SOLUTIONS TO CHAPTER 5 PROBLEMS – SYNCHRONIZATION

1. Q: Name at least three sources of delay that can be introduced between WWV broadcasting the time and the processors in a distributed system setting their internal clocks.
A: First we have signal propagation delay in the atmosphere. Second we might have collision delay while the machines with the WWV receivers fight to get on the Ethernet. Third, there is packet propagation delay on the LAN. Fourth, there is delay in each processor after the packet arrives, due to interrupt processing and internal queueing delays.

2. Q: Consider the behavior of two machines in a distributed system. Both have clocks that are supposed to tick 1000 times per millisecond. One of them actually does, but the other ticks only 990 times per millisecond. If UTC updates come in once a minute, what is the maximum clock skew that will occur?
A: The second clock ticks 990,000 times per second, giving an error of 10 msec per second. In a minute this error has grown to 600 msec. Another way of looking at it is that the second clock is one percent slow, so after a minute it is off by 0.01 ··60 sec, or 600 msec.

3. Q: Add a new message to Fig. 5-7 that is concurrent with message *A*, that is, it neither happens before *A* nor happens after *A*.
A: The solution cannot involve 0 or it would be ordered. Thus it must be a message from 1 to 2 or from 2 to 1. If it departs or arrives from 1 after 16, it will be ordered with respect to *A*, so it must depart or arrive before 16. The possibilities are a message leaving process 2 at 0 and arriving at process 1 at 8, or a message leaving process 1 at 0 and arriving at process 2 at 10. Both of these are concurrent with *A*.

4. Q: To achieve totally-ordered multicasting with Lamport timestamps, is it strictly necessary that each message is acknowledged?
A: No, it is sufficient to multicast any other type of message, as long as that message has a timestamp larger than the received message. The condition for delivering a message *m* to the application, is that another message has been received from each other process with a large timestamp. This guarantees that there are no more messages underway with a lower timestamp.

5. Q: Consider a communication layer in which messages are delivered only in the order that they were sent. Give an example in which even this ordering is unnecessarily restrictive.
A: Imagine the transfer of a large image which, to that end, has been divided into consecutive blocks. Each block is identified by its position in the original image, and possibly also its width and height. In that case, FIFO ordering is not necessary, as the receiver can simply paste each incoming block into the correct position.

6. Q: Suppose that two processes detect the demise of the coordinator simultaneously and both decide to hold an election using the bully algorithm. What happens?
A: Each of the higher-numbered processes will get two *ELECTION* messages, but will ignore the second one. The election will proceed as usual.

7. Q: In Fig. 5-12 we have two *ELECTION* messages circulating simultaneously. While it does no harm to have two of them, it would be more elegant if one could be killed off. Devise an algorithm for doing this without affecting the operation of the basic election algorithm.
A: When a process receives an *ELECTION* message, it checks to see who started it. If it itself started it (i.e., its number is at the head of the list), it turns the message into a

COORDINATOR message as described in the text. If it did not start any *ELECTION* message, it adds its process number and forwards it along the ring. However, if it did send its own *ELECTION* message earlier and it has just discovered a competitor, it compares the originator's process number with its own. If the other process has a lower number, it discards the message instead of passing it on. If the competitor is higher, the message is forwarded in the usual way. In this way, if multiple *ELECTION* messages are started, the one whose first entry is highest survives. The rest are killed off along the route.

8. Q: Many distributed algorithms require the use of a coordinating process. To what extent can such algorithms actually be considered distributed? Discuss.

A: In a centralized algorithm, there is often one, fixed process that acts as coordinator. Distribution comes from the fact that the other processes run on different machines. In distributed algorithms with a nonfixed coordinator, the coordinator is chosen (in a distributed fashion) among the processes that form part of the algorithm. The fact that there is a coordinator does not make the algorithm less distributed.

9. Q: In the centralized approach to mutual exclusion (Fig. 5-13), upon receiving a message from a process releasing its exclusive access to the critical region it was using, the coordinator normally grants permission to the first process on the queue. Give another possible algorithm for the coordinator.

A: Requests could be associated with priority levels, depending on their importance. The coordinator could then grant the highest priority request first.

10. Q: Consider Fig. 5-13 again. Suppose that the coordinator crashes. Does this always bring the system down? If not, under what circumstances does this happen? Is there any way to avoid the problem and make the system able to tolerate coordinator crashes?

A: Suppose that the algorithm is that every request is answered immediately, either with permission or with denial. If there are no processes in critical regions and no processes queued, then a crash is not fatal. The next process to request permission will fail to get any reply at all, and can initiate the election of a new coordinator. The system can be made even more robust by having the coordinator store every incoming request on disk *before* sending back a reply. In this way, in the event of a crash, the new coordinator can reconstruct the list of active critical regions and the queue by reading file from the disk.

11. Q: Ricart and Agrawala's algorithm has the problem that if a process has crashed and does not reply to a request from another process to enter a critical region, the lack of response will be interpreted as denial of permission. We suggested that all requests be answered immediately, to make it easy to detect crashed processes. Are there any circumstances where even this method is insufficient? Discuss.

A: Suppose a process denies permission and then crashes. The requesting process thinks that it is alive, but permission will never come. One way out is to have the requester not actually block, but rather go to sleep for a fixed period of time, after which it polls all processes that have denied permission to see if they are still running.

12. Q: How do the entries in Fig. 5-16 change if we assume that the algorithms can be implemented on a LAN that supports hardware broadcasts?

A: Only the entries for the distributed case change. Because sending a pointto-point message is as expensive as doing a broadcast, we need only send one broadcast message to all processes requesting the entry to the critical section. Likewise, only one exit broadcast message is needed. The delay becomes $1 \cdot (n - \cdot 1)$: one delay coming from the broadcast request, and an additional $n - \cdot 1$ as we still need to receive a message from each other process before being allowed to enter the critical section.

13. Q: A distributed system may have multiple, independent critical regions. Imagine that process 0 wants to enter critical region *A* and process 1 wants to enter critical region *B*. Can Ricart and Agrawala's algorithm lead to deadlocks? Explain your answer.
A: It depends on the ground rules. If processes enter critical regions strictly sequentially, that is, a process in a critical region may not attempt to enter another one, then there is no way that it can block while holding a resource (i.e., a critical section) that some other process wants. The system is then
deadlock free. On the other hand, if process 0 may enter critical region *A* and then try to enter critical region *B*, a deadlock can occur if some other process tries to acquire them in the reverse order. The Ricart and Agrawala algorithm itself does not contribute to deadlock since each critical region is handled independently of all the others.

14. Q: In Fig. 5-17 we saw a way to update an inventory list atomically using magnetic tape. Since a tape can easily be simulated on disk (as a file), why do you think this method is not used any more?
A: The primary reason is probably that people have gotten greedier and want to do more than they used to. If the users were content to simply maintain the inventory by making a run once a day, one could do it on disk. The problem is that now everyone is demanding instantaneous online access to the data base, which makes it impossible to store the inventory as a tape image.

15. Q: In Fig. 5-25(d) three schedules are shown, two legal and one illegal. For the same transactions, give a complete list of all values that *x* might have at the end, and state which are legal and which are illegal.
A: The legal values are 1, 2, and 3. The illegal values are 4, 5, and 6. The 4 is achieved by first running the middle transaction, then incorrectly interleaving the other two. The 5 is achieved in schedule 3. The 6 is achieved by first setting *x* to 0 three times, then incrementing it three times.

16. Q: When a private workspace is used to implement transactions on files, it may happen that a large number of file indices must be copied back to the parent's workspace. How can this be done without introducing race conditions?
A: One way is by setting a lock at the top of the system to prevent any activity at all until all the indices have been overwritten. It would probably be wise to create an intentions list before starting to guard against crashes.

17. Q: Give the full algorithm for whether an attempt to lock a file should succeed or fail. Consider both read and write locks, and the possibility that the file was unlocked, read locked, or write locked.
A: The algorithm is as follows. If the file is unlocked, the attempt always succeeds. If the file is read locked and the attempt is for another read lock, it also succeeds. In all other cases, it fails (i.e., write locking a locked file fails, as does read locking a file already locked for writing).

18. Q: Systems that use locking for concurrency control usually distinguish read locks from write locks. What should happen if a process has already acquired a read lock and now wants to change it into a write lock? What about changing a write lock into a read lock?
A: A read lock can only be converted to a write lock if there are no other readers. Doing this conversion is effectively releasing the lock, then immediately trying to reacquiring it as a write lock. This operation fails if there are other readers. If there are other processes waiting to acquire a write lock, it is a matter of policy whether it should succeed; there is no technical objection. Downgrading a lock from write to read is always legal and should always

work. (Doing so, however, implicitly gives priority to waiting readers over waiting writers, but this is a legal choice.)

19. Q: With timestamp ordering in distributed transactions, suppose a write operation *write(T*1*,x)* can be passed to the data manager, because the only, possibly conflicting operation *write(T*2*,*x) had a lower timestamp. Why would it make sense to let the scheduler postpone passing *write(T*1*,x)* until transaction *T*2 finishes?
A: By waiting until transaction *T*2 finishes, the scheduler may avoid a cascaded abort in the case *T*2 aborts.

20. Q: Is optimistic concurrency control more or less restrictive than using timestamps? Why?
A: It is more restrictive because by using the optimistic scheme, if a transaction tries to commit and discovers that another transaction has modified a file it has used, it always aborts. With timestamps, the transaction can sometimes commit, namely, if the other transaction has a lower timestamp.

21. Q: Does using timestamping for concurrency control ensure serializability? Discuss.
A: Yes. Stronger yet, it either completely serializes the transactions in their timestamp order, or it aborts some of them.

22. Q: We have repeatedly said that when a transaction is aborted, the world is restored to its previous state, as though the transaction had never happened. We lied. Give an example where resetting the world is impossible.
A: Any situation in which physical I/O has occurred cannot be reset. For example, if the process has printed some output, the ink cannot be removed from the paper. Also, in a system that controls any kind of industrial process, it is usually impossible to undo work that has been done.

SOLUTIONS TO CHAPTER 6 PROBLEMS – CONSISTENCY AND REPLICATION

1. Q: Access to shared Java objects can be serialized by declaring its methods as being synchronized. Is this enough to guarantee serialization when such an object is replicated?
A: No. The problem is that access to each replica is serialized. However, different operations at different replicas may be executed at the same time, leaving the replicated instance variables in an inconsistent state.

2. Q: Consider a monitor as discussed in Chap. 1. If threads are allowed to block in a replicated monitor, what do we need to guarantee when signaling a condition variable?
A: That the same thread is awakened in each replica, so that exactly the same flow of control takes place in all replicas. In practice, this means that the runtime system should be in full control of thread scheduling at each replica.

3. Q: Explain in your own words what the main reason is for actually considering weak consistency models.
A: Weak consistency models come from the need to replicate for performance. However, efficient replication can be done only if we can avoid global synchronizations, which, in turn, can be achieved by loosening constistency constraints.

4. Q: Explain how replication in DNS takes place, and why it actually works so well.
A: The basic idea is that name servers cache previously looked up results. These results can be kept in a cache for a long time, because DNS makes the assumption that name-to-address mappings do not change often.

5. Q: During the discussion of consistency models, we often referred to the contract between the software and data store. Why is such a contract needed?
A: If a program expects a sequentially consistent data store and cannot live with anything less, the store must provide sequential consistency. However, to improve performance, some systems provide a weaker model. It is then essential that the software agrees to abide by the rules imposed by this model.
Generally, it means that programs obeying the rules will perceive what looks like a sequentially consistent data store.

6. Q: Linearizability assumes the existence of a global clock. However, with strict consistency we showed that such an assumption is not realistic for most distributed systems. Can linearizability be implemented for physically distributed data stores?
A: Yes. Linearizability assumes loosely synchronized clocks, that is, it assumes that several events may happen within the same time slot. Those events need to be ranked adhering to sequential consistency.

7. Q: A multiprocessor has a single bus. Is it possible to implement strictly consistent memory?
A: Yes. The bus serializes requests so they appear at the memory in absolute time order.

8. Q: Why is *W*1(*x*)*a R*2(*x*)*NIL R*3(*x*)*a* not legal for Fig. 6-7(b)?
A: It violates data coherence.

9. Q: In Fig. 6-0, is 000000 a legal output for a distributed shared memory that is only FIFO consistent? Explain your answer.

A: Yes. Suppose (a) runs first. It prints 00. Now (b) runs. If the store into (a) has not arrived yet, it also prints 00. Now (c) runs. If neither store has arrived yet, it too prints 00.

10. Q: In Fig. 6-8, is 001110 a legal output for a sequentially consistent memory? Explain your answer.
A: Yes. If the processes run in the order (a), (c), (b), this result is obtained.

11. Q: At the end of Sec. 6.2.2, we discussed a formal model that said every set of operations on a sequentially consistent data store can be modeled by a string, *H*, from which all the individual process sequences can be derived. For processes *P*1 and *P*2 in Fig. 6-9, give all the possible values of *H*. Ignore
processes *P*3 and *P*4 and do not include their operations in *H*.
A: It is clear that *P*2 cannot read 1 before it is written, so all values of *H* will have the write of 1 before the read of 1. Program order is respected, so the write of 2 must come after the read of 1. The only possibilities that remain are these:
H = *W*(*x*)*a R*(*x*)*a W*(*x*)*b W*(*x*)*c*
H = *W*(*x*)*a R*(*x*)*a W*(*x*)*c W*(*x*)*b*

12. Q: In Fig. 6-13, a sequentially consistent memory allows six possible statement interleavings. List them all.
A: The six statement interleavings are as follows:
(1) a=1; if (b==0); b=1; if (a==0);
(2) a=1; b=1; if (a==0); if (b==0);
(3) a=1; b=1; if (b==0); if (a==0);
(4) b=1; if (a==0); a=1; if (b==0)
(5) b=1; a=1; if (b==0); if (a==0);
(6) b=1; a=1; if (a==0); if (b==0);

13. Q: It is often argued that weak consistency models impose an extra burden for programmers. To what extent is this statement actually true?
A: It really depends. Many programmers are used to protect their shared data through synchronization mechanisms such as locks or transactions. The main idea is that they require a coarser grain of concurrency than one offered at the level of only read and write operations. However, programmers do expect that operations on synchronization variables adhere to sequential consistency.

14. Q: In most implementations of (eager) release consistency in distributed shared memory systems, shared variables are synchronized on a release, but not on an acquire. Why is acquire needed at all then?
A: Acquire is needed to delay a process trying to access shared variables while another process is doing so.

15. Q: Does Orca offer sequential consistency or entry consistency? Explain your answer.
A: Formally, Orca provides only entry consistency, as concurrent operations on the same object are properly serialized. However, when using totallyordered broadcasting as the means to implement ordering on operations, *all* operations are globally ordered, independent of the object on which they are performed. In that case, it offers sequential consistency.

16. Q: Does totally-ordered multicasting by means of a sequencer and for the sake of consistency in active replication, violate the end-to-end argument in system design?

A: Yes. The end-to-end argument states that problems should be solved at the same level in which they occur. In this case, we are dealing with the problem of totally-ordered multicasting for achieving consistency in active replication. In primary-based protocols, consistency is achieved by first forwarding all operations to the primary. Using a sequencer, we are actually doing the same but at a lower level of abstraction. In this case, it may have been better to use a primary-based protocol in which updates are propagated by sending operations.

17. Q: What kind of consistency would you use to implement an electronic stock market? Explain your answer.
A: Causal consistency is probably enough. The issue is that reactions to changes in stock values should be consistent. Changes in stocks that are independent can be seen in different orders.

18. Q: Consider a personal mailbox for a mobile user, implemented as part of a wide-area distributed database. What kind of client-centric consistency would be most appropriate?
A: All of them, actually. What it boils down to is that the owner should always see the same mailbox, no matter whether he is reading or updating it. In fact, the simplest implementation for such a mailbox may well be that of a primary-based local-write protocol, where the primary is always located on
the user's mobile computer.

19. Q: Describe a simple implementation of read-your-writes consistency for displaying Web pages that have just been updated.
A: The simplest implementation is to let the browser always check whether it is displaying the most recent version of a page. This requires sending a request to the Web server. This scheme is simple as it is already implemented by many systems.

20. Q: Give an example where client-centric consistency can easily lead to write-write conflicts.
A: In our description of client-centric consistency models, we assumed that each data item had a single owner, which had exclusive write access. Dropping this assumption is enough to generate write-write conflicts. For example, if two independent users of the same data eventually bind to the same replica, their respective updates may need to be propagated to that replica. At that point, update conflicts will become apparent.

21. Q: When using a lease, is it necessary that the clocks of a client and the server, respectively, are tightly synchronized?
A: No. If the client takes a pessimistic view concerning the level at which its clock is synchronized with that of the server, it will attempt to obtain a new lease far before the current one expires.

22. Q: Consider a nonblocking primary-backup protocol used to guarantee sequential consistency in a distributed data store. Does such a data store always provide read-your-writes consistency?
A: No. As soon as the updating process receives an acknowledgement that its update is being processed, it may disconnect from the data store and reconnect to another replica. No guarantees are given that the update has already reached that replica. In contrast, with a blocking protocol, the updating process can disconnect only after its update has been fully propagated to the other replicas as well.

23. Q: For active replication to work in general, it is necessary that all operations be carried out in the same order at each replica. Is this ordering always necessary?
A: No. Consider read operations that operate on nonmodified data or commutative write operations. In principle, such situations allow ordering to be different at different replicas. However, it can be hard or impossible to detect whether, for example, two write operations are commutative.

24. Q: To implement totally-ordered multicasting by means of a sequencer, one approach is to first forward an operation to the sequencer, which then assigns it a unique number and subsequently multicasts the operation. Mention two alternative approaches, and compare the three solutions.
A: Another approach is to multicast the operation, but defer delivery until the sequencer has subsequently multicast a sequence number for it. The latter happens after the operation has been received by the sequencer. A third approach is to first get a sequence number from the sequencer, and then multicast the operation.
The first approach (send operation to sequencer), involves sending one pointto-point message with the operation, and a multicast message. The second approach requires two multicast messages: one containing the operation, and one containing a sequence number. The third approach, finally, costs one point-to-point message with the sequence number, followed by a multicast message containing the operation.

25. Q: A file is replicated on 10 servers. List all the combinations of read quorum and write quorum that are permitted by the voting algorithm.
A: The following possibilities of (read quorum, write quorum) are legal. (1, 10), (2, 9), (3, 8), (4, 7), (5, 6), (6, 5), (7, 4), (8, 3), (9, 2), and (10, 1).

26. Q: In the text, we described a sender-based scheme for preventing replicated invocations. In a receiver-based scheme, a receiving replica recognizes copies of incoming messages that belong to the same invocation. Describe how replicated invocations can be prevented in a receiver-based scheme.
A: Assume object *A* invokes object *B*. As before, each invocation is assigned the same, unique identifier by all replicas of *A*. Each replica subsequently multicasts its invocation request to the replicas of *B*. When a replica of *B* receives an invocation request, it checks whether it had already received that request from one of *A*'s replicas. If not, the invocation is carried out at the local replica, and a reply message is multicast to all replicas of *A*. If the invocation had already been done, the incoming request is further ignored. Likewise, a replica of *A* passes only the first incoming reply for an outstanding invocation request to its local copy of the object, while all subsequent replies to that same invocation request are simply ignored.

27. Q: Consider causally-consistent lazy replication. When exactly can an operation be removed from a write queue?
A: When it is known that the operation has been performed everywhere. This can be detected by keeping track of the last update (from, say *Li*) carried out in each copy *Lj* . If each *Lj* has performed an operation *W* from *Li* with timestamp *ts* (*W*), *Li* can remove all operations *W*′ with a smaller timestamp (i.e., with each $W'[k] \leq W[k]$). Therefore, each *Lj* sends an acknowledgement to *Li* containing the timestamp of the most recently performed write operation.

28. Q: For this exercise, you are to implement a simple system that supports multicast RPC. We assume that there are multiple, replicated servers and that each client communicates with a server through an RPC. However, when dealing with replication, a client will need to

send an RPC request to each replica. Program the client such that to the application it appears as if a single RPC is sent. Assume you are replicating for performance, but that servers are susceptible to failures.

SOLUTIONS TO CHAPTER 7 PROBLEMS – FAULT TOLERANCE

1. Q: Dependable systems are often required to provide a high degree of security. Why?
A: If, for example, the responses given by servers cannot be trusted to be correct because some malicious party has tampered with them, it hardly makes sense to talk about a dependable system. Likewise, servers should be able to trust their clients.

2. Q: What makes the fail-stop model in the case of crash failures so difficult to implement?
A: The fact that, in practice, servers simply stop producing output. Detecting that they have actually stopped is difficult. As far as another process can see, the server may just be slow, or communication may (temporarily) be failing.

3. Q: Consider a Web browser that returns an outdated cached page instead of a more recent one that had been updated at the server. Is this a failure, and if so, what kind of failure?
A: Whether or not it is a failure depends on the consistency that was promised to the user. If the browser promises to provide pages that are at most T time units old, it may exhibit performance failures. However, a browser can never live up to such a promise in the Internet. A weaker form of consistency is to provide one of the client-centric models discussed in Chap. 7. In that case, simply returning a page from the cache without checking its consistency may lead to a response failure.

4. Q: Can the model of triple modular redundancy described in the text handle Byzantine failures?
A: Absolutely. The whole discussion assumed that failing elements put out random results, which are the same as Byzantine failures.

5. Q: How many failed elements (devices plus voters) can Fig. 7-2 handle? Give an example of the worst case that can be masked.
A: In each row of circles, at most one element can fail and be masked. Furthermore, one voter in each group can also fail provided it is feeding a faulty element in the next stage. For example, if all six elements at the top of their respective columns all fail, two of the three final outputs will be correct, so
we can survive six failures.

6. Q: Does TMR generalize to five elements per group instead of three? If so, what properties does it have?
A: Yes, any odd number can be used. With five elements and five voters, up to two faults per group of devices can be masked.

7. Q: For each of the following applications, do you think at-least-once semantics or at most once semantics is best? Discuss.
(a) Reading and writing files from a file server.
(b) Compiling a program.
(c) Remote banking.
A: For (a) and (b), at least once is best. There is no harm trying over and over. For (c), it is best to give it only one try. If that fails, the user will have to intervene to clean up the mess.

8. Q: With asynchronous RPCs, a client is blocked until its request has been *accepted* by the server (see Fig. 2-12). To what extent do failures affect the semantics of asynchronous RPCs?

A: The semantics are generally affected in the same way as ordinary RPCs. A difference lies in the fact that the server will not be processing the request while the client is blocked, which introduces problems when the client crashes in the meantime. Instead, the server simply does its work, and
attempts to contact the client later on, if necessary.

9. Q: Give an example in which group communication requires no message ordering at all.
A: Multicasting images in small fragments, where each fragment contains the (x,y) coordinate as part of its data. Likewise, sending the pages of a book, with each page being numbered.

10. Q: In reliable multicasting, is it always necessary that the communication layer keeps a copy of a message for retransmission purposes?
A: No. In many cases, such as when transferring files, it is necessary only that the data is still available at the application level. There is no need that the communication layer maintains its own copy.

11. Q: To what extent is scalability of atomic multicasting important?
A: It really depends on how many processes are contained in a group. The important thing to note is, that if processes are replicated for fault tolerance, having only a few replicas may be enough. In that case, scalability is hardly an issue. When groups of different processes are formed, scalability may
become an issue. When replicating for performance, atomic multicasting itself may be overdone.

12. Q: In the text, we suggest that atomic multicasting can save the day when it comes to performing updates on an agreed set of processes. To what extent can we guarantee that each update is actually performed?
A: We cannot give such guarantees, similar to guaranteeing that a server has actually performed an operation after having sent an acknowledgement to the client. However, the degree of fault tolerance is improved by using atomic multicasting schemes, and makes developing fault-tolerant systems easier.

13. Q: Virtual synchrony is analogous to weak consistency in distributed data stores, with group view changes acting as synchronization points. In this context, what would be the analogon of strong consistency?
A: The synchronization resulting from individual multicasts, be they totally, causally, or FIFO ordered. Note that view changes take place as special multicast messages, which are required to be properly ordered as well.

14. Q: What are the permissable delivery orderings for the combination of FIFO and total-ordered multicasting in Fig. 7-14?
A: There are six orderings possible:

Order1	order2	order3	order4	order5	order6
m1	m1	m1	m3	m3	m3
m2	m3	m3	m1	m1	m4
m3	m2	m4	m2	m4	m1
m4	m4	m2	m4	m2	m2

15. Q: Adapt the protocol for installing a next view $Gi \cdot 1$ in the case of virtual synchrony so that it can tolerate process failures.

A: When a process *P* receives *Gi* · *k*, it first forwards a copy of any unstable message it has, regardless to which previous view it belonged, to every process in *Gi* · *k* , followed by a flush message for *Gi* · *k* . It can then safely mark the message as being stable. If a process *Q* receives a message *m* that was
sent in *Gj* (with *j* < *i* ··*k*), it discards it when *Q* was never in *Gj* . If the most recently installed view at *Q* is *Gl* with *l* > *j*, message *m* is also discarded (it is a duplicate). If *l* ··*j* and *m* had not yet been received, process *Q* delivers *m* taking any additional message ordering constraints into account. Finally, if *l* < *j*, message *m* is simply stored in the communication layer until *Gj* has been installed.

16. Q: In the two-phase commit protocol, why can blocking never be completely eliminated, even when the participants elect a new coordinator?
A: After the election, the new coordinator may crash as well. In this case, the remaining participants can also not reach a final decision, because this requires the vote from the newly elected coordinator, just as before.

17. Q: In our explanation of three-phase commit, it appears that committing a transaction is based on majority voting. Is this true?
A: Absolutely not. The point to note is that a recovering process that could not take part in the final decision as taken by the other process, will recover to a state that is consistent with the final choice made by the others.

18. Q: In a piecewise deterministic execution model, is it sufficient to log only messages, or do we need to log other events as well?
A: More logging is generally needed: it concerns *all* nondeterministic events, including local I/O and, in particular, system calls.

19. Q: Explain how the write-ahead log in distributed transactions can be used to recover from failures.
A: The log contains a record for each read and write operation that took place within the transaction. When a failure occurs, the log can be replayed to the last recorded operation. Replaying the log is effectively the opposite from rolling back, which happens when the transaction needed to be aborted.

20. Q: Does a stateless server need to take checkpoints?
A: It depends on what the server does. For example, a database server that has been handed a complete transaction will maintain a log to be able to redo its operations when recovering. However, there is no need to take checkpoints for the sake of the state of the distributed system. Checkpointing is done only for local recovery.

21. Q: Receiver-based message logging is generally considered better than sender-based logging. Why?
A: The main reason is that recovery is entirely local. In sender-based logging, a recovering process will have to contact its senders to retransmit their messages.

SOLUTIONS TO CHAPTER 8 PROBLEMS – SECURITY

1. Q: Which mechanisms could a distributed system provide as security services to application developers that believe only in the end-to-end argument in system's design, as discussed in Chap. 5?
A: None. Applying the end-to-end argument to security services means that developers will not trust anything that is not provided by their own applications. In effect, the distributed system as a whole is considered to be untrusted.

2. Q: In the RISSC approach, can all security be concentrated on secure servers or not?
A: No, we still need to make sure that the local operating systems and communication between clients and servers are secure.

3. Q: Suppose you were asked to develop a distributed application that would allow teachers to set up exams. Give at least three statements that would be part of the security policy for such an application.
A: Obvious requirements would include that students should not be able to access exams before a specific time. Also, any teacher accessing an exam before the actual examination date should be authenticated. Also, there may be a restricted group of people that should be given read access to any exam in preparation, whereas only the responsible teacher should be given full access.

4. Q: Would it be safe to join message 3 and message 4 in the authentication protocol shown in Fig. 8-15, into $KA,B(RB,RA)$?
A: Yes, there is no reason why the challenge sent by Alice for Bob cannot be sent in the same message.

5. Q: Why is it not necessary in Fig. 8-12 for the KDC to know for sure it was talking to Alice when it receives a request for a secret key that Alice can share with Bob?
A: Suppose that Chuck had sent the message "I'm Alice and I want to talk to Bob." The KDC would just return $KA,KDC(KA,B)$ which can be decrypted only by Alice because she is the only other entity holding the secret key KA,KDC.

6. Q: What is wrong in implementing a nonce as a timestamp?
A: Although a timestamp is used only once, it is far from being random. Implementations of security protocols exist that use timestamps as nonces, and which have been succusfully attacked by exploiting the nonrandomness of the nonces.

7. Q: In message 2 of the Needham-Schroeder authentication protocol, the ticket is encrypted with the secret key shared between Alice and the KDC. Is this encryption necessary?
A: No. Because Bob is the only one who can decrypt the ticket, it might as well have been sent as plaintext.

8. Q: Can we safely adapt the authentication protocol shown in Fig. 8-19 such that message 3 consists only of RB?
A: In principle, if RB is never used again, then returning it unencrypted should be enough. However, such randomness is seldom found. Therefore, by encrypting RB, it becomes much more difficult for Chuck to break in and forge message 3.

9. Q: Devise a simple authentication protocol using signatures in a public-key cryptosystem.

A: If Alice wants to authenticate Bob, she sends Bob a challenge *R*. Bob will be requested to return *KB*(*R*), that is, place his signature under *R*. If Alice is confident that she has Bob's public key, decrypting the response back to *R* should be enough for her to know she is indeed talking to Bob.

10. Q: Assume Alice wants to send a message *m* to Bob. Instead of encrypting *m* with Bob's public key *KB*•, she generates a session key *KA,B* and then sends [*KA,B*(*m*),*KB*•(*KA,B*)]. Why is this scheme generally better? (*Hint:* consider performance issues.)
A: The session key has a short, fixed length. In contrast, the message *m* may be of arbitrary length. Consequently, the combination of using a session key and applying public-key cryptography to a short message will generally provide much better performance than using only a public key on a large message.

11. Q: How can role changes be expressed in an access control matrix?
A: Roles, or protection domains in general, can be viewed as objects with basically a single operation: enter. Whether or not this operation can be called depends on the role from which the request is issued. More sophisticated approaches are also possible, for example, by allowing changes back to
previous roles.

12. Q: How are ACLs implemented in a UNIX file system?
A: Each file has three associated entries: one for the owner, one for a group that is associated with the file, and one for everyone else. For each entry, the access rights can essentially be specified as *read*, *write*, *execute*.

13. Q: How can an organization enforce the use of a Web proxy gateway and prevent its users to directly access external Web servers?
A: One way is to use a packet-filtering gateway that discards all outgoing traffic except that directed to a few, well-known hosts. Web traffic is accepted provided it is targeted to the company's Web proxy.

14. Q: Referring to Fig. 8-29, to what extent does the use of Java object references as capabilities actually depend on the Java language?
A: It is independent of the Java language: references to secured objects still need to be handed out during runtime and cannot be simply constructed. Java helps by catching the construction of such references during compile time.

15. Q: Name three problems that will be encountered when developers of interfaces to local resources are required to insert calls to enable and disable privileges to protect against unauthorized access by mobile programs as explained in the text.
A: An important one is that no thread switching may occur when a local resource is called. A thread switch could transfer the enabled privileges to another thread that is not authorized to access the resource. Another problem occurs when another local resource needs to be called before the current invocation is finished. In effect, the privileges are carried to the second resource, while it may happen that the caller is actually not trusted to access that second resource. A third problem is that explicitly inserting calls to enable and disable privileges is suspect to programming errors, rendering the mechanism useless.

16. Q: Name a few advantages and disadvantages of using centralized servers for key management.
A: An obvious advantage is simplicity. For example, by having *N* clients share a key with only a centralized server, we need to maintain only *N* keys. Pairwise sharing of keys would

add up to $N(N-1)/2$ keys. Also, using a centralized server allows efficient storage and maintenance facilities at a single site. Potential disadvantages include the server becoming a bottleneck with respect to performance as well as availability. Also, if the server is compromised, new keys will need to be established.

17. Q: The Diffie-Hellman key-exchange protocol can also be used to establish a shared secret key between three parties. Explain how.
A: Suppose Alice, Bob, and Chuck want to set up a shared secret key based on the two publicly known large primes *n* and *g*. Alice has her own secret large number *x*, Bob has *y*, and Chuck has *z*. Alice sends *gx mod n* to Bob; Bob sends *gy mod n* to Chuck; and Chuck sends *gz mod n* to Alice. Alice can now compute *gxz mod n*, which she sends to Bob. Bob, in turn, can then compute *gxyz mod n*. Likewise, after receiving *gx mod n* from Alice, Bob can compute *gxy mod n*, which he sends to Chuck. Chuck can then compute *gxyz mod n*. Similarly, after Chuck receives *gy mod n* from Bob, he computes
gyz mod n and sends that to Alice so that she can compute *gxyz mod n*.

18. Q: There is no authentication in the Diffie-Hellman key-exchange protocol. By exploiting this property, a malicious third party, Chuck, can easily break into the key exchange taking place between Alice and Bob, and subsequently ruin the security. Explain how this would work.
A: Assume Alice and Bob are using the publicly known values *n* and *g*. When Alice sends *gx mod n* to Bob, Chuck need simply intercept that message, return his own message *gz mod n*, and thus make Alice believe she is talking to Bob. After intercepting Alice's message, he sends *gz mod n* to Bob, from which he can expect *gy mod n* as reply. Chuck is now in the middle.

19. Q: Give a straightforward way how capabilities in Amoeba can be revoked.
A: The object's owner simply requests that the server discard all registered *(rights, check)*-pairs for that object. A disadvantage is that *all* capabilities are revoked. It is difficult to revoke a capability handed out to a specific process.

20. Q: Does it make sense to restrict the lifetime of a session key? If so, give an example how that could be established.
A: Session keys should always have a restricted lifetime as they are easier to break than other types of cryptographic keys. The way to restrict their lifetime is to send along the expiration time when the key is generated and distributed. This approach is followed, for example, in SESAME.

SOLUTIONS TO CHAPTER 9 PROBLEMS – DISTRIBUTED OBJECT-BASED SYSTEMS

1. Q: Why is it useful to define the interfaces of an object in an Interface Definition Language?
A: There are several reasons. First, from a software-engineering point of view, having precise and unambiguous interface definitions is important for understanding and maintaining objects. Furthermore, IDL-based definitions come in handy for generating stubs. Finally, and related to the latter, if an IDL definition has been parsed and stored, supporting dynamic invocations becomes easier, as the client-side proxy can be automatically constructed at runtime from an interface definition.

2. Q: Give an example in which CORBA's dynamic invocation mechanisms come in handy.
A: When a process implements a gateway between two different ORBs, it can dynamically construct an invocation for an object at the target ORB without knowing in advance what that object actually looks like. Instead, it can find out dynamically.

3. Q: Which of the six forms of communication discussed in Sec. 2.4 are supported by CORBA's invocation model?
A: Transient asynchronous communication (by means of CORBA one-way requests) and response-based transient synchronous communication (by means of CORBA synchronous requests). Deferred synchronous request can be viewed as a combination of two one-way requests, possibly with a blocking client waiting for the response.

4. Q: Outline a simple protocol that implements at-most-once semantics for an object invocation.
A: A simple solution is to let a proxy retransmit an invocation request, but telling the server explicitly that it is a retransmission. In that case, the server may decide to ignore the request and return an error to the client. In a more sophisticated solution, the server can cache results of previous invocations
and check whether it still has those results when receiving a retransmitted request.

5. Q: Should the client and server-side CORBA objects for asynchronous method invocation be persistent?
A: In general, they should be persistent, allowing the client or server to shut down and later restart and fetch the objects from disk. However, in theory, there is no hard reason to demand that these objects should be persistent.

6. Q: In a GIOP *Request* message, the object reference and method name are part of the message header. Why can they not be contained in the message body only?
A: The server will need to demultiplex incoming messages to the appropriate object adapter. It can only look at message headers as it should know nothing about the actual invocation. Demultiplexing can be done on an object reference alone, but further demultiplexing may be needed by also looking at the method that is to be invoked.

7. Q: Does CORBA support the thread-per-object invocation policy that we explained in Chap. 3?
A: Yes, although it can be done only by associating a single object per POA, which in turn should provide the single-threading policy. In other cases, the thread-per-object policy depends on what the underlying ORB provides.

8. Q: Consider two CORBA systems, each with their own naming service. Outline how the two naming services could be integrated into a single, federated naming service.
A: Integration can be straightforward. Because naming contexts are regular objects, a naming context in a foreign name space can be registered under a specific name in the naming graph. Resolving that name will return an IOR referring to the foreign naming context. To a client, a regular object reference is returned for which it can invoke the *resolve* method as usual.

9. Q: In the text, we state that when binding to a CORBA object, additionalsecurity services may be selected by the client ORB based on the object reference. How does the client ORB know about these services?
A: In principle, they are encoded as part of an IOR, for example, in the *Components* field of a tagged profile. Such information can be made available by the object itself when its associated object server (or rather, POA) generates its IOR.

10. Q: If a CORBA ORB uses several interceptors that are not related to security, to what extent is the order in which interceptors are called important?
A: It really depends, but in general, if security is needed, most interceptors (at the client side) will be called after calling the access control interceptor, but certainly before the secure invocation interceptor. The latter is strictly required because the secure invocation interceptor produces encrypted messages in which only the destination is visible. Any modification to such a message will lead to a fault when checked for integrity.

11. Q: Illustrate how a transactional queue can be part of a distributed transaction as mentioned in the text.
A: In this case, an application may need to send and receive messages from components running on other machines, and which possibly need to access local databases. By turning the operations on a transactional queue into a nested transaction, and likewise using nested transactions for the other groups of operations, it becomes possible to construct one big distributed transaction.

12. Q: In comparing DCOM to CORBA, does CORBA provide standard marshaling or custom marshaling? Discuss.
A: CORBA provides standard marshaling, but allows customization of proxies and skeletons by means of interceptors. In this sense, custom marshaling in DCOM can be seen as a work-around for the lack of general interception techniques. However, there is no reason why a CORBA developer should not write his own custom proxies, as long as they adhere to language-dependent mapping of the IDL specifications of the object's interfaces. The drawback of this approach, however, is that it requires these custom proxies to be stored in an interface repository from where they can be retrieved by clients. Unlike DCOM, CORBA provides no standard support for storing and retrieving customized proxies. In DCOM, such matters are handled by the registry.

13. Q: In DCOM, consider a method *m* that is contained in interface *X*, and which accepts a pointer to interface *Y* as input parameter. Explain what happens with respect to marshaling when a client holding an interface pointer to a proxy implementation of *X*, invokes *m*.
A: When *m* is invoked, the proxy implementing *X* will have to marshal the proxy implementing *Y*. If *Y* is based on standard marshaling, then, in principle, only the interface identifier of *Y* needs to be marshaled into the message comprising the marshaled invocation of *m*. If *Y* is based on custom marshaling, *X* will have to marshal *Y* using the class object associated with *Y* that handles the marshaling of *Y*. The CLSID of that class object is

prepended to the marshaled version of *Y* in order to allow the receiver to load the appropriate code to unmarshal *Y* again.

14. Q: Does custom marshaling in DCOM require that the process receiving a marshaled proxy runs on the same type of machine as the sending process?
A: No, provided that a marshaled proxy can be represented in a machineindependent way. A developer of object-specific proxies will have to take this design issue into account. For example, assume the sending process runs on an Intel-based Windows machine whereas the receiver is running as a Solaris process on a SPARC station. When the sender marshals its proxy, it provides a CLSID identifying the class object that is needed to unmarshal the proxy by the receiver. When the Solaris implementation of DCOM receives this CLSID, it will have to be able to load its specific implementation of the identified class object. That class object is then used to instantiate an object to
unmarshal the proxy and return an interface pointer to the receiving process. If the proxy can be unmarshaled to something that can be readily understood by the Solaris implementation of DCOM, then no problems need to occur.
15. Q: Outline an algorithm for migrating an object in DCOM to another server.
A: The main issues that need to be dealt with is migrating the state to the target object server, and having each client rebind to the migrated object. One possible approach is the following. First, a copy of the object is created on the target server using the object's CLSID. Then, the state is marshaled and
transferred to the new server, where the newly created object is instructed to unmarshal that state. The old copy is replaced by an object implementing a forwarding pointer that offers the same interface as the moved object. Each time an invocation request is forwarded, the invoking client will be notified
that it should rebind to the object at its new location.

16. Q: To what extent does JIT activation violate or comply with the notion of objects?
A: JIT activation essentially assumes that objects have no state. In other words, an object is reduced to a collection of (stateless) methods. In general, such collections are hardly considered to resemble objects.

17. Q: Explain what happens when two clients on different machines use the same file moniker to bind to a single object. Will the object be instantiated once or twice?
A: It really depends. If no special measures are taken, two copies of the object will be constructed and initialized with the data from the same file. However, with a special DCOM object server and possibly adapted moniker, it should be possible to create only a single, shared instance.

18. Q: What would be the obvious way to implement events in Globe without violating the principle that objects are passive?
A: The best way would be to implement a simple layer on top of Globe objects in which a thread would occasionally poll the underlying object to see if any state changes have occurred. Such a thread could then invoke a callback interface as provided by a client application. In effect, such an implementation changes a pull-style model into a push-style model.

19. Q: Give an example in which the (inadvertent) use of callback mechanisms can easily lead to an unwanted situation.
A: If a callback leads to another invocation on the same object, a deadlock may arise if locks are needed to protect shared resources. Situations as these are very hard to control in a general way.

20. Q: In CORBA, nodes in naming graph, such as directories, are also considered as objects. Would it make sense to also model directories as distributed shared objects in Globe?
A: Yes. This is best explained by considering a root directory in a worldwide naming service. In most cases, the root directory hardly changes. Consequently, its state, which consists of a collection of object references (in the form of object handles), can be widely replicated. A similar reasoning will
show that for many lower-level nodes, a different approach to replication is needed as the update-to-read ratio is very different. These differences can be dealt with by means of distributed shared objects.

21. Q: Assume a Globe object server has just installed a persistent local object. Also, assume that this object offers a contact address. Should that address be preserved as well when the server shuts down?
A: If the address is not preserved, the server should remove it from the Globe location service before shutting down. When the object is loaded again, the server should offer a new address and insert that address into the location service. For efficiency reasons, Globe also supports persistent contact addresses that remain the same even when a server shuts down.

22. Q: In our example of using Kerberos for security in Globe, would it be useful to encrypt the ticket with a key shared between the object and the ticket granting service, instead of the key K_B, TGT?
A: It depends. Using a single object key would allow secure and efficient multicasting of a shared session key to all replicas. However, because the object key needs to be shared between all processes that have a replica, we need to trust that all processes will indeed keep that key a secret.

SOLUTIONS TO CHAPTER 10 PROBLEMS – DISTRIBUTED FILE SYSTEMS

1. Q: Is a file server implementing NFS version 3 required to be stateless?
A: No, but the NFS protocols are such that it is possible to provide an implementation using stateless servers.

2. Q: NFS does not provide a global, shared name space. Is there a way to mimic such a name space?
A: A global name space can easily be mimiced using a local name space for each client that is partially standardized, and letting the automounter mount the necessary directories into that name space.

3. Q: Give a simple extension to the NFS lookup operation that would allow iterative name lookup in combination with a server exporting directories that it mounted from another server.
A: If a lookup operation always returns an identifier for the server from which a directory was mounted, transparent iterative name lookups across multiple servers would be easy. Whenever a server looks up a mount point on which it mounted a remote file system, it simply returns the server's ID for that file system. The client can then automatically mount the directory as well, and contact its associated server to continue name resolution.

4. Q: In UNIX-based operating systems, opening a file using a file handle can be done only in the kernel. Give a possible implementation of an NFS file handle for a user-level NFS server for a UNIX system.
A: The problem to be solved is to return a file handle that will allow the server to open a file using the existing name-based file system interface. One approach is to encode the file name into the file handle. The obvious drawback is that as soon as the file name changes, its file handles become invalid.

5. Q: Using an automounter that installs symbolic links as described in the text makes it harder to hide the fact that mounting is transparent. Why?
A: After the symbolic link has been followed, the user will not be in the expected directory, but in a subdirectory used by the automounter. In other words, the local home directory for Alice will be */tmp3mnt/home/alice* instead of what she thinks it is, namely */home/alice*. Special support from the
operating system or shell is needed to hide this aspect.

6. Q: Suppose the current denial state of a file in NFS is *WRITE*. Is it possible that another client can first successfully open that file and then request a write lock?
A: Yes. If the second client requires read/write access (i.e., value *BOTH*) but no denial (i.e., value *NONE*), it will have been granted access to the file. However, although a write lock may actually be granted, performing an actual write operation will fail. Remember that share reservation is completely
independent from locking.

7. Q: Taking into account cache coherence as discussed in Chap. 6, which kind of cache-coherence protocol does NFS implement?
A: Because multiple write operations can take place before cached data is flushed to the server, NFS clients implement a write-back cache.

8. Q: Does NFS implement entry consistency? What about release consistency?

A: Yes. Because share reservations and file locking are associated with specific files, and because a client is forced to revalidate a file when opening it and flush it back to the server when closing it, it can be argued that NFS implements entry consistency. Note that this scheme also comes close to
eager release consistency, except that not all files need to be flushed, as would be the case with release consistency, which, by definition, works on all shared data.

9. Q: We stated that NFS implements the remote access model to file handling. It can be argued that it also supports the upload/download model. Explain why.
A: Because a server can delegate a file to a client, it can effectively support whole-file caching and putting that client in charge of further handling of the file. This model comes close to uploading a file to a client and downloading it later when when the client is finished.

10. Q: In NFS, attributes are cached using a write-through cache coherence policy. Is it necessary to forward all attributes changes immediately?
A: No. For example, when appending data to a file, the server does not really need to be informed immediately. Such information may possibly be passed on when the client flushes its cache to the server.

11. Q: To what extent will the duplicate-request cache as described in the text actually succeed in implementing at-most-once semantics?
A: Although the scheme will generally work, there are still numerous problems. For example, the duplicate-request cache is often implemented in volatile memory. Consequently, when the server crashes, the cache is lost, which may eventually lead to processing duplicate requests when the server later recovers. Another problem is that cache entries, in general, will eventually need to be removed to make space for new requests. To do this safely, the client should acknowledge previous replies, possibly by piggybacking ACKs on next requests.

12. Q: In NFS, what kind of security measures can be taken to prevent a malicious client from reclaiming a lock it never was granted when a server enters its grace period?
A: One simple solution is to pass an attribute certificate to a client when it first claims a lock. That certificate should be shown again when reclaiming the lock later.

13. Q: Fig. 10-21 suggests that clients have complete transparent access to Vice. To what extent is this indeed true?
A: All issues concerning file access and caching are handled by Venus, so in that sense, user processes can indeed remain unaware of the organization of servers and the placement of volumes. However, when disconnected, it may be impossible to achieve failure transparency if it turns out that a file is not in the cache, or that a conflict needs to be manually resolved.

14. Q: Using RPC2's side effects is convenient for continuous data streams. Give another example in which it makes sense to use an application-specific protocol next to RPC.
A: File transfer. Instead of using a pure RPC mechanism, it may be more efficient to transfer very large files using a protocol such as FTP.

15. Q: If a physical volume in Coda is moved from server *A* to server *B*, which changes are needed in the volume replication database and volume location database?
A: The volume location database needs to be updated with the new location. No other changes are needed.

16. Q: What calling semantics does RPC2 provide in the presence of failures?

A: Considering that the client will be reported an error when an invocation does not complete, RPC2 provides at-most-once semantics.

17. Q: Explain how Coda solves read-write conflicts on a file that is shared between multiple readers and only a single writer.
A: The problem is solved by "defining it away." The semantics of transactions underlying file sharing in Coda, permit treating all readers as accessing the shared file before the writer opened it. Note that read-write conflicts within a specific time interval cannot be solved in this way.

18. Q: In Coda, is it necessary to encrypt the clear token that Alice receives from the AS when she logs in? If so, where does encryption take place?
A: Yes, otherwise the session key in the clear token can be seen by any other process. Encryption takes place as part of the secure RPC. As is also shown in Fig. 10-32, after mutual authentication between Alice and the AS has taken place, the AS generates a session key as part of secure RPC that is used to encrypt the clear token when returning it to Alice.

19. Q: If a file on a Vice server needs its own specific access control list, how can this be achieved?
A: A solution is to create a separate directory containing only that file, and to associate the required access control list with that directory. If necessary, a symbolic link to the file can be created in the directory where that file logically belongs.

20. Q: Does it make sense for a Vice server to offer only *WRITE* permissions on a file? (Hint: consider what happens when a client opens a file.)
A: No. The point is that in Coda, whenever a client opens a file, it receives a copy of that file to cache. This effectively already gives it *READ* permission. Only a Venus process may possibly enforce write-only permission, but in that case, Vice servers will need to trust clients.

21. Q: Can union directories in Plan 9 replace the *PATH* variable in UNIX systems?
A: Yes. By mounting various file systems in a specific order at the same mount point, and subsequently looking only for executables at that mount point, there is no need to look in any other subdirectory.

22. Q: Can Plan 9 be efficiently used as wide-area distributed system?
A: Without modifications, probably not. An important obstacle is keeping files at a server and using a write-through caching policy. In effect, all update operations need to be passed between a client and the server, which introduces a considerable latency in the case of wide-area networks.

23. Q: What is the main advantage of using stripe groups in xFS compared to the approach in which a segment is fragmented across all storage servers?
A: The main advantage is scalability. In a system with many servers, writing a segment requires that all servers are contacted. Using stripe groups, it becomes possible to limit the size of a group and in this way also limits the amount of network traffic and servers that need to be contacted.

24. Q: Using self-certifying path names, is a client always ensured it is communicating with a nonmalicious server?
A: No. SFS does not solve naming problems. In essence, a client will have to trust that the server named in the path can actually be trusted. In other words, a client will have to put its

trust in the name and name resolution process. It may very well be the case that a malicious SFS server is spoofing another server by using its IP address and passing the other server's public key.

SOLUTIONS TO CHAPTER 12 PROBLEMS – DISTRIBUTED COORDINATION BASED SYSTEMS

1. Q: What type of coordination model would you classify the message-queuing systems discussed in Chap. 2?
A: Considering that in message-queuing systems processes are temporally uncoupled, but will otherwise have to agree on message format and destination queues, they classify as mailbox coordination models.

2. Q: Referential uncoupling in TIB/Rendezvous is supported by subject-based addressing. What else contributes to referential uncoupling in this system?
A: The fact that messages are self describing. In that way, there is no need for processes to agree in advance to the format of the messages they exchange. Instead, a receiving process can dynamically determine what an incoming message actually contains. Of course, semantical issues are still left to he application.

3. Q: Outline an implementation of a publish/subscribe system based on a message-queuing system like that of IBM MQSeries.
A: Such an implementation can be accomplished by using a message broker. All publish/subscribe messages are published to the broker. Subscribers pass their subscriptions to the broker as well, which will then take care to forward published messages to the appropriate subscribers. Note that the broker makes use of the underlying message-queuing network.

4. Q: To what is a subject name in TIB/Rendezvous actually resolved, and how does name resolution take place?
A: A name is resolved to the current group of subscribers. Name resolution takes place by properly routing message to those subscribers. In TIB/Rendezvous, routing takes place through a combination of multicasting and filtering messages at rendezvous and router daemons.

5. Q: Outline a simple implementation for totally-ordered message delivery in a TIB/Rendezvous system.
A: Use a sequencer to which all messages are sent. Subscribers pass subscriptions to this sequencer. The FIFO-ordered message delivery of the TIB/Rendezvous system will then guarantee that all messages multicast by the sequencer are delivered to every subscriber in the same order.

6. Q: When a message is delivered to a process *P* as part of a TIB/Rendezvous transaction, *P* confirms that delivery. Is it possible for the transaction layer at *P* to automatically send a confirmation to the transaction daemon?
A: Yes, it could. When a message is delivered to *P* as part of a transaction, what happens is that the message is removed from an event queue local to *P*.
Removal of a message can be interpreted by the transaction layer as delivery, so that it can automatically send a confirmation to the transaction daemon.

7. Q: Assume a process is replicated in a TIB/Rendezvous system. Give two solutions to avoid so that messages from this replicated process are not published more than once.
A: A first solution is to attach a message identifier to each published message, and to let subscribers discard duplicates by taking a look at the identifiers. The main drawback of this

approach is the waste of network bandwidth. Another solution is to assign a coordinator among the replicas, and let only
the coordinator actually publish messages. This solution is similar to assigning a coordinator in the case of invocations with replicated distributed objects.

8. Q: To what extent do we need totally-ordered multicasting when processes are replicated in a TIB/Rendezvous system?
A: Assuming that duplicate messages are either detected by subscribers, or avoided altogether, total ordering is not an issue at all. In this case, the FIFOordering delivery semantics are sufficient to let subscribers process the messages in the order they were published by the replicated process.

9. Q: Describe a simple scheme for PGM that will allow receivers to detect missing messages, even the last one in a series.
A: A simple scheme is to use sequence numbers. Whenever a sender has no more data to send, it should announce by multicasting a special control message. If that control message is lost, a receiver will start complaining in the usual way. The sender can then simply retransmit the control message. In
essence, this is also the solution adopted in PGM.

10. Q: Can ledgers in TIB/Rendezvous that are implemented as files guarantee that messages are never lost, even in the presence of crashing processes?
A: No. If a ledger is not flushed to disk before the sending process crashes, messages may be lost that later require retransmission by a receiver. The idea of certified message delivery is to increase the reliability of message delivery. By no means is it a solution to 100 percent guaranteed delivery.

11. Q: How could a coordination model based on generative communication be implemented in TIB/Rendezvous?
A: This is actually not very difficult. What makes TIB/Rendezvous different from, for example, the JavaSpaces in Jini, is that processes are still temporally coupled. If published messages are temporarily stored, it would be possible for subscribers to read them even when the publisher of messages no longer exists. What is needed is for each subscriber to record a published message that it has already read, so that receiving duplicates can be avoided.

12. Q: Apart from the temporal uncoupling in Jini, in your opinion, what is the most important difference between Jini and TIB/Rendezvous when considering coordination models?
A: An important difference is the fact that TIB/Rendezvous uses characterstring naming to match publishers and subscribers, whereas JavaSpaces uses marshaled object references. Another important difference is that JavaSpaces allows tuple instances to be removed upon reading, thus providing a means to synchronize processes. A comparable facility is not available in TIB/Rendezvous.

13. Q: A lease period in Jini is always specified as a duration and not as an absolute time at which the lease expires. Why is this done?
A: Especially in wide-area systems, it may happen that clocks on different machines give very different times. In that case, specifying the expiration of a lease as an absolute time is simply too inaccurate as the holder of the lease may have a very different idea when the lease expires than the processes that handed out the lease. With durations, this difference

becomes less an issue, provided some guarantees can be given that the transmission time of a lease is relatively low.

14. Q: What are the most important scalability problems in Jini?
A: One important problem is related to the fact the Jini uses a multicast protocol to locate lookup services. In a wide-area system, this protocol will have to be replaced by something different if Jini is to scale to large numbers of users and processes. A second problem is related to matching templates to tuples in JavaSpaces. Again, special measures will be needed to search a JavaSpace that may be potentially distributed across a large-scale network. No efficient solutions to these problems are yet known.

15. Q: Consider a distributed implementation of a JavaSpace in which tuples are replicated across several machines. Give a protocol to delete a tuple such that race conditions are avoided when two processes try to delete the same tuple.
A: Use a two-phase commit protocol. In phase one, the process doing the delete sends a message to all the JavaSpace servers holding the tuple asking them to lock the tuple. When all of them are locked, the delete is sent. If a second delete happens simultaneously, it can happen that some servers have
locked one tuple and some have locked the other. If a server cannot grant a request because the tuple is already locked, it sends back a negative acknowledgement, which causes the initiator to abort the delete, unlock all the tuples it has locked, wait a random time, and try again later.

16. Q: Suppose a transaction T in Jini requires a lock on an object that is currently locked by another transaction T'. Explain what happens.
A: Transaction T will continue to block until the lock is either released or until the lease on the transaction expires. In the latter case, transaction T is simply aborted.

17. Q: Suppose a Jini client caches the tuple it obtained from a JavaSpace so that it can avoid having to go to the JavaSpace the next time. Does this caching make any sense?
A: Caching is senseless because the tuple will have been removed from the JavaSpace when it was returned; it is ready for the client to keep. The main idea behind caching is to keep data local to avoid another server access. In the case of Jini, a JavaSpace is often used to explicitly synchronize processes. Caching does not play a role when process synchronization is explicitly needed.

18. Q: Answer the previous question, but now for the case that a client caches the results returned by a lookup service.
A: This is a completely different situation. The lookup service stores information on the whereabouts of services. In this case, it may indeed make sense for a client to cache previously returned results and try to contact the returned services before going to the lookup service again.

19. Q: Outline a simple implementation of a fault-tolerant JavaSpace.
A: The simplest approach is to implement a JavaSpace on a single server with stable storage. Write operations succeed only if the tuple has been safely written to storage. In a more advanced setting, a distributed JavaSpace can be used in which a server group is used to mask process failures. In that case, the servers may need to follow a two-phase commit protocol for each write operation.