

Integration of a key-value database in the parameter management system of the ALFA framework

Tom Van Steenkiste

Supervisor: Predrag Buncic

Detector parameters

- › geometry
- › position
- › calibration
- › ...



- › ROOT
- › ASCII
- › (SQL)
- › ?

Key-value database

› requirements

- 1kB - 50MB
- 200 files read - 60 files written
- 200MB read - 30MB written
- C/C++ interface

Key-value database

› requirements

- 1kB, 10kB, 100kB, 1MB, 10MB, and 50MB
- 3.33 reads per write
- random access pattern
- C/C++ interface

Key-value database

› candidates

- RAMCloud
- Riak
- ...

Case-study: RAMCloud

- › **low latency**
- › **scalability**
- › **C++**

- › **1MB limit**
 - chunking?

Case-study: RIAK

- › **high availability**
- › **scalability**
- › **fault tolerance**
- › **configurable**

Case-study: Riak

› C/C++ client

- very buggy
- last commit 1 year ago
- no future support

Case-study: Riak

› 3rd party C/C++ client

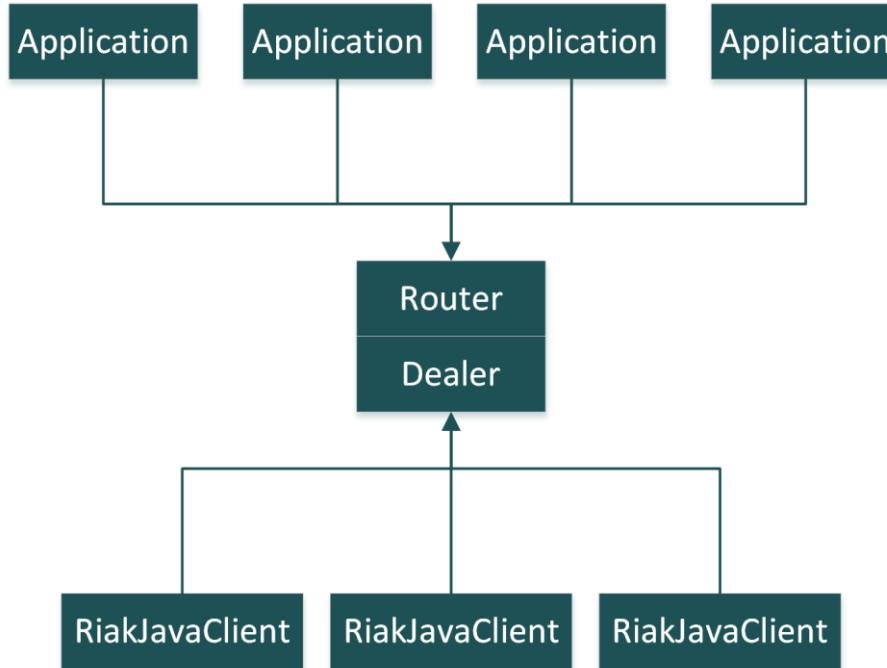
- very buggy
- last commit 1 year ago
- incomplete

Case-study: Riak

- › **Java client**
 - official library
 - future support
 - available now



Case-study: Riak



Case-study: Riak

› Java client

- easily configurable
- any protobuf language
- distributed

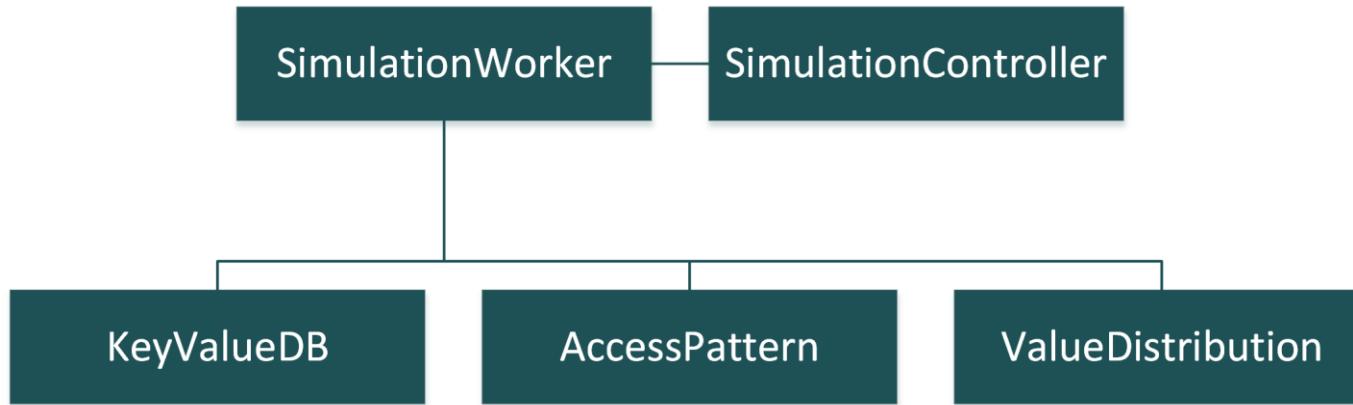


Performance framework

› existing benchmarks

- not full pipeline
- only certain use cases
- sometimes wrong

Performance framework



Performance framework

› **keyvalueclusterperf**

- full pipeline
- universal
- highly configurable
- easily extendable

Case-study: RAMCloud

› object write speed

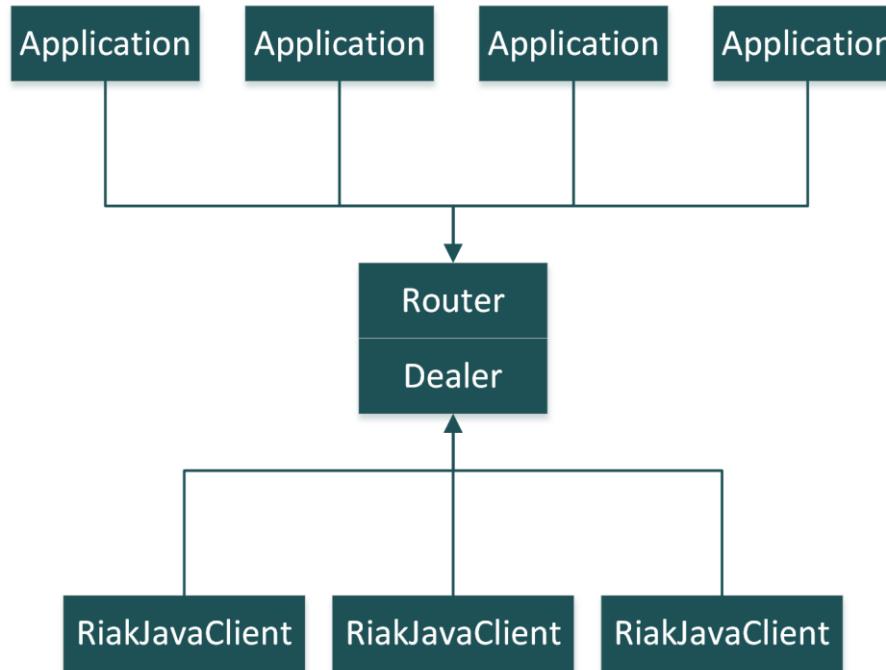
- 0B: 15 μ s
- 1MB: very slow
- +1MB: not possible

Case-study: Riak

› requirements

- 1kB, 10kB, 100kB, 1MB, 10MB, and 50MB
- 3.33 reads per write
- random access pattern
- C/C++ interface

Case-study: Riak



Case-study: Riak read vs write latency

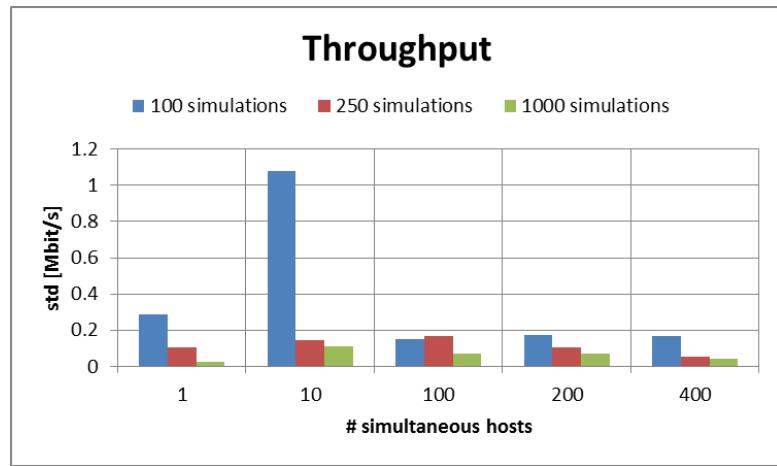
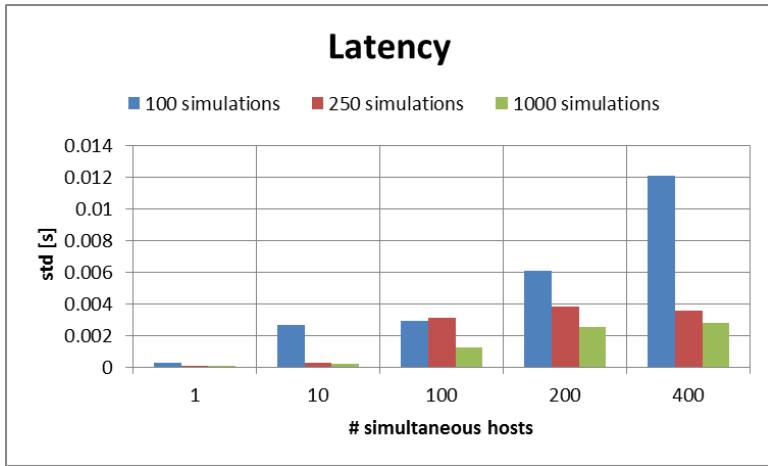
Size	Read duration[s]	Write duration[s]	% Difference
1kB	0.003456	0.003455	-0.033927
10kB	0.003673	0.003675	0.066285
100kB	0.008708	0.008708	0.005512
1MB	0.085394	0.085384	-0.010775
10MB	0.936064	0.935816	-0.026501

Case-study: Riak read vs write latency

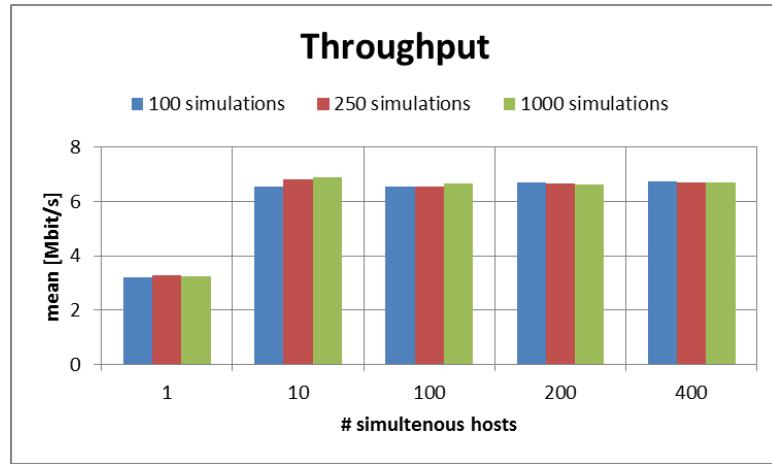
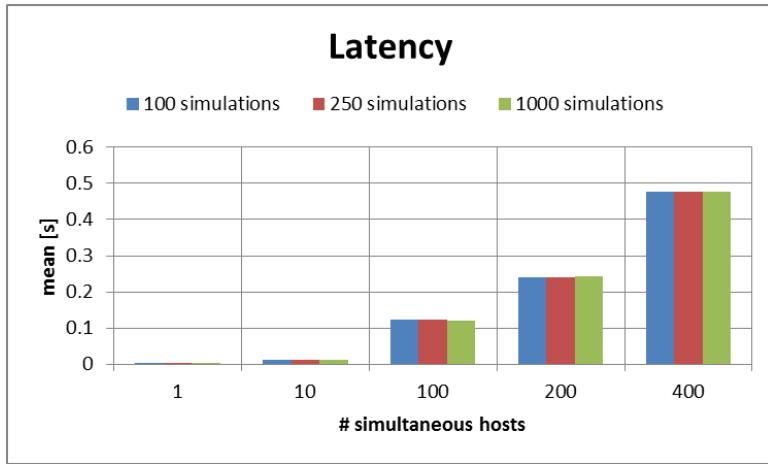
› small differences

- under load
- costs amortized
- no need for separate analysis

Case-study: Riak sample size



Case-study: Riak sample size



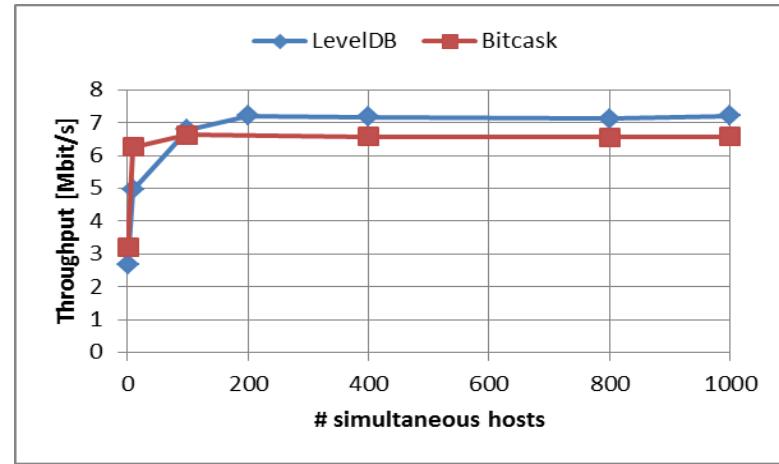
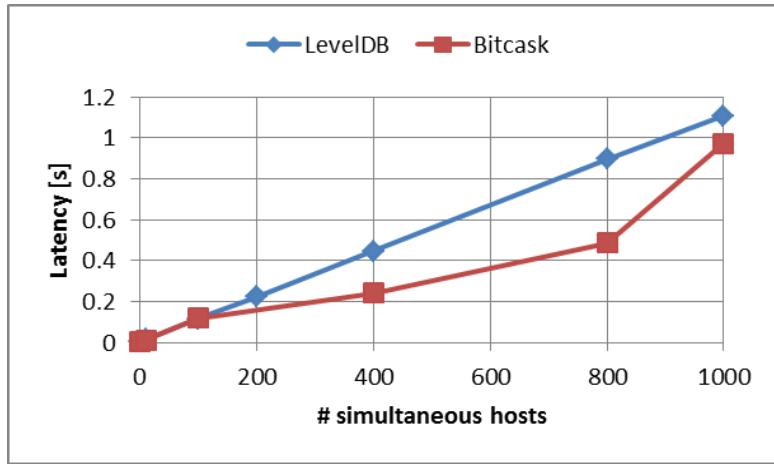
Case-study: Riak sample size

- › **amount of simulations**
 - 250 is sufficient
- › **convergence**
 - small amount of simulations
 - multiple iterations

Case-study: Riak storage backend

- › **Bitcask**
 - log-structured
- › **LevelDb**
 - compression

Case-study: Riak storage backend



object size: 1kB

Case-study: Riak storage backend

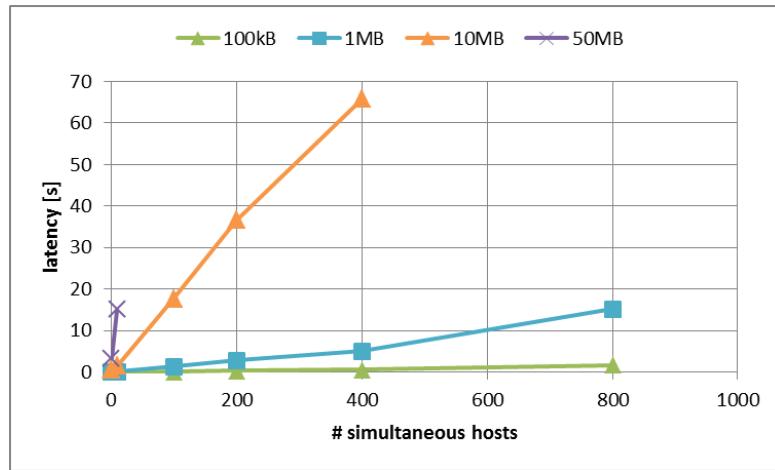
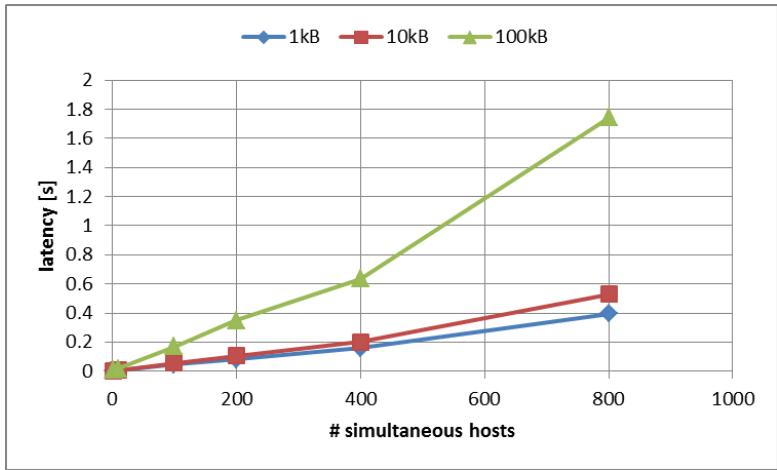
› Bitcask

- best latency
- nodes out-of-memory

› LevelDb

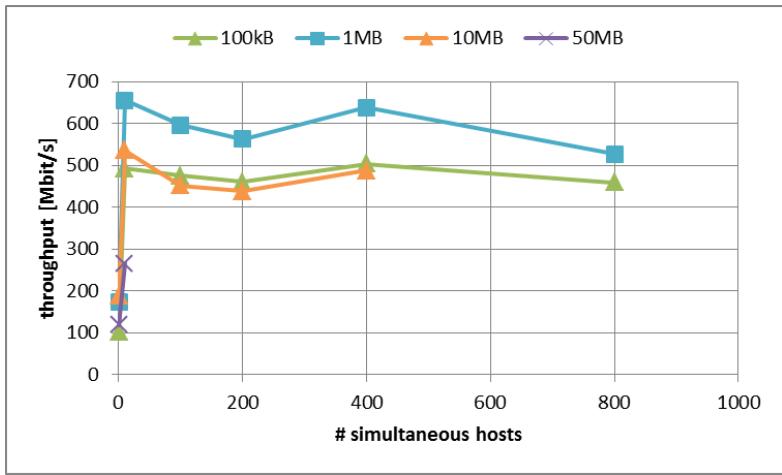
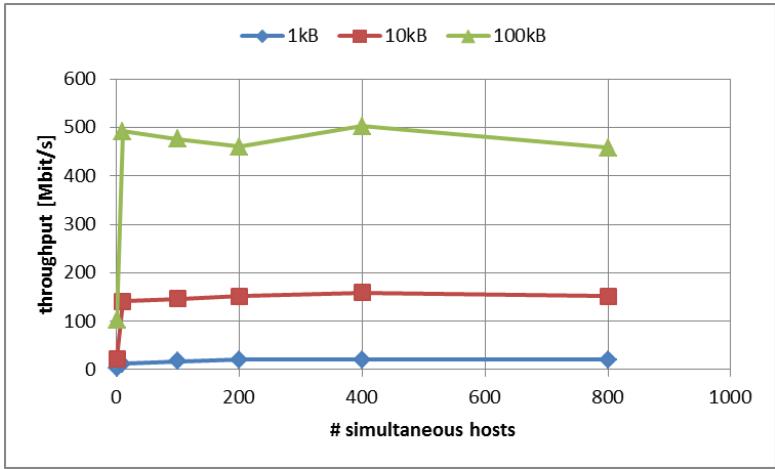
- similar performance
- compressed storage

Case-study: Riak object sizes: latency



Riak nodes: 4

Case-study: Riak object sizes: throughput

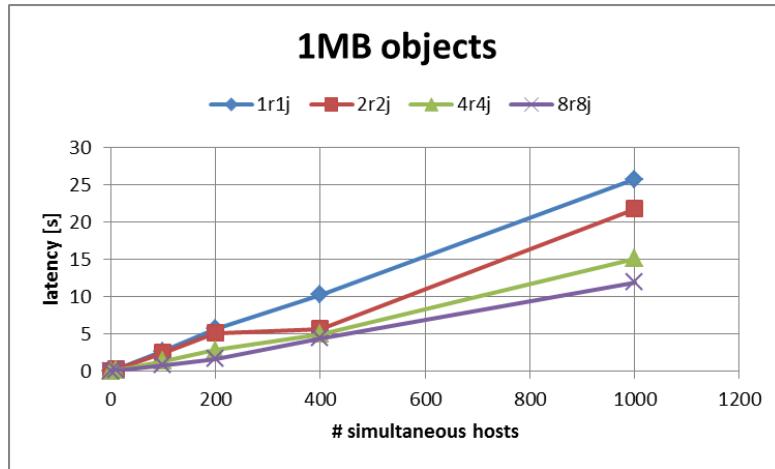
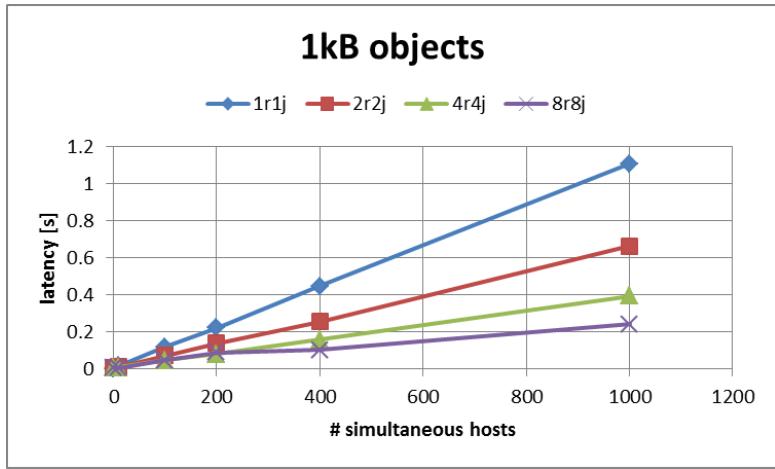


Riak nodes: 4

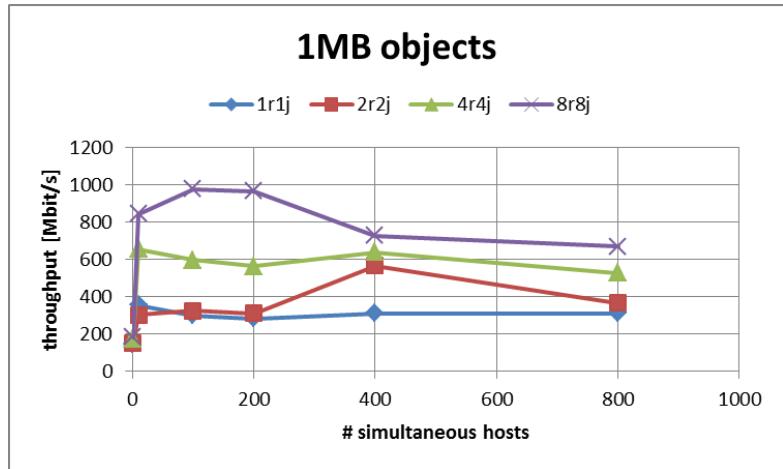
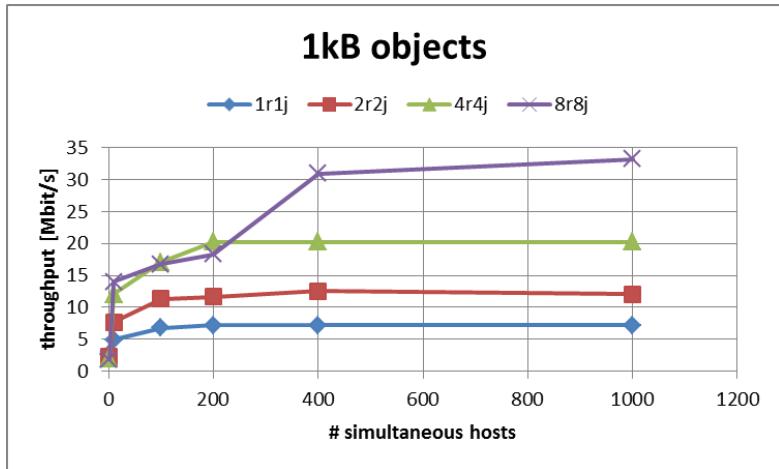
Case-study: Riak object sizes

- › **always same trend**
 - strong increase in latency
 - constant throughput
- › **large objects**
 - high latency
 - but only large objects (not realistic)

Case-study: Riak cluster size: latency



Case-study: Riak cluster size: throughput

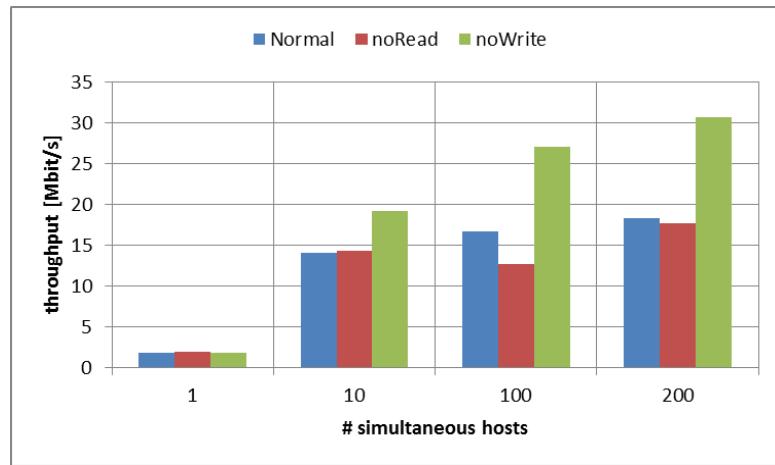
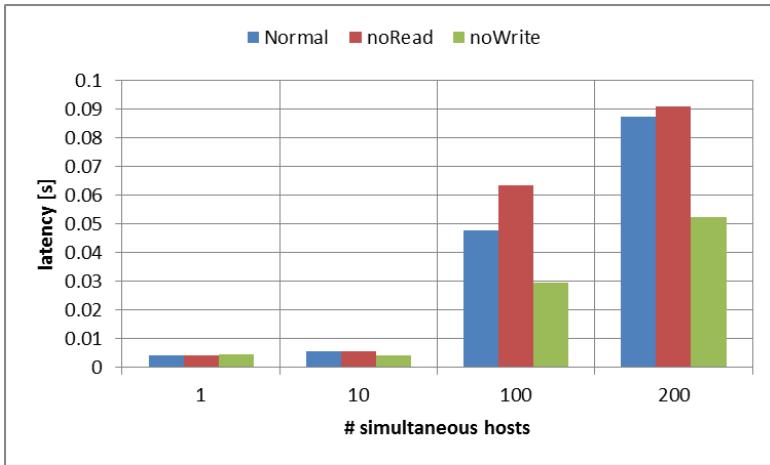


Case-study: Riak cluster size

› larger cluster

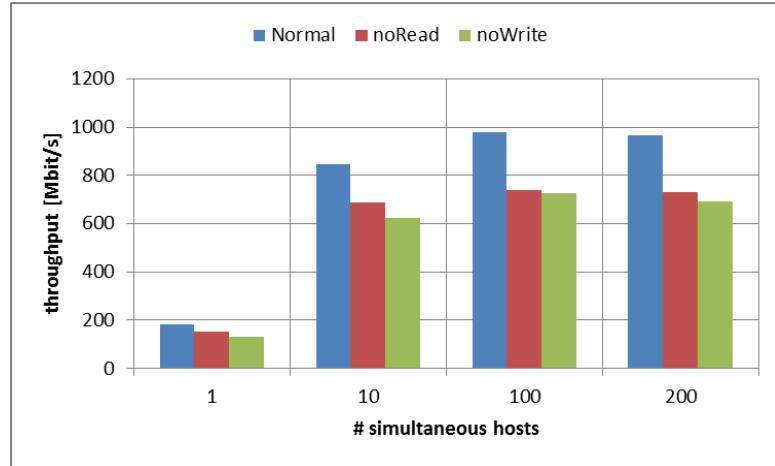
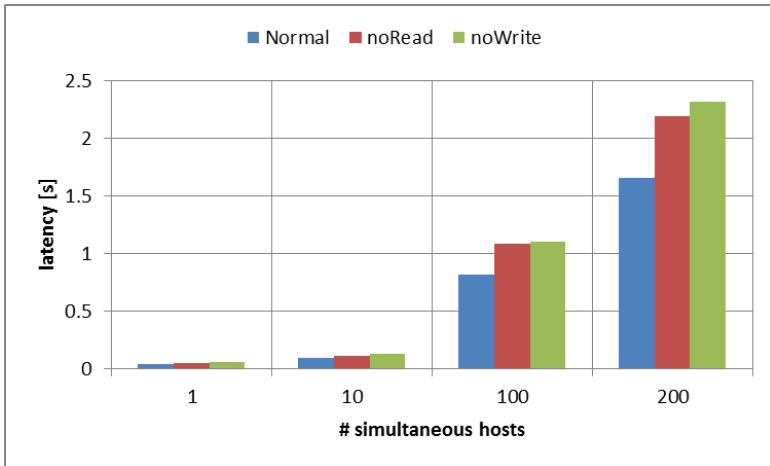
- better latency
- better throughput
- (better availability)

Case-study: Riak configuration



object size: 1kB

Case-study: Riak configuration



object size: 1MB

Case-study: Riak configuration

› small objects

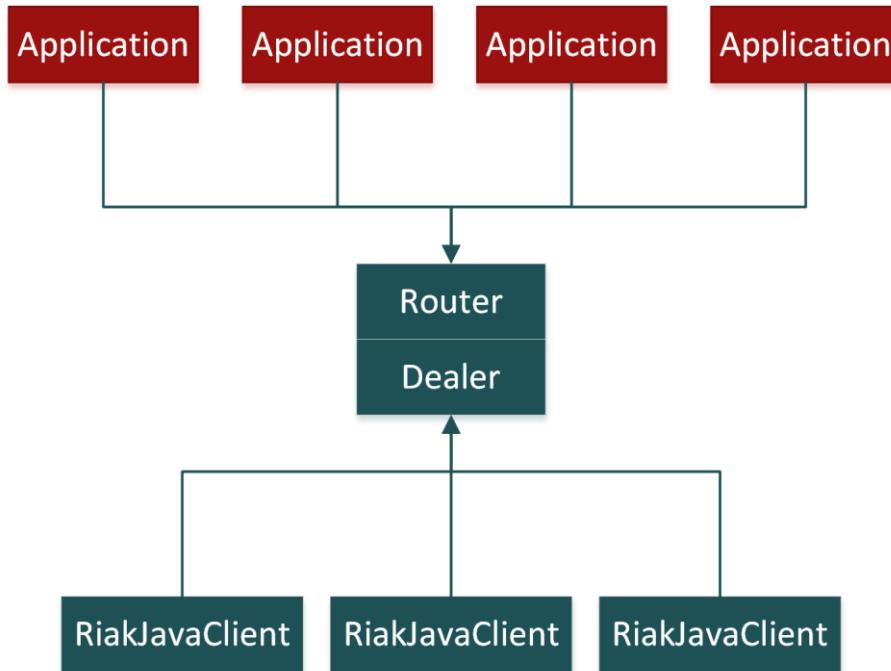
- noWrite: improved performance
- noRead: degraded performance

› large objects

- noWrite: degraded performance
- noRead: degraded performance

see further...
(difficulties
in simulation)

Case-study: Riak security



Case-study: Riak security

- › authentication
 - weak collision resistance (pre-image)
- › encryption
 - not required
- › 1 entry point
 - load balancer
 - firewall

Case-study: Riak security

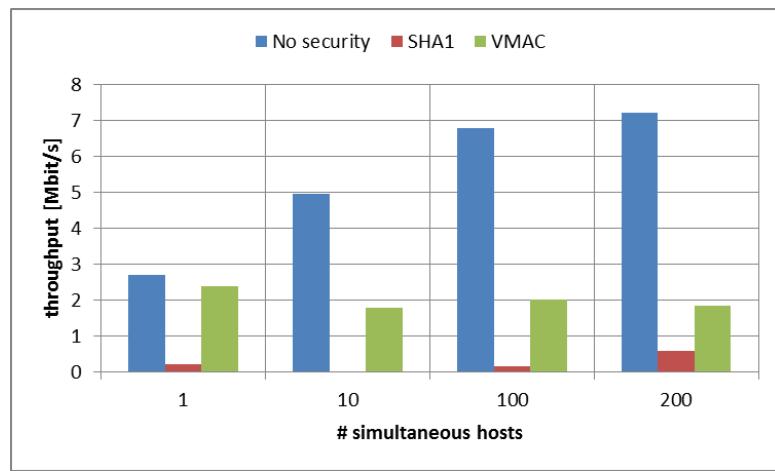
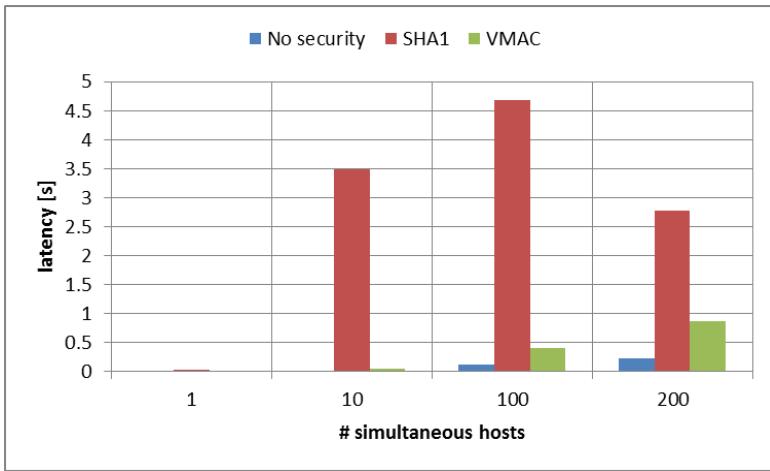
› SHA1

- sufficient for weak collision resistance
- a lot of implementations

› VMAC

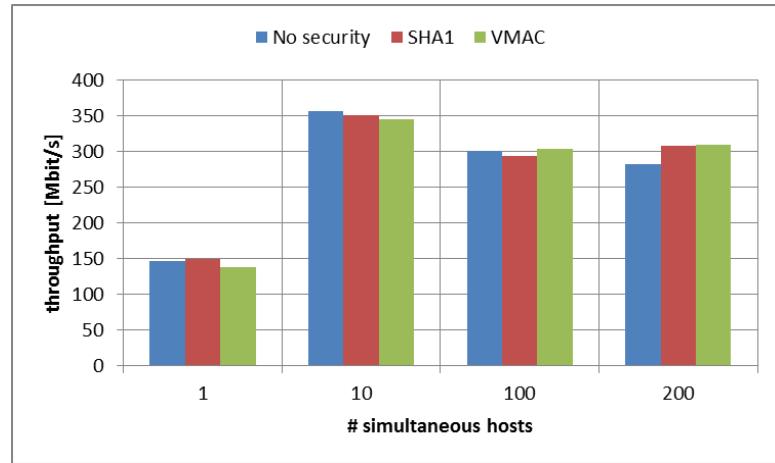
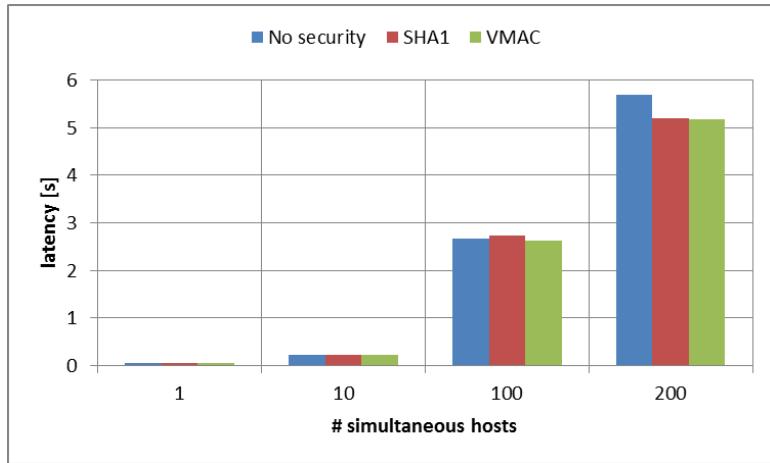
- build for speed
- only C++, no Java

Case-study: Riak security



object size: 1kB

Case-study: Riak security



object size: 1MB

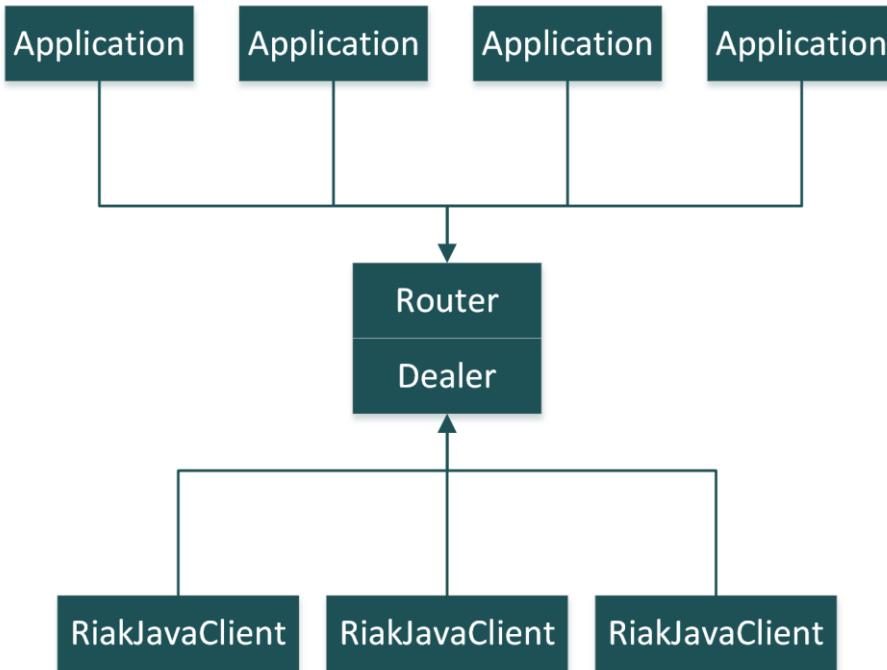
Case-study: Riak security

- › **small objects**
 - VMAC very efficient
- › **large objects**
 - strange results
 - IO bound operation?
 - incorrect results?

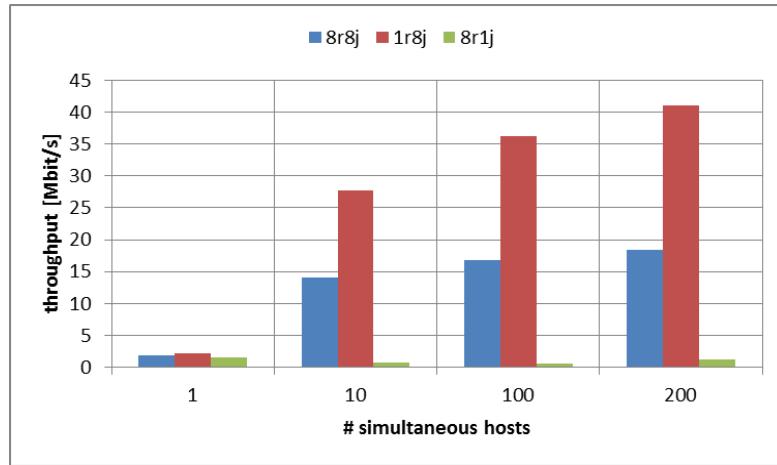
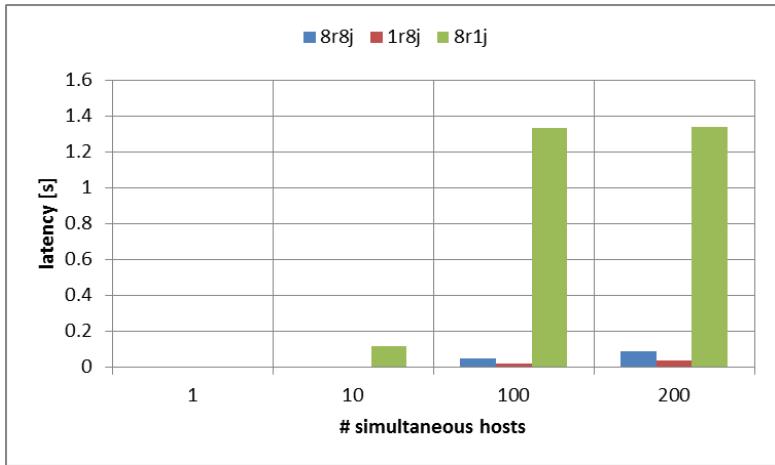


see further...
(difficulties
in simulation)

Case-study: Riak amount of Java clients

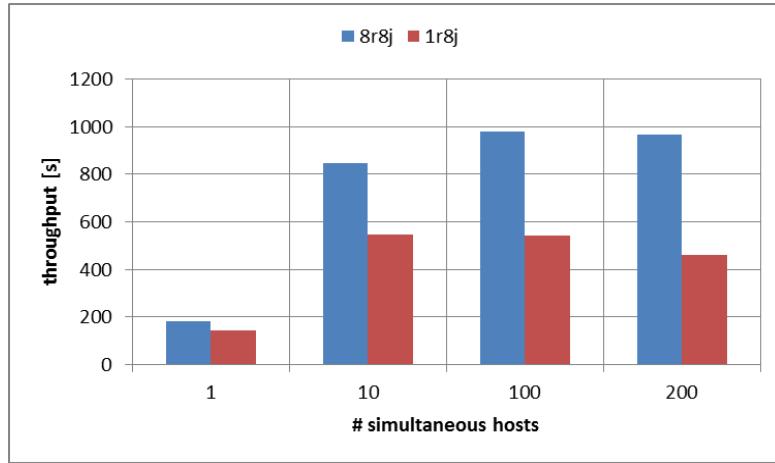
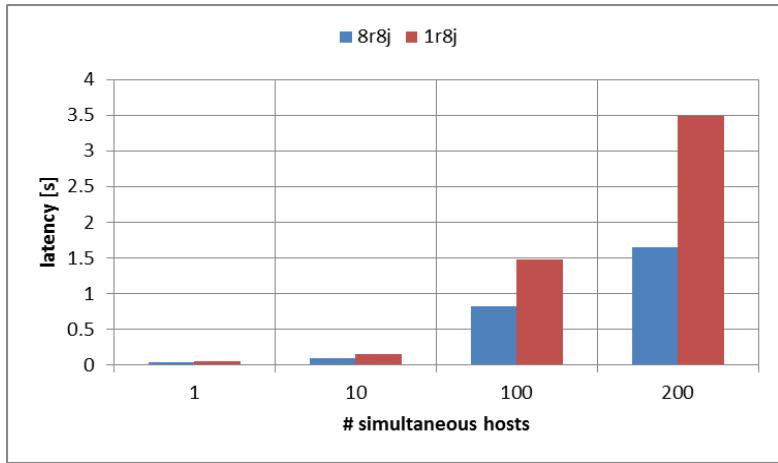


Case-study: Riak amount of Java clients



object size: 1kB

Case-study: Riak amount of Java clients



object size: 1MB

Case-study: Riak amount of Java clients

- › **small objects**
 - more Java clients better
 - less Riak nodes better
- › **large objects**
 - more Riak nodes needed

Case-study: Riak availability and fault tolerance

- › **replication policy**
 - configurable
 - default: 3 replicas
 - multi-node
 - multi-datacentre
- › **tuneable read/write consistency**

Case-study: Riak simulation difficulties

- › **clean database**
- › **background processes**
 - e.g. Bitcask merging
- › **other processes**
 - same nodes
 - same network



Be careful
when interpreting
results!

Case-study: Riak future work

- › other databases
- › C/C++ client for Riak
- › further investigation in security
- › different load balancer
- › exact value distribution and access pattern

Case-study: Riak more information

- › **all code on GitHub**
 - Doxygen
 - JavaDoc
- › **discussion results in report**
 - including software
 - user manual in appendix

Thank you!