DISTRIBUTED SYSTEMS

Shivaram Venkataraman
CS 537, Spring 2023
Project 6 grades
Project 7
Project 8 – Extra credit (4%)

Midterm 3 conflicts

AEFIS feedback
What are some basic building blocks for systems that span across machines?
RECAP
SSD OPERATIONS

Read a page: Retrieve contents of entire page (e.g., 4 KB)
   - Cost: 25—75 microseconds
   - Independent of page number, prior request offsets

Erase a block: Resets each page in the block to all 1s
   - Cost: 1.5 to 4.5 milliseconds
   - Much more expensive than reading!
   - Allows each page to be written

Program (i.e., write) a page: Change selected 1s to 0s
   - Cost is 200 to 1400 microseconds
   - Faster than erasing a block, but slower than reading a page
FTL: DIRECT MAPPING

Cons?

- Write amplification
- No wear-leveling
### FTL: LOG-BASED MAPPING

**Idea:** Treat the physical blocks like a log

<table>
<thead>
<tr>
<th>Table: 100</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Page:</strong></td>
<td>00 01 02 03 04 05 06 07 08 09 10 11</td>
</tr>
<tr>
<td><strong>Content:</strong></td>
<td>a1</td>
</tr>
<tr>
<td><strong>State:</strong></td>
<td>V E E E i i i i i i i i</td>
</tr>
</tbody>
</table>
GARBAGE COLLECTION

Steps:

1. Read all pages in physical block
2. Write out the alive entries to the end of the log
3. Erase block (freeing it for later use)
## SSD VS HDD PERFORMANCE

<table>
<thead>
<tr>
<th>Device</th>
<th>Random Reads (MB/s)</th>
<th>Random Writes (MB/s)</th>
<th>Sequential Reads (MB/s)</th>
<th>Sequential Writes (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung 840 Pro SSD</td>
<td>103</td>
<td>287</td>
<td>421</td>
<td>384</td>
</tr>
<tr>
<td>Seagate 600 SSD</td>
<td>84</td>
<td>252</td>
<td>424</td>
<td>374</td>
</tr>
<tr>
<td>Intel SSD 335 SSD</td>
<td>39</td>
<td>222</td>
<td>344</td>
<td>354</td>
</tr>
<tr>
<td>Seagate Savvio 15K.3 HDD</td>
<td>2</td>
<td>2</td>
<td>223</td>
<td>223</td>
</tr>
</tbody>
</table>
SSD VS HDD COST

1TB ~ $150 on average
~15 cents / GB

~1.5 cents / GB
Managing I/O devices is a significant part of OS!
Disk drives: storage media with specific geometry
SSDs: Pages, Blocks

Filesystems: OS provided API to access disk

Simple FS: FS layout with SB, Bitmaps, Inodes, Datablocks
FFS: Split simple FS into groups. Key idea: put inode, data close to each other
LFS: Puts data where it’s fastest to write, hope future reads cached in memory

https://www.eecs.harvard.edu/~margo/papers/usenix95-lfs/supplement/

FSCK, Journaling
DISTRIBUTED SYSTEMS
A distributed system is one where a machine I’ve never heard of can cause my program to fail.

— Leslie Lamport

Definition: More than one machine working together to solve a problem

Examples:
- client/server: web server and web client
- cluster: page rank computation
WHY GO DISTRIBUTED?
WHY GO DISTRIBUTED?

More computing power

More storage capacity

Fault tolerance

Data sharing
NEW CHALLENGES

System failure: need to worry about partial failure

Communication failure: links unreliable
  - bit errors
  - packet loss
  - node/link failure
Raw messages: UDP
Reliable messages: TCP
Remote procedure call: RPC
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
RAW MESSAGES: UDP

Advantages
  – Lightweight
  – Some applications make better reliability decisions themselves (e.g., video conferencing programs)

Disadvantages
  – More difficult to write applications correctly
NOT A QUIZ?

Course feedback: https://aefis.wisc.edu
TCP: Transmission Control Protocol

Using software to build

reliable logical connections over unreliable physical connections
Technique #1: ACK

Sender
[send message]
[recv ack]

Receiver
[recv message]
[send ack]

Ack: Sender knows message was received
What to do about message loss?
TECHNIQUE #2: TIMEOUT

Sender
[send message]
[start timer]

... waiting for ack ...

[timer goes off]
[send message]

[recv ack]

Receiver

[recv message]
[send ack]
How long to wait?

Too long?
  – System feels unresponsive

Too short?
  – Messages needlessly re-sent
  – Messages may have been dropped due to overloaded server. Resending makes overload worse!
LOST ACK PROBLEM

Sender
(send message)
[timeout]
(send message)
(recv ack)

Receiver
(recv message)
(send ack)
[ignore message]
[send ack]
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 ?
TCP: Transmission Control Protocol

Most popular protocol based on seq nums
Buffers messages so arrive in order
Timeouts are adaptive
COMMUNICATIONS OVERVIEW

Raw messages: UDP

Reliable messages: TCP

Remote procedure call: RPC
Remote Procedure Call

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!
int main(…) {
    int x = foo("hello");
}

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

Machine A

Machine B

int foo(char *msg) {
    ...
}

void foo_listener() {
    while(1) {
        recv, call foo
    }
}
int main(…) {
    int x = foo("hello");
}

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

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

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
NEXT STEPS

Distributed Filesystems