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 size($l$), prefix($l$, $e$) (excluding $e$ and later elements), and $l \cdot l'$ (concatenation) and the predicate prefix-of($l$, $l'$) on lists.

**Definition 1** (Refinement Relation).

For all $K$, $W$ and $\tau$, $\text{refines}(K, W, \tau)$ if

\[
\text{let } [p_i \mapsto \sigma_j, A_i, S_i, F_i, L_i]_{i \in \{1..|P|\}} = K,
\]

and $\langle [p_i \mapsto \sigma_j]_{i \in \{1..|P|\}}, [p_i \mapsto x_j]_{i \in \{1..|P|\}} \rangle = W$ in

(R0) For all $i, j \in \{1..|P|\}$ and $u$,

\[
\text{prefix-of}(x_j[p_i], u, x_j[p_i][u]) = \text{prefix-of}(x_j[p_i], [p_i \mapsto x_j])
\]

(R1) For all $i \in \{1..|P|\}$,

\[
\sigma_i = \text{Apply}(S_i)_{\text{prefix}(\sigma_i)}
\]

(R2) For all $i, j, k \in \{1..|P|\}$, $u, v, r$ and $u'$,

\[
\text{let } c = u(v)[p_i, r] \rightarrow
\]

\[
\langle c, D \rangle \in F_j(p_i) \land u' \in \text{Dep}(u) \rightarrow
\]

\[
\text{size}(\text{prefix}(x_i, c)[p_k][u']) = D(p_k, u')
\]

(R2') For all $i, j \in \{1..|P|\}$, $u, v, r, g$ and $u'$,

\[
\text{let } c = u(v)[p_i, r] \rightarrow
\]

\[
\langle c, D \rangle \in L_j(g) \land \text{Leader}(u) = g \land \text{Leader}(u) = p_i \rightarrow
\]

\[
\text{size}(\text{prefix}(x_i, c)[p_k][u']) = D(p_k, u')
\]

(R3) For all $j, k \in \{1..|P|\}$ and $u$,

\[
\text{size}(x_j[p_k][u]) = A_j(p_k, u)
\]

(R4) For all $i, j, k \in \{1..|P|\}$, $u, v$ and $r$,

\[
\text{let } c = u(v)[p_i, r] \rightarrow
\]

\[
\langle c, \_ \rangle \in F_j(p_i) \rightarrow k = i \land c \in x_i \land c \notin x_j
\]

(R5) For all $i, j \in \{1..|P|\}$, $u, v, r, g$ and $u'$,

\[
\text{let } c = u(v)[p_i, r] \rightarrow
\]

\[
\langle c, \_ \rangle \in L_j(g) \land \text{SyncGroup}(u) = g \land \text{Leader}(g) = p_i \rightarrow
\]

\[
k = i \land c \in x_i \land c \notin x_j
\]

(R6) For all $i, j \in \{1..|P|\}$, $u, v, r$ and $u'$,

\[
\text{let } c = u(v)[p_i, r] \rightarrow
\]

\[
x_j[p_k][u] \rightarrow (\text{map}(\text{fst}, F_j(p_k))) = x_j[p_k][u] \cdot (\text{map}(\text{fst}, F_j(p_k)))
\]

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

**Proof.** The proof is by induction on the concrete steps with the refinement relation defined in **Definition 1**.

Case analysis on the concrete step:

Case Reduce:
The abstract steps are Call for $p_j$ and Prop for other processes.

Since SyncGroup($u$) = ⊥ and Dep($u$) = Ø, the conditions CallConfSync and PropDepPres trivially hold. Thus, the abstract steps are enabled.

R0:

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

R1:

The relation $R_1$ for the post-states holds by $R_0$ for the pre-states, the summarization property, and the state-commutativity property of $u$.

R2:

$F$ maps stay the same and $x_j$ is only extended.

R3:

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

R4:

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 $(c, \_) \notin F_j(p_j)$. Therefore, the addition of $c$ to $x_j$ does not invalidate $R_4$ for any element in $F_j(p_j)$.

R5:

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 $(c, \_) \notin L_j(g)$. Therefore, the addition of $c$ to $x_j$ does not invalidate $R_5$ for any element in $L_j(g)$.

R6:

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

R7:

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 \( \text{CALL} \) for \( p_j. \)

Since \( \text{SyncGroup}(u) = \bot, \) the conditions \( \text{CallConfSync} \) 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 \( \text{CALL} \), the call \( c \) is appended to the history \( x_j \) in the post-state, \( \text{prefix}(x_j, c) = x_j \) before the step. By the rule \( \text{Free,} \) the dependencies \( D \) are a projection over \( A. \) By \( R_3, A \) represents the size of the sub-histories.

\( R_3': \) \( L \) maps stay the same and \( x_j \) is only extended.

\( R_5: \) The map \( L \) stays unchanged. Similar to the case \( \text{Reduce}, \) 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.

**Case Conf:**

The abstract step is \( \text{CALL} \) for \( p_j. \)

Let \( c = u(\nu)p_{jr}. \)

We show that the condition \( \text{CallConfSync} \) 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 \( c' = u'(\nu')p_{jr}, \) such that \( c' \in x_k \) and \( c' \not< c. \)

From \( c' < c, \) and \( \text{SyncGroup}(u) = g, \) we have \( u' \in g. \)

By \( R_0 \) and \( \text{Leader}(g) = p_j, \) we have \( k' = j. \)

By \( R_6, \) we have \( c' \in x_j. \)

Thus, the \( \text{CallConfSync} \) 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 \( \text{CALL} \), the call \( c \) is appended to the history \( x_j \) in the post-state, \( \text{prefix}(x_j, c) = x_j \) before the step. By the rule \( \text{Conf,} \) 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 \( \text{Reduce}, \) 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 \( c \notin x_j \). Therefore, \( R_5 \) holds in the post-state for the new call \( c \) in the \( L \) maps. Further, similar to the case \( \text{REDUCE} \), \( R_5 \) 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 \( \text{Conf} \).

\( R_9: \)
We have that \( p_j = \text{Leader}(g) \). 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 \( j \). Further, \( L_{\text{Leader}(g)}(g) \) stays empty and the equality is preserved for any pair of \( i \) and \( j \).

\( R_{10}: \)
This case does not apply since \( \text{SyncGroup}(g) = \perp \).

Case \( \text{Free-App} \):
Let the concrete step be for the process \( p_j \).
The abstract step is \( \text{Prop} \) for \( p_j \).
The condition \( \text{PropConfSync} \) hold by \( C_1 \).
The condition \( \text{PropDepPres} \) holds as follows:
Let \( c = u(v)_{p_i,r} \) and \( u' \in \text{Dep}(u) \).
By \( R_3, D \leq A \) and \( R_2 \), for all \( k \in \{1..|P|\} \), we have \( \text{size}(x_j, c)|p_k|u'| \leq \text{size}(x_j, p_k|u') \).
By \( R_6 \), we have that \( x_j|p_k|u' \) and \( x_j|p_k|u' \) are prefixes of \( x_j|p_k|u' \). Thus, one is a prefix of another:
- \( \text{prefix-of}(x_j, p_k|u', x_j|p_k|u) \lor \text{prefix-of}(x_j, p_k|u, x_j|p_k|u) \).
From the size equation above, we have:
- \( \text{prefix-of}(\text{prefix}(x_i, c)|p_k|u', x_j|p_k|u) \).
Thus, for all \( v', c' = u'(v')_{p_k,r} \),
\( c' \triangleq_{x_j} c \mapsto c' \notin x_j \).
Thus, the condition \( \text{PropDepPres} \) holds.
The condition \( c \in \text{xs}(p') \setminus \text{xs}(p) \) 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.

\( R'_0: \)
\( L \) maps stay the same and \( x_j \) is only extended.

\( R_3: \)
The call from \( p_i \) 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 \( C_2 \), \( \text{SyncGroup}(u) = \perp \); 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 \( \text{Conf-App} \):
Let the concrete step be for the process \( p_j \) and call \( c = u(v) \).
The abstract step is \( \text{Prop} \) for \( p_j \).
The condition \( \text{PropDepPres} \) holds similar to the case \( \text{Conf-App} \) except that instead of the relation \( R_9 \), the relation \( R'_2 \) is used.
We show that the condition \( \text{PropConfSync} \) holds:
Consider arbitrary \( i \in \{1..|P|\} \) and \( c' = u'(v')_{p_i,r} \) such that \( c' \triangleq_{x_j} c \) and \( c' = c \). From \( C_2 \), we have \( u' \in g \). Thus, we consider the group \( g \).
By \( R_6 \), we have
\[ x_j|g \cdot (\text{map}(\text{fst}, L_i(g))) = x_j|g \cdot (\text{map}(\text{fst}, L_j(g))) \]
where \( c = u(v) = \text{head}(\text{map}(\text{fst}, L_j(g))) \).
We consider two cases:
Case prefix(\(x_i|g, x_j|g\)):
From \(c' \prec x_j, c\), we have \(c' \prec x_j, c\).
Case prefix(\(x_i|g, x_j|g\)):
Thus, prefix(\(x_i|g, c\)) = \(x_j|g\).
Thus, if \(c' \prec x_j, c\) then \(c' \prec x_j, c\).
Thus, the condition PropConfSync holds.
The condition \(c \in x(s') \setminus x(s)\) 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 \(R_4\) 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 \(C_2, \text{SyncGroup}(u) \neq \perp\).

Case \textbf{Query}:
The abstract step \textbf{Query} is trivially enabled.
By \(R_1\), the two return values \(o'\) are equal.

\(R_0\):
The histories stay the same.

\(R_1\):
The states \(\sigma\) 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.

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

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

\textbf{Definition 3} (Conflict-synchronizing). A replicated execution \(xs\) is conflict-synchronizing, written as \text{ConfSync}(xs), iff
for every pair of processes $p$ and $p'$ and pair of calls $c$ and $c'$ such that $c \equiv c'$,
1. $c \in xs(p) \land c' \in xs(p') \Rightarrow c \in xs(p') \lor c' \in xs(p)$
2. $c' \sim x_{s}(p) c \rightarrow c \not\sim x_{s}(p') c'$

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

**Lemma 5** (Abstract Invariant). For all $W$ and $\tau$, if $W_0 \xrightarrow{\tau} W$, then let $\langle \tau \rangle_{i \in \{1..|P|\}} (\beta, xs) = W$ in
definelet $\{p_i \mapsto xs\}_{i \in \{1..|P|\}} = xs$ in
(A.0) For all $i \in \{1..|P|\}$, $u, v$, and $\tau$, $u(v)p \in \tau 
(A.1) For all $i \in \{1..|P|\}$, $\sigma_i = x_i(\sigma_0)$
(A.2) $\text{LocalPerm}(xs)$
(A.3) $\text{ConfSync}(xs)$
(A.4) $\text{DepPres}(xs)$

**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 $P(\sigma, c)$.

---

A.3: The condition 1 of ConfSync for the new call $c$. It follows from the premise $\text{PropConfSync}$ that $c' \in xs(p)$.
The condition 2 of ConfSync for the new call $c$: From the contra-positive of A.0, for all $p'$, $c \not\in xs(p')$. Therefore, $c \sim x_{s}(p') c'$.

---

A.4: Immediate as $p$ is the issuing process itself.

---

**Case PROP:**

---

A.0: $c = u(v)p'$. Since $c \in xs(p')$, by the induction hypothesis, $(p', (u(v)) \in \tau$.

---

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 $\text{PropConfSync}$ that $c' \in xs(p)$ and therefore, $c' \in xs'(p)$.
The condition 2 of ConfSync for the new call $c$: From the premise $\text{PropConfSync}$ we have that $c' \sim x_{s}(p') c \rightarrow c' \in xs(p)$.

---

A.4: Immediate from the premise $\text{PropDepPres}$.

---

**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 $W_0 \rightarrow^* (ss, xs)$ and $xs(p) \sim ss(p')$ then $ss(p) = xs(p')$.

**Proof.** This lemma follows from the invariant A.3 and Lemma 1 of [39].

---

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

**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 $K_0 \rightarrow K$, then

let $\left[ p_i \mapsto \sigma_i, A_i, S_i, F_i, L_i \right]_{i \in \{1..|P|\}} = K$ in

(C.1) For all $i, j \in \{1..|P|\}$, $u$ and $v$, 
\[ \langle u(v), _ \rangle \in F_i(p_j) \Rightarrow \text{SyncGroup}(u) = \perp \]
(C.2) For all $i \in \{1..|P|\}$, $u$, $v$, 

\( \langle u(v), \_ \rangle \in L_i(g) \rightarrow \text{SyncGroup}(u) = g \)

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

\((C_4)\) For all \( i, j, k \in \{1..|P|\} \) and \( g, \)
\[ \text{map}(\text{fst}, F_j(p_i)) \cdot \text{map}(\text{fst}, L_k(g)) \] 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 \( \text{Free} \) is \( \text{SyncGroup}(u) = \bot. \)

\( C_2: \)
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 contra-positive of \( C_3, \) the added call was not previously in \( F \) and \( L. \)

**Case CONF:**

\( C_1: \)
The \( F \) map stays the same.

\( C_2: \)
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.

\[ \square \]

**Corollary 3 (Convergence).** For all \( i, j \in \{1..|P|\}, \)
if \( K_0 \rightarrow^* [p_i \mapsto \sigma_{j, F_i, L_i}]_{i \in \{1..|P|\}} \) and \( F_i = F_j = \emptyset \) and \( L_i = L_j = \emptyset \) then \( \text{Apply}(S_i)(\sigma_i) = \text{Apply}(S_j)(\sigma_j). \)
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 ($R_1$).

\[ \square \]

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

Proof. Immediate from Lemma 4 ($R_i$) Lemma 7. \[ \square \]
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.