Chapter 9

Strong Guarantees in a System with Imperfect Clocks

In this chapter we assume that clocks run at the same rate as real time but are not initially synchronized, and that message delays are variable and in the range $[d-u, d]$ for some $d \geq u > 0$.

We prove that many operations in linearizable implementations of virtual shared objects must have worst-case response times that are $\Omega(u)$, under reasonable assumptions about the amount of sharing of objects by processes. We show that for many abstract data types, we can provide operations with worst-case time complexities of 0 in sequentially consistent implementations. This shows that linearizability can be more expensive than sequential consistency under our assumptions about process synchrony. Section 9.1 contains our lower bounds on the costs of operations in linearizable implementations of abstract data types, and Section 9.2 contains our sequentially consistent implementations of classes of abstract data types.

To prove our lower bounds, we use shifting techniques which were originally introduced in [LL84] to prove lower bounds on the precision achieved by algorithms for clock synchronization. Shifting is used to change the timing and ordering of events in the system without changing the local view of each process.

In an execution with a certain set of clocks, if process p's history is changed so that the real times at which the events occur are shifted by some amount s and if

p's clock is also shifted by amount s, then the result is another execution in which every process "sees" the same events happening at the same real times. p cannot detect the changes in the real times at which events occur because its clock has changed by the same amount.

The **view** of process p in history π of p with clock C is the concatenation of the sequences of steps in π, arranged in real-time order. The view does not contain the real times the steps occurred. History h of process p with clock C and history h' of process p with clock C' are **equivalent** if p's view is the same in both histories. Execution ρ of system $(P, \mathcal{C})$ and execution ρ' of system $(P, \mathcal{C}')$ are equivalent if the component histories for p in ρ and ρ' are equivalent for all p in P. This means that the processes cannot tell the difference between the two executions.

Given history π of process p with clock C and real number s, a new history $\pi' = shift(\pi, s)$ is defined by $\pi'(t) = \pi(t + s)$ for all t. This means that all tuples are shifted earlier in π' by s if $s > 0$ and later in π' by $-s$ if $s < 0$. Given a clock C and a real number s, a new clock $C' = shift(C, s)$ is defined by $C'(t) = C(t) + s$ for all t. This means that the clock is shifted forward by s if $s > 0$ and backward by s if $s < 0$.

Shifting a history of process p and p's clock by the same amount produces another history. We formally state this in the following lemma.

Lemma 9.1 *[LL84] Let π be a history of process p with clock C, and let s be a real number. Then $shift(\pi, s)$ is a history of p with clock $shift(C, s)$.*

Given execution ρ of system $(P, \mathcal{C})$ and real number s, we define a new execution $\rho' = shift(\rho, p, s)$ by replacing π, p's history in ρ, by $shift(\pi, s)$, and by retaining the same correspondence between sends and receives of messages. (We redefine the correspondence so that a pairing in ρ that involves p's event at time t will involve p's event at time $t - s$ in ρ'.) All tuples for process p are shifted by s, but no other tuples are changed. Given a set of clocks $\mathcal{C} = \{C_q\}_{q \in P}$, and real number s, we define a new set of clocks $\mathcal{C}' = shift(\mathcal{C}, p, s)$ by replacing C_p with $shift(C_p, s)$. Process p's clock is shifted forward by s, but no other clocks are changed.

Shifting the history of one process and its clock by the same amount in an execution results in an execution which is equivalent to the original. We formally state this in the following lemma.

Lemma 9.2 *[LL84] Let ρ be an execution of system $(P,\mathcal{C})$, p be a process, and s be a real number. Let $\mathcal{C}' = shift(\mathcal{C},p,s)$ and $\rho' = shift(\rho,p,s)$. Then ρ' is an execution of $(P,\mathcal{C}')$, and ρ' is equivalent to ρ.*

The following lemma tells how much message delays change when an execution is shifted. Shifting an admissible execution may produce a non-admissible execution.

Lemma 9.3 *[LL84] Let ρ be an execution of system $(P,\mathcal{C})$, p be a process, and s be a real number. Let $\mathcal{C}' = shift(\mathcal{C},p,s)$ and $\rho' = shift(\rho,p,s)$. Suppose x is the delay of message m from process q to process r in ρ. Then the delay of m in ρ' is x if $q \neq p$ and $r \neq p$, $x - s$ if $r = p$, and $x + s$ if $q = p$.*

9.1 Lower Bounds for Linearizability

We now give two algebraic properties of operations which cause individual operations to have a worst-case time complexity of $\Omega(u)$ in linearizable implementations of objects of their abstract data types, under reasonable assumptions about the amount of sharing of objects by processes.

This first theorem shows that an operation must have worst-case time complexity of $\Omega(u)$ if an operation sequence can determine the order in which two instances of the operation were executed.

Theorem 9.1 *Let T be an abstract data type with an operation OP such that $\rho \cdot op^1 \cdot op^2$ does not look like $\rho \cdot op^2 \cdot op^1$ for some operation sequence ρ and some operation instances op^1 and op^2. For any linearizable implementation of an object of type T with at least two modifying processes, $|OP| \geq u/2$.*

Proof The following proof generalizes the proofs in [AW94] that a write operation for a read/write object and the enqueue operation for a queue must take time at least $u/2$. We do not need the assumption about arbitrary initialization of objects to prove this result.

Since $\rho \cdot op^1 \cdot op^2$ does not look like $\rho \cdot op^2 \cdot op^1$, there exists an operation sequence γ such that $\rho \cdot op^1 \cdot op^2 \cdot \gamma$ is legal but $\rho \cdot op^2 \cdot op^1 \cdot \gamma$ is not legal.

Let A be an object of T. Let p_1 and p_2 be two processes that modify A, and let p_3 be a process that performs operations on A. Assume in contradiction that there is a linearizable implementation of A for which $|OP| < u/2$. By the specification of A, there is an admissible execution α such that

- $ops(\alpha)$ is $\rho[A,p_3] \cdot op^1[A,p_1] \cdot op^2[A,p_2] \cdot \gamma[A,p_3]$
- $\rho[A,p_3]$ starts at time 0 and ends at time t, $op^1[A,p_1]$ starts at time t, $op^2[A,p_2]$ starts at time $t+u/2$, and $\gamma[A,p_3]$ starts at time $t+u$
- the message delays in α are d from p_1 to p_2, $d-u$ from p_2 to p_1, and $d-u/2$ for all other ordered pairs of processes

Let $\beta = shift(shift(\alpha, p_1, -u/2), p_2, u/2)$ (shift p_1 later by $u/2$ and p_2 earlier by $u/2$). β is admissible because by Lemma 9.3 the delay of a message from p_1 or to p_2 is $d-u$, the delay of a message from p_2 or to p_1 is d, and all other delays are unchanged. But the linearization of $ops(\beta)$ is
$\rho[A,p_3] \cdot op^2[A,p_2] \cdot op^1[A,p_1] \cdot \gamma[A,p_3]$, which is not legal.

■

Note that the violation of legality in the proof of Theorem 9.1 could be immediate because γ could be empty.

Corollary 9.1 *The following are true in systems with at least two accessing processes:*

- *In any linearizable implementation of augmented or regular queues (Table A.1),* $|ENQ| \geq u/2$ *([AW94]).*

- *In any linearizable implementation of augmented or regular stacks (Table A.2),* $|PUSH| \geq u/2$ *([AW94]).*
- *In any linearizable implementation of read/write objects,* $|WRITE| \geq u/2$ *([AW94]).*
- *In any linearizable implementation of a set with operations* $FINDKEY$ *and* $CHANGEKEY$*, where* $FINDKEY$ *returns the key for a given element and* $CHANGEKEY$ *updates the key for a given element,* $|CHANGEKEY| \geq u/2$.

A variant of Theorem 9.1 can be obtained if the operation instances in the statement of Theorem 9.1 are not in the same generic operation. The conclusion is that at least one of the generic operations must take time at least $u/2$.

Theorem 9.2 *Let* T *be an abstract data type with an operation* OP *such that* $\rho \cdot op_1 \cdot op_2$ *does not look like* $\rho \cdot op_2 \cdot op_1$ *for some operation sequence* ρ *and some operation instances* op_1 *and* op_2*. For any linearizable implementation of an object of type* T *with at least two modifying processes,* $|OP_1| \geq u/2$ *or* $|OP_2| \geq u/2$.

This next theorem shows that an accessor operation must have worst-case time complexity of $\Omega(u)$ if it can determine whether an instance of an operation was executed.

Theorem 9.3 *Let* T *be an abstract data type with an operation* MOP *and an accessor* AOP *such that* $\rho \cdot aop^{before}$ *and* $\rho \cdot mop \cdot aop^{after}$ *are legal but* $\rho \cdot aop^{after}$ *and* $\rho \cdot mop \cdot aop^{before}$ *are not legal for some operation sequence* ρ *and some operation instances* mop*,* aop^{before}*, and* aop^{after}*. For any linearizable implementation of an object of type* T *with at least two accessing processes,* $|AOP| \geq u/2$.

Proof The following proof generalizes the proof in [MR92] that $|READ| \geq u/2$ for read/write objects. Their proof improved a lower bound of $u/4$ in [AW94]. We do not need the assumption about arbitrary initialization of objects to prove this result.

Let A be an object of type T. Let p_1 and p_2 be two processes that access A and let p_3 be a process that performs operations on A.

Assume in contradiction that there exists a linearizable implementation of A for which $|AOP| < u/2$. Let $w = \lceil \frac{|MOP|}{u} \rceil$. By the specification of A, there exists an admissible timed execution α which is as follows:

- $ops(\alpha)|p_3 = \rho[A, p_3] \cdot mop[A, p_3]$, where ρ starts at time 0 and ends at time t, mop's call occurs at time $t + u/2$, and its response occurs at or before time $t + u/2 + |MOP|$.
- $ops(\alpha)|p_1$ is a sequence of $w + 1$ operations $aop^{2i}[A, p_1]$, where i ranges from 0 to w and the i^{th} call occurs at time $t + iu$.
- $ops(\alpha)|p_2$ is a sequence of $w+1$ operations $aop^{2i+1}[A, p_2]$, where i ranges from 0 to w and the i^{th} call occurs at time $t + iu + u/2$.

We assume that α has the following message delays: messages from p_1 to p_2 have delay d, messages from p_2 to p_1 have delay $d - u$, and all others have delay $d - u/2$.

By the definition of w, $t + u/2 + |MOP| < t + u/2 + wu$. Thus, mop has completed before aop^{2w+1} begins. Because of this fact, the linearizability of α, and the accessor property of AOP, aop^{2w+1} is an aop^{after}. Also, since ρ finishes before aop^0 begins, aop^0 completes before time $t + u/2$, and mop does not begin until time $t + u/2$, aop^0 is an aop^{before} by the linearizability of α and the accessor property of AOP. Thus, by the linearizability of α, there exists an index i, $0 \leq i \leq 2w$, such that aop^i is an aop^{before} and aop^{i+1} is an aop^{after}. This implies that the linearization of α is $\rho \cdot aop^0 \cdot \ldots \cdot aop^i \cdot mop \cdot aop^{i+1} \cdot \ldots aop^{2w+1}$. Since AOP is an accessor and the linearization is legal, $aop^1, \ldots, aop^{i-1}$ are of the form aop^{before}, and $aop^{i+1}, \ldots, aop^{2w}$ are of the form aop^{after}. We can assume that i is even so that aop^i is performed by p_1. Let $\beta = shift(shift(\alpha, p_1, -u/2), p_2, u/2)$. β is admissible because by Lemma 9.3, the delay of a message from p_1 is $d - u$, the delay of a message from p_2 is d, and all other message delays are unchanged. In β, ρ precedes all AOP operations, and the order of the AOP operations performed on A by p_1

and p_2 is $aop^1, aop^0, aop^3, aop^2, \ldots, aop^{i+1}, aop^i, \ldots$. If mop is linearized before aop^i in β, then by the accessor property of AOP, $\rho \cdot mop \cdot aop^i$ is legal, a contradiction. If mop is linearized after aop^i in β, then by the accessor property of AOP, $\rho \cdot aop^{i+1}$ is legal, a contradiction. ■

Corollary 9.2 *The following are true in systems with at least two accessing processes:*

- *In any linearizable implementation of augmented queues (Table A.1),* $|PEEK| \geq u/2$.
- *In any linearizable implementation of augmented stacks (Table A.2),* $|PEEK| \geq u/2$.
- *In any linearizable implementation of read/write objects,* $|READ| \geq u/2$ *([MR92]).*
- *In any linearizable implementation of bank account objects (Table A.4),* $|BALANCE| \geq u/2$.
- *In any linearizable implementation of dictionary sets (Table A.3),* $|SEARCH| \geq u/2$.
- *In any linearizable implementation of reference-count sets (Table A.5),* $|FIND| \geq u/2$.

9.2 Upper Bounds for Sequential Consistency

[AW94] gives implementations of sequentially consistent read/write objects and queues in systems with approximately synchronized clocks and uncertainties in message delays. The implementations use an atomic broadcast algorithm to ensure that all processes receive and handle messages in the same order, also preserving the order of messages sent by each individual process.

[AW94] presents two implementations of read/write objects where $|READ| = 0$ and $|WRITE| = h$, and $|READ| = h$ and $|WRITE| = 0$, respectively. They also

present a sequentially consistent implementation of queues where $|ENQ| = 0$ and $|DEQ| = h$. In their atomic broadcast algorithm, $h = 2d$.

[MR92] presents a linearizable implementation of read/write objects, based on a time-slicing method, in which $|READ| = \beta d + 4u + b$ and $|WRITE| = (1-\beta)d + 3u$, where β is a trade-off parameter indicating the relative frequency of reads and b is a positive constant.

We now describe conditions on an abstract data type which allow some operations to be "fast" (i.e., take time 0).

Accessor operations can be made fast.

Theorem 9.4 *Let T be an abstract data type with m generic accessor operations (AOP_i) and n generic modifier operations (MOP_j). Then there exists a sequentially consistent implementation of T with $|AOP_i| = 0$ for each i in $\{1, \ldots, m\}$ and $|MOP_j| = h$ for each j in $\{1, \ldots, n\}$, where h is the maximum time for message delivery by the underlying atomic broadcast algorithm.*

Proof We now explain the details of the algorithm. For each accessor operation, the invoking process just applies the operation to its local copy of the object and returns the result. For each modifier operation, the invoking process uses the atomic broadcast algorithm to send a message to all processes (including itself) containing the invoking process' identification, the name of the operation, the object on which the operation is invoked, and the argument list for the operation. The modifier operation does not complete until the invoking process receives the message and handles it.

We now explain why the algorithm provides sequential consistency. Our proof generalizes the proof in [AW94] for read/write objects. Let ρ be an admissible execution. We systematically build the desired τ. We first order all modifier operations in the order that the atomic broadcast algorithm assigned to their messages. Now we determine where to place the accessor operations. We start from the beginning of ρ. $aop_i^j[X, p]$ goes immediately after the later of (1) the previous operation for process p and (2) the modifier that caused the latest update of p's copy of X before

the generation of the response for the operation. We use process identifiers to break any ties.

Now we prove that $ops(\rho)|p = \tau|p$ for all processes p. Choose some p. By the definition of τ, the relative ordering of two accessor operations in $ops(\rho)|p$ is the same as in $\tau|p$. The properties of atomic broadcast guarantee that the relative ordering of two modifier operations in $ops(\rho)|p$ is the same as in $\tau|p$. If accessor A follows modifier M in $ops(\rho)|p$, then A follows M in τ by the definition of τ. Suppose that accessor A precedes modifier M in $ops(\rho)|p$. Suppose in contradiction that M precedes A in τ. Then in ρ there is some accessor A' and some modifier M' such that the following hold:

1. A' is A or precedes A in ρ
2. M' is M or follows M in ρ
3. M' causes the latest update to p's copy of A's object that precedes A'

A' finishes before M starts in ρ. Since modifier operations are performed in ρ in atomic broadcast order, A' does not see the update performed by M', a contradiction.

We must now prove that τ is legal. We must check all operations, modifiers and accessors, for legality.

Modifiers are performed at every process in atomic broadcast order. Thus, each modifier returns correctly based on all updates handled before it, ensuring that each modifier is legal.

Consider accessor $A = aop_i^j[X, p]$ in τ. Let M be the modifier (performed by process q) in ρ that causes the latest update to p's copy of X preceding A's accessor of p's copy of X. A follows M in τ by the definition of τ. We must show that no other modifier operation on X is placed between M and A in τ. Suppose in contradiction that an M' performed by process r does. The atomic broadcast algorithm ensures that the update for M' follows the update for M at each process in ρ.

Suppose that $r = p$. M' precedes A in ρ by the definition of τ. The update for M' follows the update for M in ρ. Thus A sees M''s update and not M's, contradicting the choice of M.

Suppose that $r \neq p$. By the definition of τ, there is some operation in $ops(\rho)|p$ that precedes A and follows M' in τ (otherwise A would not follow M'). Let O be the first such operation.

Suppose O is a modifier operation on some object Y. O's update to p's copy of Y precedes A's access of p's copy of X. Since updates are performed in atomic broadcast order, the update for M' occurs at p before the update for O, and also before A's access, contradicting the choice of M.

Suppose that O is an accessor operation. By the definition of τ, O is an accessor of X, and M''s update to p's copy of X is the latest one preceding O's access (otherwise O would not follow M'). Since updates are performed in atomic broadcast order, the value from M' supersedes the value from M, contradicting the choice of M. ■

Corollary 9.3 *The following are true:*

- *There exists a sequentially consistent implementation of augmented queues (Table A.1) in which* $|PEEK| = 0$.
- *There exists a sequentially consistent implementation of augmented stacks (Table A.2) in which* $|PEEK| = 0$.
- *There exists a sequentially consistent implementation of read/write objects in which* $|READ| = 0$ *([AW94]).*
- *There exists a sequentially consistent implementation of bank account objects (Table A.4) in which* $|BALANCE| = 0$.
- *There exists a sequentially consistent implementation of dictionary sets (Table A.3) in which* $|SEARCH| = 0$.
- *There exists a sequentially consistent implementation of reference-count sets (Table A.5) in which* $|FIND| = 0$.

Pure modifiers can be made fast, too.

Theorem 9.5 *Let T be an abstract data type with m pure modifier operations (MOP_i) and n other operations (OP_j). Then there exists a sequentially consistent implementation of T with $|MOP_i| = 0$ for each i in $\{1, \ldots, m\}$ and $|OP_j| = h$ for each j in $\{1, \ldots, n\}$, where h is the maximum time for message delivery by the underlying atomic broadcast algorithm.*

Proof We now explain the details of the algorithm. For each operation, the invoking process uses the atomic broadcast algorithm to send a message to all processes (including itself) containing the invoking process' identification, the name of the operation, the name of the object on which the operation is invoked, and the argument list for the operation. MOP operations return immediately, while an OP_j operation does not complete until its invoking process receives the message and handles it.

We now explain why the algorithm guarantees sequential consistency. Let ρ be an admissible execution. We construct τ by ordering all operations in the order that the atomic broadcast algorithm assigned to their messages. Since atomic broadcast preserves the per-process message orders, $\tau|p = ops(\rho)|p$ for each process p. We now show why τ is legal. All pure modifier operations are legal in τ because the return values for pure modifier operations do not depend on the states of the objects on which they are invoked. Now we must show that each op_j^i is legal. In ρ, $op_j^i[X, p]$ returns based on the state of p's copy of object X when its message is handled. The state of p's copy of object X reflects all changes made at all processes before op_i^j's message is handled. Thus $op_j^i[X, p]$ returns a legal value list in τ. ■

Corollary 9.4 *The following are true:*

- *There exists a sequentially consistent implementation of read/write objects in which $|WRITE| = 0$ ([AW94]).*
- *There exists a sequentially consistent implementation of queues (Table A.1) in which $|ENQ| = 0$ ([AW94]).*

- *There exists a sequentially consistent implementation of stacks (Table A.2) in which* $|PUSH| = 0$ *([AW94]).*
- *There exists a sequentially consistent implementation of bank account objects (Table A.4) in which* $|DEPOSIT| = 0$.
- *There exists a sequentially consistent implementation of Increment-Half objects (Table A.6) in which* $|INC| = |HALF| = 0$.

In many abstract data types, accessor operations (respectively, pure modifier operations) satisfy the hypothesis of Theorem 9.3 (respectively, Theorem 9.1), meaning that they can determine whether an instance of an operation has been invoked (respectively, meaning that there is an accessor that can determine the order in which operation instances were invoked); thus, they must take at least $u/2$ time in linearizable implementations of their abstract data types. In contrast, they can take time 0 in a sequentially consistent implementation. Thus, in systems with only approximately synchronized clocks and uncertainty in the network message delay, sequential consistency is less expensive than linearizability for a reasonably large class of abstract data types.

Chapter 10

Hybrid Consistency

Attiya and Friedman [AF92] showed that hybrid consistency is a reasonable consistency guarantee to provide for read/write objects. If weak operations are used mostly, hybrid consistency is cheaper than sequential consistency or linearizability. We want to explore the inherent costs of providing consistent implementations of general abstract data types in order to see if hybrid consistency is always a bargain compared to stronger consistency guarantees. We have determined that hybrid consistency is not necessarily cheaper than stronger consistency guarantees.

The lower bound proofs for hybrid consistency that we present in Section 10.1 are analogous to the lower bound proofs for sequential consistency. In Section 10.2, we use these lower bounds to compare the costs of hybrid consistency to the stronger consistency guarantees of sequential consistency and linearizability.

We assume in this chapter that all processes have perfectly synchronized clocks.

10.1 Lower Bounds

10.1.1 Singles, Pairs, and Trios

This first theorem handles individual operations which immediately do not commute with themselves.

Theorem 10.1 *Let T be an abstract data type with a generic operation OP that immediately does not commute with itself. In any hybrid consistent implementation of T, $|SOP| \geq d$ and $|WOP| \geq d$.*

Proof First we prove that $|SOP| \geq d$. Theorem 8.1 applies here, letting all operations be strong.

Now we prove that $|WOP| \geq d$. This proof is very similar to the proof of Theorem 8.1. Since OP immediately does not commute with itself, there exist a sequence of operations ρ and an operation instance op such that $\rho \cdot op$ is legal but $\rho \cdot op \cdot op$ is not legal.

Let A be an object of type T. Let processes 1 and 2 access A. Suppose in contradiction that there is a hybrid consistent implementation of A for which $|WOP| < d$.

We consider A_ρ, the ρ-initialized version of A.

By the sequential specification for A_ρ, there is some admissible execution α_1 such that $ops(\alpha_1)$ is $Wop[A_\rho, 1]$. There is an admissible execution α_2 such that $ops(\alpha_2)$ is $Wop[A_\rho, 2]$. Since $|WOP| < d$, no messages are received in α_1 and α_2. Thus, replacing p_2's history in α_1 with its history in α_2 results in another admissible execution, α. By assumption, α is hybrid and must satisfy the conditions of Definition 3. Consider τ_1. τ_1 is either $Wop[A_\rho, 1] \cdot Wop[A_\rho, 2]$ or $Wop[A_\rho, 2] \cdot Wop[A_\rho, 1]$, both of which violate the sequential specification for A_ρ, a contradiction. ■

Corollary 10.1 *The following are true:*

- *In any hybrid consistent implementation of augmented or regular queues (Table A.1), $|SDEQ| \geq d$ and $|WDEQ| \geq d$.*
- *In any hybrid consistent implementation of augmented or regular stacks (Table A.2), $|SPOP| \geq d$ and $|WPOP| \geq d$.*
- *In any hybrid consistent implementation of dictionary sets (Table A.3), $|SDEL| \geq d$ and $|WDEL| \geq d$.*
- *In any sequentially consistent implementation of bank account objects (Table A.4), $|SWITHDRAW| \geq d$ and $|WWITHDRAW| \geq d$.*

This next lower bound proof handles pairs of operations which immediately do not commute.

Theorem 10.2 *Let T be an abstract data type containing at least two objects, and let OP_1 and OP_2 be distinct generic operations of T which immediately do not commute. In any hybrid consistent implementation of T, $|SOP_1| + |WOP_2| \geq d$*[1].

Proof The following proof generalizes the proofs in [AF92] that $|SREAD| + |WWRITE| \geq d$ and $|WREAD| + |SWRITE| \geq d$ for read/write objects.

Let A and B be two objects of type T. Let processes 1 and 2 use A and B. Assume in contradiction that there exists some hybrid consistent implementation of A and B for which $|SOP_1| + |WOP_2| < d$.

Since op_1 and op_2 immediately do not commute, there is a sequence α of operations such that $\alpha \cdot op_1$ and $\alpha \cdot op_2$ are legal, but (without loss of generality) $\alpha \cdot op_1 \cdot op_2$ is not legal.

By the sequential specification for A and B, there exists an admissible execution ρ_1 with $ops(\rho_1)$ equal to $S\alpha[A,1]$[2] $\cdot S\alpha[B,2] \cdot Sop_1[A,1] \cdot Wop_2[B,1]$. Assume that $Sop_1[A,1]$ starts at real time t, and $Wop_2[B,1]$ starts immediately after $Sop_1[A_\alpha,1]$ finishes. Because we have assumed that the real time after the end of ρ_1 is less than $t+d$, no process receives a message during ρ_1 after time t about $Sop_1[A,1]$ or $Wop_2[B,1]$.

By the sequential specification for A and B, there exists an admissible execution ρ_2 with $ops(\rho_2)$ equal to $S\alpha[A,1] \cdot S\alpha[B,2] \cdot Sop_1[B,2] \cdot Wop_2[A,2]$. Assume that $Sop_1[B,2]$ starts at real time t, and $Wop_2[A,2]$ starts immediately after $Sop_1[B,2]$ finishes. Because we have assumed that the real time after the end of ρ_2 is less than $t+d$, no process receives a message during ρ_2 after time t about $Sop_1[B,2]$ or $Wop_2[A,2]$.

---

[1] WOP indicates OP's weak version, and SOP indicates OP's strong version. Wop indicates a weak instance of OP, and Sop indicates a strong instance of OP.

[2] If α is an operation sequence, then $S\alpha$ (respectively, $W\alpha$) denotes the strong (respectively, weak) version of α.

Since no messages are received in ρ_1 and ρ_2 after time t, we can produce an admissible hybrid execution ρ by replacing process 2's history in ρ_1 with its history in ρ_2. Thus $ops(\rho)$ consists of the operations $Sop_1[A,1]$ followed by $Wop_2[B,1]$ and $Sop_1[B,2]$ followed by $Wop_2[A,2]$, where $Sop_1[A,1]$ is preceded by $S\alpha[A,1]$ and $Sop_1[B,2]$ is preceded by $S\alpha[B,2]$.

ρ must satisfy the conditions of Definition 3, the definition of hybrid consistency in Section 7.2. We can assume without loss of generality that $Sop_1[A,1]$ precedes $Sop_1[B,2]$ in σ, the serialization of $ops(\rho)$ from Definition 3. We now consider τ_2. Since τ_2 is legal for A and B, each Wop_2 should precede the Sop_1 for the same object. This implies that $Wop_2[A,2]$ precedes $Sop_1[B,2]$ in τ_2, contradicting the order of p_2's operations in ρ.

■

Corollary 10.2 *The following are true:*

- *In any hybrid consistent implementation of read/write objects([AF92]),* $|SREAD| + |WWRITE| \geq d$.
- *In any hybrid consistent implementation of augmented queues (Table A.1),* $|SENQ| + |WPEEK| \geq d$, $|SPEEK| + |WDEQ| \geq d$, *and* $|SENQ| + |WDEQ| \geq d$.
- *In any hybrid consistent implementation of augmented stacks (Table A.2),* $|SPUSH| + |WPEEK| \geq d$, $|SPEEK| + |WPOP| \geq d$, *and* $|SPUSH| + |WPOP| \geq d$.
- *In any hybrid consistent implementation of dictionary sets (Table A.3),* $|SINS| + |WDEL| \geq d$, $|SINS| + |WSEARCH| \geq d$, *and* $|SSEARCH| + |WDEL| \geq d$.
- *In any hybrid consistent implementation of bank account objects (Table A.4),* $|SDEPOSIT| + |WWITHDRAW| \geq d$, $|SDEPOSIT| + |WBALANCE| \geq d$, *and* $|SBALANCE| + |WWITHDRAW| \geq d$.

Changing all strong operations above to weak ones and vice versa also yields true statements.

This next theorem handles operations which have cyclic dependences.

Theorem 10.3 *Let T be an abstract data type with generic operations OP_1 and OP_2 that are cyclically dependent. In any hybrid consistent implementation of objects of type T, the following are true:*

- $|SOP_1| + |SOP_2| \geq 2d$.
- $|SOP_1| + |WOP_2| \geq 2d$.
- $|WOP_1| + |SOP_2| \geq 2d$.
- $|WOP_1| + |WOP_2| \geq 2d$.

Proof A slight variation of the proof of Theorem 8.3 (for sequential consistency) applies here. However, we need to find a violation of hybrid consistency as defined in Definition 3.

We will prove the case $|SOP_1| + |WOP_2| \geq 2d$. The same proof will apply for the other cases after making OP_1 and OP_2 strong or weak as necessary.

Since OP_1 and OP_2 are cyclically dependent, they immediately do not commute. Thus, $|SOP_1| + |WOP_2| \geq d$ by Theorem 10.2.

Since OP_1 and OP_2 are cyclically dependent, there is a sequence of operations ρ and operation instances op_1 and op_2 such that $\rho \cdot op_1$ and $\rho \cdot op_2$ are legal, but $\rho \cdot op_1 \cdot op_2$ and $\rho \cdot op_2 \cdot op_1$ are not legal.

Let A be an object of type T. Let processes 1 and 2 access A. We consider A_ρ, the ρ-initialized version of A.

Assume in contradiction that there exists a hybrid consistent implementation of A for which $|SOP_1| + |WOP_2| < 2d$. Assume that $|SOP_1| \geq |WOP_2|$. (If $|SOP_1| < |WOP_2|$, then essentially the same argument holds.)

By the sequential specification for A_ρ, there is some admissible execution α_1 such that $ops(\alpha_1)$ is $Sop_1[A_\rho, 1]$. Assume that the Sop_1 operation starts at time 0. Then the real time at the end of α_1 is at most $|SOP_1|$.

By the sequential specification for A_ρ, there is some admissible execution α_2 such that $ops(\alpha_2)$ is $Wop_2[A_\rho, 2]$. Assume that the Wop_2 operation starts at time $(|SOP_1| - |WOP_2|)/2$. Then the real time after the end of α_2 is at most $(|SOP_1| - |WOP_2|)/2 + |WOP_2|$, which is less than d. Any message sent by process 2 would not be delivered until at least time $(|SOP_1| - |WOP_2|)/2 + d$, which is more than $|SOP_1|$.

Since no messages are received in α_1 and α_2 before time d, replacing process 1's history in α_2 with its history in α_1 results in another admissible execution, α. By assumption, α is hybrid consistent and must satisfy the conditions of Definition 3. Consider τ_1. τ_1 must be legal for A_ρ and a permutation of $ops(\alpha)$. However, because of cyclic dependency, neither permutation of $ops(\alpha)$ is legal for A_ρ. We have a contradiction. ■

Corollary 10.3 *The following are true:*

- *In any hybrid consistent implementation of objects of type TWOCYCLE,*
  - $|SR1W2| + |SR2W1| \geq 2d$.
  - $|SR1W2| + |WR2W1| \geq 2d$.
  - $|WR1W2| + |SR2W1| \geq 2d$.
  - $|WR1W2| + |WR2W1| \geq 2d$.
- *In any hybrid consistent implementation of objects of type TWOFIVE,*
  - $|SADD2EVEN| + |SADD5DIV5| \geq 2d$.
  - $|SADD2EVEN| + |WADD5DIV5| \geq 2d$.
  - $|WADD2EVEN| + |SADD5DIV5| \geq 2d$.
  - $|WADD2EVEN| + |WADD5DIV5| \geq 2d$.

This next theorem is the hybrid consistent analogue for the sequentially consistent lower bound arising from the noninterleavability of a trio of operations.

Theorem 10.4 *Let T be an abstract data type, and let OP_1, OP_2, and OP_3 be operations of T such that OP_3 is noninterleavable with respect to OP_1 preceding OP_2. Then in any hybrid consistent implementation of objects of type T, at least one of the following is true:*

- $|SOP_1| + |SOP_2| \geq d$, $|SOP_1| + |WOP_2| \geq d$, $|WOP_1| + |SOP_2| \geq d$, *and* $|WOP_1| + |WOP_2| \geq d$, *or*
- $|SOP_3| \geq d$ *and* $|WOP_3| \geq d$

Proof A slight variation of the proof of Theorem 8.4 (for sequential consistency) applies here. However, we need to find a violation of hybrid consistency as defined in Definition 3.

Since OP_3 is noninterleavable with respect to OP_1 and OP_2, there exists an operation sequence ρ and operation instances op_1, op_2, and op_3 such that $\rho \cdot op_3$ and $\rho \cdot op_1 \cdot op_2$ are legal, but none of $\rho \cdot op_3 \cdot op_1 \cdot op_2$, $\rho \cdot op_1 \cdot op_3 \cdot op_2$, and $\rho \cdot op_1 \cdot op_2 \cdot op_3$ is legal.

Let A be an object of type T. Let processes 1 and 2 access A. We consider A_ρ, the ρ-initialized version of A.

Assume in contradiction that there exists a hybrid consistent implementation of A for which the following hold:

- At least one of $|SOP_1| + |SOP_2|$, $|SOP_1| + |WOP_2|$, $|WOP_1| + |SOP_2|$, and $|WOP_1| + |WOP_2|$ is less than d.
- $|SOP_3| < d$ or $|WOP_3| < d$.

Let us prove the case when $|SOP_1| + |WOP_2| < d$ and $|WOP_3| < d$. The other cases are handled similarly.

By the sequential specification for A_ρ, there is some admissible execution α_1 such that $ops(\alpha_1)$ is $Sop_1[A_\rho, 1] \cdot Wop_2[A_\rho, 1]$. Assume that the Sop_1 operation starts at time 0 and that the Wop_2 operation starts immediately after the Sop_1

operation finishes. Because the real time after the end of α_1 is less than d, no process receives a message during α_1.

By the sequential specification for A_ρ, there is some admissible execution α_2 such that $ops(\alpha_1)$ is $Wop_3[A_\rho, 2]$. Assume that the Wop_3 operation starts at time 0. Because the real time after the end of α_2 is less than d, no process receives a message during α_2.

Since no messages are received in α_1 and α_2, replacing process 1's history in α_2 with its history in α_1 results in another admissible execution, α. By assumption, α is hybrid consistent and must satisfy the conditions of Definition 3. Consider τ_1. τ_1 must be legal for A_ρ and a permutation of $ops(\alpha)$. However, because of noninterleavability, none of the three permutations of $ops(\alpha)$ which satisfies the order of process 1's operations is legal for A_ρ. We have a contradiction. ■

Corollary 10.4 *In any hybrid consistent implementation of objects of the abstract data type described in Corollary 8.4, at least one of the following is true:*

- $|SW1| + |SR2| \geq d$, $|SW1| + |WR2| \geq d$, $|WW1| + |SR2| \geq d$, *and* $|WW1| + |WW2| \geq d$, *or*
- $|SR1W2| \geq d$ *and* $|WR1W2| \geq d$.

10.1.2 All Operations

As in the sequentially consistent case (Section 8.3), we use the lower bound results about single operations and pairs of operations from the previous subsection and the structure of the commutativity graphs to determine lower bounds on the worst-case time complexity for all operations of abstract data types in hybrid consistent implementations.

We now give lower bounds on the worst-case completion time for all operations in hybrid consistent implementations of abstract data types with cliques in their commutativity graphs.

Theorem 10.5 *Let T be an abstract data type with operations $OP_1, OP_2, \ldots, OP_n$ such that for all $i \in \{1, \ldots, n\}$, OP_i immediately does not commute with OP_j if $i \neq j$. Suppose s of the operations immediately do not commute with themselves. In any hybrid consistent implementation of T,*

- *If $n - s$ is odd, then $\sum_{i=1}^{n}(|SOP_i| + |WOP_i|) \geq 2sd + (n - s - 1)d$.*
- *If $n - s$ is even, then $\sum_{i=1}^{n}(|SOP_i| + |WOP_i|) \geq 2sd + (n - s)d$.*

Proof Let $OP_{i_1}, \ldots, OP_{i_s}$ be the operations which immediately do not commute with themselves. By Theorem 10.1, $|SOP_{i_k}| \geq d$ and $|WOP_{i_k}| \geq d$ for all k in $\{1, \ldots, s\}$. Thus, $\sum_{k=1}^{s}(|SOP_{i_k}| + |WOP_{i_k}|) \geq 2sd$.

Let $OP_{j_1}, \ldots, OP_{j_{n-s}}$ be the remaining operations. By Theorem 10.2, $|SOP_{j_l}| + |WOP_{j_m}| \geq d$ and $|WOP_{j_l}| + |SOP_{j_m}| \geq d$ for all $l \neq m$.

Suppose that $n - s$ is odd. We can add the specific inequalities $|SOP_{j_k}| + |WOP_{j_{k+1}}| \geq d$ and $|WOP_{j_k}| + |SOP_{j_{k+1}}| \geq d$ for each odd k in $\{1, \ldots, n - s - 2\}$, rearranging terms where necessary, to obtain $\sum_{k=1}^{n-s-1}(|SOP_{j_k}| + |WOP_{j_k}|) \geq (n - s - 1)d$. We can add this inequality to the one in the first paragraph to obtain the desired result ($|SOP_{j_{n-s}}|$ and $|WOP_{j_{n-s}}|$ do not affect the outcome since they are nonnegative).

Now suppose that $n - s$ is even. We can add the specific inequalities $|SOP_{j_k}| + |WOP_{j_{k+1}}| \geq d$ and $|WOP_{j_k}| + |SOP_{j_{k+1}}| \geq d$ for each odd k in $\{1, \ldots, n - s - 1\}$, rearranging terms where necessary, to obtain $\sum_{k=1}^{n-s}(|SOP_{j_k}| + |WOP_{j_k}|) \geq (n - s - 1)d$. We can add this inequality to the one in the first paragraph to obtain the desired result. ■

Corollary 10.5 *The following are true:*

- *In any hybrid consistent implementation of read/write objects, $|SREAD| + |WREAD| + |SWRITE| + |WWRITE| \geq 2d$ ([AF92]).*
- *In any hybrid consistent implementation of reference-count sets, $|SINS| + |WINS| + |SUP| + |WUP| + |SFIND| + |WFIND| \geq 2d$.*

- *In any hybrid consistent implementation of increment/read objects,* $|SREAD| + |WREAD| + |SINC| + |WINC| \geq 2d$.

- *In any hybrid consistent implementation of half/read objects,* $|SREAD| + |WREAD| + |SHALF| + |WHALF| \geq 2d$.

- *In any hybrid consistent implementation of read/square root objects,* $|SREAD| + |WREAD| + |SSQRT| + |WSQRT| \geq 2d$.

- *In any hybrid consistent implementation of a reference-count set with a delete operation,* $|SDEL| + |WDEL| + |SINS| + |WINS| + |SUP| + |WUP| + |SFIND| + |WFIND| \geq 2d + 2d = 4d$.

- *In any hybrid consistent implementation of a bounded double-ended peek queue (where the peek operation returns the contents at each end of the queue),* $|SBACKDEQ| + |WBACKDEQ| + |SFRONTDEQ| + |WFRONTDEQ| + |SBACKENQ| + |WBACKENQ| |SFRONTENQ| + |WFRONTENQ| + |SPEEK| + |WPEEK| \geq 4d + 2d = 6d$.

As in the sequentially consistent case, we can give lower bounds on the costs of hybrid consistent implementations abstract data types with more general commutativity graphs.

Theorem 10.6 *Let* T *be an abstract data type with operations* $OP_1, OP_2, \ldots, OP_n$ *and commutativity graph* $CG(T)$. *In any hybrid consistent implementation of* T, $\sum_{i=1}^{n}(|SOP_i| + |WOP_i|) \geq 2(|NSC(T)| + |Maxdom(RCG(T))|)d$.

Proof If OP_i immediately does not commute with itself, then $|SOP_i| \geq d$ and $|WOP_i| \geq d$ by Theorem 10.1. Thus, $\sum_{OP_i \in NSC(T)} |OP_i| \geq 2|NSC(T)|d$. Let (OP_i, OP_j) be an edge in $Maxdom(RCG(T))$. By Theorem 10.2, $|SOP_i| + |WOP_j| \geq d$ and $|WOP_i| + |SOP_j| \geq d$. By adding together these inequalities and rearranging terms, we obtain the desired result. ■

10.2 Comparison with Sequential Consistency and Linearizability

We have displayed a number of impossibility results for hybrid consistent implementations of abstract data types. Some of these impossibility results hold for individual weak operations, meaning that they must be slow. Since hybrid consistency is weaker than sequential consistency and linearizability, intuition tells us that there may be some abstract data types in which hybrid consistent implementations of weak operations are faster than their sequentially consistent (or linearizable) counterparts. [AF92] describes a hybrid consistent implementation of read/write objects in which weak reads and weak writes are "fast" (take time 0) and strong reads and strong writes take time $O(d)$. Is it always possible to develop hybrid consistent implementations with similar time bounds (ideally, weak operations that are faster than strong operations)? We show below that it is not always possible to develop hybrid consistent implementations in which weak operations are faster than strong operations.

Why could weak operations for read/write objects be optimized? One plausible explanation is that read/write objects have simple semantics. It may be possible to optimize weak operations for other abstract data types with simple semantics. Let us investigate the possibility of optimizing weak operations for one such class of abstract data types, the PMR objects defined in Section 8.4.

For concreteness, we will consider read/add/multiply objects. $read()(v)[X,p]$ is legal if the value stored in X is v. The effect of $add(v)()[X,p]$ is to add v to the value stored in X. The effect of $mul(v)()[X,p]$ is to multiply v to the value stored in X.

Let X be one such object, initialized to 1. Suppose there is a hybrid consistent implementation of X for which $|WADD| = |WMUL| = |WREAD| = 0$. Consider the following admissible execution, ρ, as follows:

- $ops(\rho)|p_1$ is $Wadd(5)()[X,p_1] \cdot Wread()(6)[X,p_1]$, where the add starts at time 0 and the read completes before time d.

- $ops(\rho)|p_2$ is $Wmul(3)()[X,p_2] \cdot Wread()(3)[X,p_2]$, where the multiply starts at time 0 and the read completes before time d.

This execution is not hybrid. The reason why p_1's read must return 6 is because the execution is indistinguishable to p_1 from an execution in which p_1 is running alone. p_2's read must return 3 for a similar reason.

In any total ordering of the operations of ρ satisfying the definition of hybrid consistency, at most one of the reads will be legal.

We now use the insight gained from this example to yield a new lower bound on the worst-case time complexity for weak operations in hybrid consistent implementations of abstract data types.

A generic operation OP is **doubly noninterleavable with respect to** OP_1 **and** OP_2 if there exist operation sequence ρ and operation instances op^1, op^2, op_1, and op_2 such that $\rho \cdot op_1 \cdot op^1$ and $\rho \cdot op_2 \cdot op^2$ are legal, but there is no way to place both op^1 and op^2 after ρ in $\rho \cdot op_1 \cdot op_2$ and $\rho \cdot op_2 \cdot op_1$ to form a legal sequence.

Theorem 10.7 *Let T be an abstract data type, and let AOP, OP_1, and OP_2 be operations on objects of type T, where AOP is a generic accessor operation. Suppose AOP is doubly noninterleavable with respect to OP_1 and OP_2. Then, in any hybrid consistent implementation of objects of type T, $|WOP_1|+|WAOP| \geq d$ or $|WOP_2|+|WAOP| \geq d$.*

Proof Let A be an object. Let processes 1 and 2 use A. Assume in contradiction that there exists some hybrid consistent implementation of A for which $|WOP_1| + |WAOP| < d$ and $|WOP_2| + |WAOP| < d$.

Since AOP is doubly noninterleavable with respect to OP_1 and OP_2, there exists an operation sequence α and operation instances aop^1, aop^2, op_1, and op_2 such that $\alpha \cdot op_1 \cdot aop^1$ and $\alpha \cdot op_2 \cdot aop^2$ are legal, but there is no way to place both aop^1 and aop^2 after ρ in $\rho \cdot op_1 \cdot op_2$ and $\rho \cdot op_2 \cdot op_1$ to form a legal sequence.

We consider A_α, the α-initialized version of A.

By the sequential specification for A_α, there exists an admissible execution ρ_1 with $ops(\rho_1)$ equal to $Wop_1[A_\alpha,1] \cdot Waop^1[A_\alpha,1]$. $Wop_1[A_\alpha,1]$ starts at real time 0,

and $Waop^1[A_\alpha, 1]$ starts immediately after $Wop_1[A_\alpha, 1]$ finishes. Because we have assumed that the real time after the end of ρ_1 is less than d, no process receives a message during ρ_1.

By the sequential specification for A_α, there exists an admissible execution ρ_2 with $ops(\rho_2)$ equal to $Wop_2[A_\alpha, 2] \cdot Waop^2[A_\alpha, 2]$. $Wop_2[A_\alpha, 2]$ starts at real time 0, and $Waop^2[A_\alpha, 2]$ starts immediately after $Wop_2[A_\alpha, 2]$ finishes. Because we have assumed that the real time after the end of ρ_2 is less than d, no process receives a message during ρ_2.

Since no messages are received in ρ_1 and ρ_2, we can produce an admissible hybrid execution ρ by replacing process 2's history in ρ_1 with its history in ρ_2.

By assumption, ρ is hybrid and satisfies the conditions of Definition 3. Consider τ_1. In τ_1, $Wop_1[A_\alpha, 1]$ precedes $Waop^1[A_\alpha, 1]$. If $Wop_1[A_\alpha, 1]$ precedes $Wop_2[A_\alpha, 2]$, then by the double noninterleavability property, all possible placements of both $Waop^1[A_\alpha, 1]$ and $Waop^2[A_\alpha, 2]$ in $Wop_1[A_\alpha, 1] \cdot Wop_2[A_\alpha, 2]$ are illegal for A_α. If $Wop_2[A_\alpha, 2]$ precedes $Wop_1[A_\alpha, 1]$, then by the double noninterleavability property, all possible placements of both $Waop^1[A_\alpha, 1]$ and $Waop^2[A_\alpha, 2]$ in $Wop_2[A_\alpha, 2] \cdot Wop_1[A_\alpha, 1]$ are illegal for A_α. Thus, We cannot produce a τ_1 that is legal for A_α because we need to place all operations. ■

Corollary 10.6 *The following are true:*

- *In any hybrid consistent implementation of read/add/multiply objects,* $|WADD| + |WREAD| \geq d$ *or* $|WMUL| + |WREAD| \geq d$.
- *In any hybrid consistent implementation of PMR objects (Section 8.4) with one modifier operation* MOP *and a write operation,* $|WMOP| + |WREAD| \geq d$ *or* $|WWRITE| + |WREAD| \geq d$.

For the above classes of objects, the lower bound on total time complexity of the operations matches the upper bound given in the sequentially consistent (and linearizable) implementations from Theorem 8.10. Thus, for these classes of objects, sequential consistency and linearizability cost no more than hybrid consistency, even when weak operations are mostly used.

A natural question arises from this. Are there other abstract data types for which sequential consistency and linearizability cost no more than hybrid consistency?

Let us consider hybrid implementations and linearizable implementations of queues. By Corollary 10.1, $|SDEQ| \geq d$ and $|WDEQ| \geq d$. Hybrid dequeue operations are inherently expensive. In contrast, in systems with perfectly synchronized clocks, Attiya and Welch [AW94] presented a linearizable implementation of queues in which enqueues complete in time 0 and dequeues complete in time d. In their implementation, all operations are strong. The lower bound on the time required to implement weak operations in hybrid consistency matches the upper bound on the time required to implement linearizability in this case, and weak dequeues must be slow. Thus, hybrid consistency gives us no advantage for queues.

We conclude that hybrid consistency does not necessarily give performance gains for every abstract data type.

Theorem 10.8 *Let T be an abstract data type with operations $OP_1, \ldots, OP_n$ such that $S = \{OP_i | OP_i$ immediately does not commute with itself$\}$ is nonempty. Suppose that the users of a hybrid implementation of objects of T only use weak operations in S. Then the users of the hybrid implementation can switch over to a linearizable implementation without an increase in worst-case completion time in a system with perfectly synchronized clocks.*

Proof Since we can guarantee that all processes receive and handle messages at the same time and in the same order, there exists a linearizable implementation of objects of type T in which all operations take time d. It works as follows. When an operation is invoked at a process, the process sends a message about it and returns d later when the message is handled.

By Theorem 10.1, $|WOP_i| \geq d$ for all OP_i in S. In our linearizable implementation $|OP_i| = d$ for all OP_i in S. Thus $\sum_{OP_i \in S} |WOP_i| \geq \sum_{OP_i \in S} |OP_i|$. ■

We may not always have the luxury of perfectly synchronized clocks. However, in systems with only approximately synchronized clocks, we can deduce that the

inherent cost of weak operations in S for hybrid consistent implementations is $\Omega(d)$, while the actual cost of operations in S in a sequentially consistent implementation is $O(d)$ (using a simple algorithm with atomic broadcast). Thus, providing hybrid consistency does not yield any major gains.

We have shown in this subsection that hybrid consistency is not necessarily cheaper to implement than the stronger guarantees of sequential consistency and linearizability, even when weak operations are mostly used.

Chapter 11

Summary and Partial Extensions

We have studied the impact of algebraic properties of operations, the degree of process synchronization, and the type of consistency guarantee on the time complexities of distributed implementations of abstract data types. As a result, we have shown that sequential consistency and linearizability are equally costly in systems with perfectly synchronized clocks under certain reasonable assumptions, that sequential consistency is cheaper than linearizability in systems with only approximately synchronized clocks under certain reasonable assumptions, and that hybrid consistency is not necessarily cheaper than the stronger consistency guarantees.

We were unsuccessful in finding an algebraic property of operations such that an operation could be optimized in a linearizable implementation assuming perfect process synchrony if it satisfied the property and it could not be optimized otherwise. Although we were unsuccessful, we determined a reasonably general algebraic property, self-obliviousness, such that if an operation is self-oblivious, then it can be optimized in a linearizable implementation assuming perfect process synchrony. We described a general implementation in which a self-oblivious operation is optimized in Theorem 8.7. However, we still do not know if we can optimize an operation which is not self-oblivious.

In this chapter, we discuss how our implementation works when we attempt to optimize an operation that is not self-oblivious (Section 11.1), improvements to our implementation for specific data types (Section 11.2), and progress towards

optimizing *multiple* operations of abstract data types (Section 11.3). We assume that processes have perfectly synchronized clocks.

11.1 Optimizing an Operation Which Is Not Self-oblivious

Theorem 8.7 describes a general linearizable implementation of an abstract data type in which a single self-oblivious operation is optimized. Does our implementation from Theorem 8.7 still work when the operation to be optimized is not self-oblivious? The operation must immediately commute with itself, because otherwise the impossibility result from Theorem 8.1 would apply, causing the operation to take at least time d. The answer to our question is "No", and we explain why with a specific abstract data type.

Consider the abstract data type CONDARRAY, which we describe in Figure 11.1. The objects are two-element arrays. $r2w1$ commutes with itself. Suppose we want to optimize $|R2W1|$ in an instantiation of the algorithm described in the proof of Theorem 8.7.

Suppose that $X[1]$ is initialized to 4 and $X[2]$ is initialized to 5. The following execution ρ is a possible execution of the CONDARRAY implementation:

- At time 0, process 1 starts a $r1condw2$ operation with input argument 4 on object X. It completes at time $2d$.
- At time $d/2$, process 2 executes an $r2w1$ operation with input argument 1 on object X.
- At time $5d/4$, process 3 executes an $r2w1$ operation with input argument 3 on object X.

Process 2 returns 5 for its $r2w1$ operation when it completes. By the time process 3 executes its operation, it has only received the message about process 1's operation. Thus, it returns 4 for its $r2w1$ operation when it completes.

$r2w1(X, v)$:

$X[1] := v$

$return(X[2])$

$r1condw2(X, v)$:

if $X[1] = v$ **then**

$X[2] := v$

end if

$ack(Ok)$

Figure 11.1: The CONDARRAY Abstract Data Type

Our τ is as follows:
$r2w1(1)(5)[X, 1] \cdot r1condw2(4)(Ok)[X, 2] \cdot r2w1(3)(4)[X, 3]$, which is illegal.

The problem is that the fast operations do not necessarily behave in the same way when they are globally executed as when they are locally executed. Although $R2W1$ operations do not immediately affect each other, they may indirectly affect each other due to other kinds of operations. Thus, $R2W1$ is not self-oblivious. However, $R1CONDW2$ is self-oblivious, and we can instantiate the implementation in Theorem 8.7 to optimize $|R1CONDW2|$.

This work shows that we need a different approach when trying to optimize an operation which is not self-oblivious.

11.2 Improvements for Specific Data Types

The implementation from Theorem 8.7 optimizes one operation but is overly conservative for other operations; in essence, it assumes that the optimized operation has a cyclic dependence with every other operation because every other operation takes time $2d$ (Theorem 8.3).

If the operation to be optimized has no cyclic dependences with other operations, we may be able to improve the time bounds for the other operations. We now give