Paper-A Scalable File Based Data Store For Forensic Analysis

By

Flavio Cruz, Andreas Moser and Michael Cohen

Presented At

The Digital Forensic Research Conference

DFRWS 2015 EU Dublin, Ireland (Mar 23rd- 26th)

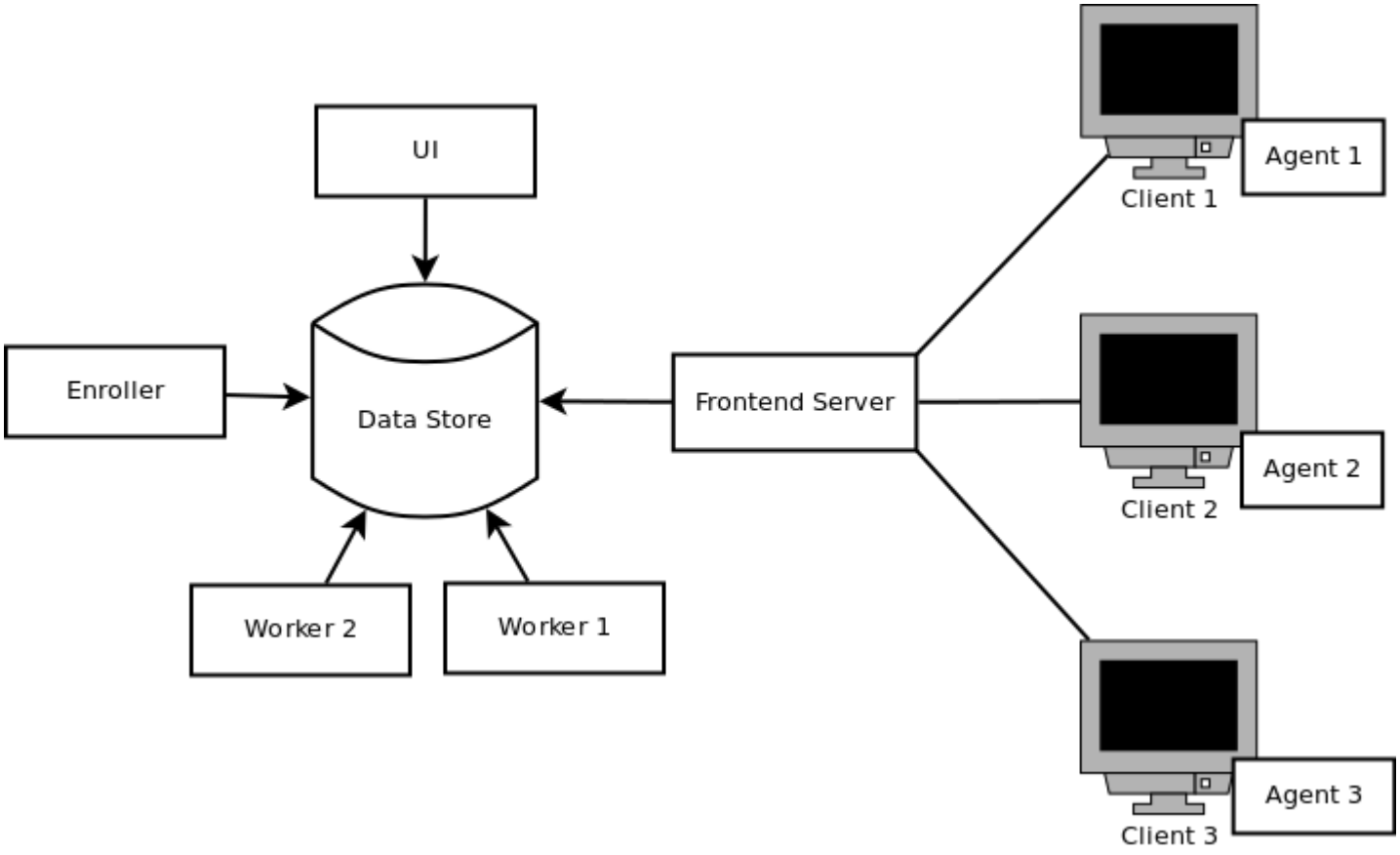
A scalable file based data store for forensic analysis

Flavio Cruz, Andreas Moser, Michael Cohen

GRR

- Distributed Incident Response Framework
  - Agent running on client machines
  - Server collects information from agents running remotely and stores it in the data store
- A large number of machines implies huge amounts of forensic data

GRR Overview



AFF4 Object Space

- AFF4 is an object store abstraction
- Made of many objects:
  - Object: aff4:/C.34a62f06/fs/os/boot.ini
  - An object has a set of attributes
  - Attribute: set of pairs (value, timestamp)
- AFF4 namespace universe is assumed to be incomplete
  - AFF4 objects cannot be enumerated
  - We must store references to children of objects as an attribute of the object

Data Store Requirements

- Single object access
- Support for both synchronous and asynchronous operations
- Object locking
- Concurrent: multi-process and multi-threaded
- Time-stamped attributes

Data Store

- Due to the characteristics of AFF4, most key-value “NoSQL” tend to be a good fit
- Available options in GRR:
  - Mongo (objects as documents)
  - MySQL (single table where each row represents a timestamped value)
- Those data stores have shown some performance problems

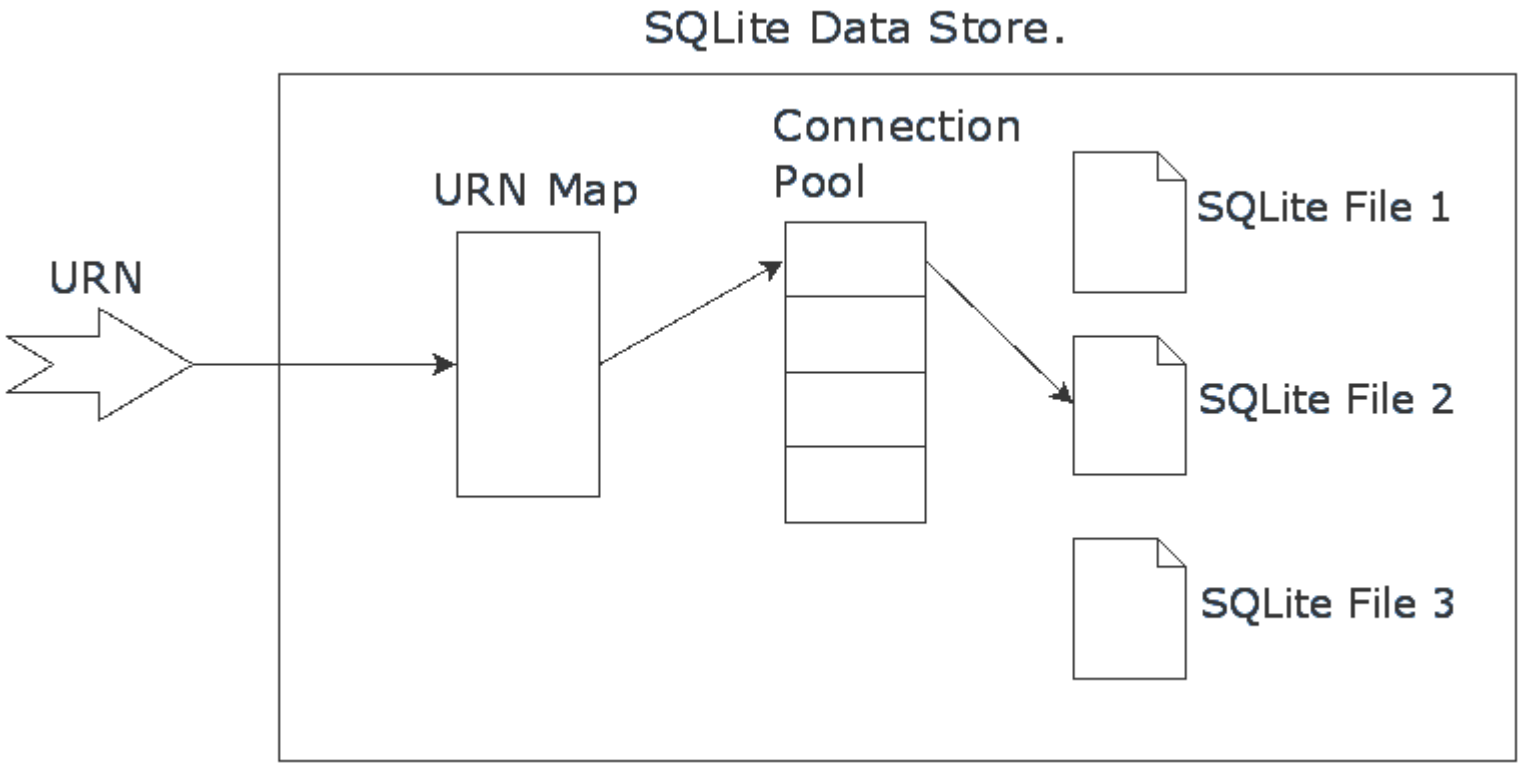
SQLite Data Store

- Partition the AFF4 object space into several files
  - Each file may have multiple objects
  - Attributes of a single object are stored in a single file
- Files are independent databases since each operation only works with an object
- Granularity can be customized

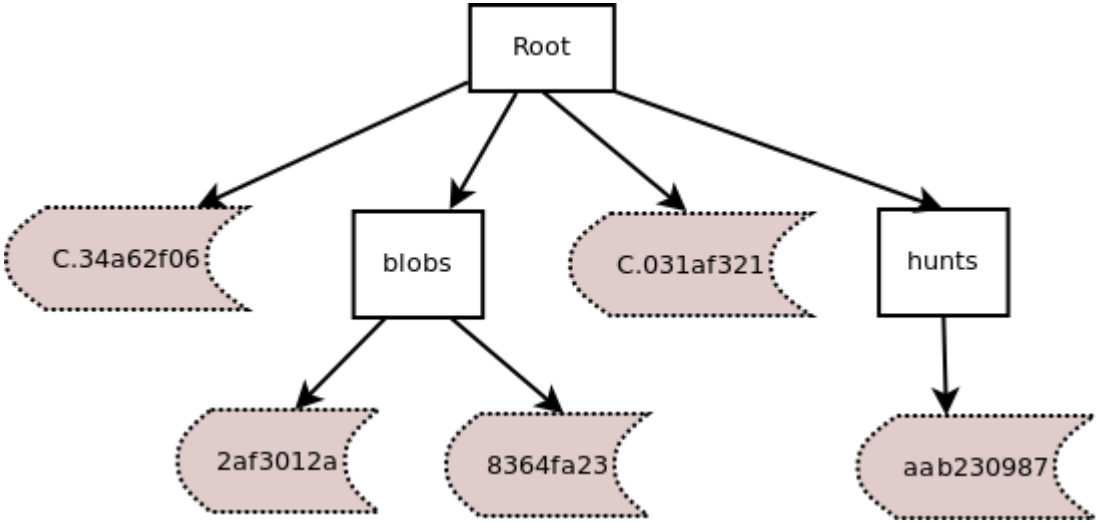
Granularity: Path Configuration

- aff4:/**C.34a62f06**/fs/os/boot.ini
  - C.34a62f06.sqlite
- aff4:/**C.34a62f06**/fs/os/cmd.exe
  - C.34a62f06.sqlite
- aff4:/**blobs/2af3012a**
  - blobs/2af3012a.sqlite
- Pathing configuration:
  - (?P<path>C\.\{1,16\}?)($|/.*) - One file per client
  - (?P<path>blobs/[^/]+).* - One file per blob
  - (?P<path>hunts/[^/]+).* - One file per hunt
  - (?P<path>[^/]+).* - One file per top-level directory

SQLite Data Store



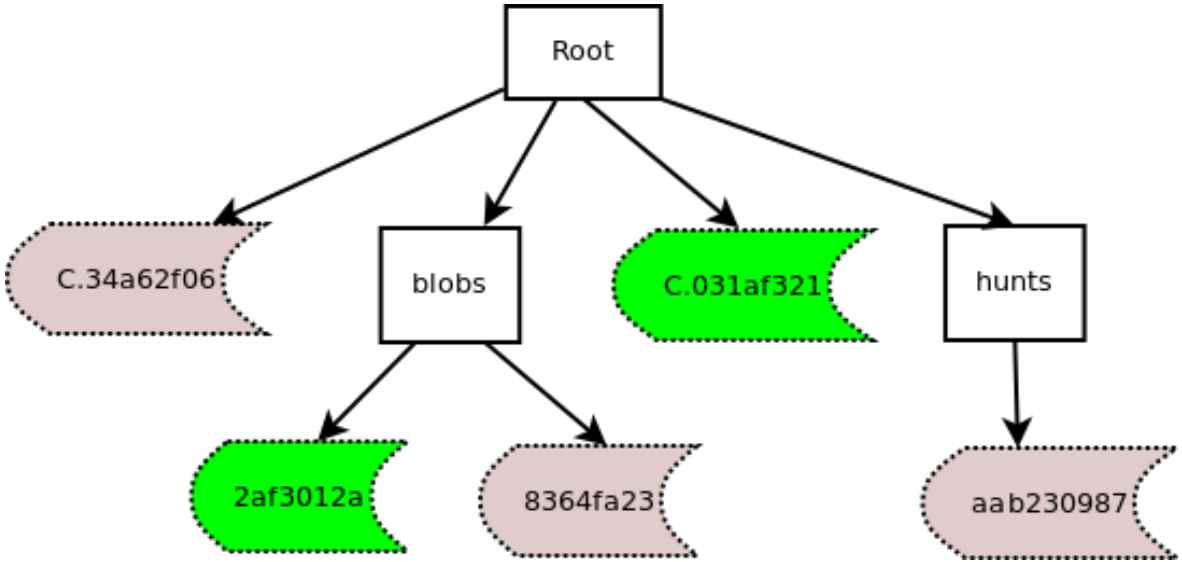
Directory of files



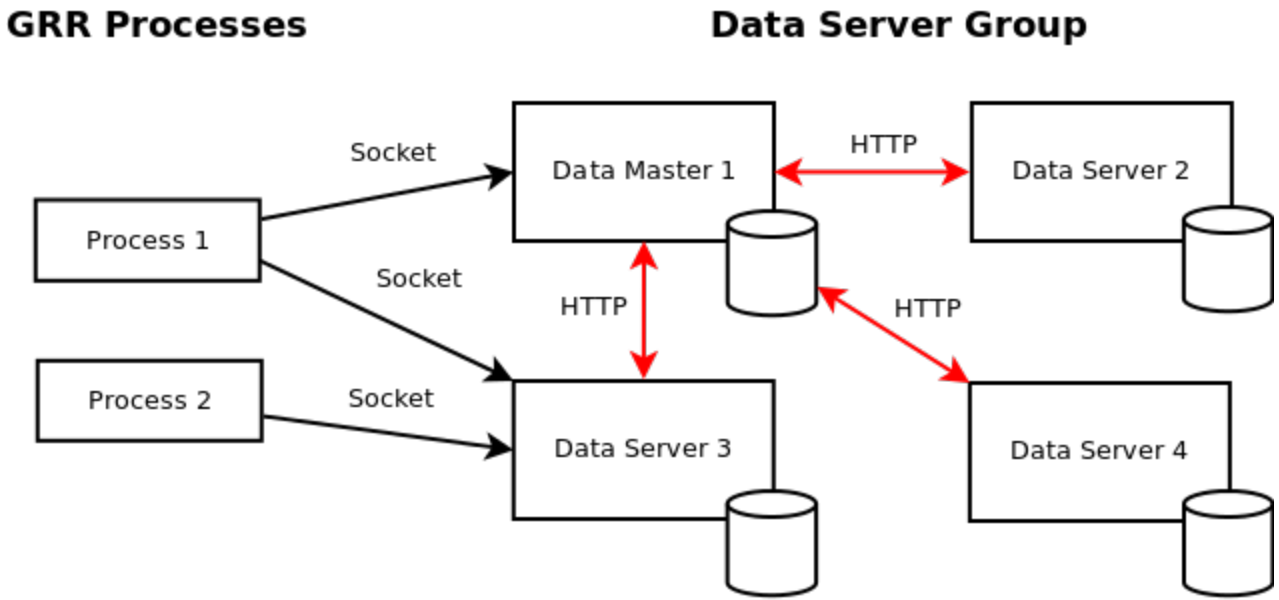
Distributed data store

- While the SQLite data store has great performance, it will not scale well because all the server processes must run on a single machine
  - A single machine can only hold so much data
- Partition the SQLite files across N servers

Partitioning the directory

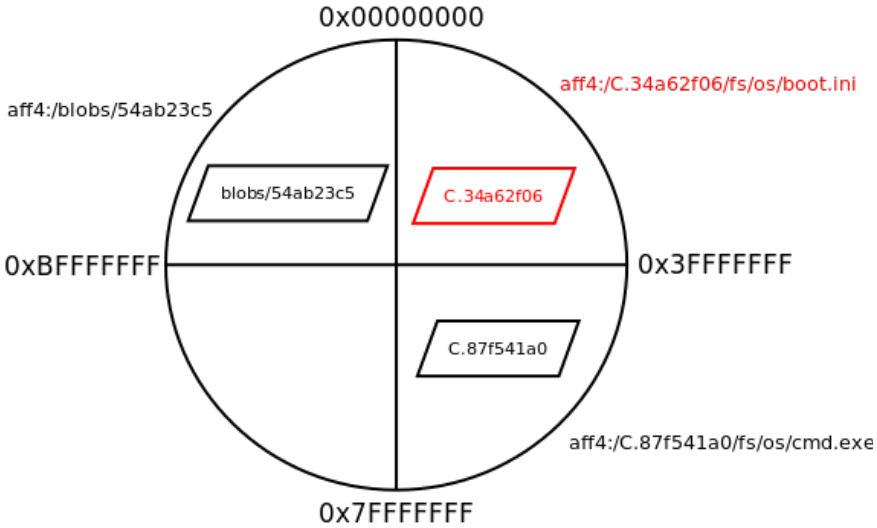


Architecture



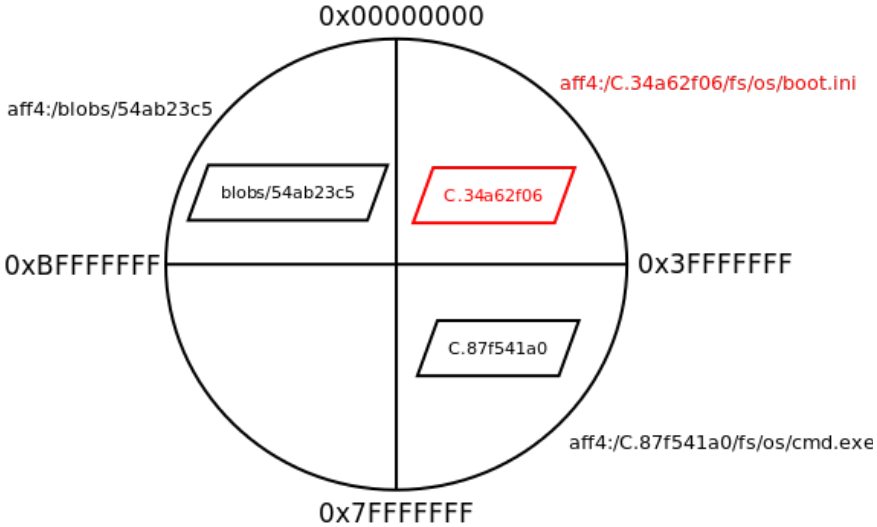
Mapping AFF4 objects

- We need to map objects to data servers
  - Solved problem: objects to files
  - We need to map files to data servers



Consistent hashing

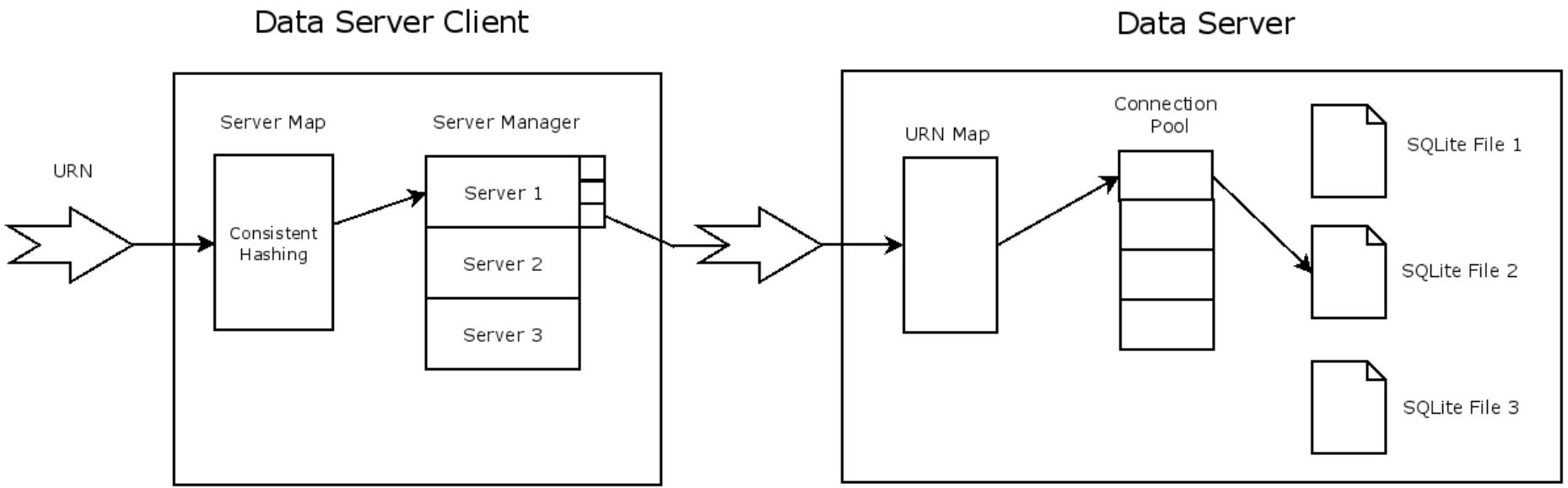
- Server hash space ranges from 0 to 2 ** 64
- ‘aff4:/C.34a62f06/fs/os/boot.ini’ -> ‘C.34a62f06.sqlite’
- hash(‘C.34a62f06.sqlite’) -> 0x10000000



How is the mapping configuration used?

- Data Master:
  - Startup: check if mapping configuration exists
    - Create it if that is not the case
- Data Server:
  - Read master location
  - Register with master and receive mapping configuration
- Client:
  - Read list of servers
  - Ask a random server for the mapping
  - Master has a limited role

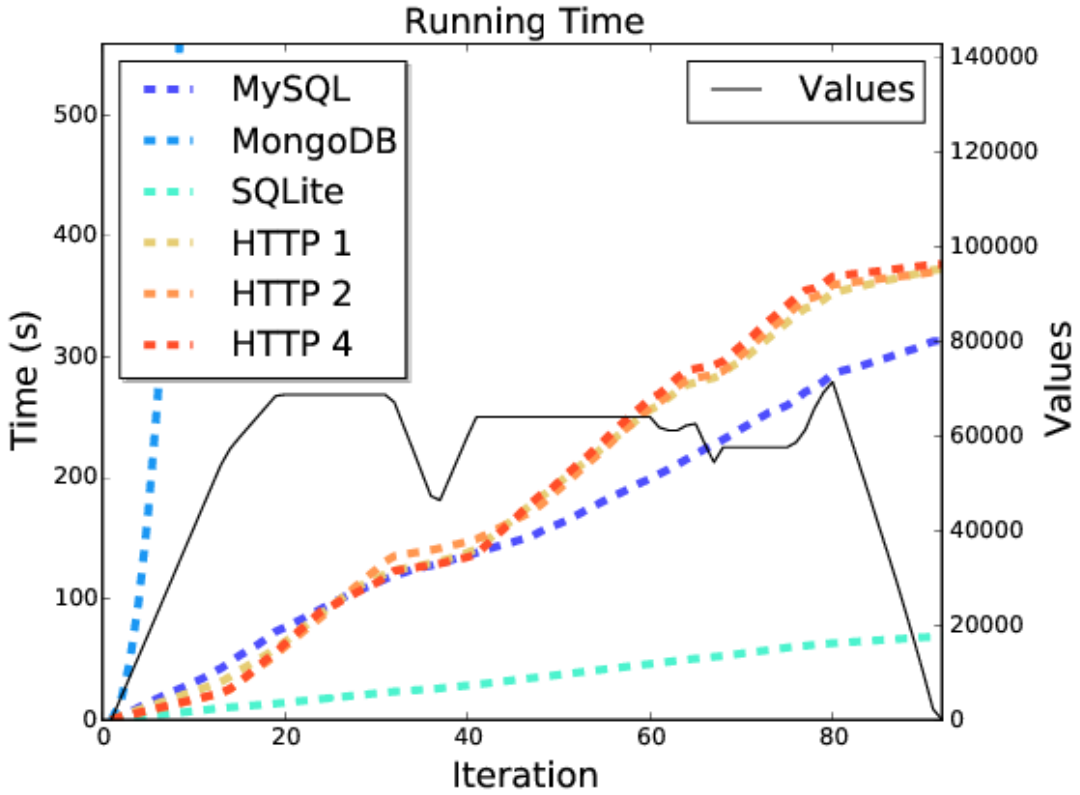
Data Store Overview



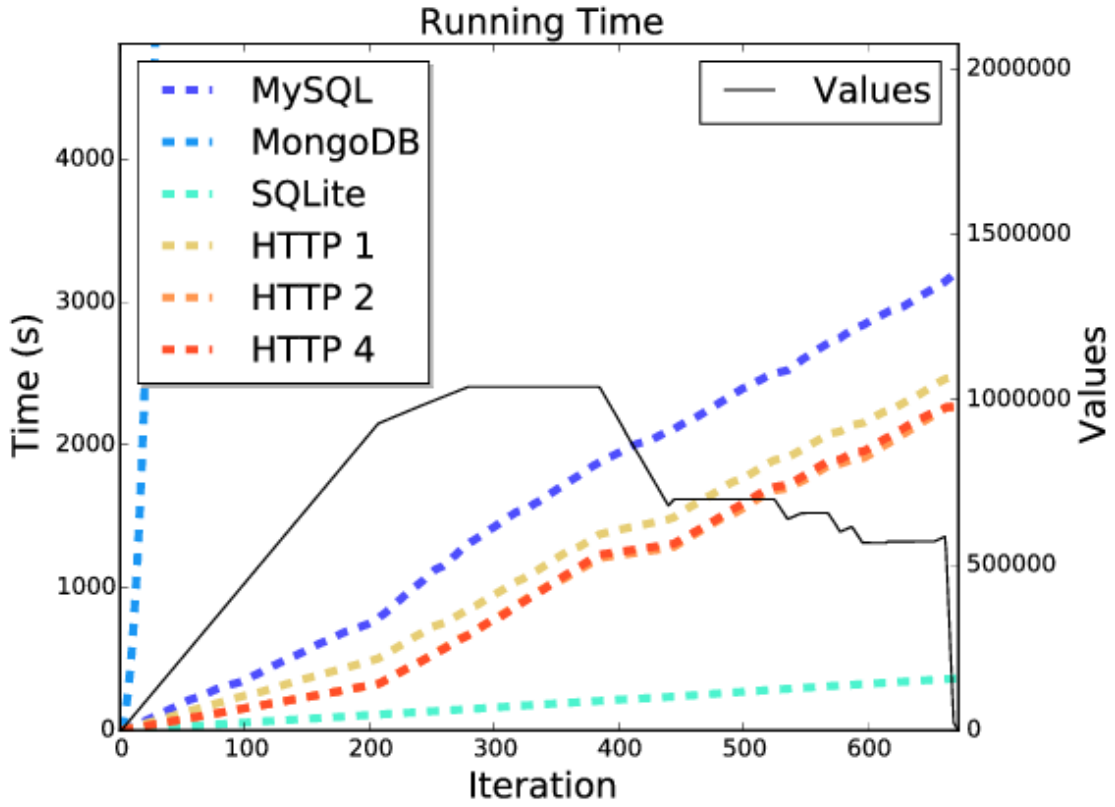
Micro Benchmarks

- Measured the performance of the new data store using micro-benchmarks
  - Benchmark script performs a sequence of data store operations
- Used two scenarios

(1) Objects with few attributes



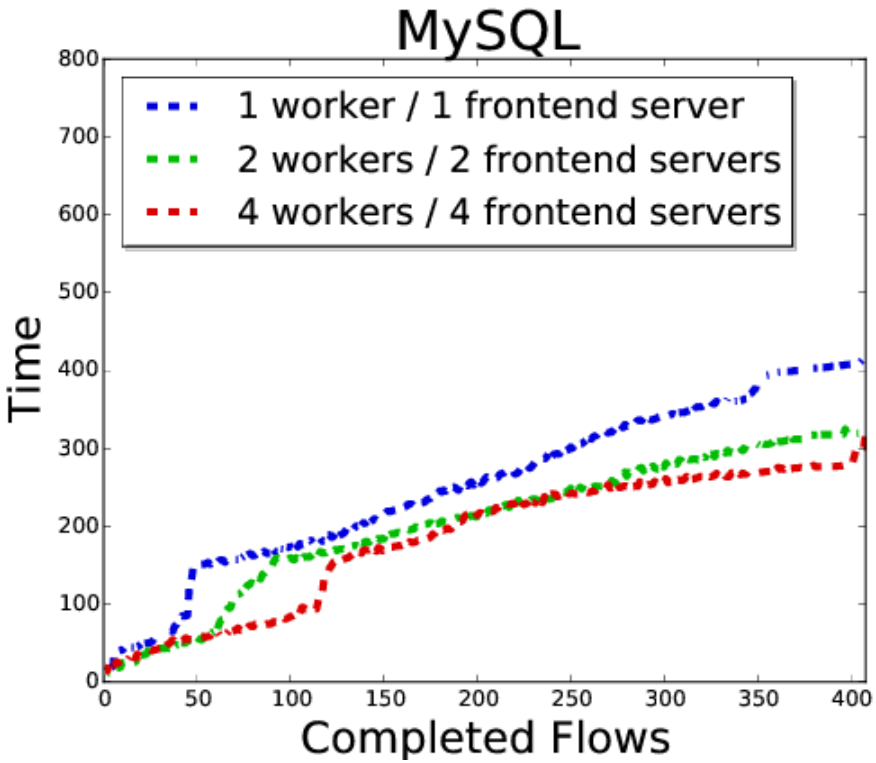
(2) Objects with many attributes

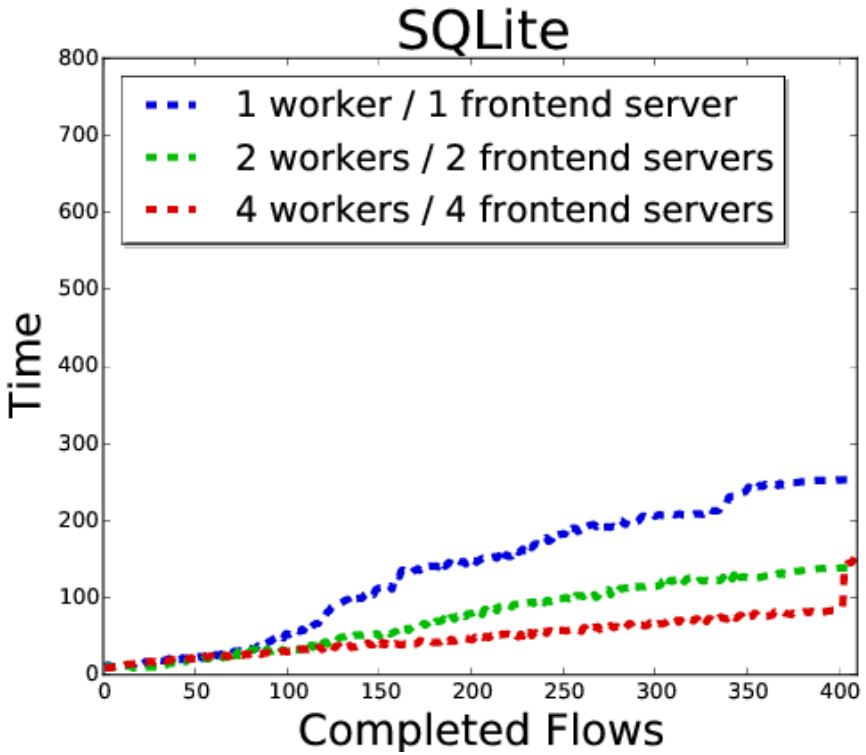


End to End Benchmarks

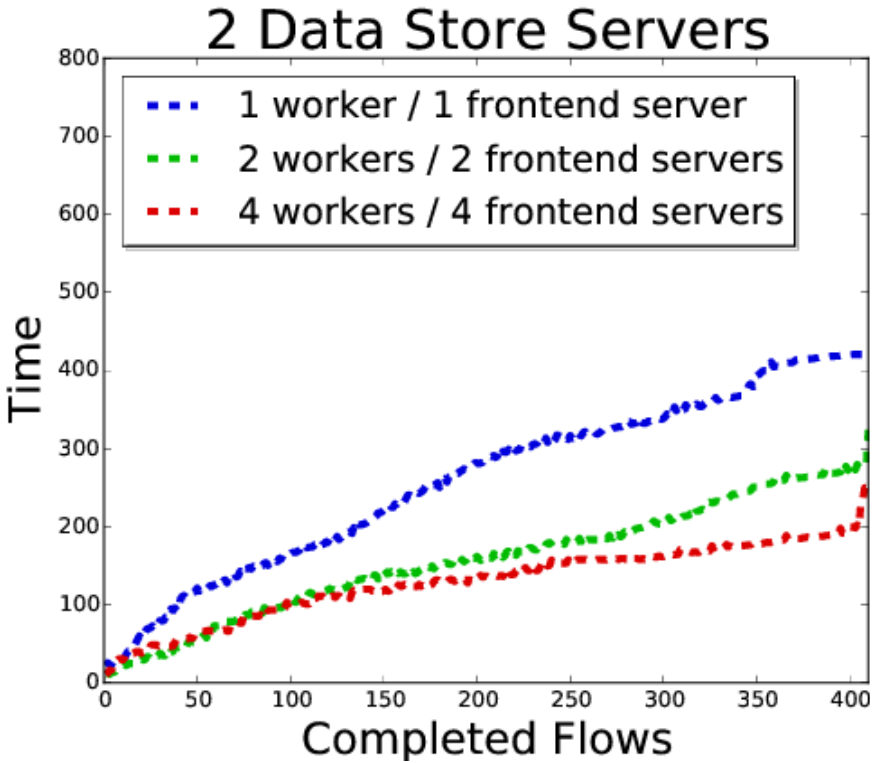
- Execute several “flows” on multiple clients running on a single machine
- We measured the scalability of the data stores by increasing the number of:
  - Workers
  - Frontend servers
  - Data servers

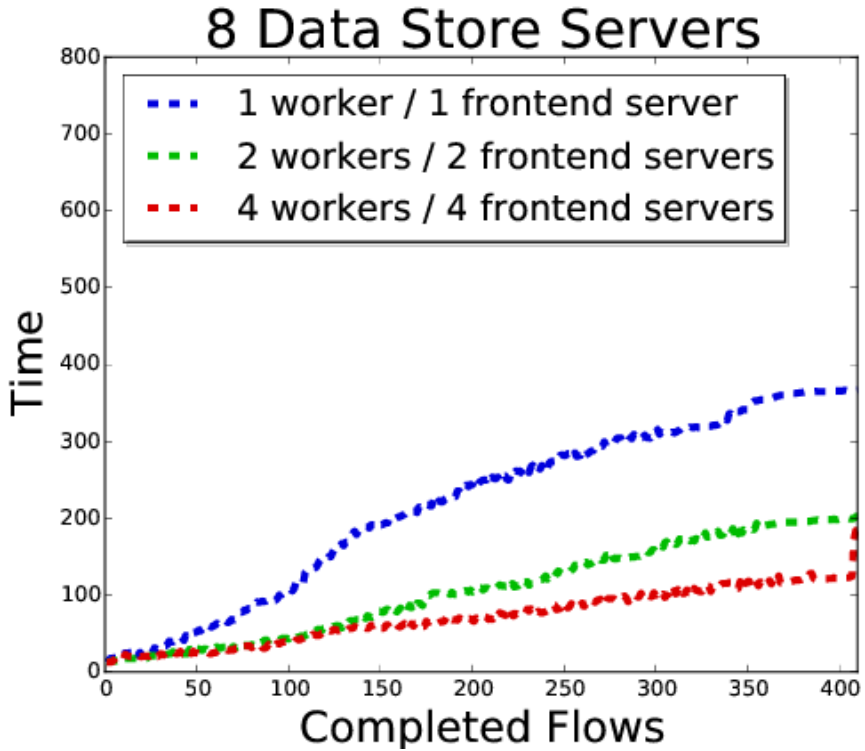
End to End Benchmarks





End to End Benchmarks





Using the Data Store with NSRL

- NSRL is a library of hashes of files from known software packages
- GRR can de-duplicate files from multiple clients
- Idea: use the NSRL dataset to avoid storing well known files
  - Need to populate the data store with the NSRL library

Storing the NSRL library

- New AFF4 object that represents a NSRL file: aff4:/files/nsrl/sha1-hash
- Use the 3 characters of the sha1-hash for the path configuration
  - 000.sqlite … fff.sqlite: 4096 x 33MB files
- Took 5 hours to import and files are easily re-sharded if needed

Using the NSRL library on Windows 7

Statistic	With NSRL	Without NSRL
Files found	1605	1605
Files skipped	1245	2
Files fetched	360	1603
Data store size	148MB	314MB
Client sent	117MB	243MB
Client received	5MB	8MB
Client time	293s	400s

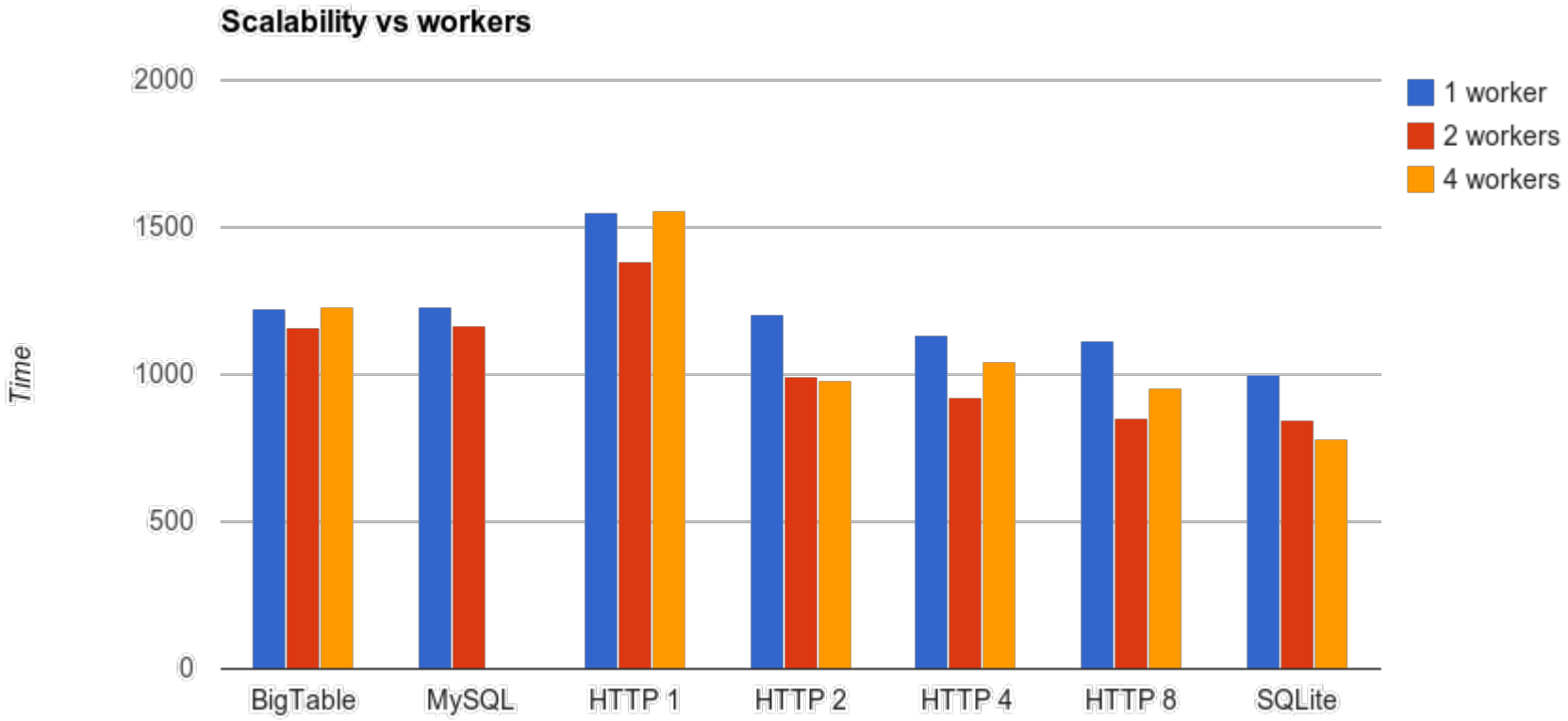
Thanks for your time!

Flavio Cruz
<fmfernan@cs.cmu.edu>

Authentication

- Data servers use an username/password when registering with the master
- After data server registration, the master also sends the encrypted client credentials to the data server
- Client credentials are a set of (user, password, permissions)

Scalability Results



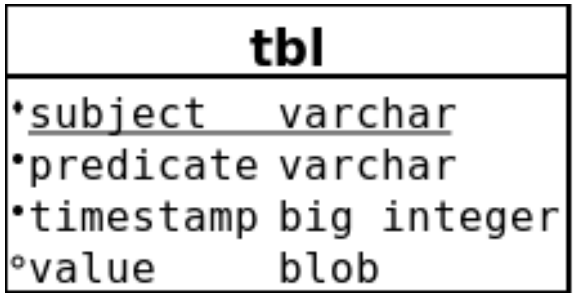
Server processes

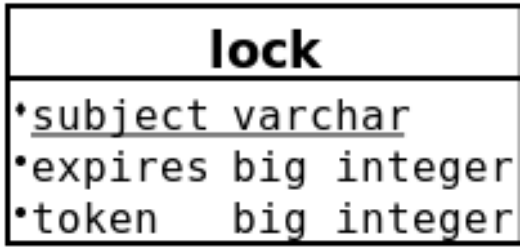
- Server processes share the data store and are allowed to manipulate it.
  - Frontend server
  - Worker: used by the server to perform tasks retrieved from a data store queue
  - Enroller: special worker responsible for the initial enrollment of clients
  - Admin UI: user interface that allows users to interact with the system

Data Store Operations

- MultiSet(subject, values, sync, replace)
- MultiResolveRegex(subject, predicate_regex, timestamp)
- MultiResolve(subject, predicates, timestamp)
- DeleteSubject(subject)
- DeleteAttributes(subject, predicates, start, end, sync)
- DeleteAttributesRegex(subject, regexes)
- Transaction(subject, lease_time)
- Flush()
- Size()

Sqlite Data Store





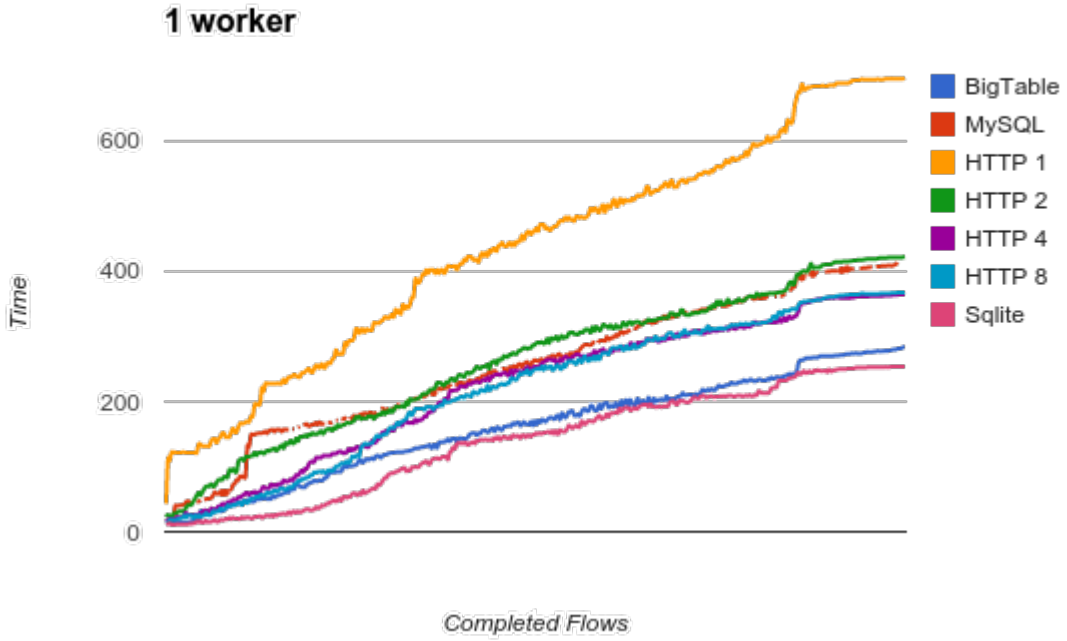
Asynchronous Requests

- Each client may create up to N connections to each data server
- When a data store request is made, we pick the connection with the least number of pending requests
- If request is synchronous, we write it to the socket and wait until we receive a reply
- If request is asynchronous, we do not wait for a response

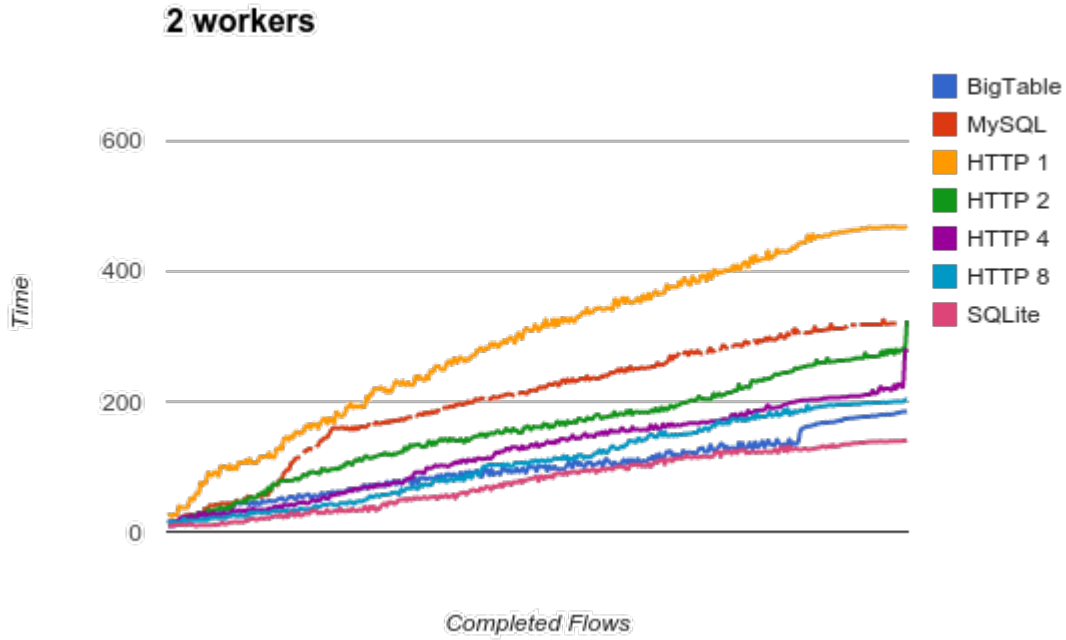
Threading

- Data servers create one thread per client connection
- Improved CPU utilization since network I/O and data store I/O takes time
- The GIL may still become a problem since each data server can only use one core at a time
  - Solution: use more data servers

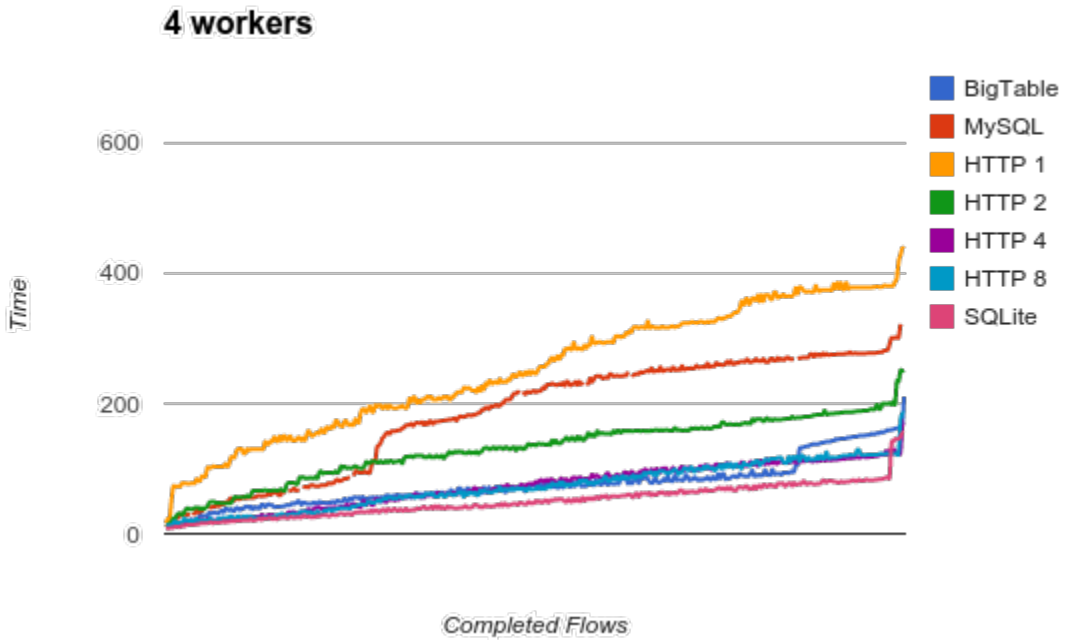
1 worker


1 worker
600
400
200
0
Time
Completed Flows
BigTable
MySQL
HTTP 1
HTTP 2
HTTP 4
HTTP 8
Sqlite

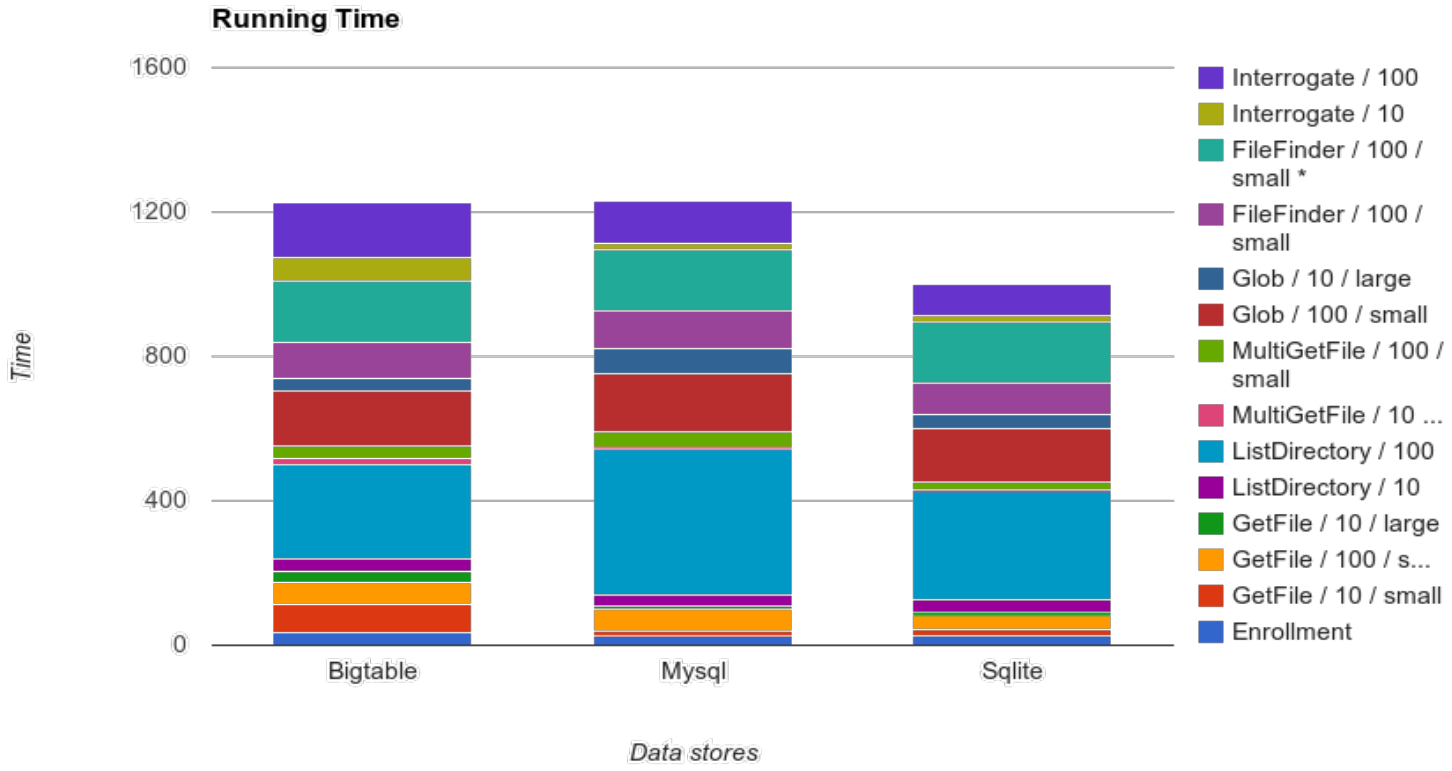
2 workers


2 workers
BigTable
MySQL
HTTP 1
HTTP 2
HTTP 4
HTTP 8
SQLite
600
400
200
0
Time
Completed Flows

4 workers


4 workers
BigTable
MySQL
HTTP 1
HTTP 2
HTTP 4
HTTP 8
SQLite
600
400
200
0
Time
Completed Flows

Push Benchmarks (Time)



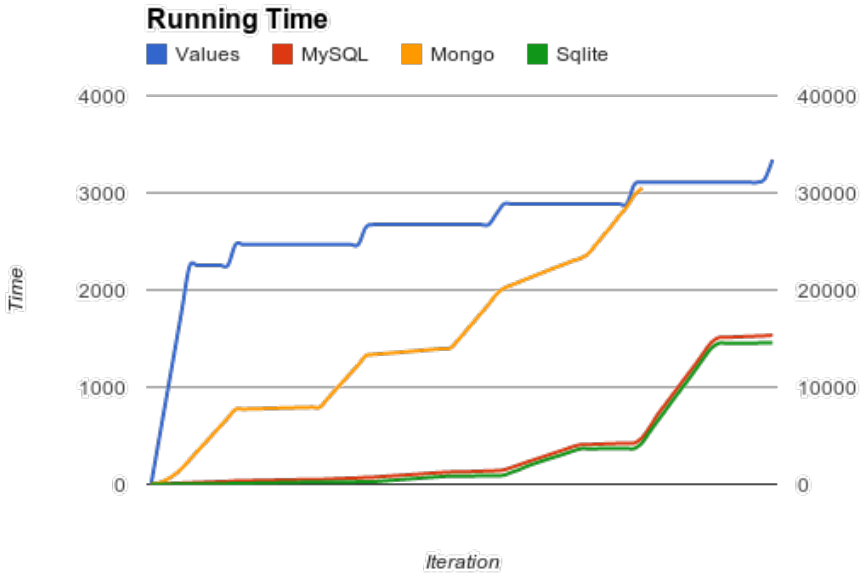
Re-balancing

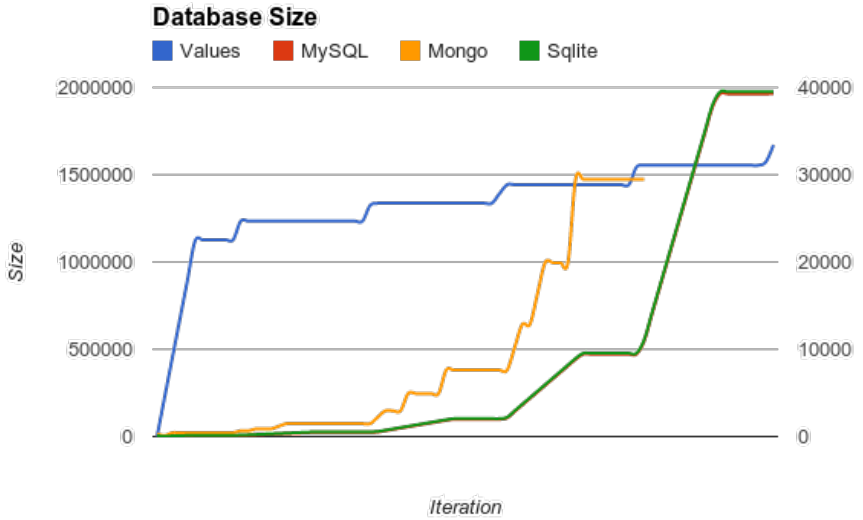
- Two-phase commit:
  - Data servers send files to target data servers
    - Sent files are kept in temporary storage
    - And that also includes list of files to remove
  - All data servers apply the transaction
    - Move files to real places
    - Delete unneeded files
- Even if the second phase fails, we can always resume the process
  - recover <transaction-id>

Communication

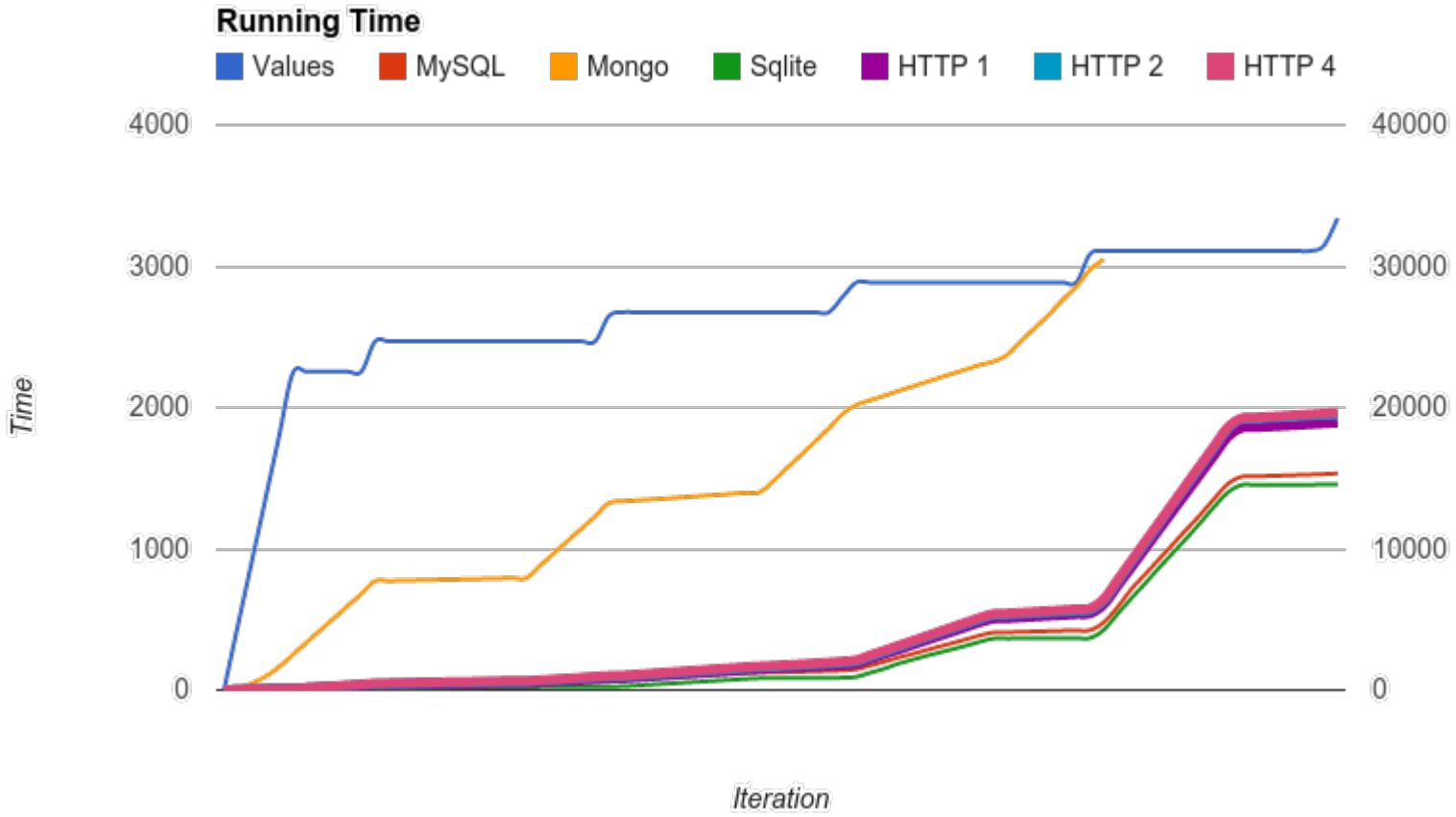
- Mostly using HTTP, except for the the data store requests
- After the initial handshake, client and server “upgrade” to a simple socket protocol, which allows multiple requests at once and faster serialization
  - protocol buffers

Blobs

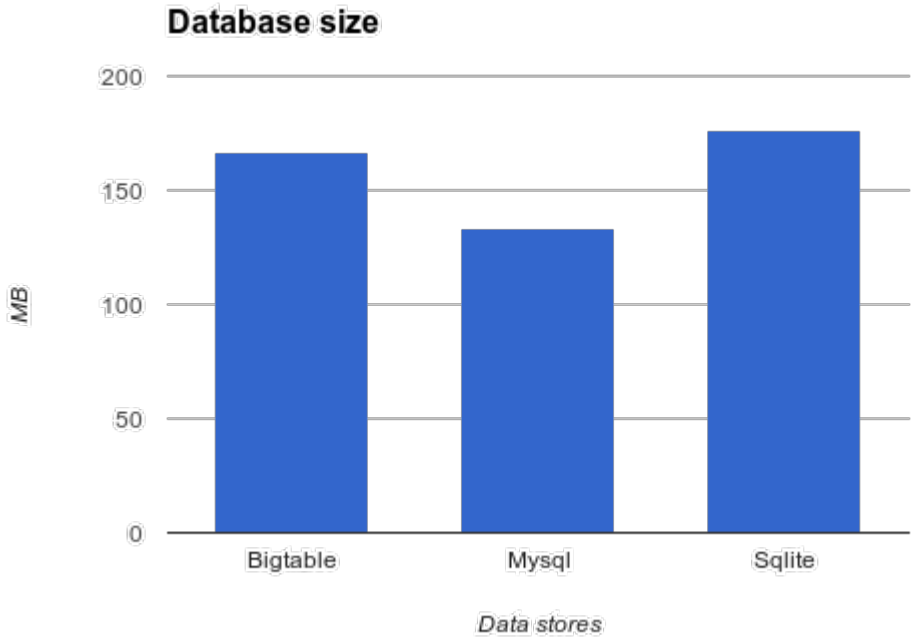

Running Time
Values
MySQL
Mongo
Sqlite
4000
3000
2000
1000
0
40000
30000
20000
10000
0
Time
Iteration


Database Size
Values
MySQL
Mongo
Sqlite
2000000
1500000
1000000
500000
0
40000
30000
20000
10000
0
Size
Iteration

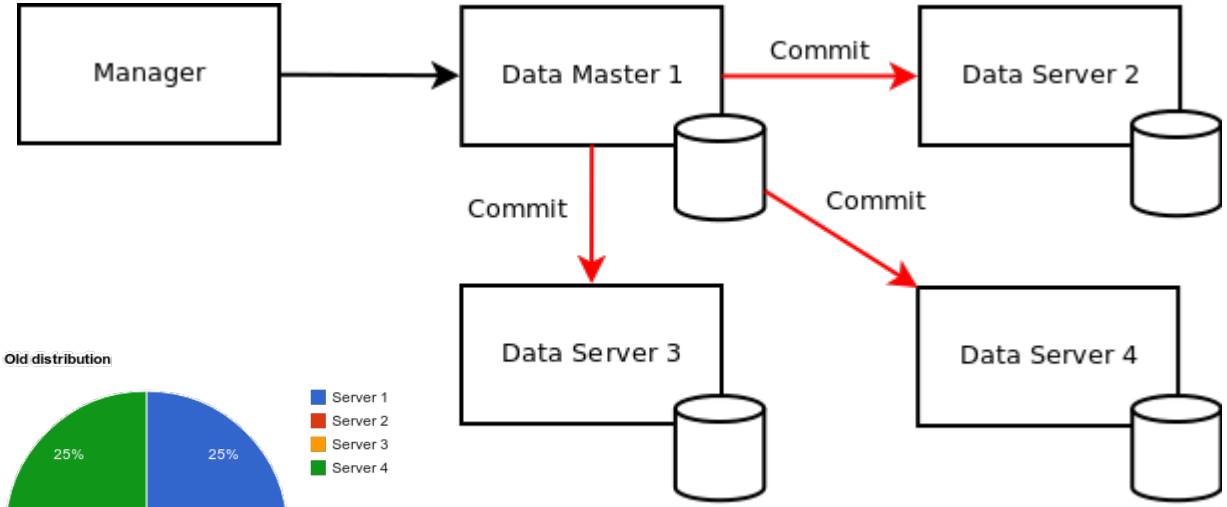
Blobs

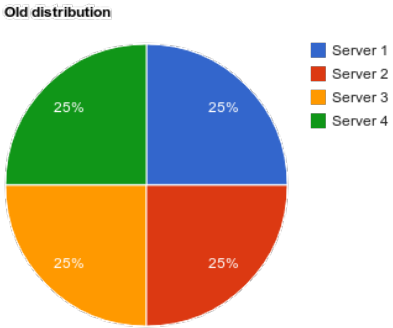

Running Time
Values
MySQL
Mongo
Sqlite
HTTP 1
HTTP 2
HTTP 4
4000
3000
2000
1000
0
40000
30000
20000
10000
0
Time
Iteration

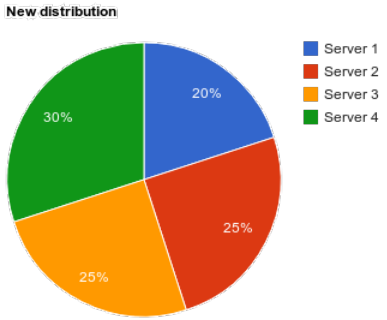
End to End Benchmarks (Size)



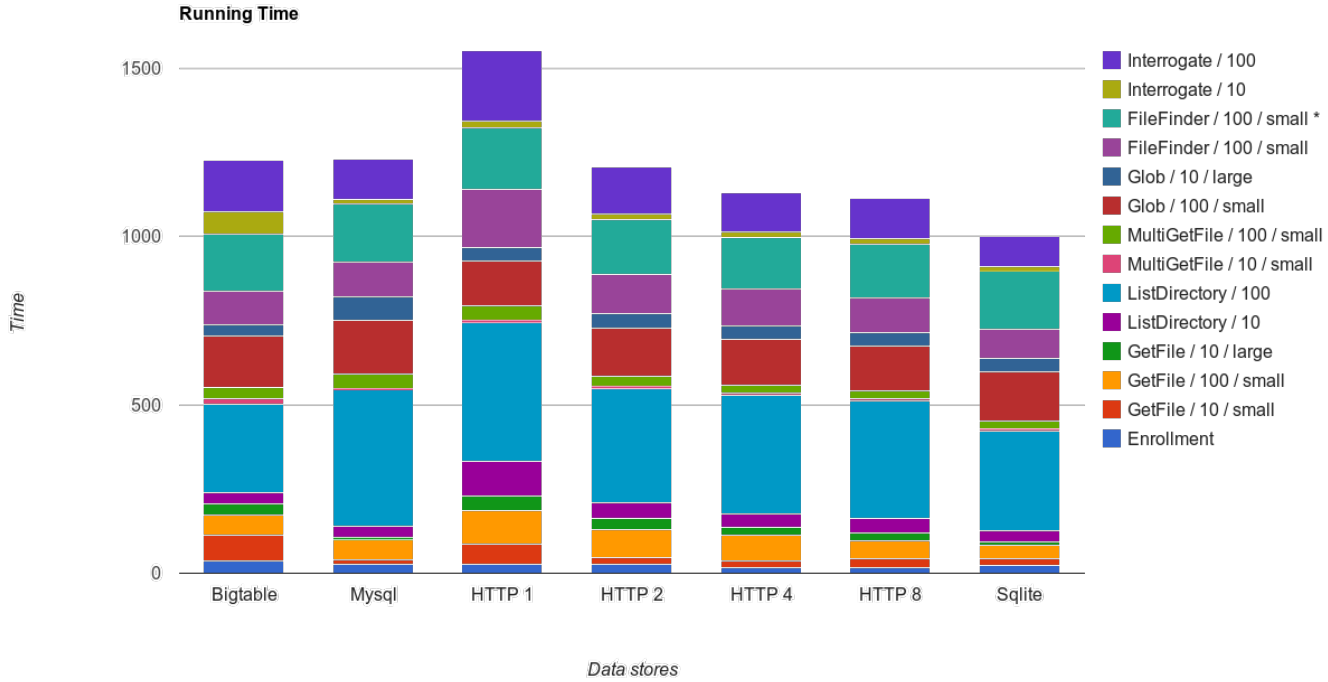
Manager


Manager
Data Master 1
Commit
Data Server 2
Commit
Commit
Data Server 3
Data Server 4


Old distribution
25%
25%
25%
25%
Server 1
Server 2
Server 3
Server 4


New distribution
20%
25%
25%
30%
Server 1
Server 2
Server 3
Server 4

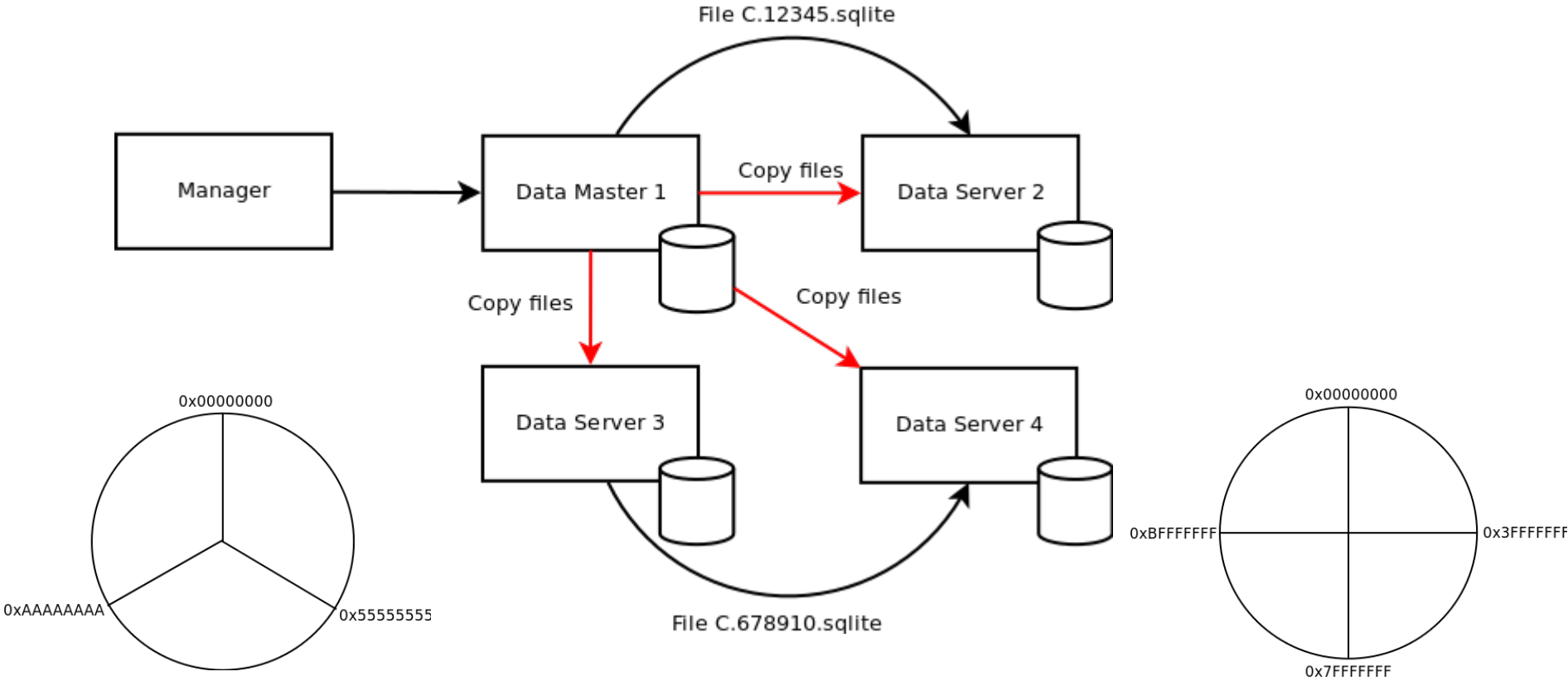
End to End Benchmarks



Re-sharding

- What if we want to add more data servers?
  - Need new mapping configuration
  - Need to move files because some of them will be in the wrong server
- Manager solves this problem
  - Creates new mapping configuration
  - Files are moved automatically from server to server
  - Only works in “offline” mode

Adding new server



Removing data servers

- To delete a server, we first re-balance and set the target server hash range to [x, x[
  - Afterwards it is just a matter of stopping the server and removing the server from the configuration

(2) Few objects and many attributes