DISTRIBUTED SYSTEMS, NFS

Shivaram Venkataraman

CS 537, Spring 2020

ADMINISTRIVIA

Project 5: Due today!

AEFIS feedback

Optional project

Discussion today: Optional project (short?)

AGENDA / LEARNING OUTCOMES

What are some basic building blocks for systems that span across machines?

How to design a distributed file system that can survive partial failures?

RECAP

DISTRIBUTED SYSTEMS: CHALLENGES

System failure: need to worry about partial failure

Communication failure: links unreliable

- bit errors
- packet loss
- node/link failure

RAW MESSAGES: UDP

UDP : User Datagram Protocol

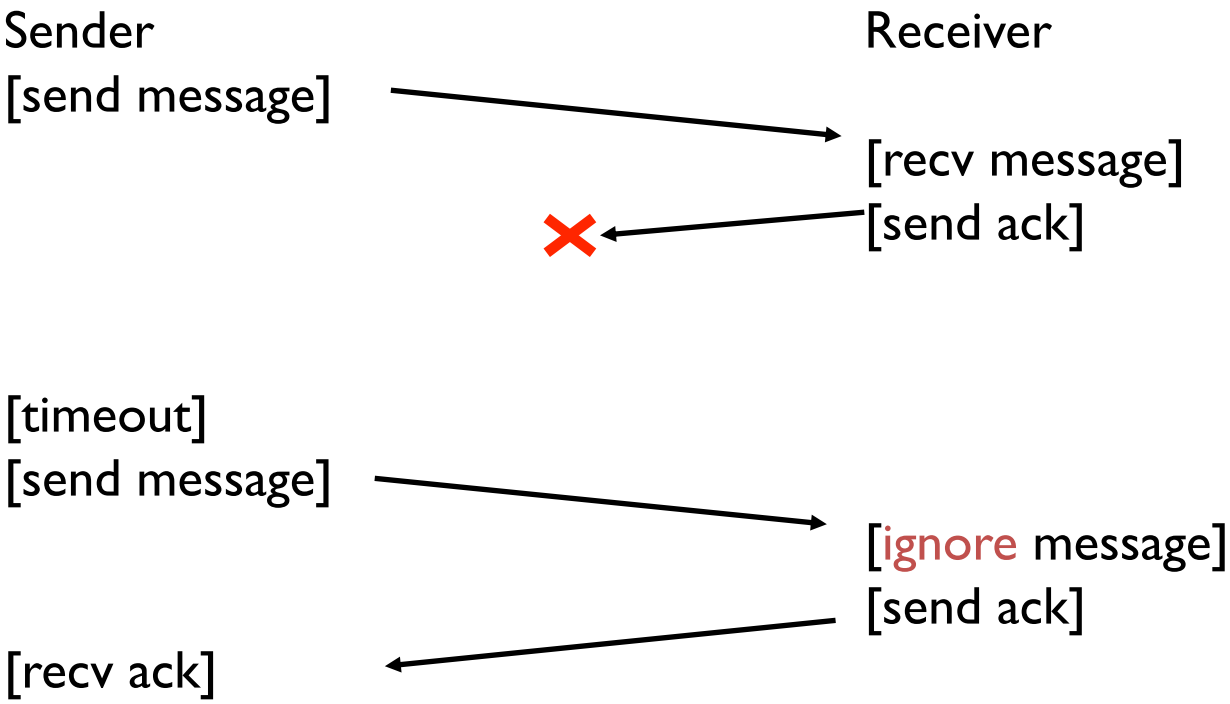
API:

- reads and writes over socket file descriptors
- messages sent from/to ports to target a process on machine

Provide minimal reliability features:

- messages may be lost
- messages may be reordered
- messages may be duplicated
- only protection: checksums to ensure data not corrupted

TCP: ACKS, TIMEOUTS



TCP: SEQUENCE NUMBERS

Sequence numbers

- senders gives each message an increasing unique seq number
- receiver knows it has seen all messages before N

Suppose message K is received.

- if K <= N, Msg K is already delivered, ignore it
- if K = N + 1, first time seeing this message
- if K > N + 1 ?

RPC

Remote **P**rocedure **C**all

What could be easier than calling a function?

Approach: create wrappers so calling a function on another machine feels just like calling a local function!

RPC

Machine A

```
int main(…) {
        int x = foo(”hello”);
}

int foo(char *msg) {
        send msg to B
        recv msg from B
}
```

client wrapper

Machine B

```
int foo(char *msg) {
        …
}

void foo_listener() {
        while(1) {
                recv, call foo
        }
}
```

server wrapper

RPC TOOLS

RPC packages help with two components

(1) Runtime library

- Thread pool
- Socket listeners call functions on server

(2) Stub generation

- Create wrappers automatically
- Many tools available (rpcgen, thrift, protobufs)

WRAPPER GENERATION

Wrappers must do conversions:

- client arguments to message
- message to server arguments
- convert server return value to message
- convert message to client return value

Need uniform endianness (wrappers do this)

Conversion is called marshaling/unmarshaling, or serializing/deserializing

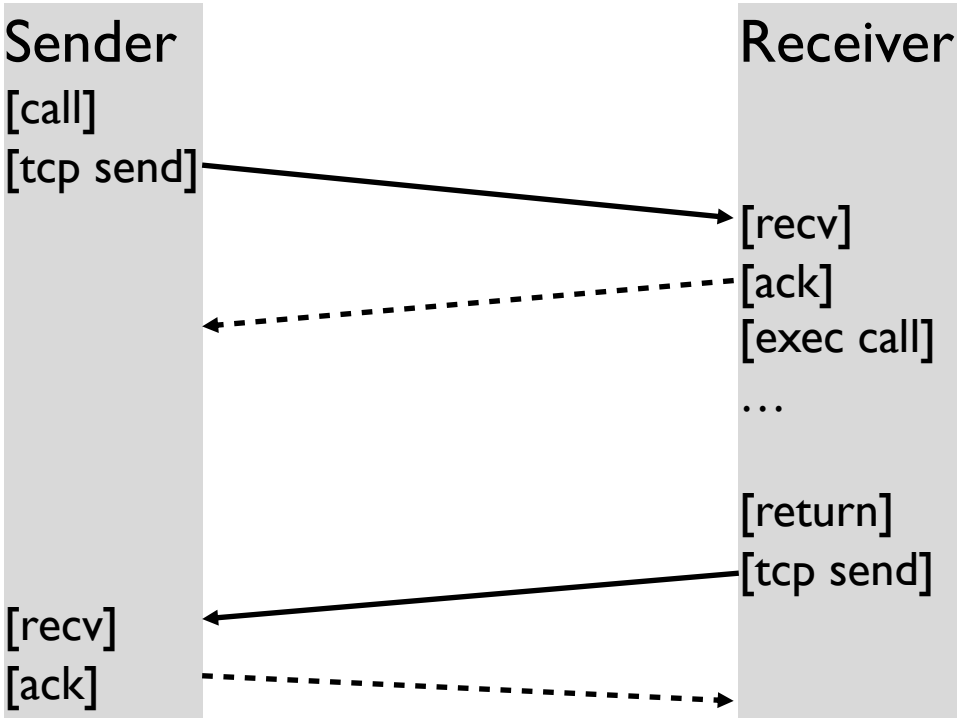
WRAPPER GENERATION: POINTERS

Why are pointers problematic?

Address passed from client not valid on server

Solutions? Smart RPC package: follow pointers and copy data

RPC OVER TCP?


Sender
[call]
[tcp send]
[recv]
[ack]
Receiver
[recv]
[ack]
[exec call]
…
[return]
[tcp send]

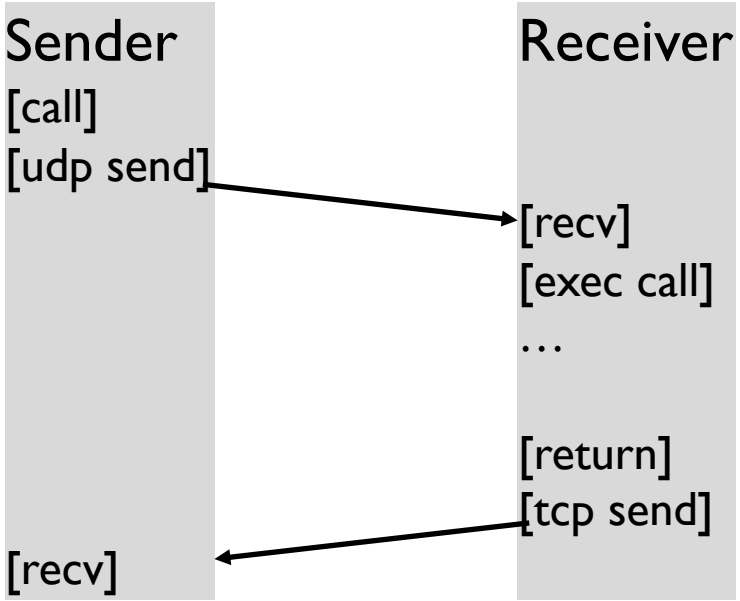
RPC OVER UDP

Strategy: use function return as implicit ACK

Piggybacking technique

What if function takes a long time?

then send a separate ACK



QUIZ 30

https://tinyurl.com/cs537-sp20-quiz30

Who can initiate a Remote Procedure Call?

Which are the functions of client wrapper generator?

Which are the functions of server wrapper generator?

DISTRIBUTED FILE SYSTEMS

Local FS: processes on same machine access shared files

Network FS: processes on different machines access shared files in same way

GOALS FOR DISTRIBUTED FILE SYSTEMS

Transparent access

- can't tell accesses are over the network
- normal UNIX semantics

Fast + simple crash recovery: both clients and file server may crash

Reasonable performance?

NETWORK FILE SYSTEM: NFS

NFS: more of a protocol than a particular file system

Many companies have implemented NFS: Oracle/Sun, NetApp, EMC, IBM

We're looking at NFSv2. NFSv4 has many changes

Why look at an older protocol? Simpler, focused goals

OVERVIEW

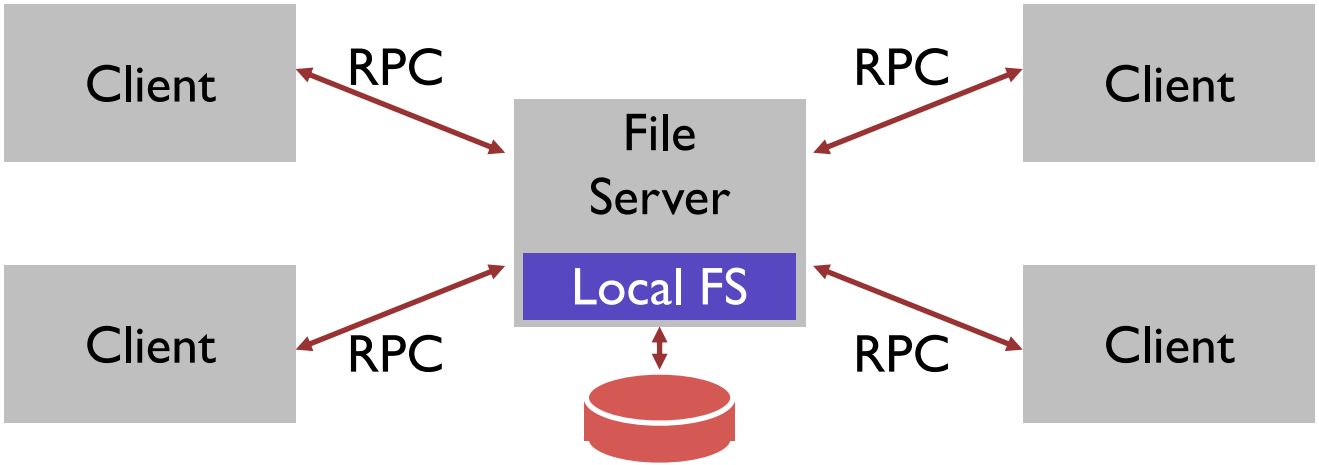
Architecture

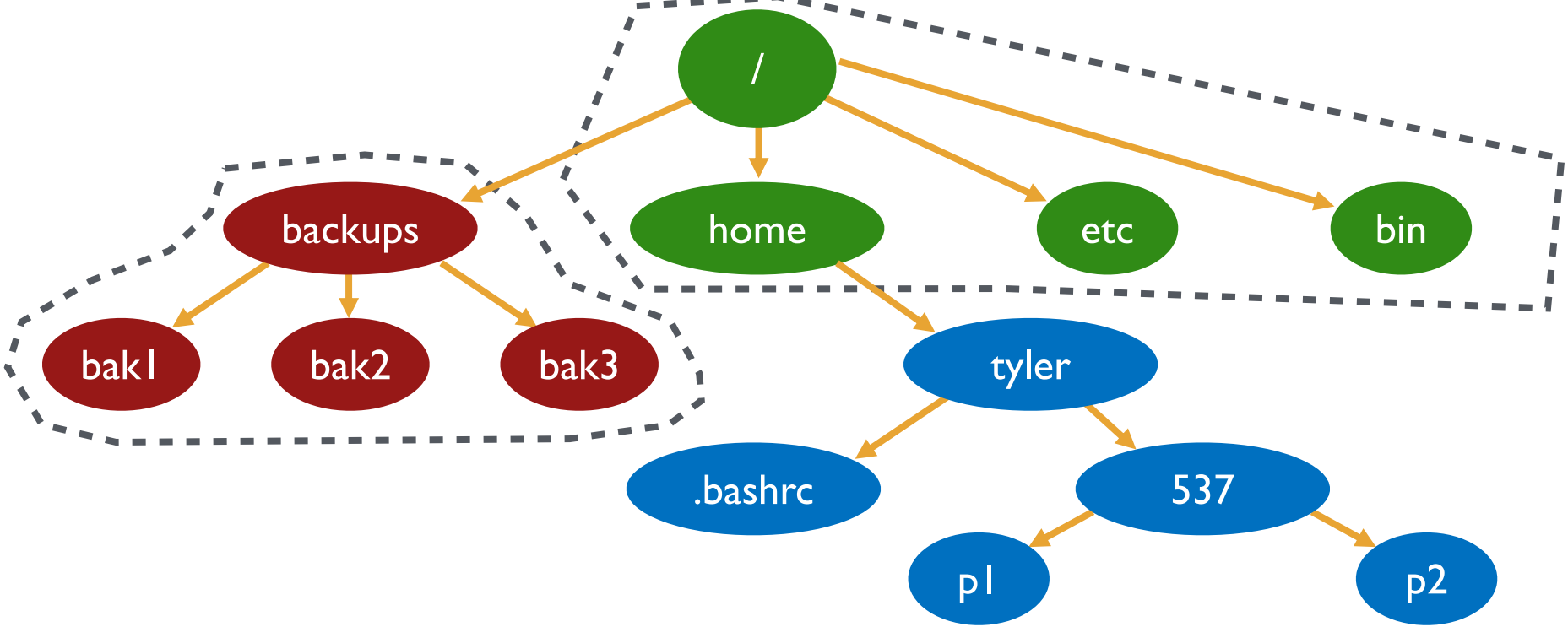
Network API

Write Buffering

Cache

NFS ARCHITECTURE

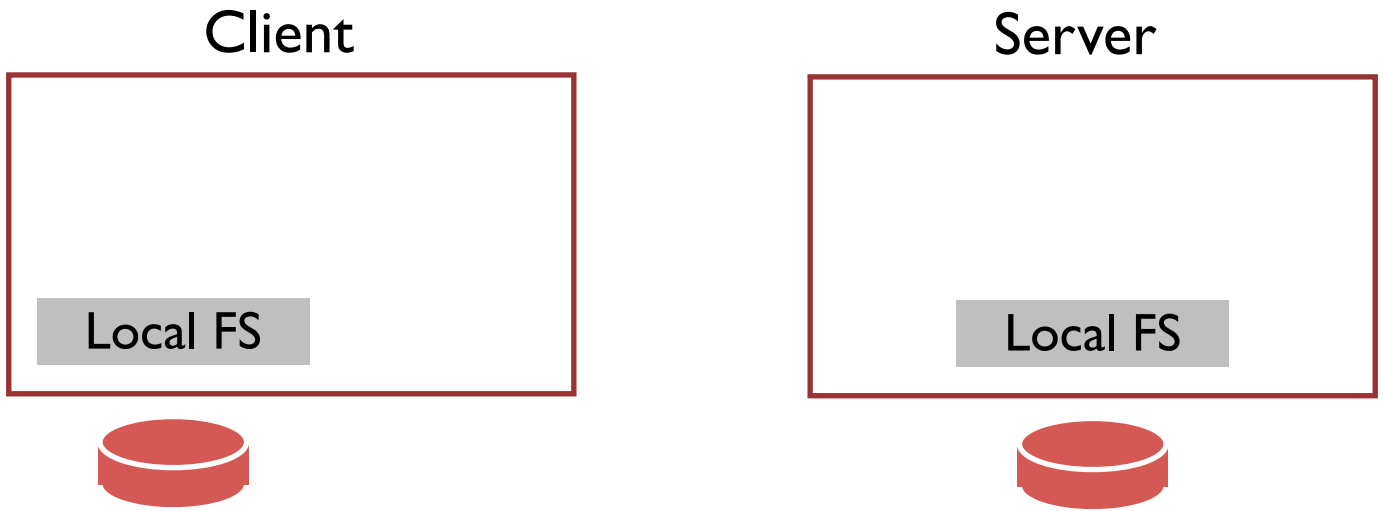




/dev/sda1 **on** /

/dev/sdb1 **on** /backups

NFS **on** /home


Client
Local FS
Server
Local FS

OVERVIEW

~~Architecture~~

Network API

Write Buffering

Cache

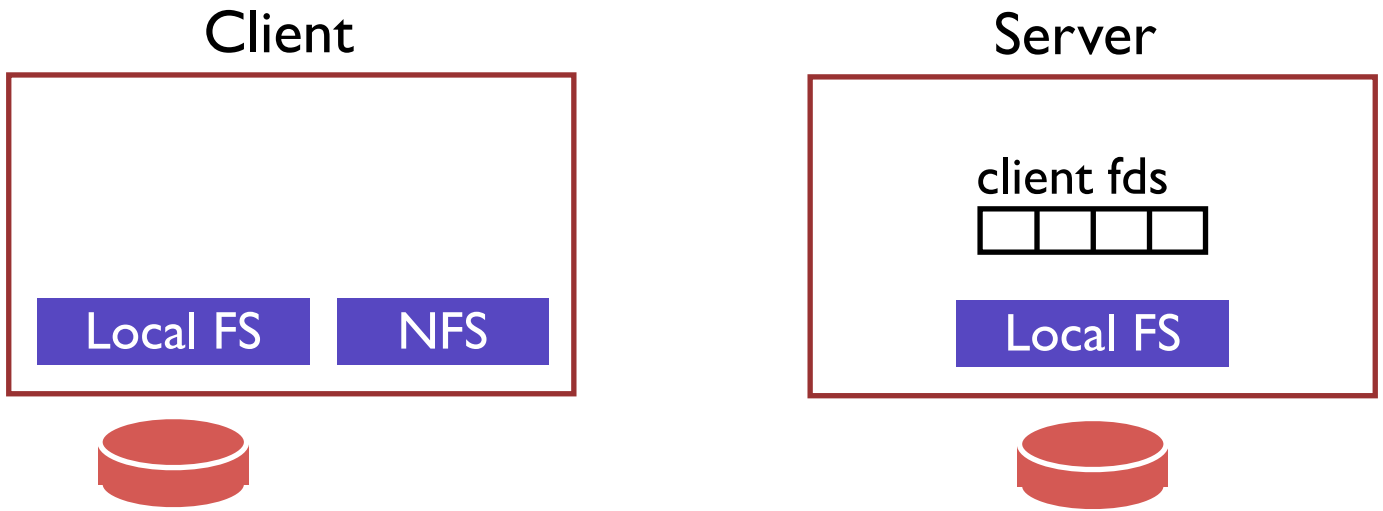
STRATEGY 1

Attempt: Wrap regular UNIX system calls using RPC

open() on client calls open() on server
open() on server returns fd back to client

read(fd) on client calls read(fd) on server
read(fd) on server returns data back to client

FILE DESCRIPTORS



Examples

- open
- read

STRATEGY 1: WHAT ABOUT CRASHES

```
int fd = open(“foo”, O_RDONLY);
read(fd, buf, MAX);
read(fd, buf, MAX);
…
read(fd, buf, MAX);
```

Server crash!

POTENTIAL SOLUTIONS

1. Run some crash recovery protocol upon reboot
   - Complex

2. Persist fds on server disk.
   - Slow
   - What if client crashes? When can fds be garbage collected?

STRATEGY 2: PUT ALL INFO IN REQUESTS

Use “stateless” protocol!

- server maintains no state about clients
- server still keeps other state, of course

STRATEGY 2: PUT ALL INFO IN REQUESTS

"Stateless" protocol: server maintains no state about clients

Need API change. One possibility:

pread(char *path, buf, size, offset);

pwrite(char *path, buf, size, offset);

Specify path and offset each time. Server need not remember anything from clients.

Pros?

Cons?

STRATEGY 3: FILE HANDLES

```
fh = open(char *path);
pread(fh, buf, size, offset);
pwrite(fh, buf, size, offset);
```

File Handle = <volume ID, inode #, **generation #**>

Opaque to client (client should not interpret internals)

```
NFSPROC_GETATTR
  expects: file handle
  returns: attributes
NFSPROC_SETATTR
  expects: file handle, attributes
  returns: nothing
NFSPROC_LOOKUP
  expects: directory file handle, name of file/directory to look up
  returns: file handle
NFSPROC_READ
  expects: file handle, offset, count
  returns: data, attributes
NFSPROC_WRITE
  expects: file handle, offset, count, data
  returns: attributes
NFSPROC_CREATE
  expects: directory file handle, name of file, attributes
  returns: nothing
NFSPROC_REMOVE
  expects: directory file handle, name of file to be removed
  returns: nothing
NFSPROC_MKDIR
  expects: directory file handle, name of directory, attributes
  returns: file handle
NFSPROC_RMDIR
  expects: directory file handle, name of directory to be removed
  returns: nothing
NFSPROC_READDIR
  expects: directory handle, count of bytes to read, cookie
  returns: directory entries, cookie (to get more entries)
```

Client	Server
fd = open("/foo", ...);	
Send LOOKUP (rootdir FH, "foo")	
	Receive LOOKUP request look for "foo" in root dir return foo's FH + attributes
Receive LOOKUP reply allocate file desc in open file table store foo's FH in table store current file position (0) return file descriptor to application	

NEXT STEPS

Next class: More NFS

P5 is due TODAY!

AEFIS feedback

Optional project discussion