

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro, Neil Conway, Joseph M. Hellerstein, William R. Marczak

UC Berkeley

November 19, 2010

Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

1 Background

- The CALM Conjecture
- Introducing Bloom
- Writing distributed programs in Bloom

2 Shopping Carts

- Cart client
- A key/value store
- A destructive cart
- A disorderly cart

Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A disorderly cart

1 Background

- The CALM Conjecture
- Introducing Bloom
- Writing distributed programs in Bloom

2 Shopping Carts

- Cart client
- A key/value store
- A destructive cart
- A disorderly cart

Some insights from logic programming

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A disorderly cart

- Datalog rules (function-free Horn clauses) can be evaluated in any order and produce the same result
 - The logical consequence relation is monotonic
- We can evaluate (monotonic) Datalog in the network without any coordination
 - in spite of delay and reordering
 - e.g., “Pipelined Semi-naive evaluation” of Loo et al

Some insights from logic programming

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

- The logical consequence relation is monotonic *until* we admit negation or aggregation into the language
 - Both entail universal quantification over (possibly distributed) state
 - Intuition: you can only (safely) assert a predicate over *all* elements in a set when the set is “complete;” otherwise you may need to retract conclusions
- When we do, it imposes a partial order of evaluation on the program
 - Stratification order
- Distributed stratified evaluation is nontrivial
 - Need to do distributed coordination.

CALM: Consistency and Logical Monotonicity

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A disorderly cart

Conjecture: Eventually consistent \Leftrightarrow monotonic

- \Leftarrow is easy
 - We can express purely monotonic programs in Datalog, and all Datalog programs are eventually consistent
- \Rightarrow will take some proving
 - NoSQL \equiv Datalog ?

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro, Neil Conway, Joseph M. Hellerstein, William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A disorderly cart

CALM Analysis: leverage dataflow analyses from Datalog literature to:

- Identify “points of order” in distributed programs
 - where coordination may be required to ensure that all orderings of inputs produce the same output
 - a first crack: syntactic nonmonotonicity (negation, aggregation, deletion) in the dataflow
- Ensure that the program is well-formed
 - Free from contradictions
 - After composition, all dataflow components are connected

Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A disorderly cart

1 Background

- The CALM Conjecture
- **Introducing Bloom**
- Writing distributed programs in Bloom

2 Shopping Carts

- Cart client
- A key/value store
- A destructive cart
- A disorderly cart

Introducing Bloom

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A disorderly cart

BUD: Bloom Under Development

- Ruby internal DSL
- Semantics based on Dedalus (\neg Datalog with temporal extensions)
- Set-comprehension style of programming

A BUD rule

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

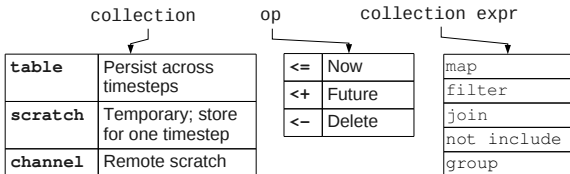
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

```
multicast < + join([message, members]).map |mes, mem|  
[members.address, message.id, message.payload]
```



Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

1 Background

- The CALM Conjecture
- Introducing Bloom
- Writing distributed programs in Bloom

2 Shopping Carts

- Cart client
- A key/value store
- A destructive cart
- A disorderly cart

Abstract Interfaces and Declarations

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

```
module DeliveryProtocol
  def state
    super
    interface input, :pipe_in,
      ['dst', 'src', 'ident'], ['payload']
    interface output, :pipe_out,
      ['dst', 'src', 'ident'], ['payload']
    channel :pipe_chan,
      ['@dst', 'src', 'ident'], ['payload']
  end
end
```

Concrete Implementations

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

A best-effort delivery program

```
module BestEffortDelivery
  include DeliveryProtocol
  declare
  def snd
    pipe_chan <~ pipe_in.map{|p| p }
  end

  declare
  def done
    pipe_out <= pipe_in.map{|p| p }
  end
end
```

Concrete Implementations

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

Reliable (ack'd) delivery extends best-effort delivery, overriding the “done” declaration.

```
module ReliableDelivery
  include BestEffortDelivery
  def state
    super
    table :pipe, ['dst', 'src', 'ident'], ['payload']
    channel :ack, ['@src', 'dst', 'ident']
  end

  declare
  def remember
    pipe <= pipe.in.map {|p| p }
  end

  declare
  def done
    ack <~ pipe_chan.map {|p| [p.src, p.dst, p.ident] }
    pipe_out <= join([ack, pipe], [ack.ident, pipe.ident]).map {|a, p| p }
  end
end
```

Predicate Dependency Graph Legend

Show and Tell: Building a consistent, replicated shopping cart in Bloom

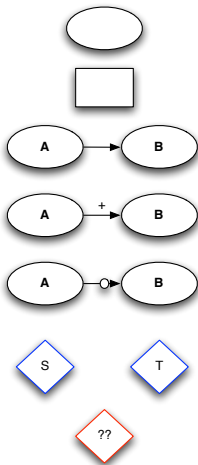
Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart



Scratch

Persistent Table

If B appears in LHS, A in RHS of a rule R

if R is a temporal (<+ or <-) rule

if R is nonmonotonic (uses negation, aggregation or deletion)

Distinguished nodes for dataflow source and sink (respectively)

Distinguished node for underspecified dataflow

Dependency Graphs

Show and Tell: Building a consistent, replicated shopping cart in Bloom

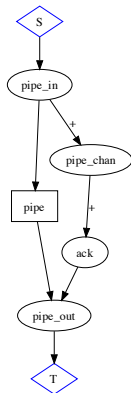
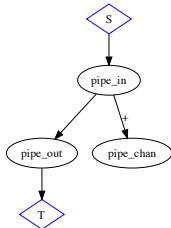
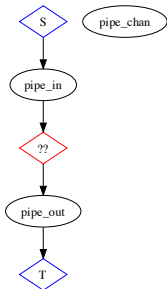
Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart



DeliveryProtocol, BestEffortDelivery and ReliableDelivery

Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

- 1 Background
 - The CALM Conjecture
 - Introducing Bloom
 - Writing distributed programs in Bloom
- 2 Shopping Carts
 - Cart client
 - A key/value store
 - A destructive cart
 - A disorderly cart

Shopping Carts

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

Example application: a replicated shopping cart

- 1 Replicated to achieve high availability and low latency
- 2 Clients are associated with unique session ids
- 3 Add item, delete item and “checkout” operations
- 4 A slightly more complicated (and more modern) version of the classic escrow transaction model

Challenge: ensure that replicas are “eventually consistent”

Rule of thumb: use commutative operations.

- Easier said than done

Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

1 Background

- The CALM Conjecture
- Introducing Bloom
- Writing distributed programs in Bloom

2 Shopping Carts

- **Cart client**
- A key/value store
- A destructive cart
- A disorderly cart

Cart Client Abstract Interfaces

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

```
module CartClientProtocol
  def state
    super
    interface input, :client_checkout,
      ['server', 'client', 'session', 'reqid']
    interface input, :client_action,
      ['server', 'client', 'session', 'item', 'action', 'reqid']
    interface output, :client_response,
      ['client', 'server', 'session', 'item', 'cnt']
  end
end
```

A simple realization of a cart client

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

```
module CartProtocol
  def state
    super
    channel :action_msg ,
      [ '@server', 'client', 'session', 'item', 'action', 'reqid' ]
    channel :checkout_msg ,
      [ '@server', 'client', 'session', 'reqid' ]
    channel :response_msg ,
      [ '@client', 'server', 'session', 'item', 'cnt' ]
  end
end

module CartClient
  include CartProtocol
  include CartClientProtocol

  declare
  def client
    action_msg <~ client_action.map { |a| a }
    checkout_msg <~ client_checkout.map { |a| a }
    client_response <= response_msg.map { |r| r }
  end
end
```

Dependency Graphs: abstract and concrete client

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

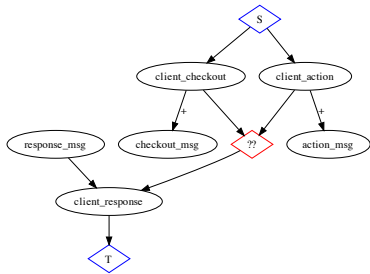
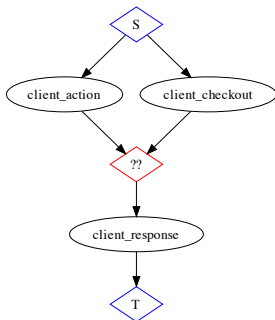
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store
A destructive cart
A disorderly cart



Note that the concrete client is still underspecified: we haven't supplied an implementation of the cart yet!

Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

- 1 Background
 - The CALM Conjecture
 - Introducing Bloom
 - Writing distributed programs in Bloom
- 2 Shopping Carts
 - Cart client
 - **A key/value store**
 - A destructive cart
 - A disorderly cart

A simple key/value store

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

```
module KVSProtocol
  def state
    super
    interface input, :kvput, ['client', 'key', 'reqid'], ['value']
    interface input, :kvget, ['reqid'], ['key']
    interface output, :kvget_response, ['reqid'], ['key', 'value']
  end
end
```

A simple key/value store

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A disorderly cart

```
module BasicKVS
  include KVSProtocol

  def state
    super
    table :bigtable, ['key'], ['value']
  end

  declare
  def mutate
    bigtable <+ kvput.map {|p| [p.key, p.value] }
    jst = join [bigtable, kvput], [bigtable.key, kvput.key]
    bigtable <- jst.map {|b, p| b }
  end

  declare
  def get
    kvget_response <= join([kvget, bigtable], [kvget.key, bigtable.key]).map do
      [g.reqid, t.key, t.value]
    end
  end
end
```

A simple key/value store

Show and Tell: Building a consistent, replicated shopping cart in Bloom

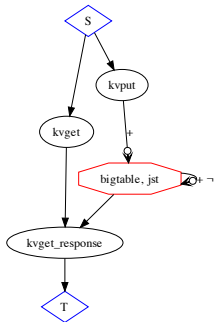
Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart



A simple key/value store

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

The analysis shows that any path through *kvput* crosses both a point of order and a temporal edge. Where is the nonmonotonicity?

A simple, conservative, syntactic check

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

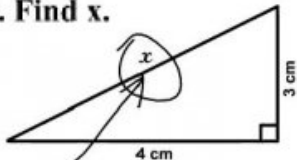
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

3. Find x .



Here it is

A simple, conservative, syntactic check

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping

Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

```
module BasicKVS
  include KVSProtocol

  def state
    super
    table :bigtable, ['key'], ['value']
  end

  declare
  def mutate
    bigtable <+ kvput.map {|p| [p.key, p.value] }
    jst = join [bigtable, kvput], [bigtable.key, kvput.key]

    # dude, it's here! (<-)
    bigtable <- jst.map {|b, p| b }
  end

  declare
  def get
    kvget_response <= join([kvget, bigtable], [kvget.key, bigtable.key]).map do
      [g.reqid, t.key, t.value]
    end
  end
end
```

Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

- 1 Background
 - The CALM Conjecture
 - Introducing Bloom
 - Writing distributed programs in Bloom
- 2 Shopping Carts
 - Cart client
 - A key/value store
 - **A destructive cart**
 - A disorderly cart

Destructive Cart

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro, Neil Conway, Joseph M. Hellerstein, William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

We'll use the KVSPProtocol as the storage abstraction for a key/value store:

```
module DestructiveCart
  include CartProtocol
  include KVSPProtocol
  declare
  def queueing
    kvput <= action_msg.map do |a|
      if a.action == "A" and !bigtable.map{|b| b.key}.include? a.session
        [a.server, a.client, a.session, a.reqid, [a.item]]
      end
    end
    joldstate = join [bigtable, action_msg], [bigtable.key, action_msg.session]
    kvput <= joldstate.map do |b, a|
      if a.action == "A"
        [a.server, a.client, a.session, a.reqid, (b.value.push(a.item))]
      elsif a.action == "D"
        [a.server, a.client, a.session, a.reqid, delete_one(b.value, a.item)]
      end
    end
  end
  end
  declare
  def finish
    kvget <= checkout_msg.map{|c| [c.reqid, c.session]}
    response_msg <~ join([kvget_response, checkout_msg],
      [kvget_response.key, checkout_msg.session]).map do |r, c|
      [r.client, r.server, r.key, r.value]
    end
  end
end
```

Destructive Cart

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

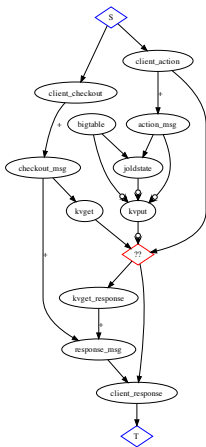
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A **destructive** cart
A disorderly cart

Looks ok, but we have committed to an abstract KVS implementation.



Destructive Cart

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

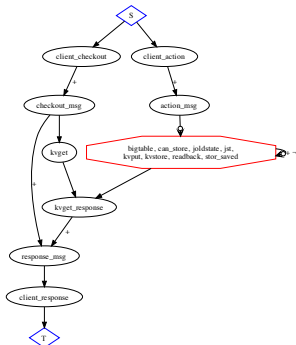
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

When we mix in our basic KVS, it interposes its point of order into the dataflow



Destructive Cart

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

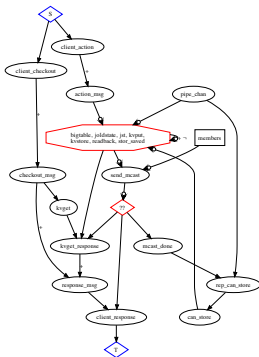
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

and finally best-effort multicast, completing the concrete implementation of an asynchronously-replicated key/value store:



Final Analysis

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

- 1 There is a point of order at each cart update from the client
- 2 There is a point of order as each tuple is forwarded to replicas
- 3 All this was evident even in the abstract cart (once we committed to using a KVS)
- 4 Solutions:
 - 1 Assert that all operations commute, and leave as is.
 - Informal and bug-prone
 - E.g., a deletion for a given item doesn't commute with that item's addition.
 - 2 Add a round of distributed coordination for each update
 - E.g., Two-phase commit, Paxos
 - Overkill?
 - 3 Is there a better cart abstraction?

Outline

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

- 1 Background
 - The CALM Conjecture
 - Introducing Bloom
 - Writing distributed programs in Bloom
- 2 Shopping Carts
 - Cart client
 - A key/value store
 - A destructive cart
 - A disorderly cart

A simple skeleton for a “disorderly” cart

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro, Neil Conway, Joseph M. Hellerstein, William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

```
module DisorderlyCart
  include CartProtocol
  def state
    super
    table :cart_action, ['session', 'item', 'action', 'reqid']
    table :action_cnt, ['session', 'item', 'action'], ['cnt']
    scratch :status, ['server', 'client', 'session', 'item'], ['cnt']
  end
  declare
  def saved
    cart_action <= action_msg.map { |c| [c.session, c.item, c.action, c.reqid] }
    action_cnt <= cart_action.group(
      [cart_action.session, cart_action.item, cart_action.action],
      count(cart_action.reqid))
    action_cnt <= cart_action.map do |a|
      unless cart_action.map{|c| [c.session, c.item] if c.action == "D"}.include
        [a.session, a.item, 'D', 0]
      end
    end
  end
  declare
  def consider
    status <= join([action_cnt, action_cnt, checkout_msg]).map do |a1, a2, c|
      if a1.session == a2.session and a1.item == a2.item
        and a1.session == c.session and a1.action == "A" and a2.action == "D"
          [c.client, c.server, a1.session, a1.item, a1.cnt - a2.cnt]
        end
      end
    response_msg <~ status.map { |s| s }
  end
end
```

A simple skeleton for a “disorderly” cart

Show and Tell: Building a consistent, replicated shopping cart in Bloom

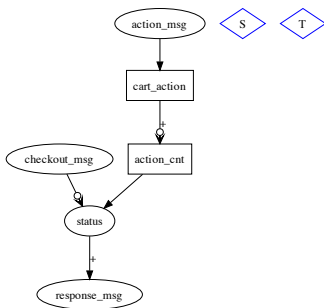
Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart



... and its composition with the client code

Show and Tell: Building a consistent, replicated shopping cart in Bloom

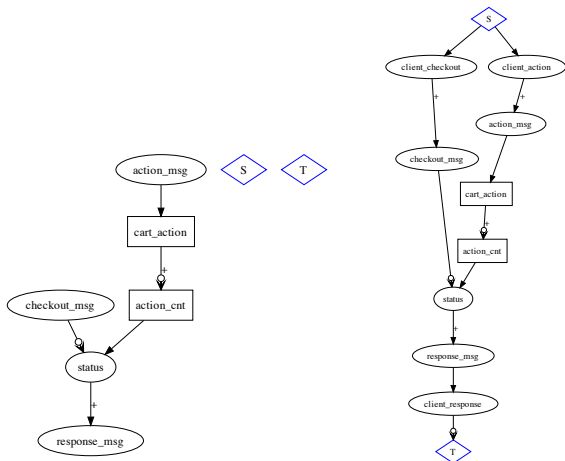
Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart



Note the points of order (circles) corresponding to aggregation in the dataflow.

Replication

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro, Neil Conway, Joseph M. Hellerstein, William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

We use the abstract class Multicast...

```
module MulticastProtocol
  def state
    super
    table :members, ['peer']
    interface input, :send_mcast, ['ident'], ['payload']
    interface output, :mcast_done, ['ident'], ['payload']
  end
end

module Multicast
  include MulticastProtocol
  include DeliveryProtocol
  include Anise
  annotator :declare

  declare
  def snd_mcast
    pipe_in <= join([send_mcast, members]).map do |s, m|
      [m.peer, @addy, s.ident, s.payload]
    end
  end

  declare
  def done_mcast
    # override me
    mcast_done <= pipe_out.map{|p| [p.ident, p.payload]}
  end
end
```

Replication

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

... and extend the disorderly cart to use it.

```
module ReplicatedDisorderlyCart
  include DisorderlyCart
  include Multicast

  declare
  def replicate
    send_mcast <= action_msg.map {|a| [a.reqid, a] }
    action_msg <= mcast_done.map {|m| m.payload }
  end
end
```

Replication

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

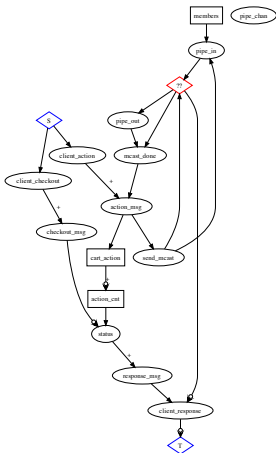
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

But our analysis tells us that this combination is underspecified: it lacks a realization of Multicast



Replication

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

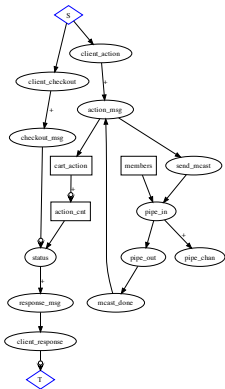
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

We complete the replicated, disorderly cart by supplying a concrete implementation of Multicast (including BestEffortDelivery).



Final Analysis

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A disorderly cart

- 1 Our implementation is fully specified
- 2 Our concrete implementation has the same points of order that were implied by the abstraction
- 3 Client updates and replication of cart state may be coordination-free
- 4 Some coordination may be necessary to handle a *checkout* message

Replication Overkill

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro, Neil Conway, Joseph M. Hellerstein, William R. Marczak

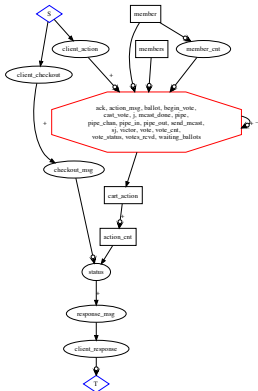
Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart

Too much coordination! By using ReliableMulticast (voting-based, not shown) we add coordination to an already monotonic program component (update/replication), and still do not address the downstream point of order.



Two abstract carts

Show and Tell: Building a consistent, replicated shopping cart in Bloom

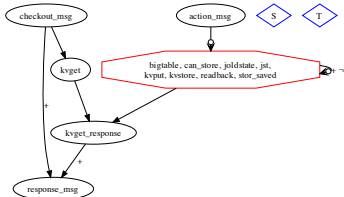
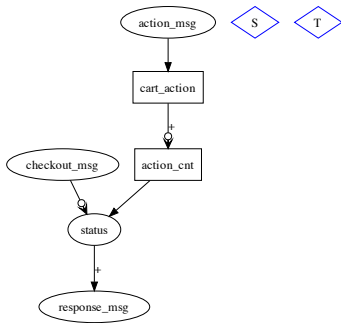
Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture
Introducing Bloom
Writing distributed programs in Bloom

Shopping Carts

Cart client
A key/value store
A destructive cart
A disorderly cart



Thanks!

Show and Tell: Building a consistent, replicated shopping cart in Bloom

Peter Alvaro,
Neil Conway,
Joseph M. Hellerstein,
William R. Marczak

Background

The CALM Conjecture

Introducing Bloom

Writing distributed programs in Bloom

Shopping Carts

Cart client

A key/value store

A destructive cart

A **disorderly** cart