Distributed Causal Memory: Modular Specification and Verification in Higher-Order Distributed Separation Logic

Technical Appendix

LÉON GONDELMAN, Aarhus University, Denmark
SIMON ODDERSHEDE GREGERSEN, Aarhus University, Denmark
ABEL NIETO, Aarhus University, Denmark
AMIN TIMANY, Aarhus University, Denmark
LARS BIRKEDAL, Aarhus University, Denmark

A Summary of the Specification of the Causally Consistent Distributed Database

Given client-provided Addrlist and *Keys*, we provide types, functions, predicates, and lemmas as in Section 3 (Mathematical Model), and given those, the specification of the causally-consistent distributed database is as follows. The specification existentially quantifies over abstract predicates; then follows all the laws governing the abstract predicates from Section 4 (Specification) and Section 7 (HOCAP Style Specification for Write) (here we only show a few) and finally the specification for the initialization operation.

$$
\begin{array}{l}
\exists\ \mathsf{GlobalInv} : \mathit{iProp}. \\
\exists\ (\cdot) \rightharpoonup_s (\cdot) : \mathit{Keys} \to \mathscr{P}_{\mathrm{fin}}(\mathit{WriteEvent}) \to \mathit{iProp}. \\
\exists\ (\cdot) \rightharpoonup_u (\cdot) : \mathit{Keys} \to \mathscr{P}_{\mathrm{fin}}(\mathit{WriteEvent}) \to \mathit{iProp}. \\
\exists\ \mathsf{Seen} : \mathbb{N} \to \mathit{LocalHistory} \to \mathit{iProp}. \\
\exists\ \mathsf{Snap} : \mathit{Keys} \to \mathscr{P}_{\mathrm{fin}}(\mathit{WriteEvent}) \to \mathit{iProp}. \\
\exists\ \mathsf{initToken} : \mathbb{N} \to \mathit{iProp}. \\
\exists\ \Phi_{\mathrm{DB}} : \mathit{Message} \to \mathit{iProp}. \\
\qquad \mathsf{Snap}(x, h) * \mathsf{Snap}(x, h') \vdash \mathsf{Snap}(x, h \cup h') \\
\wedge \quad x \rightharpoonup_u h \vdash x \rightharpoonup_u h * \mathsf{Snap}(x, h) \\
\wedge \quad \ldots \\
\wedge \quad \boxed{\mathsf{GlobalInv}}^{\mathcal{N}_{\mathrm{GI}}} * \mathsf{Seen}(i, s) * x \rightharpoonup_u h \vdash \Rrightarrow_{\mathcal{E}} \forall a \in s, w \in h.\ w.t < a.t \Rightarrow \exists a' \in s_{|x}.\ \lfloor a' \rfloor = w \\
\wedge \quad \mathsf{True} \vdash \Rrightarrow_{\mathcal{E}} \boxed{\mathsf{GlobalInv}}^{\mathcal{N}_{\mathrm{GI}}} * \Big(\mathop{\Huge *}_{0 \le i < \mathit{length}(\mathsf{Addrlist})} \mathsf{initToken}(i) \Big) * \Big(\mathop{\Huge *}_{k \in \mathit{Keys}} k \rightharpoonup_u \emptyset \Big) * \mathsf{initSpec}(\mathsf{init})
\end{array}
$$

where

$$
\begin{array}{l}
\mathsf{initSpec}(\mathsf{init}) \triangleq \\
\qquad \left\{ \begin{array}{l} \mathsf{initToken}(i) * \mathsf{Fixed}(A) * \mathsf{IsNode}(ip_i) * \mathsf{Addrlist}[i] = (ip_i, p) * \\ (ip_i, p) \in A * \mathit{isList}(\mathsf{Addrlist}, v) * \mathsf{FreePorts}(ip_i, \{p\}) * \mathop{\Huge *}_{z \in \mathsf{Addrlist}} z \Rightarrow \Phi_{\mathrm{DB}} \end{array} \right\} \\
\qquad\quad \langle ip_i;\ \mathsf{init}(i, v) \rangle \\
\qquad \{ (\mathsf{rd}, \mathsf{wr}).\ \ \mathsf{Seen}(i, \emptyset) * \mathsf{readSpec}(\mathsf{rd}, i) * \mathsf{writeSpec}(\mathsf{wr}, i) \quad \} \\
\mathsf{readSpec}(\mathsf{read}, i) \triangleq
\end{array}
$$

Authors' addresses: Léon Gondelman, Aarhus University, Denmark, gondelman@cs.au.dk; Simon Oddershede Gregersen, Aarhus University, Denmark, gregersen@cs.au.dk; Abel Nieto, Aarhus University, Denmark, abeln@cs.au.dk; Amin Timany, Aarhus University, Denmark, timany@cs.au.dk; Lars Birkedal, Aarhus University, Denmark, birkedal@cs.au.dk.

$$
\begin{array}{l}
\quad \{\mathsf{Seen}(i, s)\} \\
\quad\quad \langle ip_i;\ \mathsf{read}(x)\rangle \\
\quad \left\{\begin{array}{l}
v.\ \exists s' \supseteq s.\ \mathsf{Seen}(i, s') \ast \\
\Big((v = \mathsf{None} \wedge s'_{|x} = \emptyset) \vee \\
(\exists a \in s'_{|x}.\ v = \mathsf{Some}(a.v) \ast \mathrm{Snap}(x, \{\lfloor a \rfloor\}) \ast a \in \mathrm{Maximals}(s'_{|x}) \ast \mathrm{Observe}(s'_{|x}) = a)\Big)
\end{array}\right\} \\
\mathsf{writeSpec}(\mathsf{write}, i) \triangleq \\
\quad \forall \mathcal{E}, k, v, s, P, Q.\ \mathcal{N}_{\mathrm{GI}} \subseteq \mathcal{E} \Rightarrow \\
\quad \Box\,(\forall s', a.\ (s \subseteq s' \ast a \notin s' \ast a.k = k \ast a.v = v \ast P) \\
\quad\quad {}^{\top}\!\!\Rrightarrow\!\!\ast^{\mathcal{E}}\ \forall h. \begin{pmatrix} \lfloor a \rfloor \notin h \ast \lfloor a \rfloor \in \mathrm{Maximals}(h \uplus \{\lfloor a \rfloor\}) \ast k \rightharpoonup_s h \ast \\ \mathsf{Seen}(i, s' \uplus \{a\}) \ast \mathrm{Maximum}(s' \uplus \{a\}) = a \end{pmatrix} \\
\quad\quad \Rrightarrow\!\!\ast_{\mathcal{E} \setminus \mathcal{N}_{\mathrm{GI}}}\ k \rightharpoonup_s h \uplus \{\lfloor a \rfloor\} \ast {}^{\mathcal{E}}\!\!\Rrightarrow^{\top} Q\ a\ h\ s') \mathrel{-\!\!\ast} \\
\quad \{P \ast \mathsf{Seen}(i, s)\}\ \big\langle ip_i; \mathsf{write}(k, v)\big\rangle\ \{v.v = () \ast \exists h, s', a.\ s \subseteq s' \ast Q\ a\ h\ s'\}
\end{array}
$$

B Client Reasoning about Causality: Indirect Causal Dependency

The proof of the indirect causal dependency example [Lloyd et al. 2011] makes use of predicates

$$
\begin{aligned}
\mathsf{pending} &: \mathit{iProp} \\
\mathsf{shot} &: \mathit{WriteEvent} \longrightarrow \mathit{iProp} \\
\mathsf{hist} &: \mathscr{P}_{\mathrm{fin}}(\mathit{WriteEvent}) \xrightarrow{\mathrm{fin}} \mathit{iProp}
\end{aligned}
$$

satisfying the laws below. The predicates pending and shot are defined using the *oneshot* resource algebra [Jung et al. 2016] whereas the hist predicate is defined using a fractional agreement resource algebra. We refer to the Coq formalization for all the details and the full proof.

Laws for ghost resources

$$
\begin{array}{l}
\mathsf{pending} \Rrightarrow\!\!\ast\ \mathsf{shot}(w) \\
\mathsf{shot}(w) \ast \mathsf{shot} w' \vdash w = w' \\
\mathsf{pending} \ast \mathsf{shot}(w) \vdash \mathsf{False} \\
\boxed{\diamond} \ast \boxed{\diamond} \vdash \mathsf{False} \\
\mathsf{hist}(h_1) \ast \mathsf{hist}(h_2) \vdash h_1 = h_2 \\
\mathsf{hist}(h_1) \ast \mathsf{hist}(h_1) \vdash \mathsf{hist}(h_2) \ast \mathsf{hist}(h_2)
\end{array}
$$

Invariants

$$
\begin{array}{l}
\mathsf{Inv}_x \triangleq \exists h.\ x \rightharpoonup_u h \ast \mathsf{hist}(h) \ast \big((\mathsf{pending} \ast \forall w \in h.\ w.v \neq 1) \vee \\
\qquad\qquad (\boxed{\diamond} \ast \exists w.\ \mathsf{shot}(w) \ast w.v = 37 \ast \mathrm{Maximum}(h) = \mathrm{Some}(w) \ast \\
\qquad\qquad\quad \forall w' \in h.\ w'.v = 37 \Rightarrow w = w')\big) \\
\mathsf{Inv}_y \triangleq \exists h.\ y \rightharpoonup_u h \ast \forall w \in h.\ w.v = 1 \Rightarrow \exists w'.\ w'.t < w.t \ast \mathsf{shot}(w')
\end{array}
$$

Proof sketch

Node i, proof outline

$$\left\{\mathsf{Seen}(i, s) * \mathsf{hist}(h) * \boxed{\diamond} * h \subseteq \lfloor s \rfloor * \boxed{\mathsf{Inv}_x}\right\}$$

open Inv_x

$$\left\{\mathsf{hist}(h) * \mathsf{hist}(h) * x \rightharpoonup_u h * \mathsf{pending} * \boxed{\diamond}\right\}$$

$$\quad \mathsf{write}(x, 0)$$

$$\left\{\begin{array}{l}\exists a, s' \supseteq s.\, \mathsf{Seen}(i, s' \uplus \{a\}) * \mathsf{hist}(h \uplus \{\lfloor a \rfloor\}) \\ * \mathsf{hist}(h \uplus \{\lfloor a \rfloor\}) * x \rightharpoonup_u h \uplus \{\lfloor a \rfloor\} * \mathsf{pending} * \boxed{\diamond}\end{array}\right\}$$

$$\left\{\exists h', s'.\, \mathsf{Seen}(i, s') * \mathsf{hist}(h) * \boxed{\diamond} * h' \subseteq s' * \boxed{\mathsf{Inv}_x}\right\}$$

$$\left\{\mathsf{Seen}(i, s') * \mathsf{hist}(h') * \boxed{\diamond} * h' \subseteq \lfloor s' \rfloor * \boxed{\mathsf{Inv}_x}\right\}$$

open Inv_x

$$\left\{\mathsf{hist}(h) * \mathsf{hist}(h) * x \rightharpoonup_u h * \mathsf{pending} * \boxed{\diamond}\right\}$$

$$\quad \mathsf{write}(x, 37)$$

$$\left\{\begin{array}{l}\exists s'' \supseteq s', a'.\, \mathsf{Seen}(i, s'' \uplus \{a'\}) * \mathsf{hist}(h \uplus \{\lfloor a' \rfloor\}) * \mathsf{hist}(h \uplus \{\lfloor a' \rfloor\}) * x \rightharpoonup_u h \uplus \{\lfloor a' \rfloor\} \\ * \boxed{\diamond} * \mathrm{Maximum}(h \uplus \{\lfloor a' \rfloor\}) = \mathsf{Some}(\lfloor a' \rfloor) * \mathsf{shot}(\lfloor a' \rfloor)\end{array}\right\}$$

$$\left\{\exists h', s''.\, \mathsf{Seen}(i, s'') * \mathsf{hist}(h') * \boxed{\mathsf{Inv}_x}\right\}$$

Node j, proof outline

$$\left\{\mathsf{Seen}(j, s) * \boxed{\mathsf{Inv}_x} * \boxed{\mathsf{Inv}_y}\right\}$$

$$\quad \mathsf{wait}(x = 37)$$

$$\{\exists s' \supseteq s, a_x.\, \mathsf{Seen}(j, s) * \mathsf{shot}(\lfloor a_x \rfloor) * w_x \in s' * \ldots\}$$

$$\{\mathsf{Seen}(j, s') * \mathsf{shot}(w_x)\}$$

open Inv_y

$$\{\exists h_y, s'.\, x \rightharpoonup_u h_y * \ldots\}$$

$$\quad \mathsf{write}(y, 1)$$

$$\left\{\begin{array}{l}\exists a_y, s'' \supseteq s'.\, \mathsf{Seen}(j, s'' \uplus \{a_y\}) * x \rightharpoonup_u h_y \uplus \{\lfloor a_y \rfloor\} * \mathsf{shot}(w_x) * \\ a_y.v = 1 * a_y \in s'' * \lfloor a_x \rfloor.t < \lfloor a_y \rfloor.t\end{array}\right\}$$

$$\left\{\mathsf{Seen}(j, s'') * \boxed{\mathsf{Inv}_y}\right\}$$

Node k, proof outline

$$\left\{\mathsf{Seen}(k, s) * \boxed{\mathsf{Inv}_x} * \boxed{\mathsf{Inv}_y} *\right\}$$

$$\quad \mathsf{wait}(y = 1)$$

$$\{\exists s' \subseteq s, a_y \in s', w_x.\, \mathsf{Seen}(k, s') * \mathsf{shot}(w_x) * w_x.t < \lfloor a_y \rfloor.t * a_y.v = 1\}$$

$$\{\mathsf{Seen}(k, s') * \mathsf{shot}(w_x)\}$$

open Inv_x

$$\{\exists h_x.\, x \rightharpoonup_u h_x * \mathsf{shot}(w_x) * \mathsf{shot}(w_x) * \mathrm{Maximum}(h_x) = \mathsf{Some}(w_x) * w_x.v = 37\}$$

$$\quad \mathsf{read}(x)$$

$$\{v.\, \exists s''.\, \mathsf{Seen}(k, s'') * v = \mathsf{Some}(37)\}$$

$$\{v.\, \exists s''.\, \mathsf{Seen}(k, s'') * v = \mathsf{Some}(37)\}$$

C Case Study: Towards Session Guarantees for Client-Centric Consistency

C.1 Session Manager Library Implementation

```
(* Client stub *)
type db_key
type db_val

type sm_req =
  InitReq
| ReadReq of db_key
| WriteReq of db_key * db_val

type sm_res =
  InitRes
| ReadRes of db_val
| WriteRes

let rec listen_wait_for_seqid skt seq_id =
  let res_raw = listen_wait skt in
  let res = deser_res (fst res_raw) in
  let tag = fst res in
  let vl = snd res in
  if (tag = !seq_id) then
    (seq_id := !seq_id + 1;
    vl)
  else
    listen_wait_for_seqid skt seq_id

let session_exec skt seq_id lock server_addr
     req =
  acquire lock;
  let msg = ser_req req in
  sendTo skt msg server_addr;
  let res = listen_wait_for_seqid skt seq_id
     in
  release lock;
  res
```

```
(* Request handler *)
let rec request_handler skt rd_fn wr_fn =
  let req_raw = listen_wait skt in
  let sender = snd req_raw in
  let req = deser_req (fst req_raw) in
  let seq_id = fst req in
  let res =
    match (snd req) with
     | Some InitReq → InitRes
     | Some ReadReq k → ReadRes (rd_fn k)
     | Some WriteReq (k, v) → wr_fn k v;
      WriteRes
     | None → assert false
  in
  sendTo skt (ser_res (seq_id, res)) sender;
  request_handler skt rd_fn wr_fn

let server dbs db_id req_addr =
  let fns = init dbs db_id in
  let rd_fn = fst fns in
  let wr_fn = snd fns in
  let skt = socket () in
  socketbind skt req_addr;
  request_handler skt rd_fn wr_fn

let sm_setup client_addr =
  let skt = socket () in
  socketbind skt client_addr;
  let seq_id = ref 0 in
  let lock = newlock () in
  let connect_fn server_addr =
    session_exec skt seq_id lock server_addr
     InitReq;
    ()
  in
  let read_fn server_addr key =
    session_exec skt seq_id lock server_addr
     (ReadReq key)
  in
  let write_fn server_addr key vl =
    session_exec skt seq_id lock server_addr
     (WriteReq (key, vl));
    ()
  in
  (connect_fn, read_fn, write_fn)
```

C.2 Session Manager Specifications

SM-INIT

$$\{\top\}\ \langle ip_{client};\ \mathsf{sconnect}(ip_i)\rangle\ \left\{\exists s. \mathsf{Seen}(i, s) * \mathop{\Large\ast}_{k \in \mathsf{Keys}} \exists h_k. \mathsf{Snap}(k, h_k) * \boxed{\mathsf{GlobalInv}}^{\mathcal{N}_{\mathsf{GI}}}\right\}$$

SM-READ

$$\{ip_i \Mapsto \Phi_i * \mathsf{Seen}(i, s) * \mathsf{Snap}(k, h)\}$$
$$\langle ip_{client};\ \mathsf{sread}(ip_i, k)\rangle$$
$$\left\{\begin{array}{l} v. \exists s' \supseteq s, h' \supseteq h.\ * \mathsf{Seen}(i, s') * \mathsf{Snap}(k, h') * \\ \quad \big((v = \mathsf{None} * s'_{|k} = \emptyset) \\ \quad \lor \big(\exists a, w.\ v = \mathsf{Some}(w) * a.v = w * a.k = k * a \in \mathrm{Maximals}(s'_{|k}) * \lfloor a \rfloor \in h'\big)\big) \end{array}\right\}$$

SM-WRITE

$$\{ip_i \Mapsto \Phi_i * \mathsf{Seen}(i, s) * \mathsf{Snap}(k, h)\}$$
$$\langle ip_{client};\ \mathsf{swrite}(ip_i, k, v)\rangle$$
$$\left\{\begin{array}{l} \exists a, s' \subseteq s, h' \subseteq h.\ a.k = k * a.v = v * \mathsf{Seen}(i, s') * \mathsf{Snap}(k, h') \\ * a \notin s * a' \in s' * \lfloor a \rfloor \notin h * \lfloor a' \rfloor \in h' * \lfloor a \rfloor \in \mathrm{Maximals}(h') * \mathrm{Maximum}(s') = \mathsf{Some}(a) \end{array}\right\}$$

C.3 Session Guarantees Specifications

SM-READ-YOUR-WRITES

$$\{ip_i \Mapsto \Phi_i\}$$
$$\langle ip_{client};\ \mathsf{sconnect}(ip_i); \mathsf{swrite}(ip_i, k, v_w); \mathsf{sread}(ip_i, k)\rangle$$
$$\left\{ v_o.\ \begin{array}{l} \exists s, a_w, a_r, v_r.\ v_o = \mathsf{Some}(v_r) * a_w.k = k * a_w.v = v_w * a_r.k = k * a_r.v = v_r \\ * \mathsf{Seen}(a, s) * a_w, a_r \in s * \neg(a_r.t < a_w.t) \end{array}\right\}$$

SM-MONOTONIC-READS

$$\{ip_i \Mapsto \Phi_i\}$$
$$\langle ip_{client};\ \mathsf{sconnect}(ip_i); \mathrm{let}\ v_1 = \mathsf{sread}(ip_i, k)\ \mathrm{in}\ \mathrm{let}\ v_2 = \mathsf{sread}(ip_i, k)\ \mathrm{in}\ (v_1, v_2)\rangle$$
$$v_o.\ \begin{array}{l} \exists v_{o1}, v_{o2}.\ v_o = (v_{o1}, v_{o2}) \\ * \left(\begin{array}{l} (\exists s. v_{o1} = \mathsf{None} * v_{o2} = \mathsf{None} * \mathsf{Seen}(a_1, s) * s_{|k} = \emptyset) \\ \lor \\ (\exists s, v_2, a_2. v_{o1} = \mathsf{None} * v_{o2} = \mathsf{Some}(v_2) * \mathsf{Seen}(a_1, s) \\ * a_2.k = k * a_2.v = v_2 * a_2 \in \mathrm{Maximals}(s_{|k})) \\ \lor \\ (\exists s, v_1, v_2, a_1, a_2. v_{o1} = \mathsf{Some}(v_1) * v_{o2} = \mathsf{Some}(v_2) * \mathsf{Seen}(a_1, s) \\ * a_1.k = k * a_1.v = v_1 * a_2.k = k * a_2.v = v_2 * a_2 \in \mathrm{Maximals}(s_{|k}) \\ * \neg(a_2.t < a_1.t)) \end{array}\right) \end{array}$$

SM-MONOTONIC-WRITES

$$\{ip_i \Mapsto \Phi_i\}$$

$$\langle ip_{client};\ \mathsf{sconnect}(ip_i); \mathsf{swrite}(ip_i, k_1, v_1); \mathsf{swrite}(ip_i, k_2, v_2)\rangle$$

$$\left\{\begin{array}{l} \exists s_1, a_1, a_2.\ a_1.k = k_1 * a_1.v = v_1 * a_2.k = k_2 * a_2.v = v_2 \\ * \ \mathsf{Seen}(a_1, s_1) * a_1, a_2 \in s_1 * a_1.t < a_2.t \\ * \ (\forall a, s_2, a_2.\mathsf{Seen}(a_2, s_2) * a \in s_2 * a_2.t \leq e.t \\ \qquad\qquad \Rrightarrow\!\!*_{\top}\ \exists a'_1, a'_2.\ \lfloor a'_1 \rfloor = \lfloor a_1 \rfloor * \lfloor a'_2 \rfloor = \lfloor a_2 \rfloor * a'_1, a'_2 \in s_2 * a'_1.t < a'_2.t) \end{array}\right\}$$

SM-WRITES-FOLLOW-READS

$$\{ip_i \Mapsto \Phi_i\}$$

$$\langle ip_{client};\ \mathsf{sconnect}(ip_i); \mathsf{let}\ v = \mathsf{sread}(ip_i, k_r)\ \mathsf{in}\ \mathsf{swrite}(ip_i, k_w, v_w); v\rangle$$

$$\left\{ v_o.\ \begin{array}{l} \exists s_1, a_w.\ a_w.k = k_w * a_w.v = v_w * \mathsf{Seen}(a_1, s_1) * a_w \in s_1 \\ * \left(\begin{array}{l} v_o = \mathsf{None} \\ \vee \\ \exists a_r, v_r.\ v_o = \mathsf{Some}(v_r) * a_r.k = k_r * a_r.v = v_r * a_r \in s_1 * a_r.t < a_w.t \\ * \ (\forall a, s_2, a_2.\mathsf{Seen}(a_2, s_2) * a \in s_2 * a_w.t \leq e.t \\ \qquad\qquad \Rrightarrow\!\!*_{\top}\ \exists a'_r, a'_w.\ \lfloor a'_r \rfloor = \lfloor a_r \rfloor * \lfloor a'_w \rfloor = \lfloor a_w \rfloor * a'_r, a'_w \in s_2 * a'_r.t < a'_w.t) \end{array}\right) \end{array}\right\}$$

References

Ralf Jung, Robbert Krebbers, Lars Birkedal, and Derek Dreyer. 2016. Higher-order ghost state. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*. 256–269. https://doi.org/10.1145/2951913.2951943

Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. 2011. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*. 401–416. https://doi.org/10.1145/2043556.2043593