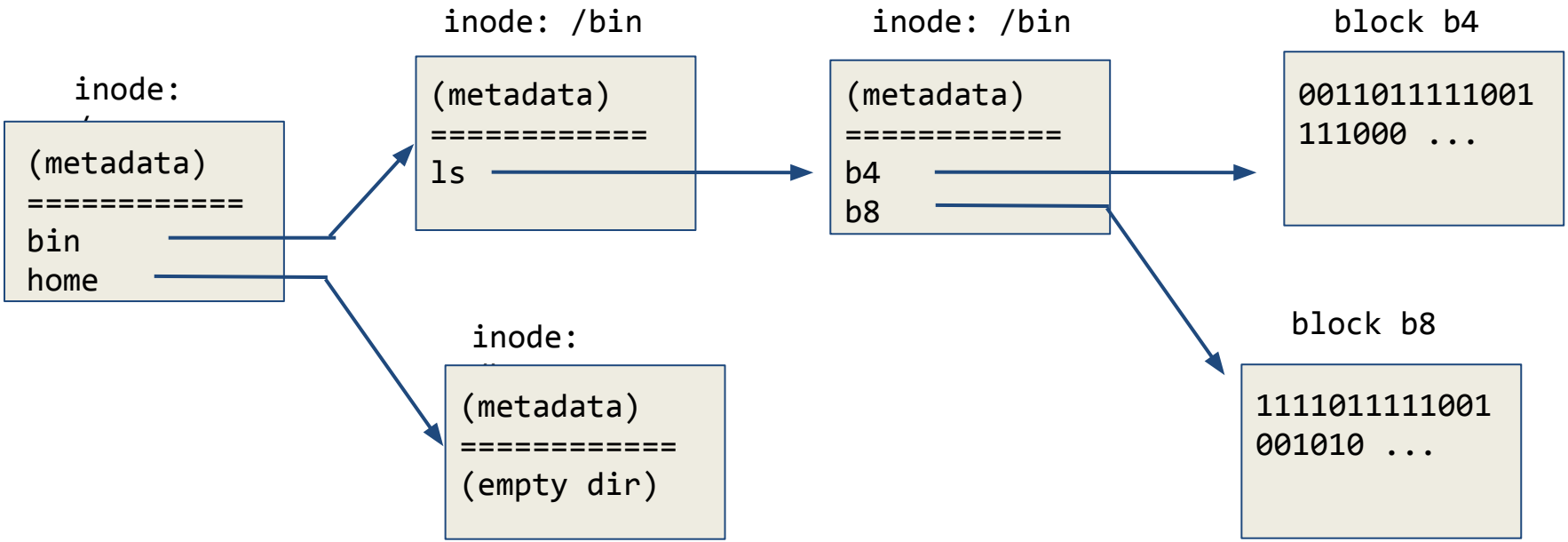
Distributed File Systems - an overview

Pierre Sutra,
Télécom SudParis

RainbowFS kick-off meeting
4-5 May 2017, Montereau



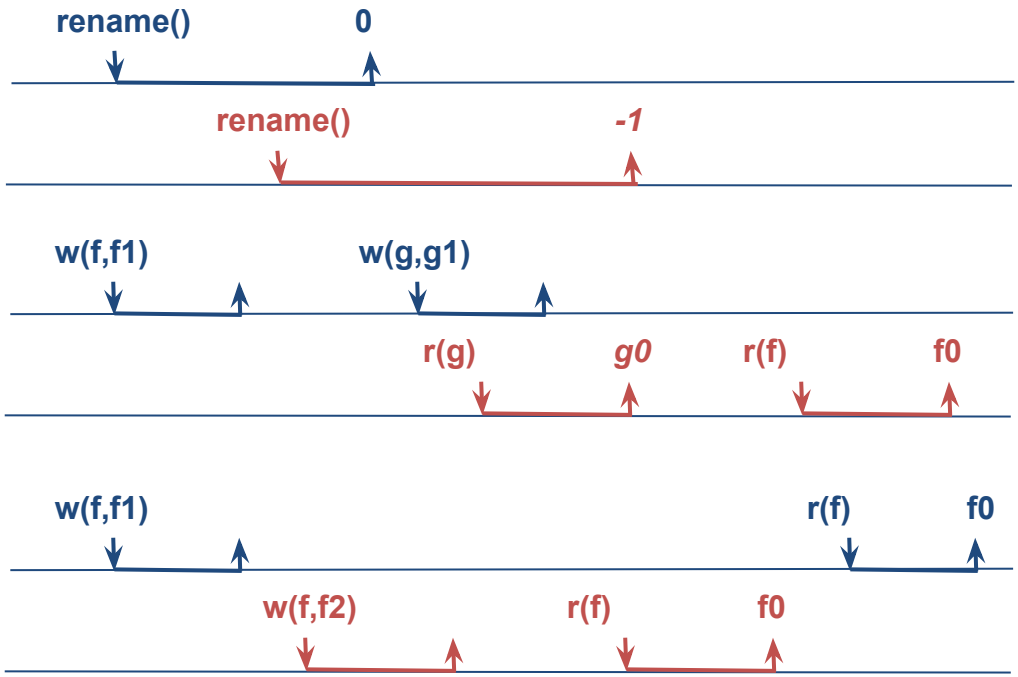
inode = container for file, directory, symbolic link, special files.
block = data content

Basics - interface*


```
int(* create )(const char *, mode_t, struct fuse_file_info *)

int(* open )(const char *, struct fuse_file_info *)

int(* read )(const char *, char *, size_t, off_t, struct fuse_file_info *)

int(* write )(const char *, const char *, size_t, off_t, struct fuse_file_info *)

int(* flush)(const char *, struct fuse_file_infor *)

int(* rename )(const char *, const char *)

int(* statfs )(const char *, struct statvfs *)

int(* unlink)(const char *)
```

** interface given for File System in User-Space (POSIX-compliant)*

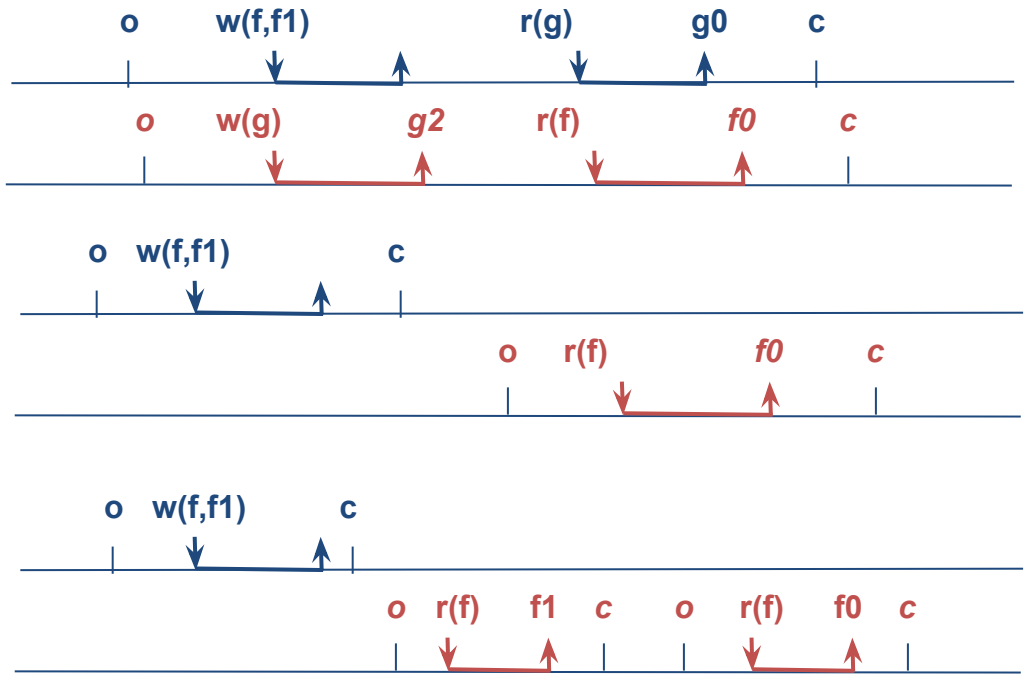
Basics - consistency*



linearizability	POSIX, Lustre
sequential consistency	Sprite, HDFS, GoogleFS
eventual consistency	Pastis

** flush() operations are missing*

Basics - consistency



linearizability	AFS, NFS
sequential consistency	SinfoniaFS
eventual consistency	Coda, Ivy

With close-to-open semantics

Idea. share transparently file systems

Features.

server-client architecture
no globally-shared file system
RPCs for mount + remote file access

Limitations.

all data served by a single server
synchronous writes (≤ v2)

/nfs

/nfs

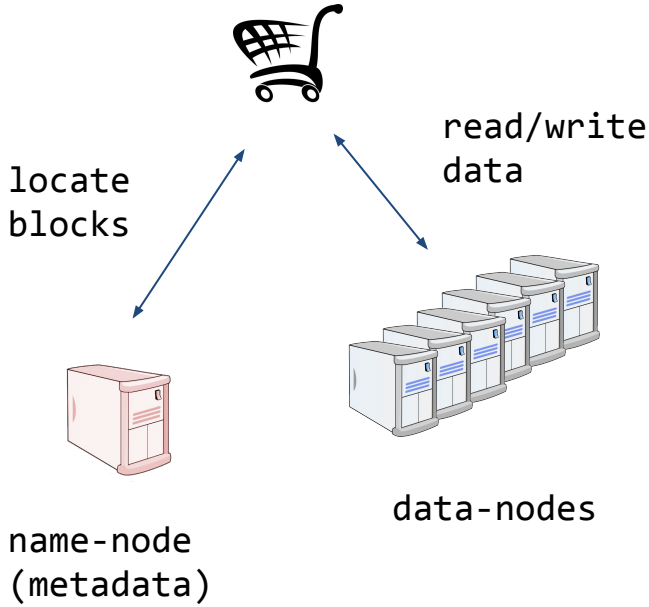
/shared/foo/bar

```
mount -t nfs 10.0.0.1:/nfs /shared/foo

mount -t nfs 10.0.0.2:/nfs /shared/foo/bar
```

Idea. metadata operation are fast
→ decouple data/metadata operations

Features.
name-node keeps metadata in-memory + log
data nodes hold (replicated) file blocks
large blocks (128MB)
location of blocks reported by data node

Limitations.
at most PB scale
log should be persistent (e.g., using ZooKeeper)
name-node = single point of failure
bottleneck on metada-heavy workload



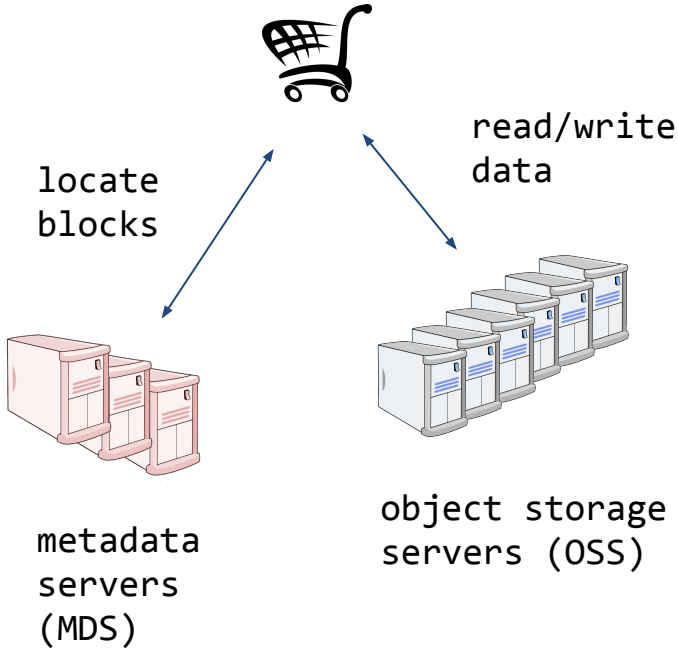
Idea. use multiple metadata servers

Features.

POSIX-compliant
directories may span multiple MDS (≥ 2.4)
OSS = object based disk server + lock server
support for Infiniband, RDMA
hadoop support (from Intel)

Limitations.

not build for commodity hardware
→ no data replication (outside of RAID)



Directions

How to scale the metadata server(s) ?

Is it possible to implement a fully distributed DFS?

What are the trade-off regarding consistency, performance and availability ?

Idea. split the file system tree such that

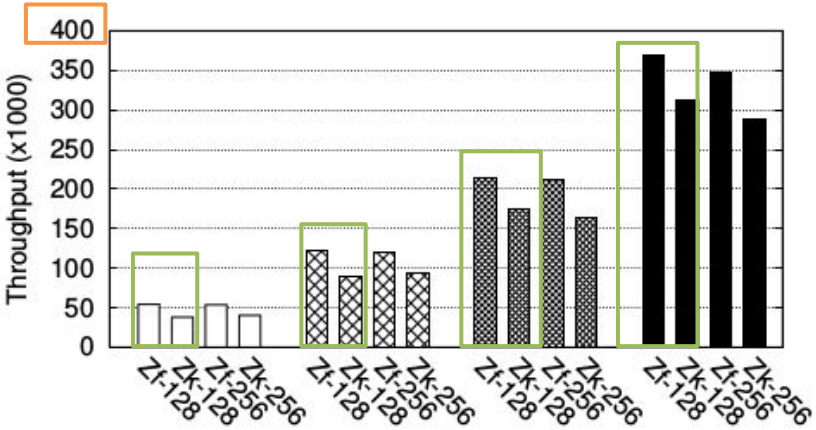
- if path *p* is stored at machine M, then M also stores *parent(p)*
- when applying command *c* to *p*, executes c on all the machines storing *p* (in a consistent and wait-free manner)

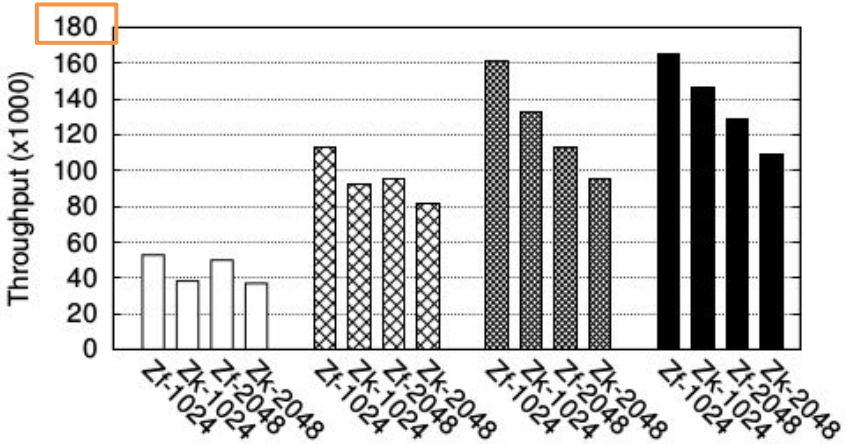
Evaluation.

- ZooFence = split ZooKeeper tree
- Bookkeeper = ZooKeeper based write-ahead log
- setting 18 nodes, rep. factor = 3
- workload

```
entry[] = random(0,B) // B bytes
open(“/new_log”,CREATE|WRITE);
write(“/new_log”,0,entry); // #times
close(“/new_log”)
```

Directions - scaling-up a metadata server





Zk = ZooKeeper
Zf = ZooFence (split tree)

log size = 128,256,1024 or 2056
#entries = 100, 250, 500 or 1000

split tree always faster (up to 45%)

Idea. implement a file system with *put*(), *get*() and *c&s*().

Example.

```
create(path,mode,info)
     iblock = new IBlock(info)
     return c&s(path,null,tmp)

open(path,mode,info)
     iblock = get(path)
     files[path] = (iblock == null) ? iblock : new IBlock()
     return check_perm(files[path],more)
```

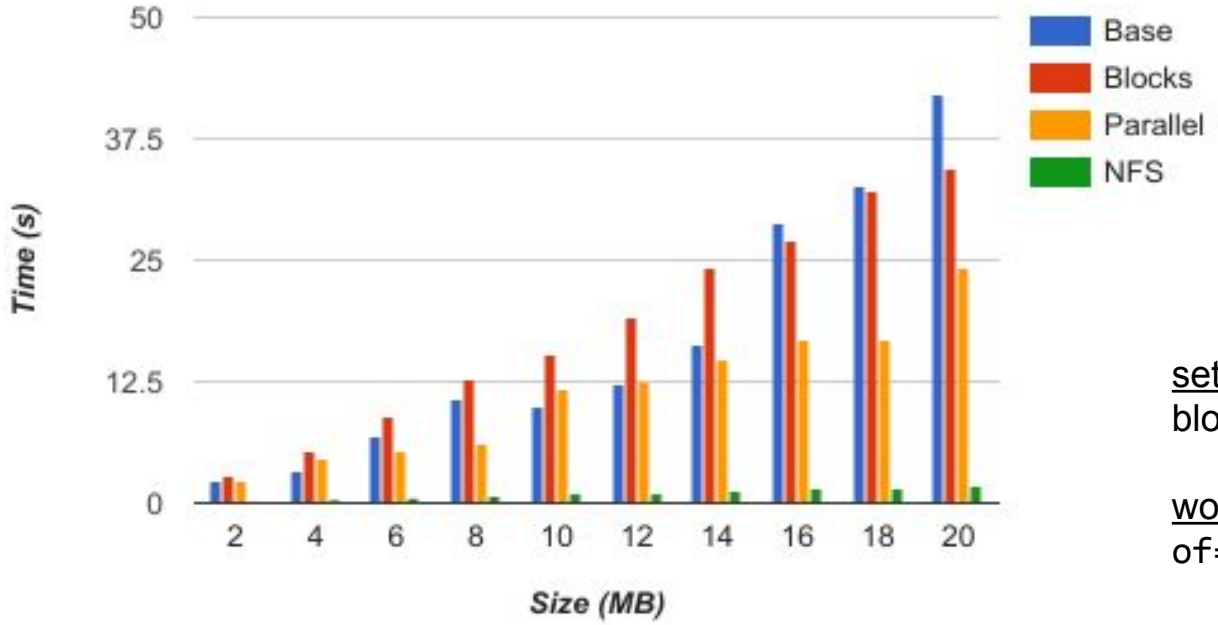
Design space.

- base solution = read/write full file each time
- variation 1 = split files in blocks
- variation 2 = read/write blocks in parallel

Evaluation.

- POSIX-compliant (single writer)
- use Cassandra
- python language (fuse.py)

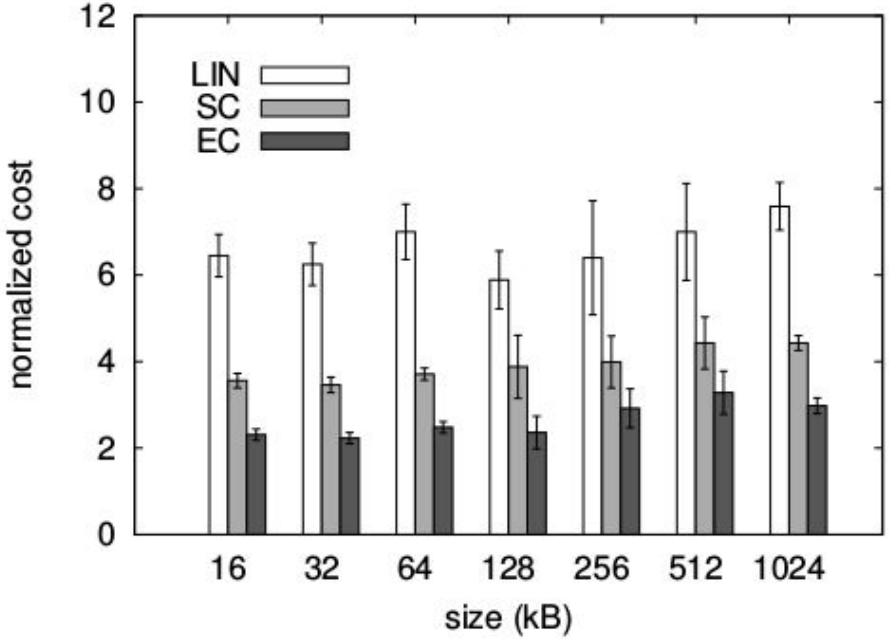
Directions - no metadata server



setting 4 nodes, rep. factor = 2, block size = 20KB

workload `time dd if=/dev/zero of=test bs=1024k count=XXX`

Idea. performance comparison of DFS consistency criteria

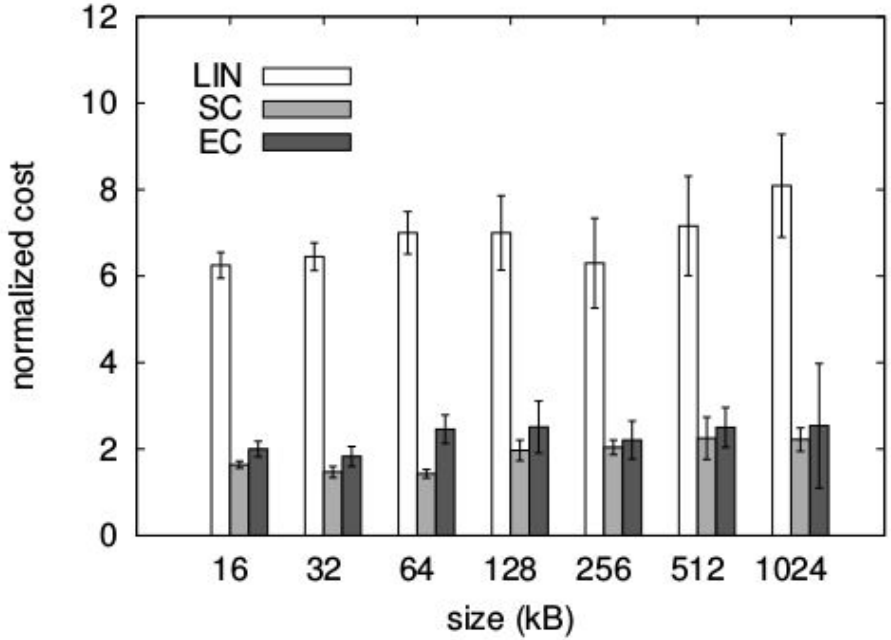


setting 3 nodes, rep. factor = 2, block size = 128 kB

workload `touch /path/to/file`

1 = roundtrip cost

Idea. performance comparison of DFS consistency criteria

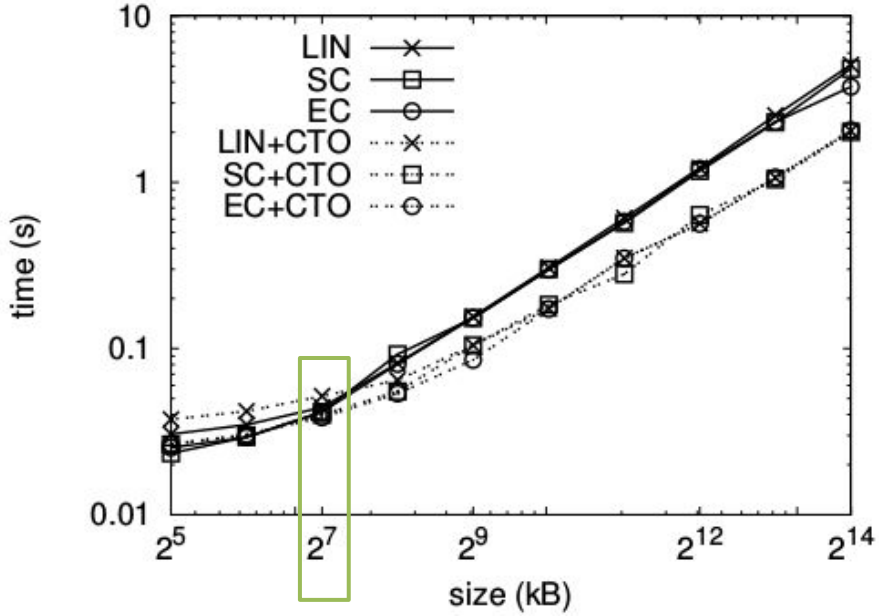


setting 3 nodes, rep. factor = 2, block size = 128 kB

workload `cat /path/to/file`

1 = roundtrip cost

Idea. performance comparison of DFS consistency criteria



setting 3 nodes, rep. factor = 2, block size = 128 kB

workload `time dd if=/dev/zero of=test bs=1024k count=XXX`

CTO = close-to-open semantics

State of the game.

- POSIX specification difficult to grasp
  (plain english, many pages)
- a lot of DFS exist, with various semantics
- over time, more distribution/parallelism
  → more performance / availability

Directions.

- scaling a metadata server (split tree) is complex
- better to use no data server (?)
  → e.g., add a read-modify-write to key/value store
- trade-off between DFS semantics / performance
  → depends on workload / block size