Maybe sticky sessions weren't a bad idea
Trends
Tradeoffs
Nuance
Status Quo

1. Write my PHP / Java / Rails
2. Connect it to an RDBMS

Buy lots of web servers & really big databases
Storing state **in** an application
• Just what the heck is state anyway?
• How do we store it today
• Some history
• Ok, but why?
• Caveats
• More information
State of the state

• Data vs. behavior

• A coupling with time and space
Time

- "happens before"
- memory barriers
- synchronization primitives
Space

- threads
- processes
- servers
State is a lens

def fact(i: Int) = { return i * fact(i - 1) }
State is a lens

def dolt(i: Int) = { log("you sent $i") }
State-less vs. State-ful

- Store data in a database
- Ship data => behavior
State-less applications

- Deployed behind a load balancer
- Most common CRUD applications
my engineers wrote this 😱
This thing "works"
don't care who's at fault
State-ful
State-ful applications

- Store data with the behavior
- Data does not move when worked on
Sticky sessions

• Sticky sessions are an http concept that glues a session to a server

• Keeps sessions in a single place
Sticky session
How

- DHT
- Non-DHT
Major motivators

• Performance
• Correctness
• Programmer ergonomics
• Resilience
Performance

<table>
<thead>
<tr>
<th>Component</th>
<th>Delay</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU cache</td>
<td>1 ns</td>
<td>1 s</td>
</tr>
<tr>
<td>Main memory</td>
<td>120 ns</td>
<td>2 min</td>
</tr>
<tr>
<td>Disk</td>
<td>50-150 μs</td>
<td>14 hours</td>
</tr>
<tr>
<td>Network</td>
<td>500 μs</td>
<td>6 days</td>
</tr>
<tr>
<td>Component</td>
<td>Time</td>
<td>Duration</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>CPU cache</td>
<td>1ns</td>
<td>1s</td>
</tr>
<tr>
<td>Main memory</td>
<td>120ns</td>
<td>2 min</td>
</tr>
<tr>
<td>Disk</td>
<td>50-150μs</td>
<td>14 hours</td>
</tr>
<tr>
<td>Network</td>
<td>500 μs</td>
<td>6 days</td>
</tr>
</tbody>
</table>
Performance

<table>
<thead>
<tr>
<th>Component</th>
<th>Delay</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU cache</td>
<td>1 ns</td>
<td>1 s</td>
</tr>
<tr>
<td>Main memory</td>
<td>120 ns</td>
<td>2 min</td>
</tr>
<tr>
<td>Disk</td>
<td>50-150 μs</td>
<td>14 hours</td>
</tr>
<tr>
<td>Network</td>
<td>500 μs</td>
<td>6 days</td>
</tr>
</tbody>
</table>
Correctness

• Data & behavior co-existing means we can reason about safely changing state
Programmer Ergonomics
Resilience

- Classes of error around txns
- Connection pools
- Failure can be handled
How do we do this...
Runtimes

- Long lived processes
- Threading model
- Control over memory
Frameworks

• Supports some way to make remote calls
• Treats concurrency as a first class citizen
• A concept of clustering
Some examples
The downside
Serialization

• De-serialize the future

• De-serialize the past
Thundering herds

- Startup time (deployment)
- Rebalance performance

But it worked on my computer?!?
Delicious memory

- Unbounded, in-memory data structures

But it worked on my computer?!?
Copying Inspiration

• Any distributed database: Riak, Cassandra, Dynamo
• Akka distributed data / cluster sharding
• Orleans
• Unison
• CRDTs
• Just what the heck is state anyway?
• How do we store it today
• Some history
• Ok, but why?
• Caveats
• More information
Thank you.