A Appendix

Given an execution history x, the history $x|p$ denotes the subsequence of x for the calls issued by the process p, and the history $x|u$ denotes the subsequence of x for the calls on the update method u. Similarly, $x|g$ denotes the subsequence of x for the calls on the update methods in the group g.

We use the familiar functions $\mathsf{size}(l)$, $\mathsf{prefix}(l, e)$ (excluding e and later elements), and $l \cdot l'$ (concatenation) and the predicate $\mathsf{prefix\text{-}of}(l, l')$ on lists.

Definition 1 (Refinement Relation).
For all K, W *and* τ,
refines(K, W, τ) *iff*
let $\overline{[p_i \mapsto \sigma'_j, A_i, S_i, F_i, L_i]}_{i\in\{1..|P|\}} = K$,
and $\langle \overline{[p_i \mapsto \sigma_i]}_{i\in\{1..|P|\}}, \overline{[p_i \mapsto x_j]}_{i\in\{1..|P|\}} \rangle = W$ *in*

(R_0) *For all* $i, j \in \{1..|P|\}$ *and* u,
$\quad$ *prefix-of*$(x_j|p_i|u, x_i|p_i|u)$.

(R_1) *For all* $i \in \{1..|P|\}$,
$\quad \sigma_i = \mathsf{Apply}(S_i)(\sigma'_i)$

(R_2) *For all* $i, j, k \in \{1..|P|\}$, u, v, r *and* u',
$\quad$ *let* $c = u(v)_{p_i,r}$ *in*
$\quad \langle c, D\rangle \in F_j(p_i) \wedge u' \in \mathsf{Dep}(u) \longrightarrow$
$\quad$ *size*$(\textit{prefix}(x_i, c)|p_k|u') = D(p_k, u')$

(R'_2) *For all* $i, j \in \{1..|P|\}$, u, v, r, g *and* u',
$\quad$ *let* $c = u(v)_{p_i,r}$ *in*
$\quad \langle c, D\rangle \in L_j(g) \wedge \mathsf{Leader}(u) = p_i \wedge u' \in \mathsf{Dep}(u) \longrightarrow$
$\quad$ *size*$(\textit{prefix}(x_i, c)|p_k|u') = D(p_k, u')$

(R_3) *For all* $j, k \in \{1..|P|\}$ *and* u,
$\quad$ *size*$(x_j|p_k|u) = A_j(p_k, u)$

(R_4) *For all* $i, j, k \in \{1..|P|\}$, u, v *and* r,
$\quad$ *let* $c = u(v)_{p_k,r}$ *in*
$\quad \langle c, \_\rangle \in F_j(p_i) \longrightarrow k = i \wedge c \in x_i \wedge c \notin x_j$

(R_5) *For all* $i, j, k \in \{1..|P|\}$, u, v, r *and* g,
$\quad$ *let* $c = u(v)_{p_k,r}$ *in*
$\quad \langle c, \_\rangle \in L_j(g) \wedge \mathsf{SyncGroup}(u) = g \wedge \mathsf{Leader}(g) = p_i \longrightarrow$
$\quad k = i \wedge c \in x_i \wedge c \notin x_j$

(R_6) *For all* $i, j \in \{1..|P|\}$, u, v, *and* r,
$\quad u(v)_{p_i,r} \in x_j \longrightarrow u(v)_{p_i,r} \in x_i$

(R_7) *For all* $i \in \{1..|P|\}$, u, v, *and* r,
$\quad$ *let* $c = u(v)_{p_i,r}$ *in*
$\quad c \in x_j \longrightarrow (p_i, (u(v)_r) \in \tau$

(R_8) *For all* g, $i, j, k \in \{1..|P|\}$, u, v *and* r,
$\quad u(v)_{p_j,r} \in x_k \wedge \mathsf{SyncGroup}(u) = g \wedge \mathsf{Leader}(g) = p_i \longrightarrow$
$\quad j = i$

(R_9) *For all* g *and* $i, j \in \{1..|P|\}$,
$\quad x_i|g \cdot (\textit{map}(\textit{fst}, L_i(g))) = x_j|g \cdot (\textit{map}(\textit{fst}, L_j(g))) \wedge$
$\quad L_{\mathsf{Leader}(g)}(g) = \emptyset$

(R_{10}) *For all* $i, j, k \in \{1..|P|\}$ *and* u
$\quad$ *such that* $\mathsf{SyncGroup}(u) = \bot$,
$\quad x_i|p_k|u \cdot (\textit{map}(\textit{fst}, F_i(p_k))) = x_j|p_k|u \cdot (\textit{map}(\textit{fst}, F_j(p_k)))$

Lemma 4 (Refinement). *For all* K *and* τ, *if* $K_0 \xrightarrow{\tau} K$, *there exists* W, *such that* $W_0 \xrightarrow{\tau} W$ *and refines*(K, W, τ).

Proof. The proof is by induction on the concrete steps with the refinement relation defined in Definition 1.
Case analysis on the concrete step:
For each concrete step, we take one or more abstract steps. We assume the refinement relation for the pre-states and show that the steps preserve it for the post-states.

Case REDUCE:
The abstract steps are CALL for p_j and PROP for other processes.
Since $\mathsf{SyncGroup}(u) = \bot$ and $\mathsf{Dep}(u) = \emptyset$, the conditions CallConfSync and PropDepPres trivially hold. Thus, the abstract steps are enabled.

R_0:
The call by p_j is only added to the history x_j of p_j.

R_1:
The relation R_1 for the post-states holds by R_1 for the pre-states, the summarization property, and the state-commutativity property of u.

R_2:
F maps stay the same and x_j is only extended.

R'_2:
L maps stay the same and x_j is only extended.

R_3:
The call is added to all processes and the record of the applied calls is advanced for all.

R_4:
The maps F stay the same. The call c is added to x_i for each $i \in 1..|P|$. However, by the uniqueness of request identifiers in the trace, $(p_j,\ c) \notin \tau$. Therefore, by the contra-positive of C_3, we have $\langle c, \_\rangle \notin F_i(p_j)$. Therefore, the addition of c to x_i does not invalidate R_4 for any element in $F_i(p_j)$.

R_5:
The map L stays the same. The call c is added to x_i for each $i \in 1..|P|$. However, by the uniqueness of request identifiers in the trace, $(p_j,\ c) \notin \tau$. Therefore, by the contra-positive of C_3, we have $\langle c, \_\rangle \notin L_i(g)$. Therefore, the addition of c to x_i does not invalidate R_5 for any element in $L_i(g)$.

R_6:
Trivial since c is applied at P_j and all other processes p_i, $i \neq j$.

R_7:
Trivial as the added call is in the label.

R_8:
The premise is refuted since $\text{SyncGroup}(u) = \bot$.

R_9:
As $\text{SyncGroup}(u) = \bot$, the step of this case do not apply a method of any synchronization group.

R_{10}:
This case adds the call to the history of all processes and does not change F maps.

Case Free:
The abstract step is Call for p_j.
Since $\text{SyncGroup}(u) = \bot$, the conditions CallConfSync trivially holds. Therefore, the abstract step is enabled.

R_0:
The call by p_j is only added to the history x_j of p_j.

R_1:
The relation R_1 for the post-states holds by R_1 for the pre-states, and the state-commutativity property of u.

R_2:
By the rule Call, the call c is appended to the history x_j In the post-state, $\text{prefix}(x_j, c)$ is equal to x_j before the step. By the rule Free, the dependencies D are a projection over A. By R_3, A represents the size of the sub-histories.

R'_2:
L maps stay the same and x_j is only extended.

R_3:
The call is added to the history of x_j and its record of the applied calls $A_j(p_j)$ is advanced.

R_4:
The call c is added to x_j and all $F_i(p_j)$, $i \neq j$. By the uniqueness of request identifiers in the trace, $(p_j,\ c) \notin \tau$. Therefore, by the contra-positive of R_7, we have $c \notin x_i$. Therefore, R_4 holds in the post-state for the new call c in the F map. Further, similar to the case Reduce, R_4 is preserved for previous elements in F maps as well.

R_5:
The map L stays unchanged. Similar to the case Reduce, by C_3, R_5 is preserved for the elements of L map.

R_6:
Trivial since $i = j$.

R_7:
Trivial as the added call is in the label.

R_8:
The premise is refuted since $\text{SyncGroup}(u) = \bot$.

R_9:
As $\text{SyncGroup}(u) = \bot$, the step of this case do not apply a method of any synchronization group.

R_{10}:
This case adds the call to the history of the local process and does not change its F maps. It adds it to the F maps of other processes and does not change their histories.

Case Conf:
The abstract step is Call for p_j.
Let $c = u(v)_{p_j,r}$.
We show that the condition CallConfSync holds:
By the contra-positive of R_7, for all $k \in \{1..|P|\}$, we have $c \notin x_k$.
Consider arbitrary $k, k' \in \{1..|P|\}$ and $c' = u'(v')_{p_{k'},r'}$ such that $c' \in x_k$ and $c' \bowtie c$.
From $c' \bowtie c$, and $\text{SyncGroup}(u) = g$, we have $u' \in g$.
By R_8 and $\text{Leader}(g) = p_j$, we have $k' = j$.
By R_6, we have $c' \in x_j$.
Thus, the CallConfSync condition holds.
Thus, the abstract step is enabled.

R_0:
The call by p_j is only added to the history x_j of p_j.

R_1:
The relation R_1 for the post-states follow from R_1 for the pre-states and the state-commutativity property of calls in S.

R_2:
F maps stay the same and x_j is only extended.

R'_2:
By the rule Call, the call c is appended to the history x_j In the post-state, $\text{prefix}(x_j, c)$ is equal to x_j before the step. By the rule Conf, the dependencies D are a projection over A. By R_3, A represents the size of the sub-histories.

R_3:
The call is added to the history of x_j and its record of the applied calls $A_j(p_j)$ is advanced.

R_4:
The maps F stay unchanged. Similar to the case Reduce, by C_3, R_4 is preserved for the elements of F maps.

R_5:

The call c is added to x_j and all $L_i(p_j)$, $i \neq j$. By the uniqueness of request identifiers in the trace, $(p_j,\ c) \notin \tau$. Therefore, by the contra-positive of R_7, we have $c \notin x_i$. Therefore, R_5 holds in the post-state for the new call c in the L maps. Further, similar to the case REDUCE, R_5 is preserved for previous elements in $L_j(g)$ as well.

R_6:
Trivial since $i = j$.

R_7:
Trivial as the added call is in the label.

R_8:
The call is added to x_j. Trivial from the premises of the rule CONF.

R_9:
We have that $p_j = \mathsf{Leader}(g)$. This step applies the call to x_j, and appends it to the L_i map for each other process p_i, $i \neq j$. Therefore, the equality is preserved for any pair of js. Further, $L_{\mathsf{Leader}(g)}(g)$ stays empty and the equality is preserved for any pair of i and j.

R_{10}:
This case does not apply since $\mathsf{SyncGroup}(g) \neq \bot$.

Case FREE-APP:
Let the concrete step be for the process p_j.
The abstract step is PROP for p_j.
The condition PropConfSync hold by C_1.
The condition PropDepPres holds as follow:
Let $c = u(v)_{p_i,r}$ and $u' \in \mathsf{Dep}(u)$.
By R_3, $D \leq A$ and R_2, for all $k \in \{1..|P|\}$, we have
$\mathsf{size}(\mathsf{prefix}(x_i, c)|p_k|u') \leq \mathsf{size}(x_j|p_k|u')$
By R_0, we have that $x_i|p_k|u'$ and $x_j|p_k|u'$ are prefixes of $x_k|p_k|u'$. Thus, one is a prefix of another:
$\mathsf{prefix\text{-}of}(x_i|p_k|u, x_j|p_k|u) \vee \mathsf{prefix\text{-}of}(x_j|p_k|u, x_i|p_k|u)$.
From the size equation above, we have:
$\mathsf{prefix\text{-}of}\,(\mathsf{prefix}(x_i, c)|p_k|u', x_j|p_k|u')$
Thus, for all v', $c' = u'(v')_{p_k}$,
$c' \prec_{x_i} c \rightarrow c' \in x_j$
Thus, the condition PropDepPres holds.
The condition $c \in xs(p') \setminus xs(p)$ hold by R_4.
Thus, the abstract step is enabled.

R_0:
Immediate from R_4.

R_1:
The relation R_1 for the post-states follow from R_1 for the pre-states and the state-commutativity property of calls in S.

R_2:
An element is only removed from the F maps and the history x_j is only extended.

R_2':
L maps stay the same and x_j is only extended.

R_3:
The call from p_i is added to the history of x_j and its record of the applied calls $A_j(p_i)$ is advanced.

R_4:
An element is only removed from F. However, the call c from $F_j(p_i)$ is applied in x_j. By C_4, there is no duplicate call in $F_j(p_i)$. Therefore, R_4 is preserved for remaining elements of $F_j(p_i)$.

R_5:
The L maps stay unchanged. However, the call c from $F_j(p_i)$ is applied in x_j. By C_4, there is no duplicate call in $F_j(p_i)$ and $L_j(g)$ (for each $i \in \{1..|P|\}$). Therefore, R_5 is preserved for the elements of $L_j(g)$.

R_6:
It follows from R_4 in the pre-state.

R_7:
Follows from C_3.

R_8:
The call from F is added to x_j. By C_2, $\mathsf{SyncGroup}(u) = \bot$; thus, the premise is refuted.

R_9:
By C_1, this rule does not change the set of methods on synchronization groups.

R_{10}:
The call is removed from F map and added to the history.

Case CONF-APP:
Let the concrete step be for the process p_j and call $c = u(v)$.
The abstract step is PROP for p_j.
The condition PropDepPres holds similar to the case CONF-APP except that instead of the relation R_2, the relation $R_2{}'$ is used.
We show that the condition PropConfSync holds:
Consider arbitrary $i \in \{1..|P|\}$ and $c' = u'(v')_{p_i,r'}$ such that $c' \prec_{x_i} c$ and $c' \bowtie c$. From C_2, we have $u' \in g$. Thus, we consider the group g.
By R_9, we have
$x_i|g \cdot (\mathsf{map}(\mathsf{fst}, L_i(g))) = x_j|g \cdot (\mathsf{map}(\mathsf{fst}, L_j(g)))$ where
$c = u(v) = \mathsf{head}(\mathsf{map}(\mathsf{fst}, L_j(g)))$
We consider two cases:

Case prefix$(x_i|g, x_j|g)$:
From $c' \prec_{x_i} c$, we have $c' \prec_{x_j} c$.
Case prefix$(x_j|g, x_i|g)$:
Thus, prefix$(x_i|g, c) = x_j|g$.
Thus, if $c' \prec_{x_i} c$ then $c' \prec_{x_j} c$.
Thus, the condition PropConfSync holds.
The condition $c \in xs(p') \setminus xs(p)$ hold by R'_4
Thus, the abstract step is enabled.

R_0:
Immediate from R_4.

R_1:
The relation R_1 for the post-states holds by R_1 for the pre-states, and the state-commutativity property of S.

R_2:
The F maps stay the same and the history x_j is only extended.

R'_2:
An element is only removed from the L maps and the history x_j is only extended.

R_3:
The call from p_i is added to the history of x_j and its record of the applied calls $A_j(p_i)$ is advanced.

R_4:
The F maps stay unchanged. However, the call c from $L_j(g)$ is applied in x_j. By C_4, there is no duplicate call in $F_j(p_i)$ and $L_j(g)$ (for each $i \in \{1..|P|\}$). Therefore, R_4 is preserved for the elements of $F_j(p_i)$.

R_5:
An element is only removed from $L_j(g)$. However, the call c from $L_j(g)$ is applied in x_j. By C_4, there is no duplicate call in $L_j(g)$. Therefore, R_5 is preserved for remaining elements of $L_j(g)$.

R_6:
It follows from R_5 in the pre-state.

R_7:
Follows from C_3.

R_8:
The call from L is added to x_j. Thus, the conclusion immediately follows from R_5.

R_9:
This step removes a call from the head of the L list and appends it to the execution history x. Thus, the equality is preserved.

R_{10}:
This case does not apply since by C_2, SyncGroup$(u) \neq \bot$

Case Query:
The abstract step Query is trivially enabled.
By R_1, the two return values v' are equal.

R_0:
The histories stay the same.

R_1:
The states σ and S stay the same.

R_2:
The map F and the histories xs stay the same.

R'_2:
The map L and the histories xs stay the same.

R_3:
The histories and the record of applied calls stay the same.

R_4:
The map F and the histories xs stay the same.

R_5:
The map L and the histories xs stay the same.

R_6:
The histories xs stay the same.

R_7:
The histories xs stay the same and the trace is extended.

R_8:
The histories xs stay the same.

R_9:
The histories xs and the maps L stay the same.

R_{10}:
The histories xs and F maps stay the same.

$\square$

Application of a call c to a state σ, $c(\sigma)$ is naturally lifted to application of an execution history x to a state σ, $x(\sigma)$.

Definition 2 (Locally permissible). *A replicated execution xs is locally permissible, written as* LocalPerm(xs)*, iff every call $c = u(v)_{p,r}$ of xs is permissible in the state resulting from the sub-history of $xs(p)$ before c, i.e., $\mathcal{P}(prefix(xs(p), c)(\sigma_0), c)$.*

Definition 3 (Conflict-synchronizing). *A replicated execution xs is conflict-synchronizing, written as* ConfSync(xs)*, iff*

for every pair of processes p and p' and pair of calls c and c' such that $c \bowtie c'$,
1. $c \in xs(p) \wedge c' \in xs(p') \rightarrow c \in xs(p') \vee c' \in xs(p)$
2. $c' \prec_{xs(p)} c \rightarrow c \not\prec_{xs(p')} c'$

Definition 4 (Dependency-Preserving)**.** *A replicated execution xs is dependency-preserving, written as* $\mathsf{DepPres}(xs)$*, iff for every pair of calls $c = u(v)_{p,r}$ and c' such that $c' \not\parallel c$, if $c \prec_{xs(p)} c'$, then for every process p', $c \prec_{xs(p')} c'$.*

Lemma 5 (Abstract Invariant)**.**
For all W and τ, if $W_0 \xrightarrow{\tau} W$, then
let $\langle \overline{[p_i \mapsto \sigma_i]}_{i\in\{1..|P|\}}, xs\rangle = W$ in
let $\overline{[p_i \mapsto x_i]}_{i\in\{1..|P|\}} = xs$ in
(A_0) For all $i \in \{1..|P|\}$, u, v, and r,
$u(v)_{p_i,r} \in x_j \rightarrow (p_i, (u(v)_r) \in \tau$
(A_1) For all $i \in \{1..|P|\}$,
$\sigma_i = x_i(\sigma_0)$
(A_2) $\mathsf{LocalPerm}(xs)$
(A_3) $\mathsf{ConfSync}(xs)$
(A_4) $\mathsf{DepPres}(xs)$

Proof. The proof is by induction on the steps.
Case analysis on the step:

Case CALL:

A_0:
The call is on the label and is added to $xs(p)$.

A_1:
By the induction hypothesis and the premise $\sigma' = u(v)(\sigma)$.

A_2:
Immediate from the premise $\mathcal{P}(\sigma, c)$.

A_3:
The condition 1 of ConfSync for the new call c: It follows from the premise CallConfSync that $c' \in xs(p)$.
The condition 2 of ConfSync for the new call c: From the contra-positive of A_0, for all p', $c \notin xs(p')$. Therefore, $c \not\prec_{xs(p')} c'$.

A_4:
Immediate as p is the issuing process itself.

Case PROP:

A_0:
$c = u(v)_{p',r}$
Since $c \in xs(p')$, by the induction hypothesis, $(p', (u(v)_r) \in \tau$.

A_1:
By the induction hypothesis and the premise $\sigma' = u(v)(\sigma)$.

A_2:
The two processes p and p' are distinct. The issuing process of the call is p and the call is applied to the process p'.

A_3:
The condition 1 of ConfSync for the new call c: It follows from the premise PropConfSync that $c' \in xs(p)$ and therefore, $c' \in xs'(p)$.
The condition 2 of ConfSync for the new call c: From the premise PropConfSync we have that $c' \prec_{xs(p')} c \rightarrow c' \in xs(p)$. Therefore, $c' \prec_{xs'(p')} c \rightarrow c' \prec_{xs'(p)} c$. Therefore, $c \not\prec_{xs(p')} c'$.

A_4:
Immediate from the premise PropDepPres.

Case QUERY:

A_0:
The histories and the states stay the same.

A_1:
The histories and the states stay the same.

A_2:
The histories stay the same.

A_3:
The histories stay the same.

A_4:
The histories stay the same.

□

Lemma 6 (Convergence)**.** *For all ss, xs, p and p', if $W_0 \rightarrow^* \langle ss, xs\rangle$ and $xs(p) \sim xs(p')$ then $ss(p) = ss(p')$.*

Proof. This lemma follows from the invariant A_3 and Lemma 1 of [39].

□

Lemma 7 (Integrity)**.** *For all ss and p, if $W_0 \rightarrow^* \langle ss, \_\rangle$ then $\mathcal{I}(ss(p))$.*

Proof. This lemma follows from the invariants A_2, A_3 and A_4 and Lemma 2 of [39].

□

Lemma 8 (Concrete Invariants)**.**
For all K, if $K_0 \xrightarrow{\tau} K$, then
let $\overline{[p_i \mapsto \sigma'_j, A_i, S_i, F_i, L_i]}_{i\in\{1..|P|\}} = K$ in
(C_1) For all $i, j \in \{1..|P|\}$, u and v,
$\langle u(v), \_\rangle \in F_i(p_j) \rightarrow \mathsf{SyncGroup}(u) = \bot$
(C_2) For all $i \in \{1..|P|\}$, u, v,

$\langle u(v), \_\rangle \in L_i(g) \longrightarrow \mathsf{SyncGroup}(u) = g$

(C_3) For all $i, j \in \{1..|P|\}$, u, v, *and* r,
let $c = u(v)_{p_i,r}$ *in*
$\langle c, \_\rangle \in F_j(p_i) \vee \langle c, \_\rangle \in L_i(g) \longrightarrow (p_i, (u(v)_r) \in \tau$

(C_4) For all $i, j, k \in \{1..|P|\}$ *and* g,
$map(fst, F_j(p_i)) \cdot map(fst, L_k(g))$ *is an isogram.*

Proof. The proof is by induction on the steps.
Case analysis on the step:

Case Reduce:

C_1:
The F map stays the same.

C_2:
The L map stays the same.

C_3:
The F and L map stays the same.

C_4:
The F and L map stays the same.

Case Free:

C_1:
A premise of the rule Free is $\mathsf{SyncGroup}(u) = \bot$.

C_2:
The L map stays the same.

C_3:
A call is added to the F map that is on the label. The L map stays the same.

C_4:
The L map stays the same. A call is added to the F map. By the uniqueness of call requests in the trace and the contrapositive of C_3, the added call was not previously in F and L.

Case Conf:

C_1:
The F map stays the same.

C_2:
A premise of the rule Conf is $\mathsf{SyncGroup}(u) = g$

C_3:
A call is added to the L map that is on the label. The F map stays the same.

C_4:
The F map stays the same. A call is added to the L map. By the uniqueness of call requests in the trace and the contrapositive of C_3, the added call was not previously in F and L.

Case Free-App:

C_1:
The L map stays the same.

C_2:
A premise of the rule Conf is $\mathsf{SyncGroup}(u) = g$

C_3:
A call is only removed from the F map.

C_4:
A call is only removed from the F map.

Case Conf-App:

C_1:
The F map stays the same.

C_2:
An element from the L is only removed.

C_3:
A call is only removed from the L map.

C_4:
A call is only removed from the L map.

Case Query:

C_1:
The F map stays the same.

C_2:
The L map stays the same.

C_3:
The F and L maps stays the same.

C_4:
The F and L maps stays the same.

□

Corollary 3 (Convergence). *For all* $i, j \in \{1..|P|\}$, *if* $K_0 \longrightarrow^* \overline{[p_i \mapsto \sigma_j, \_, S_i, F_i, L_i]}_{i \in \{1..|P|\}}$ *and* $F_i = F_j = \emptyset$ *and* $L_i = L_j = \emptyset$ *then* $\mathsf{Apply}(S_i)(\sigma_i) = \mathsf{Apply}(S_j)(\sigma_j)$.

Proof. By Lemma 4 (R_9 and R_{10}) we have $xs(p_i) \sim xs(p_j)$. Hence, the conclusion follows from Lemma 6 and Lemma 4 (R_1). □

Corollary 4 (Integrity). *For all $i \in \{1..|P|\}$, if $K_0 \rightarrow^* \overline{[p_i \mapsto \sigma_j, \_, S_i, \_, \_]}_{i \in \{1..|P|\}}$ then $\mathcal{I}(\mathsf{Apply}(S_i)(\sigma_i))$.*

Proof. Immediate from Lemma 4 (R_1) Lemma 7. □

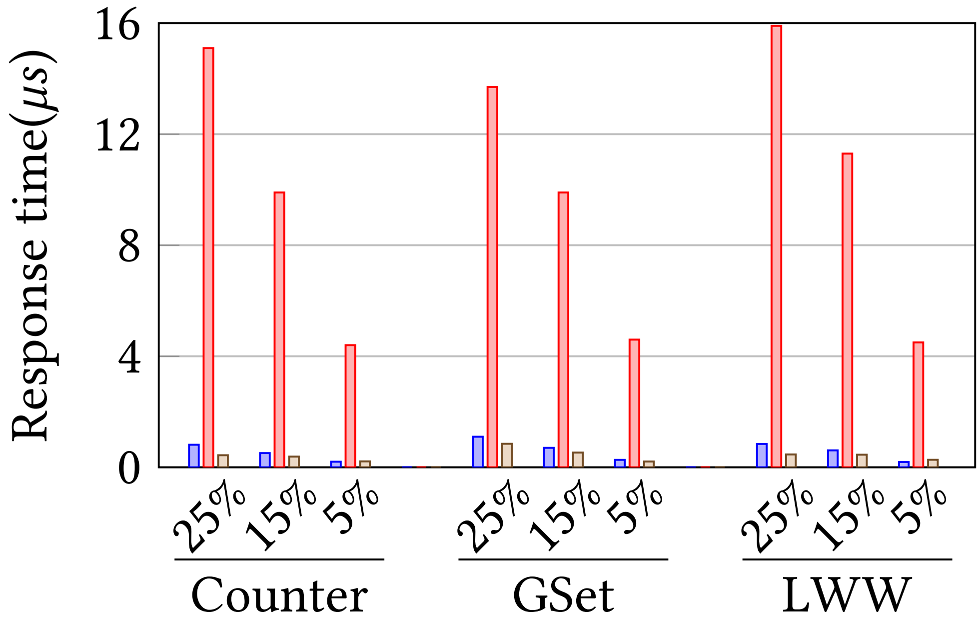

Figure 14. Effect of summarization and remote writes for on response time of reducible methods.

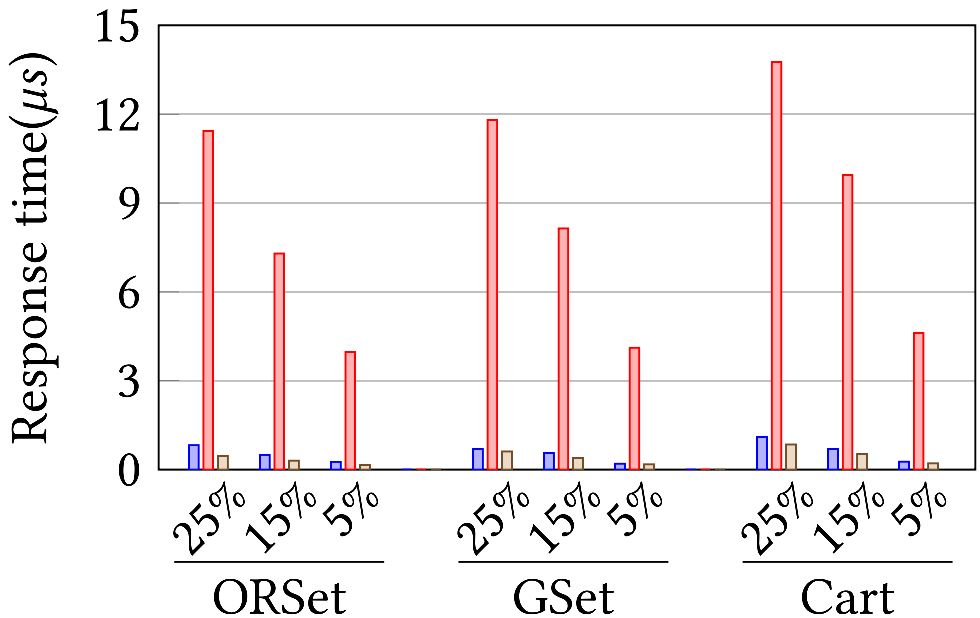

Figure 15. Effect of summarization and remote writes for on response time of irreducible methods.