Approximate Causal Consistency for Partially Replicated Geo-Replicated Cloud Storage

University of Illinois at Chicago
Ajay D. Kshemkalyani and Ta-Yuan Hsu

Agenda

- Introduction
- System model
  - Causal consistent memory
  - Underlying distributed communication system
  - Activation Predicate
- Algorithm Approx-Opt-Track
- Credits
- Conclusion and Future Work

Introduction

- Data Replication - a technique for fault tolerance in distributed systems
  - Reduces access latency in the cloud and geo-replicated systems.
- Consistency of data - a core issue in the distributed shared memory
- Consistency Models
  - linearizability (the strongest)
  - sequential consistency
  - causal consistency
    - (the stronger consistency satisfying low latency)
  - pipelined RAM
  - eventual consistency (the weakest)
- partial replication for geo-replicated cloud storage

Causally Consistent Memory

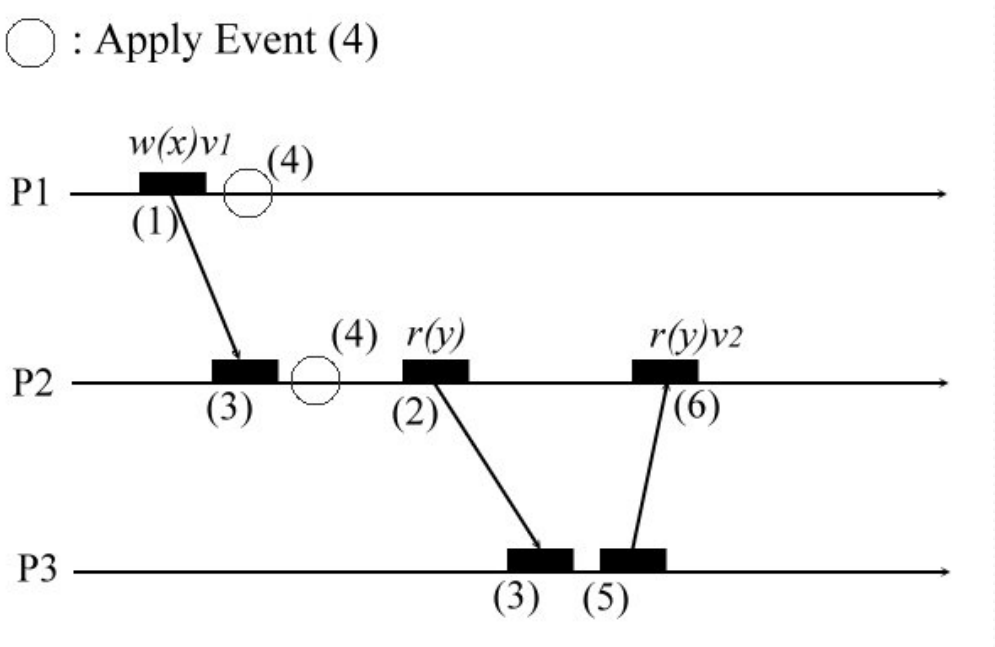
- n application processes : $ap_1, ap_2, \ldots, ap_n$
- A distributed shared memory Q
- q variables : $x_1, x_2, \ldots, x_q$ (initial value : $\perp$)
- $o_1 \prec_{co} o_2$
  - $o_1 \prec_{po} o_2$: o_1 precedes o_2 under *program order*
  - $o_1 \prec_{ro} o_2$: o_1 and o_2 from *api* and *apj* s.t. $o_1 = w(x)v$ and $o_2 = r(x)v$
    - $\exists o_3 \in O_H$ s.t. $o_1 \prec_{co} o_3$ and $o_3 \prec_{co} o_2$ (transitive closure)

Underlying Distributed System

- Distributed shared memory
- Causal consistency model
- Implemented on top of the distributed message passing system
- Connected by FIFO channels
- With each site hosting an application process

6 primitives

- (1) Send event
- (2) Fetch event
- (3) Message receipt event
- (4) Apply event
- (5) Remote return event
- (6) Return event



Activation Predicate

- $i = j$ and $send_i(m_{w(x)a})$ locally precedes $send_j(m_{w(y)a})$
- $i \neq j$ and $return_j(x, a)$ locally precedes $send_j(m_{w(y)a})$
- $i \neq j$ and $apply_l(w(x)a)$ locally precedes $remote\_return_l(r_j(x)a)$, which precedes $return_l(r_j(x)a)$, which locally precedes $send_j(m_{w(y)b})$
- $\exists send_k(m_{w(z)c})$ s.t. $send_i(m_{w(x)a}) \rightarrow_{c0} send_k(m_{w(z)c}) \rightarrow_{co} send_j(m_{w(y)b})$

- Baldoni et al. gave an optimal activation predicate as follows

$$A_{OPT}(m_w, e) \equiv \nexists m_{w'} : (send_j(m_{w'}) \rightarrow_{co} send_k(m_w) \wedge apply_i(w') \notin E_i|_e)$$

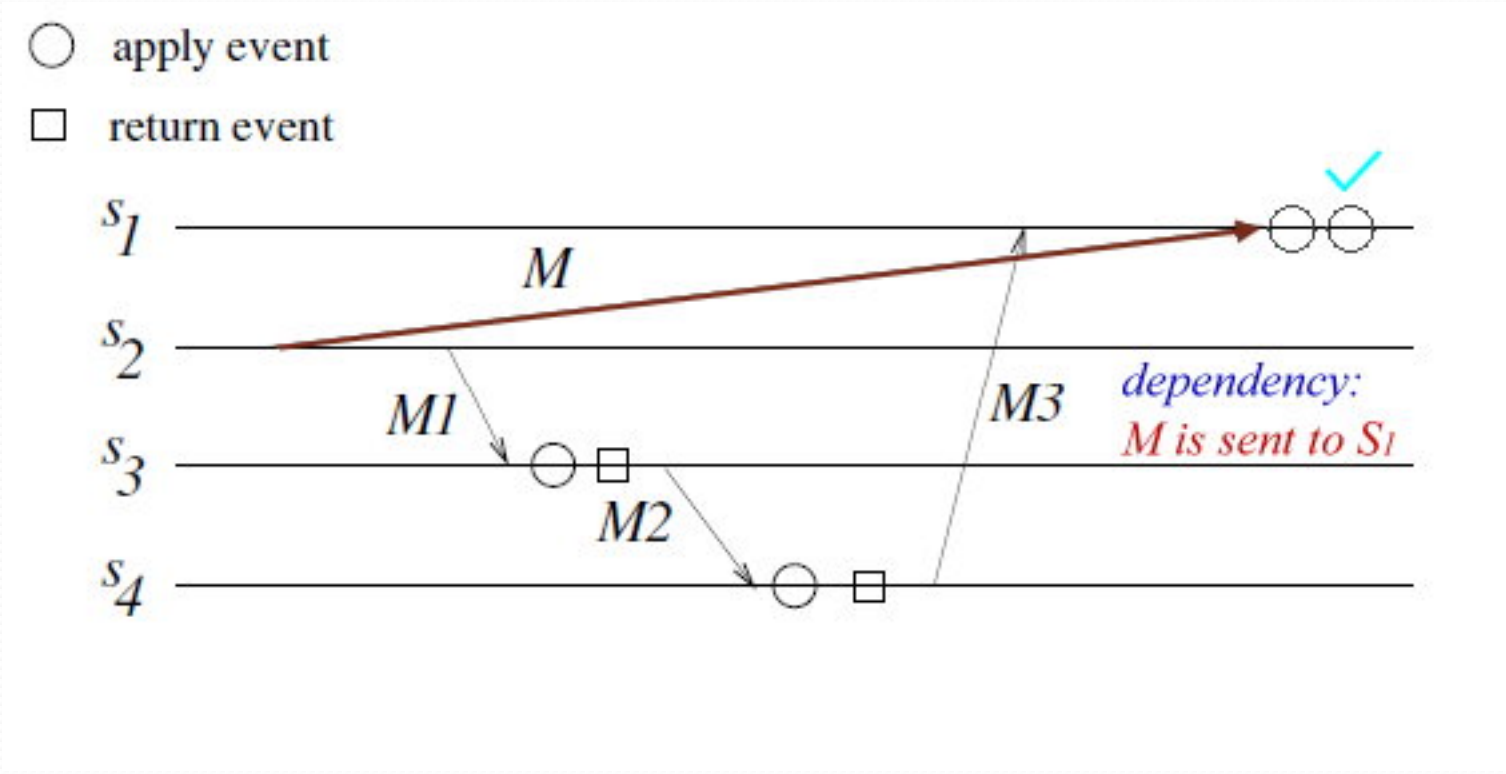
Opt-Track Algorithm[1]

- Kshemkalyani and Singhal proposed an optimization technique (the KS algorithm) reducing the message size and storage cost for causal ordering in message passing system.
- With the transitive dependency of causal deliveries of messages, each site keeps a record of recently received messages from each other site.
- The list of destinations of the message is also saved in each record. With each outgoing message, these records are also piggybacked.
- No redundant destination information is recorded.

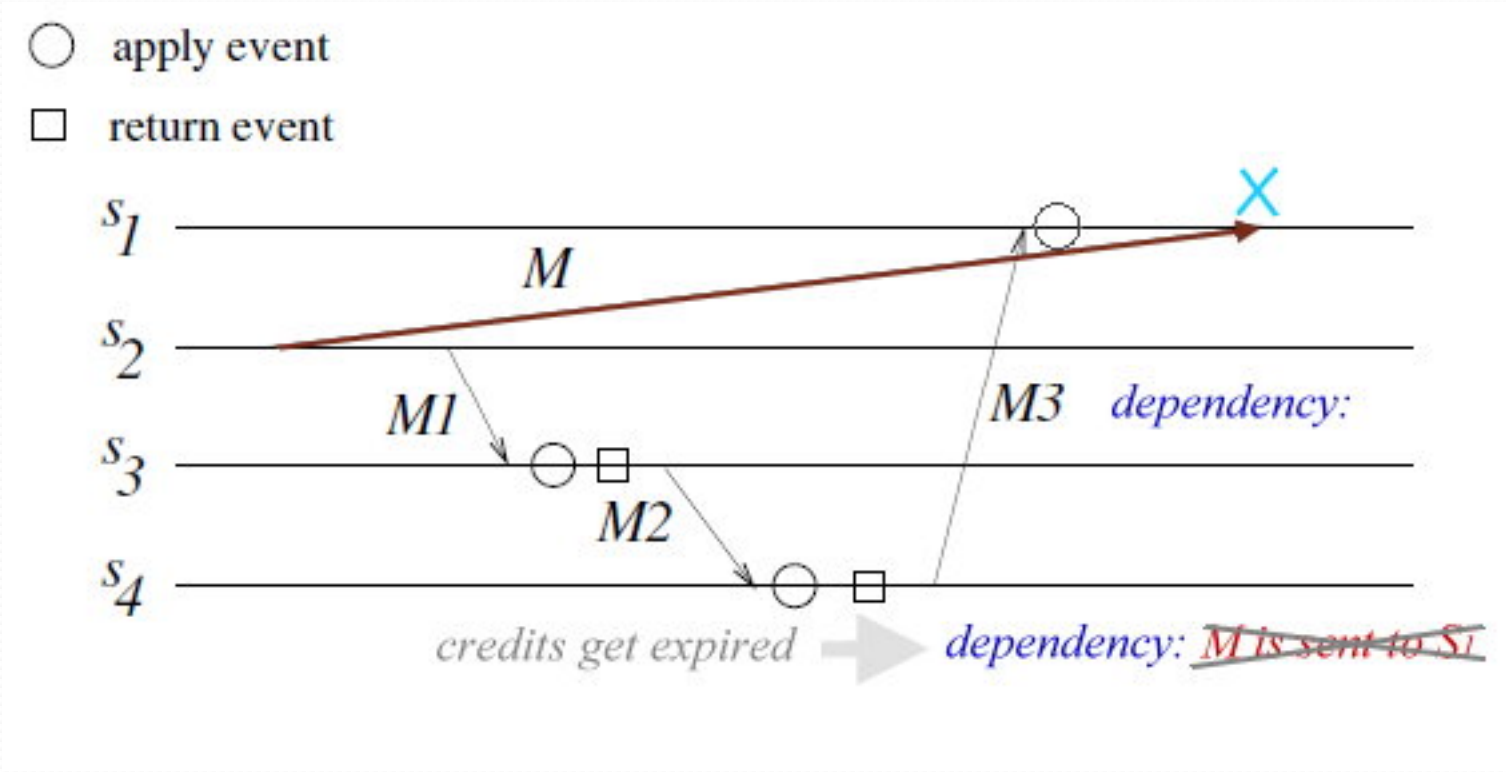
APPROX-OPT-TRACK

- With implicit knowledge about messages to be delivered in causal order, **Opt-Track** automatically prunes the meta-data.
- In the amortized case, the meta-data is manageable and linear in *n*, rather than quadratic.
- Further reduces the size of meta-data by deleting older dependencies. (With very high probability, the older the dependencies are, the more they are likely to be immediately satisfied)
- *Credits* is a parameter that let's approximate causal consistency.

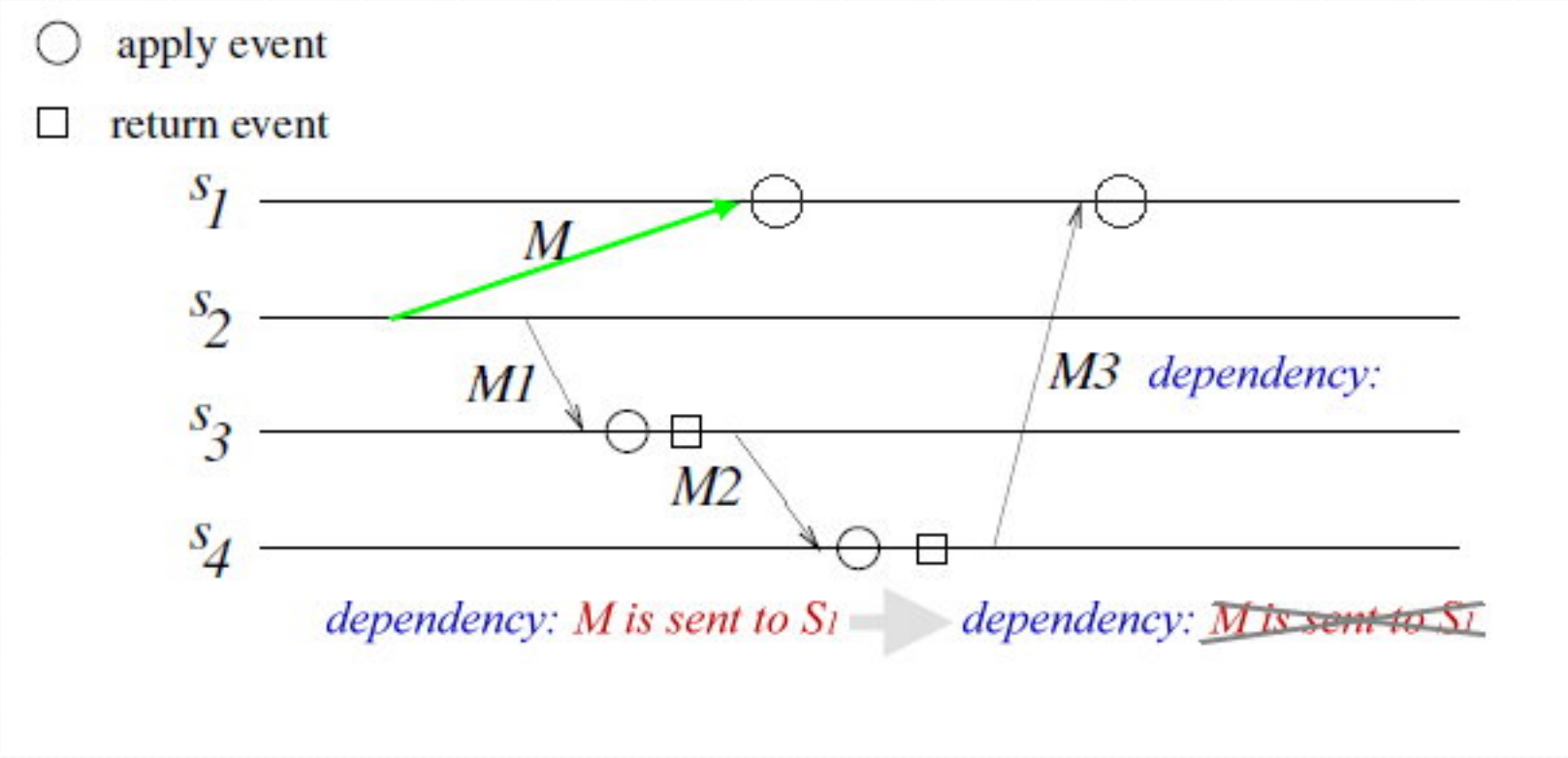
Normal Case in Opt-Track



Causal Consistency Violation if Credits are Exhausted



Meta-Data Reduction when Credits are Exhausted



Data Structures in Approx-Opt-Track

- $Clock_i$: local counter at site s_i for write operation
- $Apply_i$ [$1 \sim n$] : an integer array
  - $Apply_i$ [j] = a : a total number of a updates written at site s_j
- LOG_i = {<j, $clockj$, $Dests$, cr>} : (the local log). Each entry indicates a write operation in the causal past.
  - $Dests$: the destination list for the write operation.
  - cr : the remaining amount of credits. Only necessary destination information is stored.
- $LastWriteOn_i$< *variable* x_h, LOG > : stores the piggybacked LOG from the most recent update applied at site s_i for variable x_h.

Approx-Opt-Track Algorithm

WRITE(x_h, v):

$clock_i + +$;

(1) $cr :=$ initial credits;

for *all* $l \in LOG_i$ **do**

  $l.cr := l.cr -$ used credits;

(2) **if** $l.cr \leq 0 \wedge l.Dests \neq \varnothing$ **then** delete l;

for *all sites* $s_j (j \neq i)$ *that replicate* x_h **do**

  $L_w := LOG_i$;

  for *all* $o \in L_w$ **do**

    if $s_j \notin o.Dests$ **then**

    $o.Dests := o.Dests \setminus x_h.replicas$;

    else $o.Dests := o.Dests \setminus x_h.replicas \cup \{s_j\}$;

  for *all* $o_{z,clock_z} \in L_w$ **do**

    if $o_{z,clock_z}.Dests = \varnothing \wedge (\exists o'_{z,clock'_z} \in L_w | clock_z <$ $clock'_z)$ **then** remove $o_{z,clock_z}$ from L_w;

(3) Send $m(x_h, v, i, clock_i, x_h.replicas, cr, L_w)$ to site s_j;

for *all* $l \in LOG_i$ **do**

  $l.Dests := l.Dests \setminus x_h.replicas$;

PURGE;

(4) $LOG_i := LOG_i \cup \{\langle i, clock_i, x_h.replicas \setminus \{s_i\}, cr\rangle\}$;

if x_h *is locally replicated* **then**

  $x_h := v$;

  $Apply_i[i] + +$;

  $LastWriteOn_i\langle h\rangle := LOG_i$;

READ(x_h):

if x_h *is not locally replicated* **then**

  RemoteFetch$[f(x_h)]$ from randomly selected site s_j that replicates x_h to get x_h and $LastWriteOn_j\langle h\rangle$;

  MERGE$(LOG_i, LastWriteOn_j\langle h\rangle)$;

else MERGE$(LOG_i, LastWriteOn_i\langle h\rangle)$;

PURGE;

return x_h;

Approx-Opt-Track Algorithm

On receiving $m(x_h, v, j, clock_j, x_h.replicas, c, L_w)$ from site s_j:

for *all* $o_{z,clock_z} \in L_w$ **do**
  if $s_i \in o_{z,clock_z}.Dests$ **then** wait until $clock_z \leq Apply_i[z]$;

for *all* $o \in L_w$ **do**
  $o.cr := o.cr -$ used credits;
  if $o.cr \leq 0 \wedge o.Dests \neq \emptyset$ **then** delete o;

$x_h := v$;
$Apply_i[j] := clock_j$;
$L_w := L_w \cup \{\langle j, clock_j, x_h.replicas, c - \text{used credits}\rangle\}$;
for *all* $o_{z,clock_z} \in L_w$ **do**
  $o_{z,clock_z}.Dests := o_{z,clock_z}.Dests \setminus \{s_i\}$;
$LastWriteOn_i\langle h\rangle := L_w$;

On receiving $f(x_h)$ from site s_j:

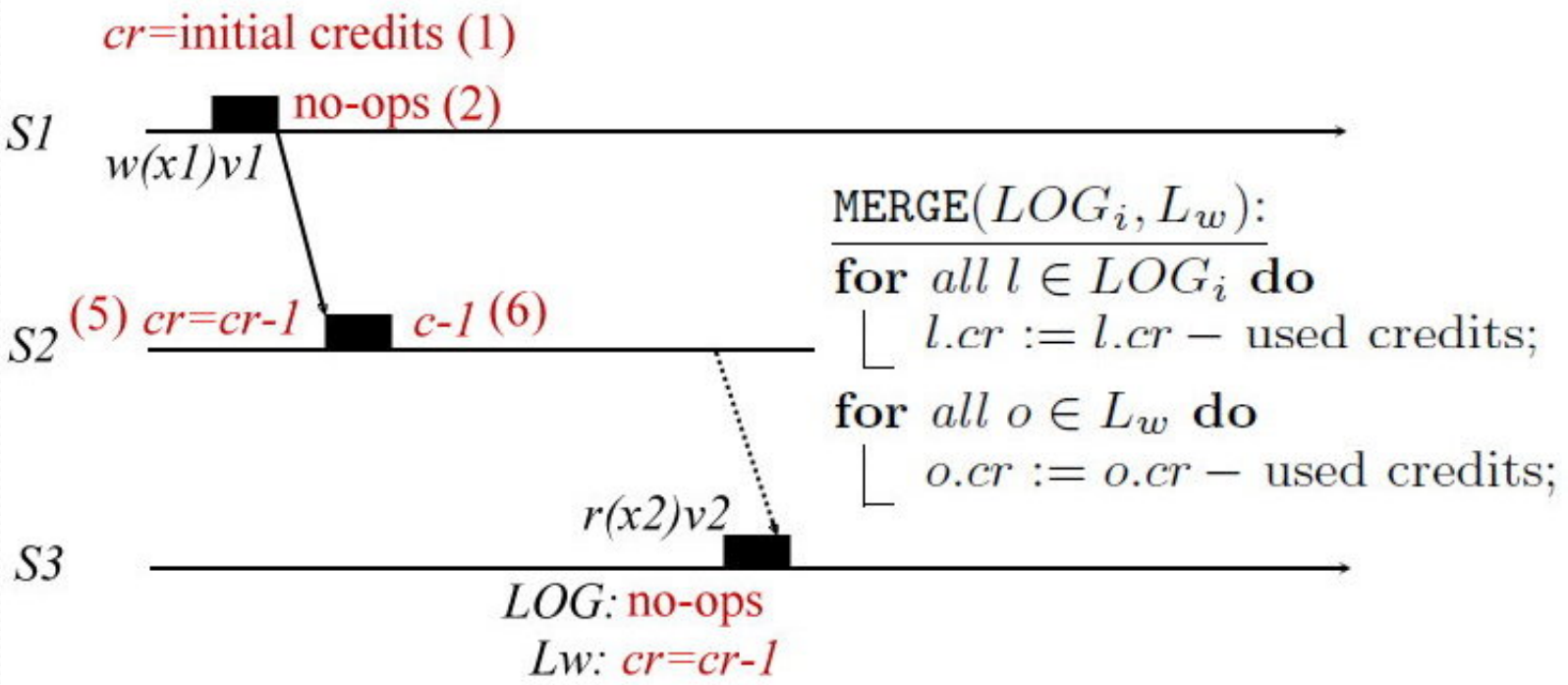
return x_h and $LastWriteOn_i\langle h\rangle$ to s_j;

PURGE:
for *all* $l_{z,t_z} \in LOG_i$ **do**
  if $l_{z,t_z}.Dests = \emptyset \wedge (\exists l'_{z,t'_z} \in LOG_i | t_z < t'_z)$ **then**
    remove l_{z,t_z} from LOG_i;

MERGE(LOG_i, L_w):
for *all* $l \in LOG_i$ **do**
  $l.cr := l.cr -$ used credits;
for *all* $o \in L_w$ **do**
  $o.cr := o.cr -$ used credits;
for *all* $o_{z,t} \in L_w$ *and* $l_{s,t'} \in LOG_i$ *such that* $s = z$ **do**
  if $t < t' \wedge l_{s,t} \notin LOG_i$ **then** mark $o_{z,t}$ for deletion;
  if $t' < t \wedge o_{z,t'} \notin L_w$ **then** mark $l_{s,t'}$ for deletion;
  delete marked entries;
  if $t = t'$ **then**
    $l_{s,t'}.Dests := l_{s,t'}.Dests \cap o_{z,t}.Dests$;
    $l_{s,t'}.cr := \min(l_{s,t'}.cr, o_{z,t}.cr)$;
    delete $o_{z,t}$ from L_w;
$LOG_i := LOG_i \cup L_w$;
for *all* $l \in LOG_i$ **do**
  if $l.cr \leq 0 \wedge l.Dests \neq \emptyset$ **then** delete l;

Credit Instantiations (Hop Count)

Credits denote the hop count available before the entry ages out.



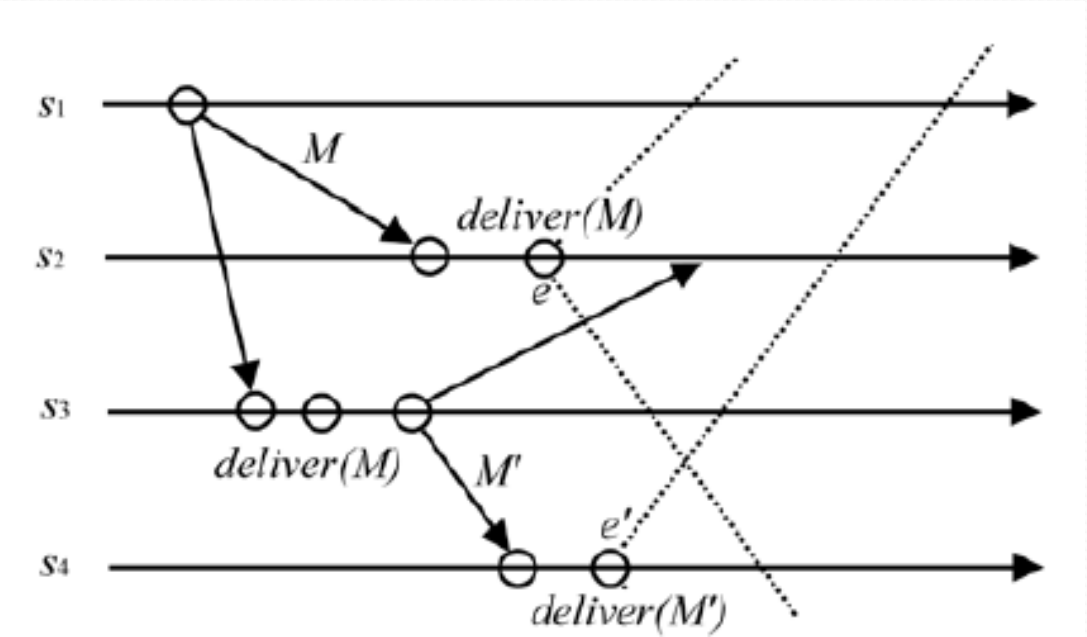
Conclusion and Future Work

- Proposed a modification of Opt-Track, called Approx-Opt-Track, provides approximate causal consistency by reducing the size of the meta-data.
- By controlling a parameter *cr*, we can trade-off the level of potential inaccuracy by the size of meta-data.
- As future work, we will study the exact nature of the trade-off experimentally for some different instantiations of credits.

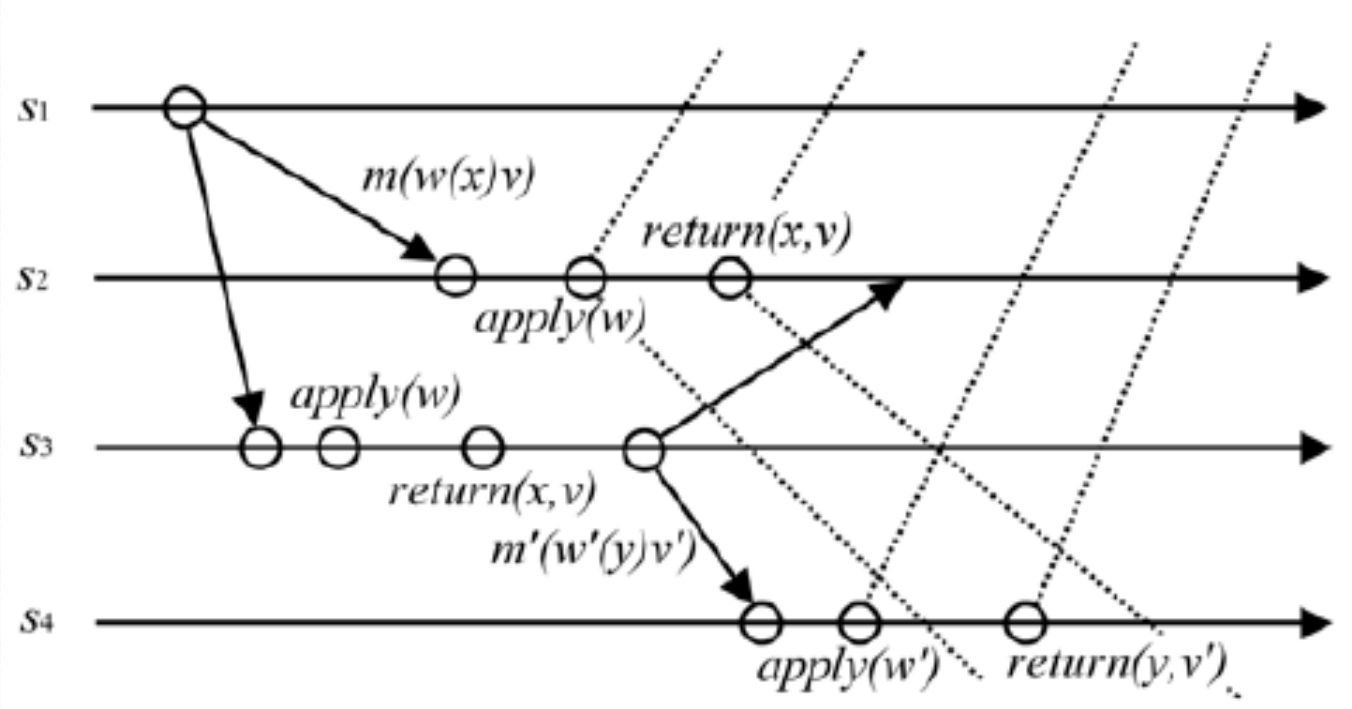
References

1. M. Shen, A.D. Kshemkalyani, and T. Hsu. Causal consistency for geo-replicated cloud storage under partial replication. IEEE Int. Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 509-518, May 2015.
2. M. Shen, A.D. Kshemkalyani, and T. Hsu. OPCAM: Optimal algorithms implementing causal memories in shared memory systems. In Int. Conference on Distributed Computing and Networking (ICDCN), Jan 2015.
3. R. Baldoni, A. Milani, and S. Piergiovanni. Optimal propagation-based protocols implementing causal memories. Distributed Computing, 18, 6, pages 461-474, 2006.

Optimality in two situation



Prune redundant destination information



How to deal with an empty record?