# What does rename() do?

**Steve Loughran**

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

# Define "Storage"?

HORTONWORKS®
POWERING THE FUTURE OF DATA™


FAT8
dBASE II & Lotus 1-2-3
int 21h
DS,DD
double side
double density
745-0
3M
DS,DD
500 Kb/50
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 Relational Model of Data for Large Shared Data Banks

E. F. Codd
*IBM Research Laboratory, San Jose, California*

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain opera-

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

### 1.2. Data Dependencies in Present Systems

# 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,

# Granularity of Locks and Degrees of Consistency in a Shared Data Base

J.N. Gray
R.A. Lorie
G.R. Putzolu
I.L. Traiger

IBM Research Laboratory
San Jose, California

ABSTRACT: In the first part of the paper the problem of choosing the granularity (size) of lockable objects is introduced and the related tradeoff between concurrency and overhead is discussed. A locking protocol which allows simultaneous locking at various granularities by different transactions is presented. It is based on the introduction of additional lock modes besides the conventional share mode and exclusive mode. A proof is given of the equivalence of this protocol to a conventional one.

In the second part of the paper the issue of consistency in a shared environment is analyzed. This discussion is motivated by the realization that some existing data base systems use automatic lock protocols which insure protection only from certain types of inconsistencies (for instance those arising from transaction backup), thereby automatically providing a limited degree of consistency. Four degrees of consistency are introduced. They

# File-System

## Directories and files

Posix with stream metaphor

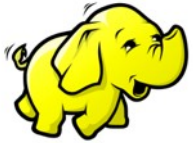
HORTONWORKS
POWERING THE FUTURE OF DATA

# Posix: hierarchical (distributed?) filesystems





`org.apache.hadoop.fs.FileSystem`



`hdfs`


`wasb`

`s3a`

`swift`

`adl`

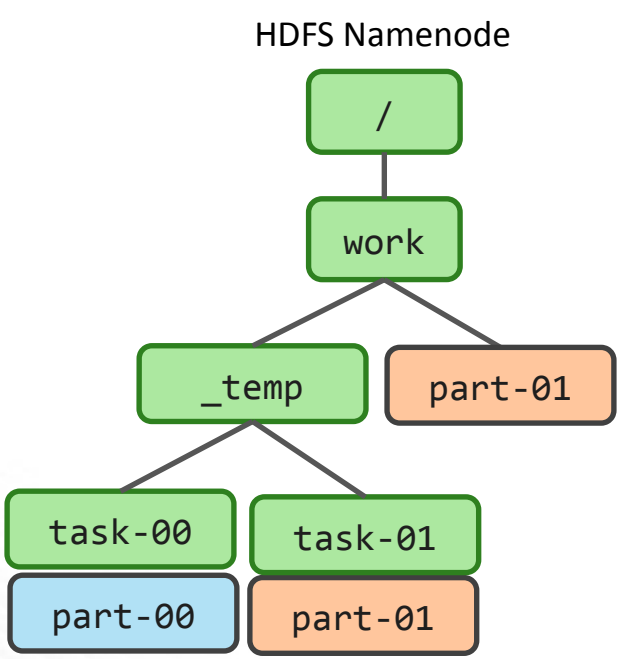
`gcs`

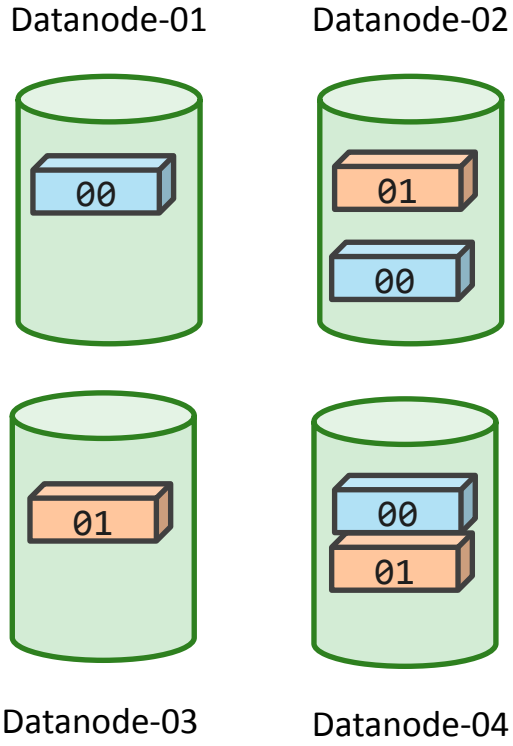



HORTONWORKS

```
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



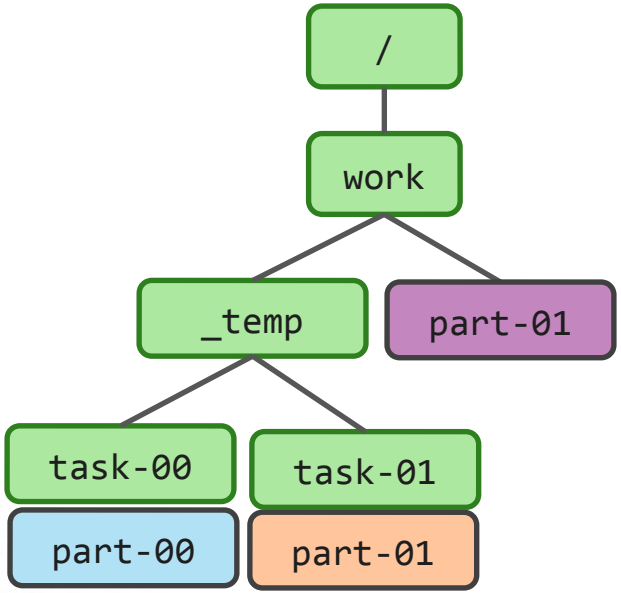



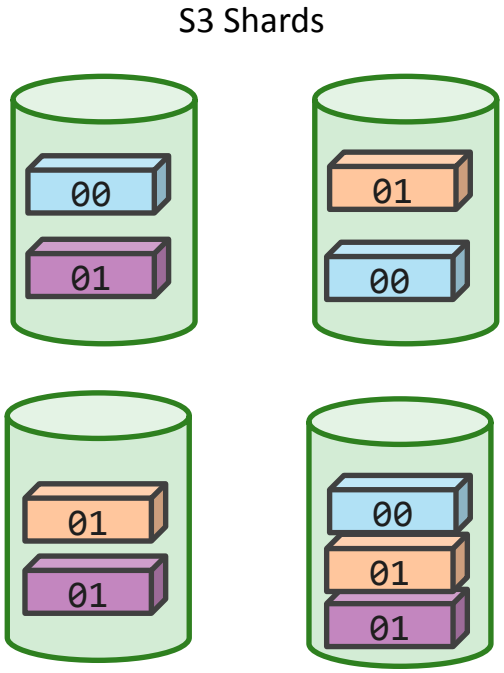

```
rename("/work/_temp/task00/*", "/work")
```




HORTONWORKS

# Amazon S3 doesn't have a rename()



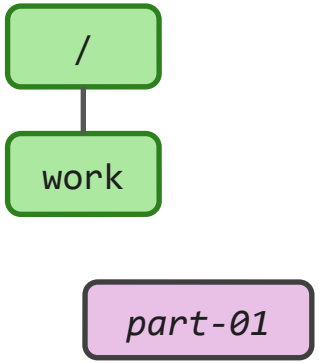



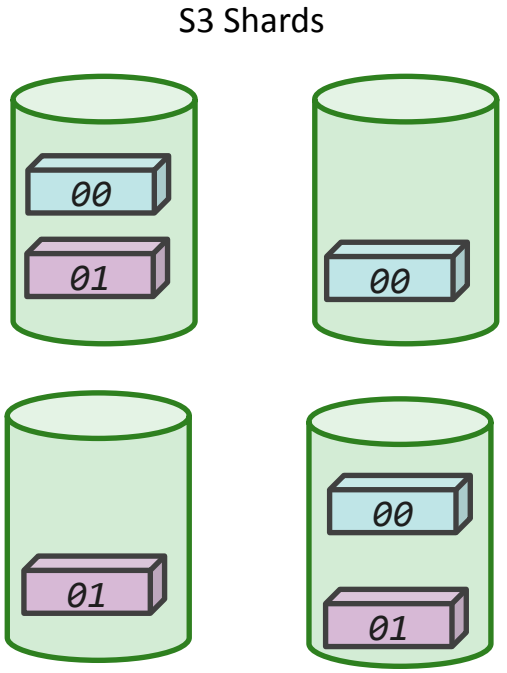

```
LIST /work/_temp/task-01/*
COPY /work/_temp/task-01/part-01 /work/part-01
DELETE /work/_temp/task-01/part-01
```




HORTONWORKS

# Fix: fundamentally rethink how we commit



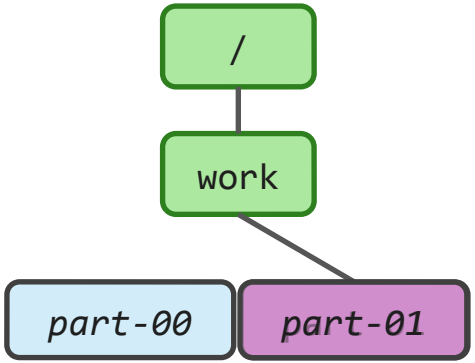



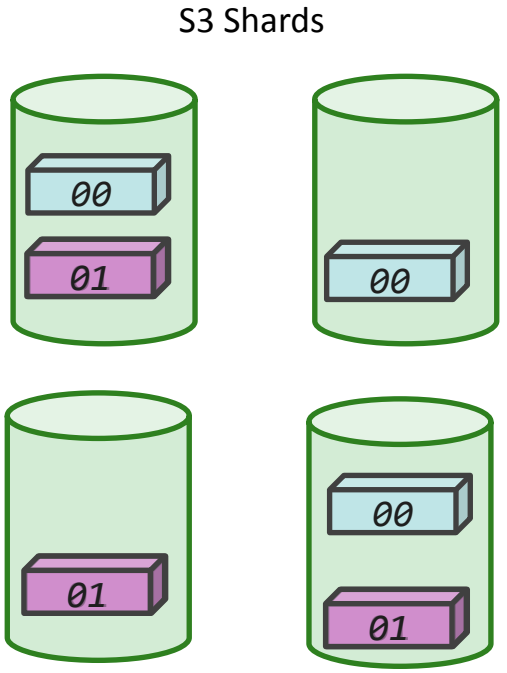

```
POST /work/part-01?uploads => UploadID

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

# S3A O(1) zero-rename commit demo!

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?

# Model #3: Storage as Memory

HORTONWORKS®
POWERING THE FUTURE OF DATA™


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);
```

# Non Volatile Memory

- SSD-backed RAM
- near-RAM-speed SSD
- Future memory stores
- RDMA access to NVM on other servers

What would a datacentre of NVM & RDMA access do?




HORTONWORKS

```
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™