What does rename() do?

Steve Loughran
stevel@hortonworks.com
@steveloughran

June 2017

How do we safely persist & recover state?

Why?

- Save state for when application is restarted
- Publish data for other applications
- Process data published by other applications
- Work with more data than fits into RAM
- Share data with other instances of same application
- Save things people care about & want to get back


FAT8
dBASE II & Lotus 1-2-3
int 21h
DS,DD
double side
double density
3M
DS, DD, 96 TPI, RH
Double Side
Double Density
Soft Sector
INDEX 01
3M
DISKETTES / DISQUETTES
DISKETTEN / DISKETTES

Linux: ext3, reiserfs, ext4
sqlite, mysql, leveldb

```
open(path, O_CREAT|O_EXCL)
rename(src, dest)
```

Windows NT, XP
NTFS
Access, Excel

```
CreateFile(path, CREATE_NEW,...)
MoveFileEx(src, dest, MOVEFILE_WRITE_THROUGH)
```


Facebook Prineville Datacentre
1+ Exabyte on HDFS + cold store
Hive, Spark, ...
FileSystem.rename()

Model and APIs

HORTONWORKS®
POWERING THE FUTURE OF DATA™

Structured data: algebra

HORTONWORKS®
POWERING THE FUTURE OF DATA™

A History and Evaluation of System R

Donald D. Chamberlin
Morton M. Astrahan
Michael W. Blasgen
James N. Gray
W. Frank King
Bruce G. Lindsay
Raymond Lorie
James W. Mehl
Thomas G. Price
Franco Putzolu
Patricia Griffiths Selinger
Mario Schkolnick
Donald R. Slutz
Irving L. Traiger
Bradford W. Wade
Robert A. Yost

IBM Research Laboratory
San Jose, California

SUMMARY: System R, an experimental database system, was constructed to demonstrate that the usability advantages of the relational data model can be realized in a system with the complete function and high performance required for everyday production use. This paper describes the three principal phases of the System R project and discusses some of the lessons learned from System R about the design of relational systems and database systems in general.

1. Introduction

Throughout the history of information storage in computers, one of the most readily observable trends has been the focus on data independence. C.J. Date [27] defined data independence as "immunity of applications to change in storage structure and access strategy." Modern database systems offer data independence by providing a high-level user interface through which users deal with the information content of their data, rather than the various bits,

File-System

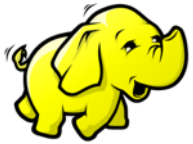
Directories and files

Posix with stream metaphor

HORTONWORKS
POWERING THE FUTURE OF DATA

Posix: hierarchical (distributed?) filesystems

`org.apache.hadoop.fs.FileSystem`



`hdfs`

`wasb`

`s3a`

`swift`

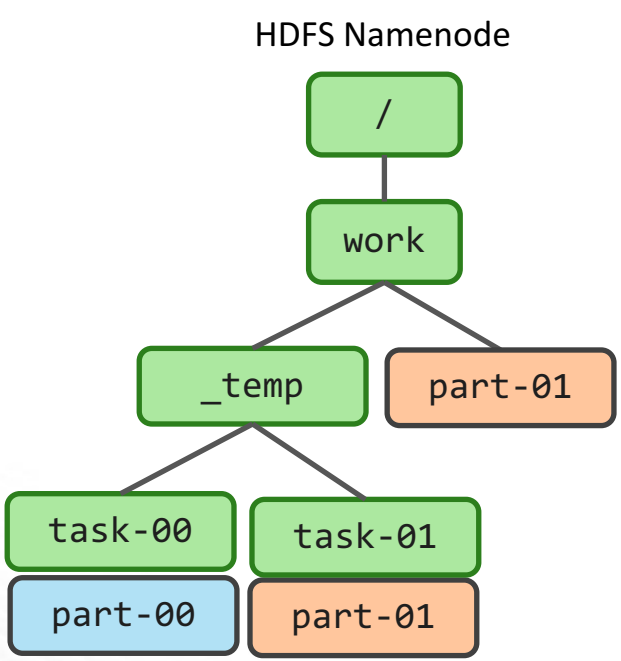
`adl`

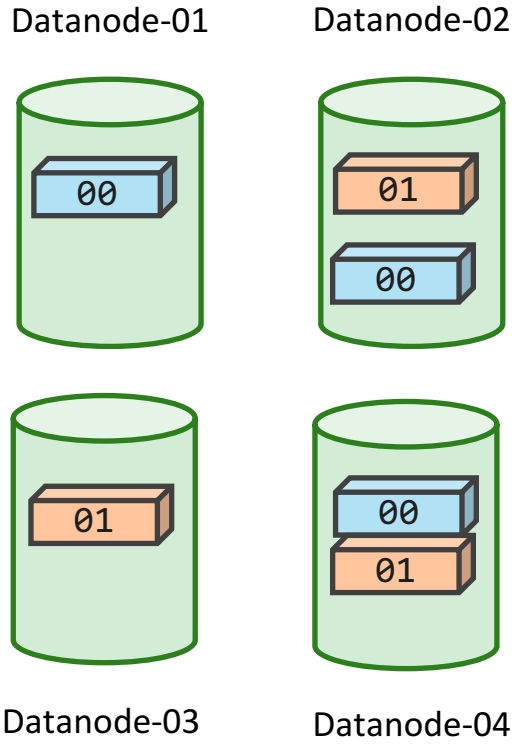
`gcs`

HORTONWORKS

```
val work = new Path("s3a://stevel-frankfurt/work")
val fs = work.getFileSystem(new Configuration())
val task00 = new Path(work, "task00")
fs.mkdirs(task00)
val out = fs.create(new Path(task00, "part-00"), false)
out.writeChars("hello")
out.close();
fs.listStatus(task00).foreach(stat =>
  fs.rename(stat.getPath, work)
)
val statuses = fs.listStatus(work).filter(_.isFile)
require("part-00" == statuses(0).getPath.getName)
```

rename() gives us O(1) atomic task commits

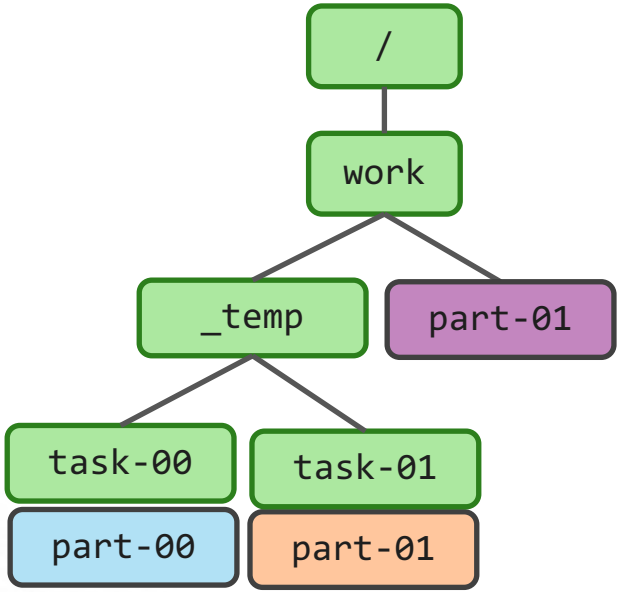


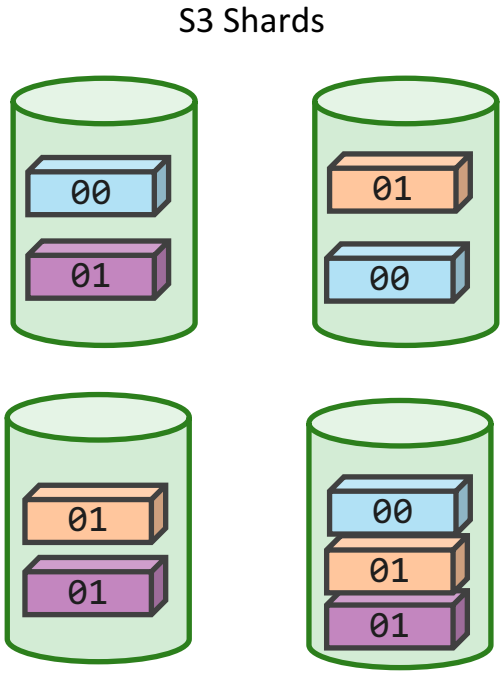



```
rename("/work/_temp/task00/*", "/work")
```


HORTONWORKS

Amazon S3 doesn't have a rename()

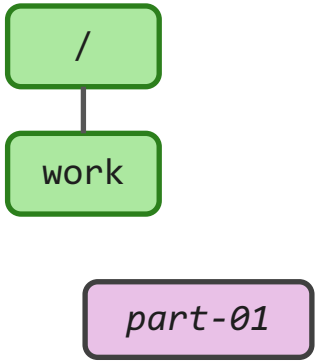


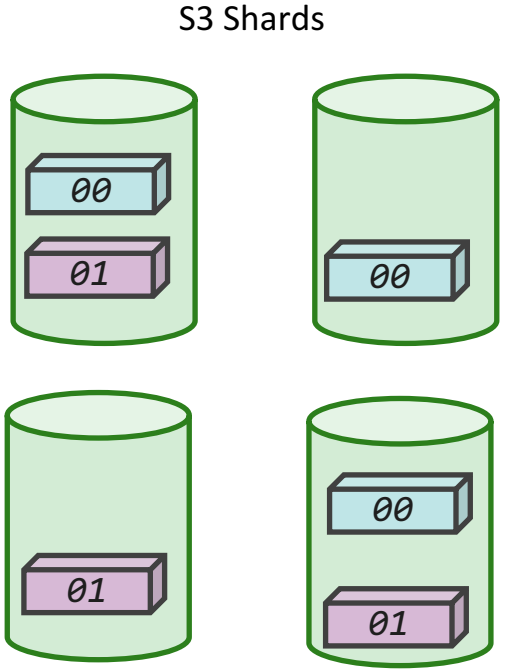



```
LIST /work/_temp/task-01/*
COPY /work/_temp/task-01/part-01 /work/part-01
DELETE /work/_temp/task-01/part-01
```


HORTONWORKS

Fix: fundamentally rethink how we commit



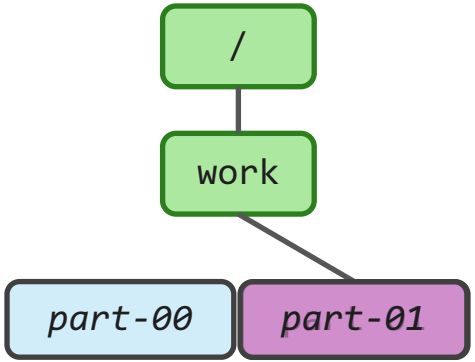


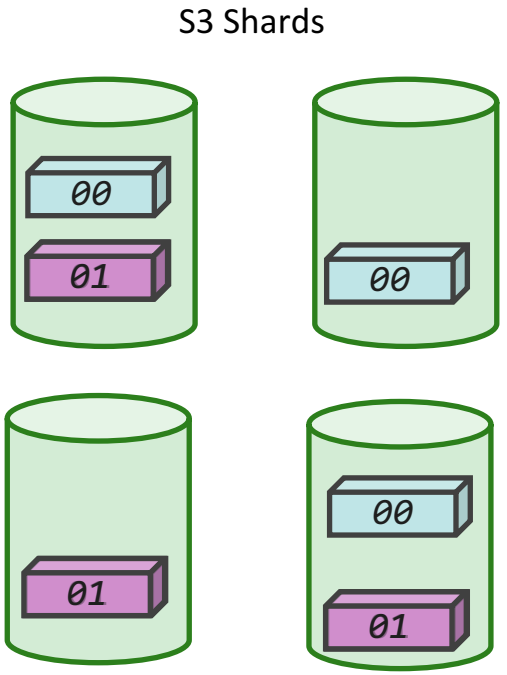

```
POST /work/part-01?uploads => UploadID

POST /work/part01?uploadId=UploadID&partNumber=01
POST /work/part01?uploadId=UploadID&partNumber=02
POST /work/part01?uploadId=UploadID&partNumber=03
```


HORTONWORKS

job manager selectively completes tasks' multipart uploads





(somehow list pending uploads of task 01)

```
POST /work/part-01?uploadId=UploadID
<CompleteMultipartUpload>
 <Part>
  <PartNumber>01</PartNumber><ETag>44a3</ETag>
  <PartNumber>02</PartNumber><ETag>29cb</ETag>
  <PartNumber>03</PartNumber><ETag>1aac</ETag>
 </Part>
</CompleteMultipartUpload>
```


HORTONWORKS

HORTONWORKS
POWERING THE FUTURE OF DATA

What else to rethink?

- Hierarchical directories to tree-walk
  ==> list & work with all files under a prefix;
- `seek() read()` sequences
  ==> HTTP-2 friendly scatter/gather IO
  `read((buffer1, 10 KB, 200 KB), (buffer2, 16 MB, 4 MB))`
- How to work with Eventually Consistent data?
- or: is everything just a K-V store with some search mechanisms?

SSD via SATA
SSD via NVMe/M.2
Future NVM technologies

```c
typedef struct record_struct {
  int field1, field2;
  long next;
} record;

int fd = open("/shared/dbase", O_CREAT | O_EXCL);
record* data = (record*) mmap(NULL, 8192,
  PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

(*data).field1 += 5;
data->field2 = data->field1;

msync(record, sizeof(record), MS_SYNC | MS_INVALIDATE);
```

```
typedef struct record_struct {
  int field1, field2;
  record_struct* next;
} record;

int fd = open("/shared/dbase");
record* data = (record*) pmem_map(fd);

// lock ?

(*data).field1 += 5;
data->field2 = data->field1;

// commit ?
```

NVM moves the commit problem into memory I/O

- How to split internal state into persistent and transient?
- When is data saved to NVM ($L1-$L3 cache flushed, sync in memory buffers, ...)
- How to co-ordinate shared R/W access over RDMA?
- How do we write apps for a world where rebooting doesn't reset our state?

Catch up: read "The Morning Paper" summaries of research

HORTONWORKS

Summary: Storage is moving in different directions

- Blobstore APIs address some scale issues, but don't match app expectations for file/dir behaviour; inefficient read/write model
- Non volatile memory is the other radical change
- Posix metaphor/API isn't suited to either —*what next?*
- SQL makes all this someone else's problem
  (leaving only O/R mapping, transaction isolation...)

Questions?

HORTONWORKS®
POWERING THE FUTURE OF DATA™