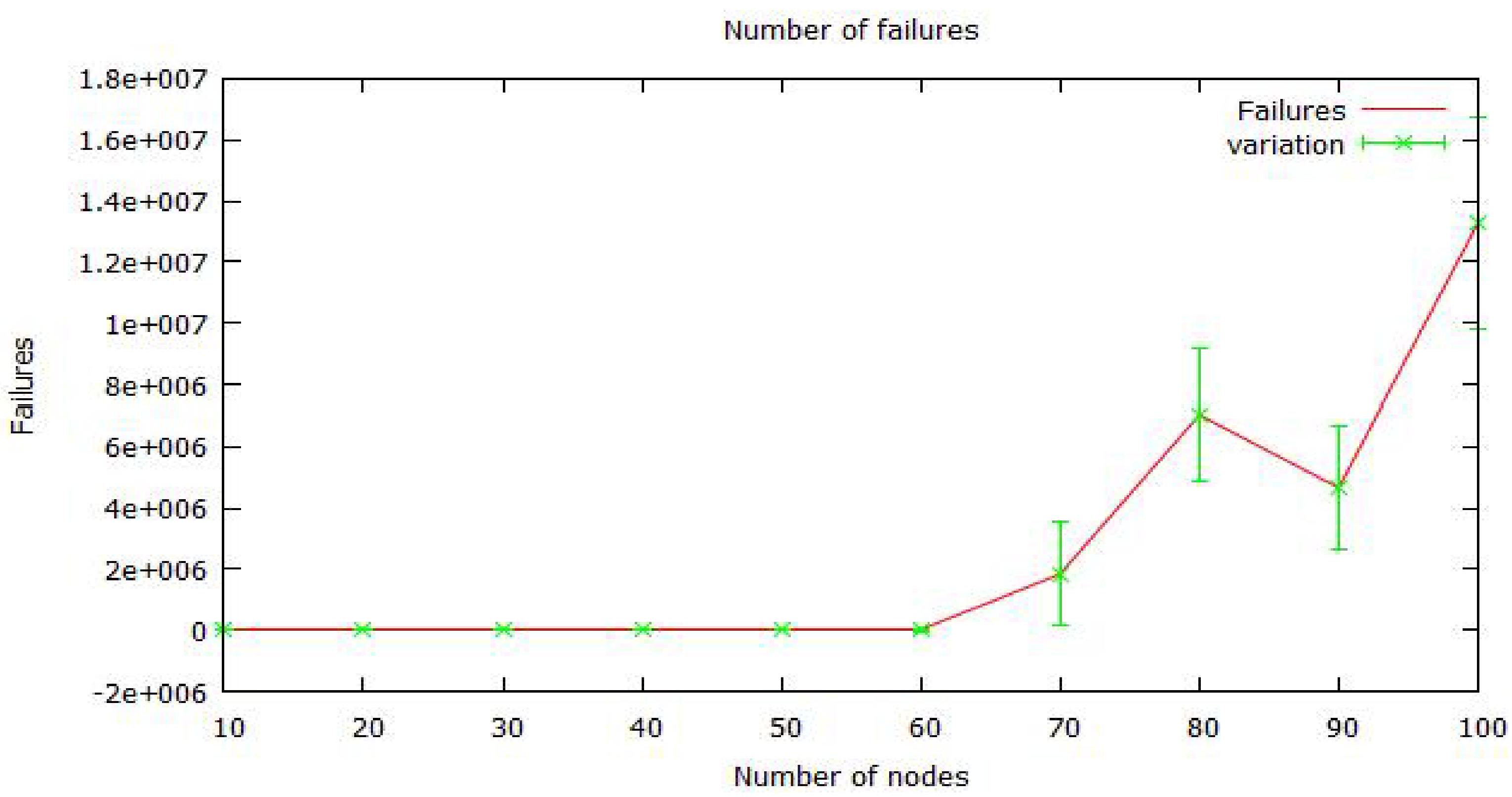
Riak Scalability



Benchmark on 348-node Kalkyl cluster at Uppsala University

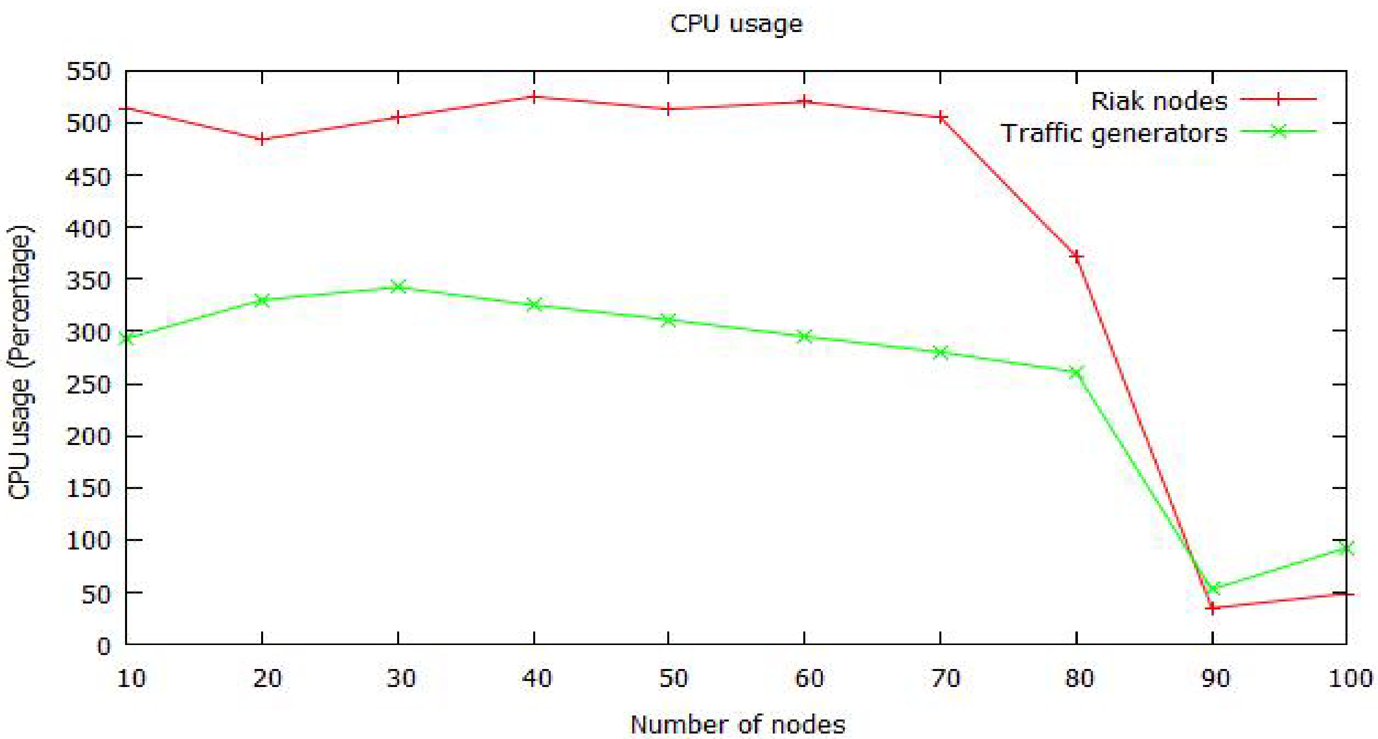
RELEASE

Failure



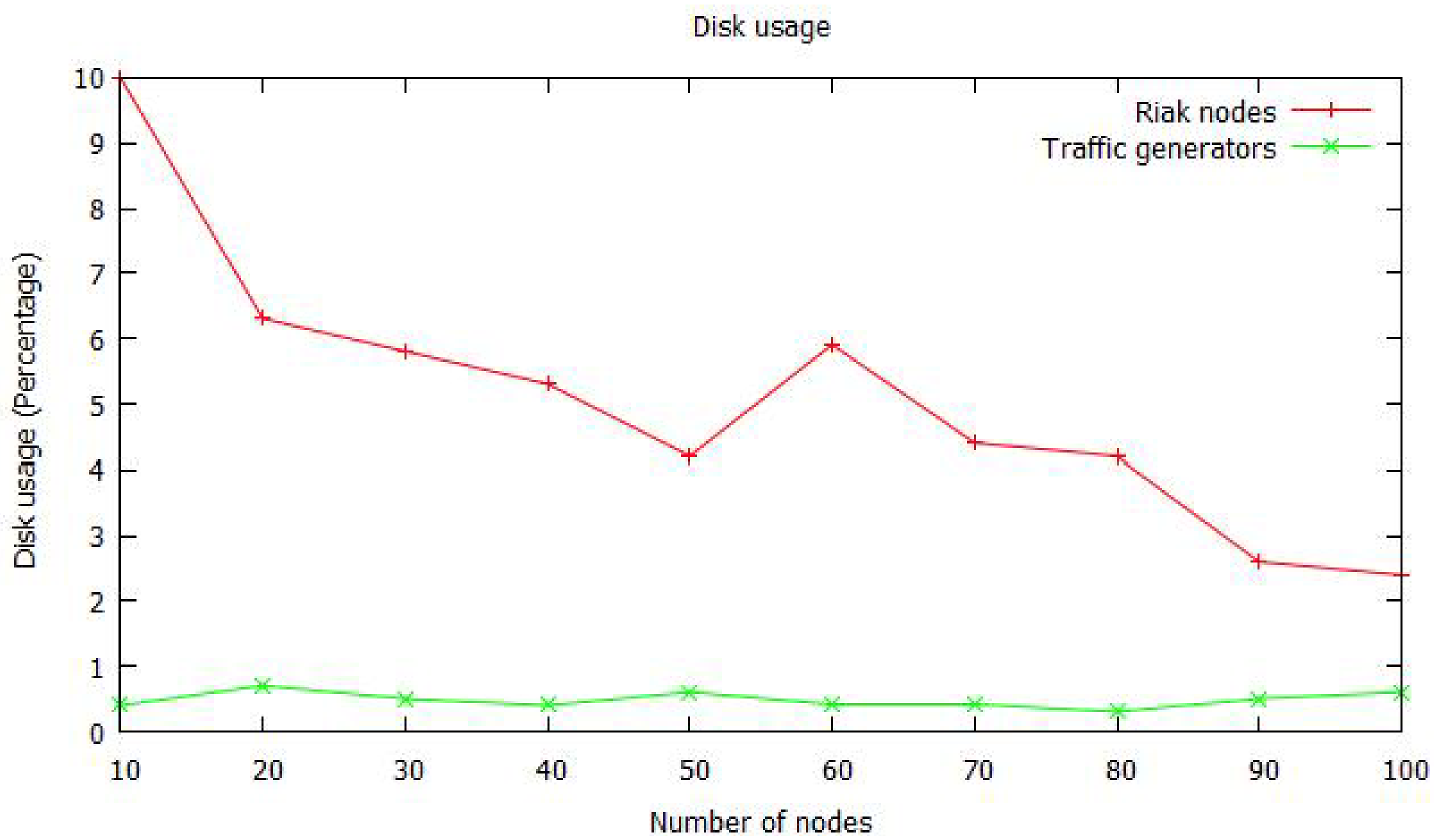
RELEASE

What is the Bottleneck?



CPU Usage

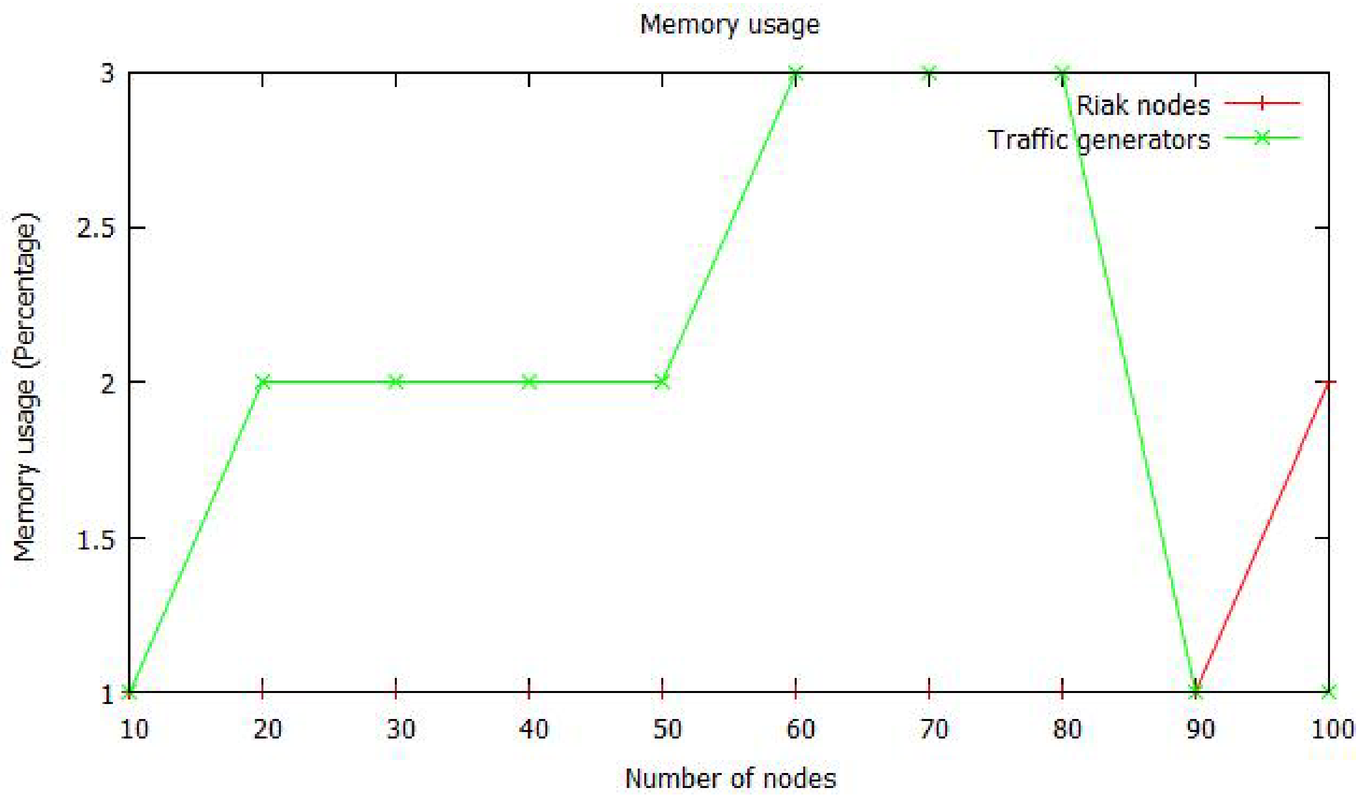
Profiling DISK



DISK Usage

RELEASE

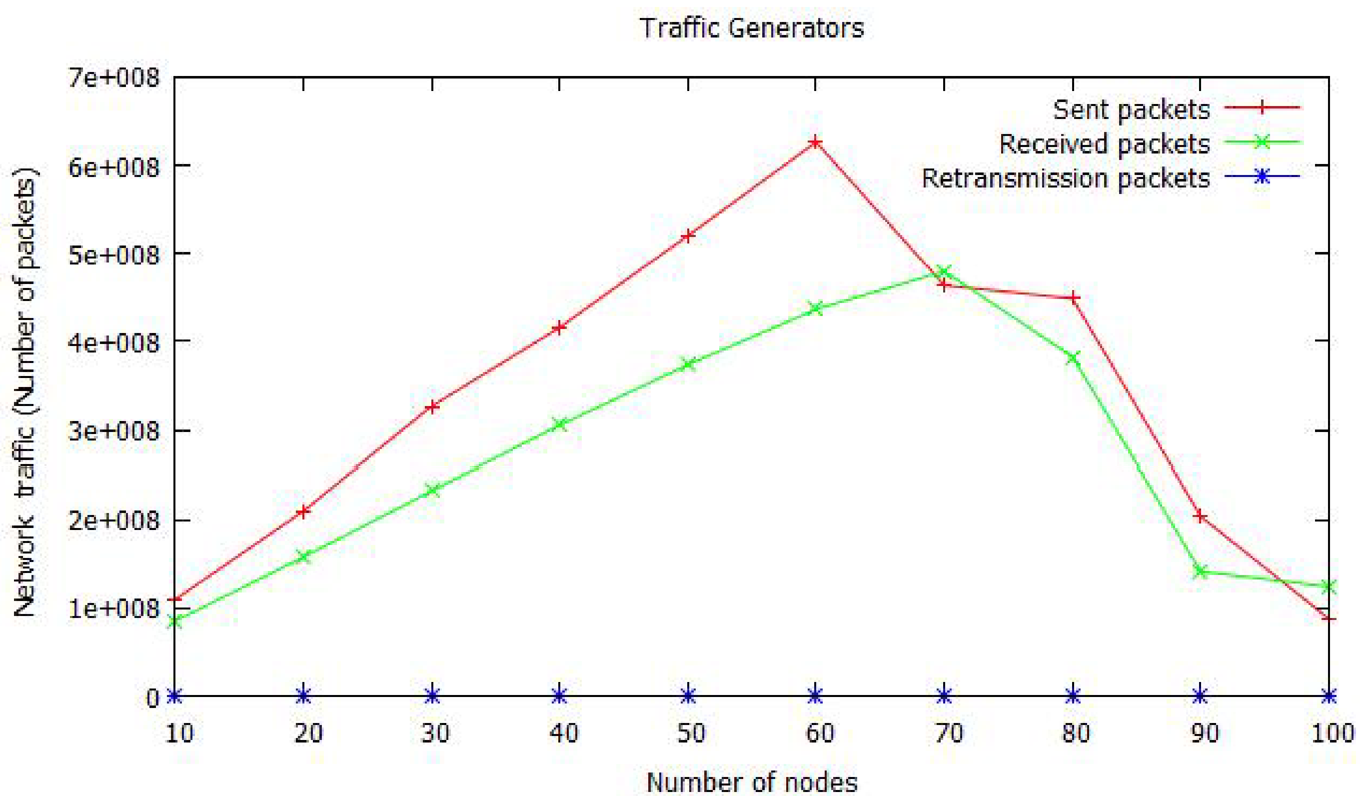
Profiling RAM



Memory Usage

RELEASE

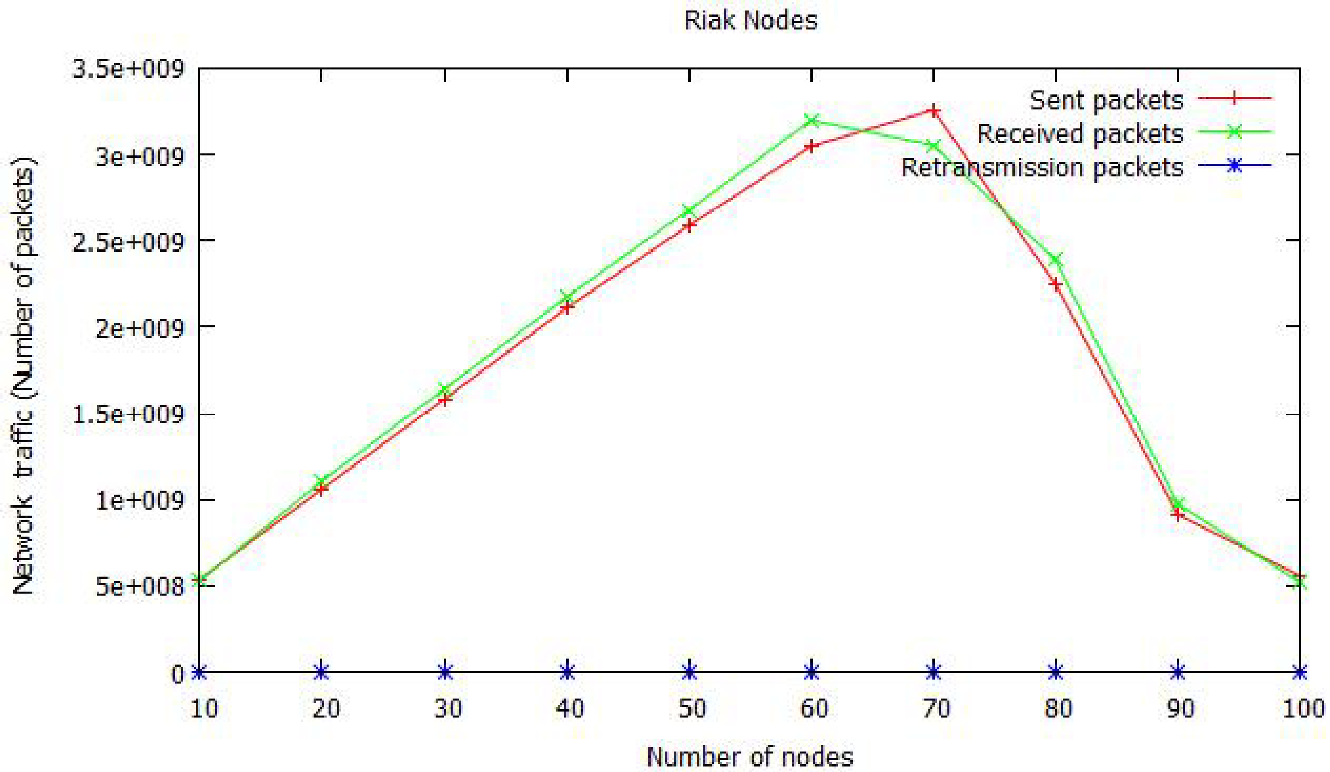
Profiling-Network (Generator)



Network Traffic of Generator Nodes

RELEASE

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.

RELEASE

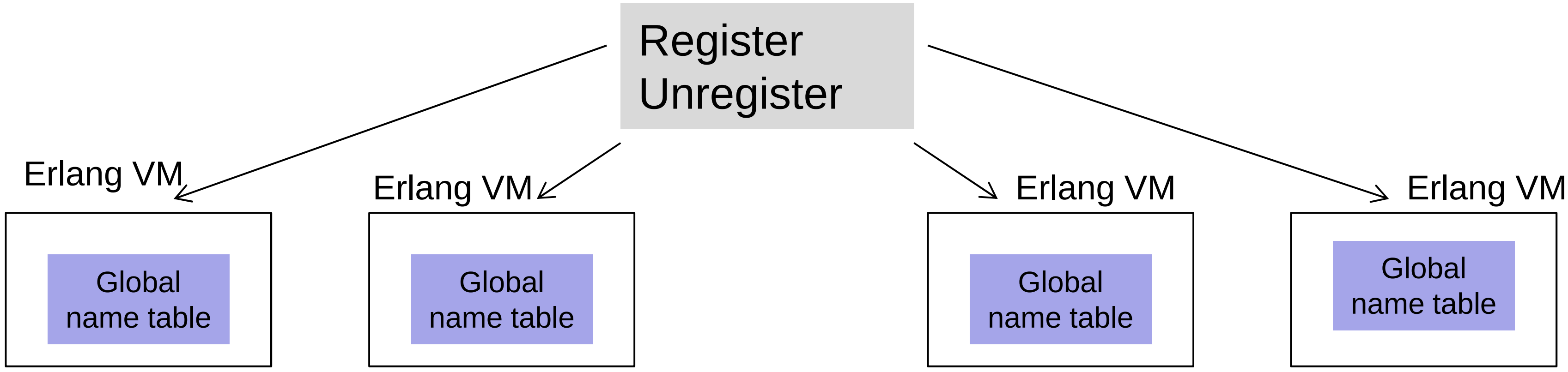
DEbench

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

RELEASE

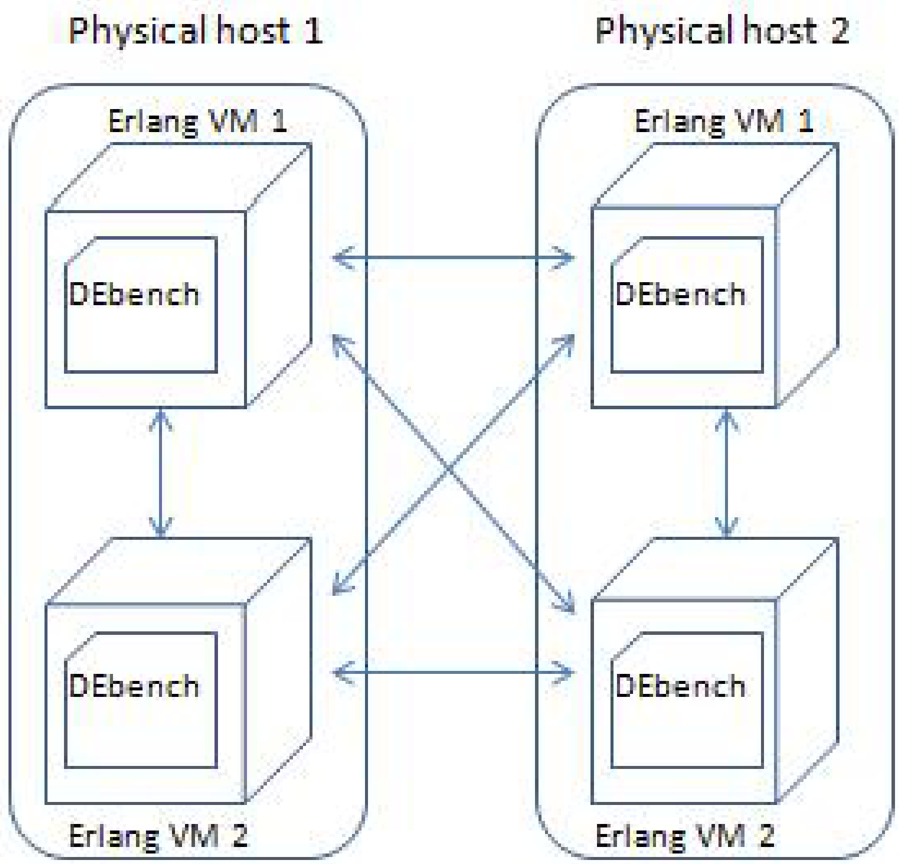
Distributed Erlang Commands

- *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



RELEASE

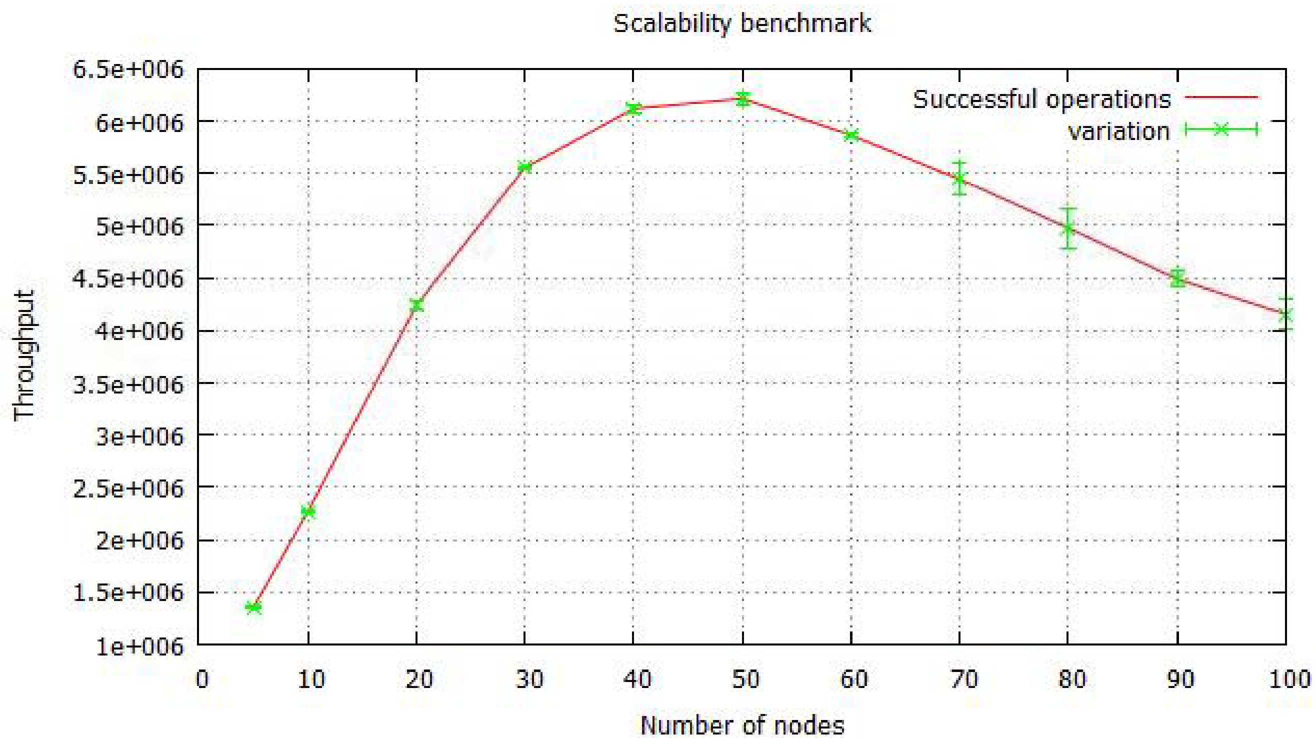
DEbench P2P Nodes



RELEASE

Scalability of Distributed Erlang

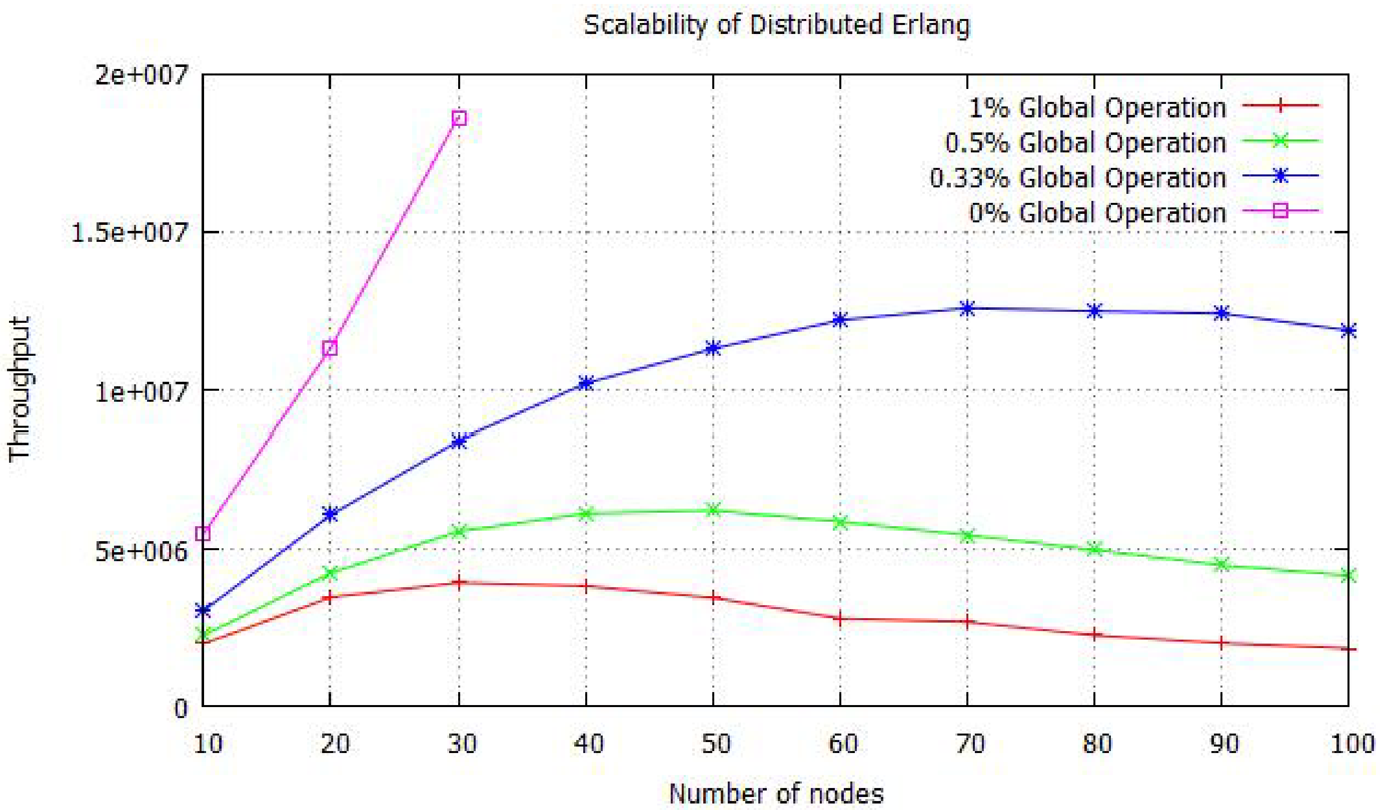
0.5% Global operation



- **Throughput peaks at 50 nodes**
- **Little improvement beyond 40 nodes**

RELEASE

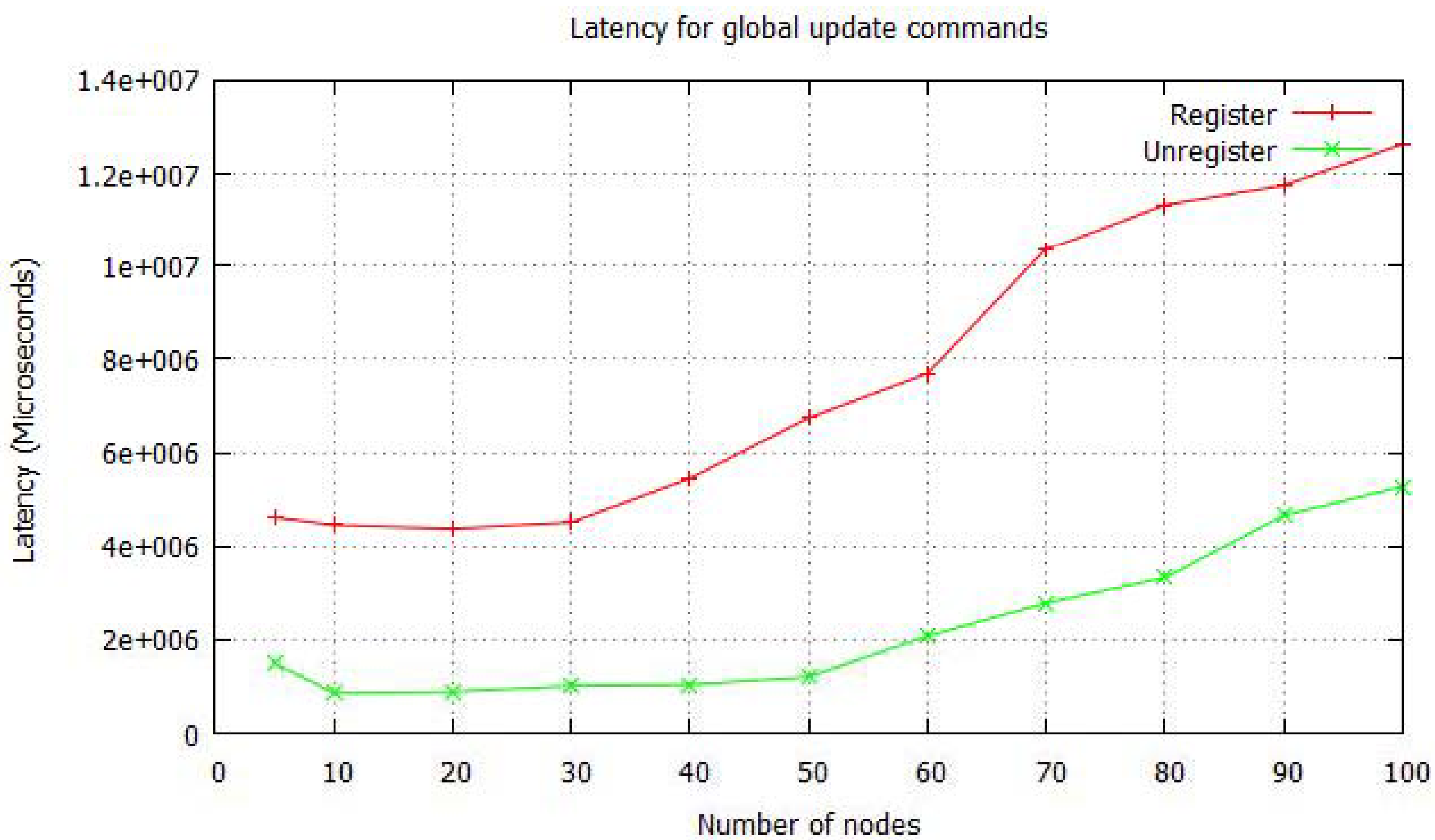
Frequency of Global Operation



Frequently	Max Throughput
1%	30 nodes
0.5%	50 nodes
0.33%	70 nodes
0%	1600 nodes

RELEASE

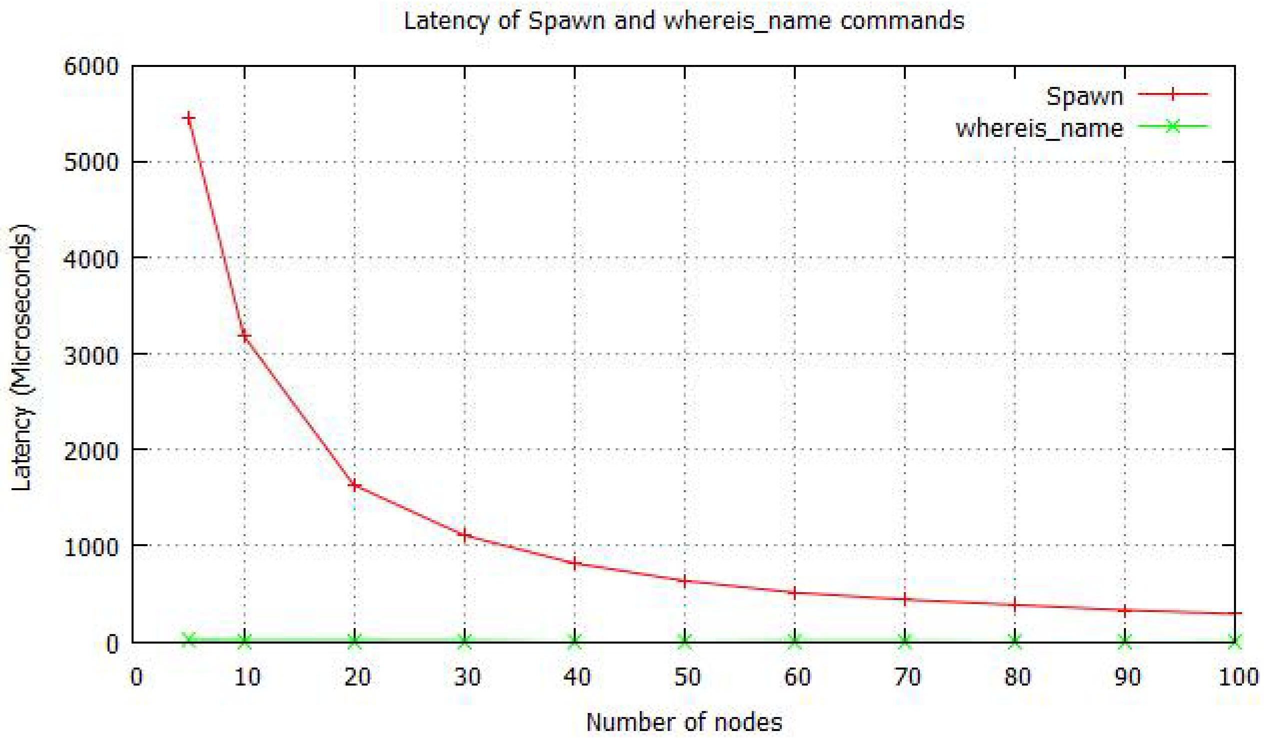
What is the Bottleneck?



Latency for *register* and *unregister for 2% global update*

RELEASE

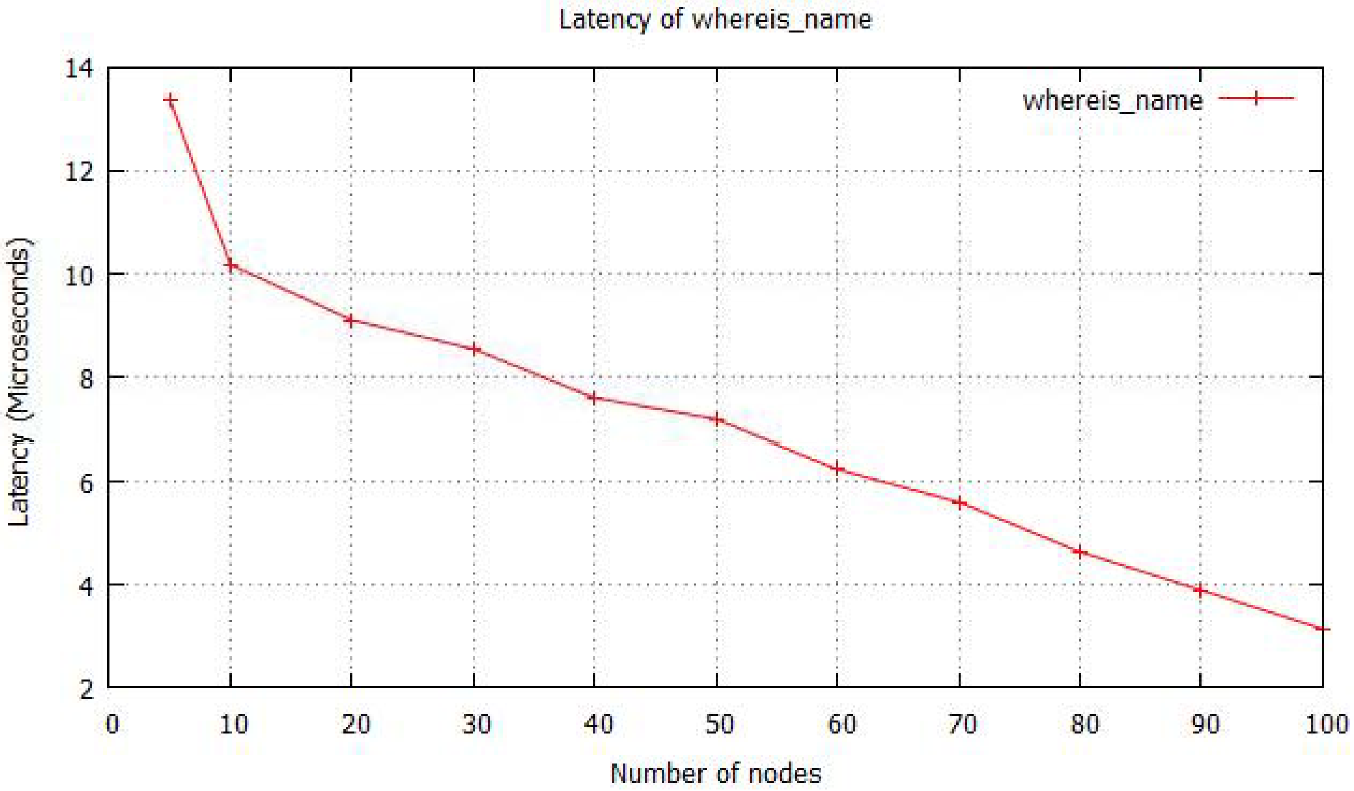
What is the Bottleneck?



Latency of *spawn*

RELEASE

What is the Bottleneck?



Latency of *whereis_name*

RELEASE

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!