Declarative,
Secure,
Convergent
Edge Computation

Christopher Meiklejohn
Université catholique de Louvain, Belgium
Internet of Things
Internet of Things

but, more generally…
Edge Computation

• Logical extremes
  Pushing both computation and data to the logical extremes of the network
Edge Computation

• Logical extremes
  Pushing both computation and data to the logical extremes of the network

• Arbitrary computation
  Support arbitrary computations regardless of location of data in the network
Edge Computation

• Logical extremes
  Pushing both computation and data to the logical extremes of the network

• Arbitrary computation
  Support arbitrary computations regardless of location of data in the network

• Self-organizing, resilient
  Directed diffusion, Cornell circa-1990; self-organizing systems that coordinate to complete computations
Example Application
Hospital Refrigerators
Hospital Refrigerators
Typical Topology
Hospital Refrigerators
Ideal Execution
Problem
Connectivity
Solution
Local Decisions
Local Decisions

• Not new for backup (80s-90s)
  Backup communication mechanisms for critical systems; POTS backup for ISDN, etc.
Local Decisions

• Not new for backup (80s-90s)
  Backup communication mechanisms for critical systems; POTS backup for ISDN, etc.

• Not new for storage (90s-00s)
  EMC’s “phone home” via POTS when disks failed in NAS devices to signal for replacement unit
Solution
Transitive Dissemination
Problem
State Transmission
Internet
Solution
Aggregate Dissemination
Local Computation

• Reduce state transmission
  Perform some local computation to reduce transmitted state on the wire
Local Computation

- Reduce state transmission
  Perform some local computation to reduce transmitted state on the wire

- Make local decisions
  Make decisions based on results of local computation
Databases

Consistency Models
Databases
Strong Consistency
I won’t diagram the **Paxos** protocol
Databases

Eventual Consistency
Read

C₁

C₂

R₁

R₂

R₃
Eventual Consistency
As The Model
Clients Own Their Data
Computations
Mergeability & Provenance
Example Application
Preliminary Results
Preliminary Results

• Conflict-Free Replicated Data Types
  Distributed data structures designed for convergence
  [Shapiro et al., 2011]
Preliminary Results

• Conflict-Free Replicated Data Types
  Distributed data structures designed for convergence
  [Shapiro et al., 2011]

• Lattice Processing
  Make decisions based on results of local computation
  [Meiklejohn & Van Roy, 2015]
Preliminary Results

• Conflict-Free Replicated Data Types
  Distributed data structures designed for convergence
  [Shapiro et al., 2011]

• Lattice Processing
  Make decisions based on results of local computation
  [Meiklejohn & Van Roy, 2015]

• Selective Hearing
  Scalable, epidemic broadcast based runtime system
  [Meiklejohn & Van Roy, 2015/2016]
Conflict-Free Replicated Data Types

- Collection of types
  Sets, counters, registers, flags, maps
Conflict-Free Replicated Data Types

- Collection of types
  Sets, counters, registers, flags, maps

- Strong Eventual Consistency (SEC)
  Objects that receive the same updates, regardless of order, will reach equivalent state
\textit{add(1)}
\(R_A\):
- add(1)

\(R_B\):
- \((1, (a), (c))\)

\(R_C\):
- add(1)
- remove(1)
- \((1, (c), (c))\)
Lattice Processing

- Distributed dataflow
  Declarative, functional programming model
Lattice Processing

- Distributed dataflow
  Declarative, functional programming model

- Convergent data structures
  Data abstraction is the CRDT
Lattice Processing

- Distributed dataflow
  Declarative, functional programming model

- Convergent data structures
  Data abstraction is the CRDT

- Enables composition
  Composition preserves SEC
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1, 2, 3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1,2,3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1,2,3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1, 2, 3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1,2,3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
Delta-based Dissemination

- Delta-state based CRDTs
  Reduces state transmission for clients
Delta-based Dissemination

- Delta-state based CRDTs
  Reduces state transmission for clients

- Operate locally
  Objects are mutated locally; deltas buffered locally and periodically gossiped
Delta-based Dissemination

• Delta-state based CRDTs
  Reduces state transmission for clients

• Operate locally
  Objects are mutated locally; deltas buffered locally and periodically gossiped

• Only fixed number of clients
  Clients resort to full state synchronization when they’ve been partitioned too long
Selective Hearing

- Epidemic broadcast protocol
  Runtime system for application state & scope
Selective Hearing

• Epidemic broadcast protocol
  Runtime system for application state & scope

• Peer-to-peer dissemination
  Pairwise synchronization between peers without a central coordinator
Selective Hearing

• Epidemic broadcast protocol
  Runtime system for application state & scope

• Peer-to-peer dissemination
  Pairwise synchronization between peers
  without a central coordinator

• No ordering guarantees on messages
  Programming model can tolerate message
  reordering and duplication
Membership Overlay
Membership Overlay
Broadcast Overlay
Membership Overlay
What can we build?
Advertisement Counter
Advertisement Counter

• Mobile game platform selling advertisement space
  Advertisements are paid according to a minimum number of impressions
Advertisement Counter

• Mobile game platform selling advertisement space
  Advertisements are paid according to a minimum number of impressions

• Clients will go offline
  Clients have limited connectivity and the system still needs to make progress while clients are offline
Ads
RoviovAdv
Counterv1
RoviovAdv
Counterv2
RiotvAdv
Counterv1
RiotvAdv
Counterv2
Contracts
Ads
Contracts
With
Contracts
RoviooAds
RoviooAds
Counterv1
RoviooAds
Counterv2
RiotAds
RiotAds
Counterv1
RiotAds
Counterv2
Remove
ClientSide,vSingleCopyatoClient
LaspoOperation
User-MaintainedoCRDT
Lasp-MaintainedoCRDT
Ads
Product
Ads Contracts
Filter
Ads With Contracts
Increment
Read
121
Advertisement Counter

- Completely monotonic
  Disabling advertisements and contracts are all modeled through monotonic state growth
Advertisement Counter

- Completely monotonic
  Disabling advertisements and contracts are all modeled through monotonic state growth

- Arbitrary distribution
  Use of convergent data structures allows computational graph to be arbitrarily distributed
Advertisement Counter

• Completely monotonic
  Disabling advertisements and contracts are all modeled through monotonic state growth

• Arbitrary distribution
  Use of convergent data structures allows computational graph to be arbitrarily distributed

• Divergence
  Divergence is a factor of synchronization period
Advertisement Counter

• “Servers” as peers to “clients”
  Servers are peers to clients that perform additional computation
Advertisement Counter

• “Servers” as peers to “clients”
  Servers are peers to clients that perform additional computation

  • Any node can disable an advertisement under this model given enough information
Advertisement Counter

• “Servers” as peers to “clients”
  Servers are peers to clients that perform additional computation
  • Any node can disable an advertisement under this model given enough information

• “Servers” as trusted nodes
  Serve as a location for performing “exactly once” side-effects
Advertisement Counter

• “Servers” as peers to “clients”
  Servers are peers to clients that perform additional computation
  • Any node can disable an advertisement under this model given enough information

• “Servers” as trusted nodes
  Serve as a location for performing “exactly once” side-effects
  • Billing customers must be done at a central point by a trusted node in the system
We’ve built up from zero synchronization
We’ve built up from zero synchronization

Instead of working to remove synchronization
Challenges
Looking Ahead
Causality
State Explosion
Set

$l$

$(l, (c), (\{\}))$

Counter

$1$

$((c, 1)), {}$
Set

Counter

{(l, (c), (c))} → (l)

{(l, (c), (c))} → (l)

{{(c, l), {}}} → 1

{{(c, l), {}}} → 1

{(a, (c), (c))} → (l)

{(a, (c), (c))} → (l)

{(l, (c), (c))} → 0

{(l, (c), (c))} → 0

{(c, (c), l, (c))} → 1

{(c, (c), l, (c))} → 1

{(c, (c), l, (c))} → 0

{(c, (c), l, (c))} → 0
Security
Computing at the Edge
Computations
Expressiveness
How restrictive is a programming model where operations must be associative, commutative, and idempotent?
How do I learn more?
Publications

• “Lasp: A Language for Distributed, Coordination-Free Programming”
  ACM SIGPLAN PPDP 2015

• “Selective Hearing: An Approach to Distributed, Eventually Consistent Edge Computation”
  IEEE W-PSDS 2015

• “The Implementation and Use of a Generic Dataflow Behaviour in Erlang”
  ACM SIGPLAN Erlang Workshop ’15

• “Lasp: A Language for Distributed, Eventually Consistent Computations with CRDTs”
  PaPoC 2015

• “Declarative, Sliding Window Aggregations for Computations at the Edge”
  IEEE EdgeCom 2016
Three independently successful techniques.
Can we combine them into a **cohesive** programming environment for distributed programming?
Thanks!

Christopher Meiklejohn
@cmeik
http://www.lasp-lang.org