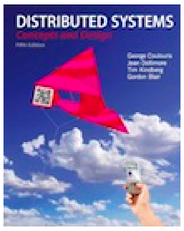

**Distributed Systems: Concepts and Design**

***Edition 5***

**By George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair**
**Addison-Wesley ©Pearson Education 2012**

# Chapter 18 Exercise Solutions

---

18.1 Four computers together provide a replicated service. The manufacturers claim that there is a 10% probability of any individual server failing over a given time period. What is the availability of the replicated service?

*18.1 Ans.*

If there is a 10% probability of any individual server failure over a given time period and if there are four servers, then the availability is $1 – 0.1^4 = 1 – 0.0001 = 99.99\%$.

---

18.2 If up to *f* servers can exhibit Byzantine failures, how many servers can provide a correct service?

*18.2 Ans.*

If up to *f* servers can exhibit Byzantine failures, then in principle a group of $2f + 1$ servers can provide a correct service.

---

18.3 In a multi-user game, the players move figures around a common scene. The state of the game is replicated at the players' workstations and at a server, which contains services controlling the game overall, such as collision detection. Updates are multicast to all replicas.

(i) The figures may throw projectiles at one another and a hit debilitates the unfortunate recipient for a limited time. What type of update ordering is required here? Hint: consider the 'throw', 'collide' and 'revive' events.

(ii) The game incorporates magic devices which may be picked up by a player to assist them. What type of ordering should be applied to the pick-up-device operation?

*18.3 Ans.*

i) The event of the collision between the projectile and the figure, and the event of the player being debilitated (which, we may assume, is represented graphically) should occur in causal order. Moreover, changes in the velocity of the figure and the projectile chasing it should be causally ordered. Assume that the workstation at which the projectile was launched regularly announces the projectile's coordinates, and that the workstation of the player corresponding to the figure regularly announces the figure's coordinates and announces the figure's debilitation. These announcements should be processed in causal order. (The reader may care to think of other ways of organising consistent views at the different workstations.)

ii) If two players move to pick up a piece at more-or-less the same time, only one should succeed and the identity of the successful player should be agreed at all workstations. Therefore total ordering is required.

The most promising architecture for a game such as this is a peer group of game processes, one at each player's workstation. This is the architecture most likely to meet the real-time update propagation requirements; it also is robust against the failure of any one workstation (assuming that at least two players are playing at the same time).

18.4 A router separating process $p$ from two others, $q$ and $r$, fails immediately after $p$ initiates the multicasting of message $m$. If the group communication system is view-synchronous, explain what happens to $p$ next.

*18.4 Ans.*

Process $p$ must receive a new group view containing only itself, and it must receive the message it sent. The question is: in what order should these events be delivered to $p$?.

If $p$ received the message first, then that would tell $p$ that $q$ and $r$ received the message; but the question implies that they did not receive it. So $p$ must receive the group view first.

---

18.5 In the primary-backup model of replication for fault tolerance, the primary replica manager executes the operations and sends copies of the updated data to the backups. What happens if the primary fails?

*18.5 Ans.*

One of the backups is promoted to act as the primary.

- The primary is replaced by a unique backup (if two clients began using two backups, then the system could perform incorrectly); and
- The replica managers that survive agree on which operations had been performed at the point when the replacement primary takes over.

---

18.6 A *sync-ordered* multicast operation is one whose delivery ordering semantics are the same as those for delivering views in a view-synchronous group communication system. In a *thingumajig* service, operations upon thingumajigs are causally ordered. The service supports lists of users able to perform operations on each particular thingumajig. Explain why removing a user from a list should be a sync-ordered operation.

*18.6 Ans.*

Sync-ordering the remove-user update ensures that all processes handle the same set of operations on a thingumajig before the user is removed. If removal were only causally ordered, there would not be any definite delivery ordering between that operation and any other on the thingumajig. Two processes might receive an operation from that user respectively before and after the user was removed, so that one process would reject the operation and the other would not.

---

18.7 What is a view-synchronous system?

*18.7 Ans.*

In a view-synchronous system, the delivery of a new view draws a conceptual line across the system and every message that is delivered at all is consistently delivered on one side or the other of that line. This enables the programmer to draw useful conclusions about the set of messages that other correct processes have delivered when it delivers a new view, based only on the local ordering of message delivery and view delivery events.

---

18.8 An operation $X$ upon an object $o$ causes $o$ to invoke an operation upon another object $o'$. It is now proposed to replicate $o$ but not $o'$. Explain the difficulty that this raises concerning invocations upon $o'$, and suggest a solution.

*18.8 Ans.*

The danger is that all replicas of $o$ will issue invocations upon $o'$, when only one should take place. This is incorrect unless the invocation upon $o'$ is idempotent and all replicas issue the same invocation.

One solution is for the replicas of $o$ to be provided with smart, replication-aware proxies to $o'$. The smart proxies run a consensus algorithm to assign a unique identifier to each invocation and to assign one of them to handle the invocation. Only that smart proxy forwards the invocation request; the others wait for it to multicast the response to them, and pass the results back to their replica.

18.9 What is the active model of replication? Explain the sequence of events when a client requests an operation to be performed under this model.

*18.9 Ans.*

In the active model of replication for fault tolerance, the replica managers are state machines that play equivalent roles and are organized as a group. Front ends multicast their requests to the group of replica managers and all the replica managers process the request independently but identically and reply.

---

18.10 What are the replication schemes that work correctly when a collection of replica managers is divided into subgroups by a network partition?

*18.10 Ans.*

- *Available copies with validation*: available copies replication is applied in each partition and when a partition is repaired, a validation procedure is applied and any inconsistencies are dealt with.
- *Quorum consensus*: a subgroup must have a quorum (meaning that it has sufficient members) in order to be allowed to continue providing a service in the presence of a partition. When a partition is repaired (and when a replica manager restarts after a failure) replica managers get their objects up-to-date by means of recovery procedures.
- *Virtual partition*: a combination of quorum consensus and available copies. If a virtual partition has a quorum, it can use available copies replication.

---

18.11 The gossip system makes two guarantees even though replica managers may be temporarily unable to communicate with one another. Explain these guarantees.

*18.11 Ans.*

*Each client obtains a consistent service over time*: In answer to a query, replica managers only ever provide a client with data that reflects at least the updates that the client has observed so far.

*Relaxed consistency between replicas*: All replica managers eventually receive all updates and they apply updates with ordering guarantees that make the replicas sufficiently similar to suit the needs of the application.

---

18.12 Explain how a gossip service processes queries and update operations.

*18.12 Ans.*

1. *Request*: The front end normally sends requests to only a single replica manager at a time.
2. *Update response*: If the request is an update then the replica manager replies as soon as it has received the update.
3. *Coordination*: The replica manager that receives a request does not process it until it can apply the request according to the required ordering constraints.
4. *Execution*: The replica manager executes the request.
5. *Query response*: If the request is a query then the replica manager replies at this point.
6. *Agreement*: The replica managers update one another by exchanging *gossip messages*, which contain the most recent updates they have received.

---

18.13 In a gossip system, a front end has vector timestamp (3, 5, 7) representing the data it has received from members of a group of three replica managers. The three replica managers have vector timestamps (5, 2, 8), (4, 5, 6) and (4, 5, 8), respectively. Which replica manager(s) could immediately satisfy a query from the front end and what is the resultant time stamp of the front end? Which could incorporate an update from the front end immediately?

*18.13 Ans.*

The only replica manager that can satisfy a query from this front end is the third, with (value) timestamp (4,5,8). The others have not yet processed at least one update seen by the front end. The resultant time stamp of the front end will be (4,5,8).

Similarly, only the third replica manager could incorporate an update from the front-end immediately.

18.14 What are the types of partner-selection policy?

*18.14 Ans.*

There are several types of partner-selection policy, such as random, deterministic and topological policies.

---

18.15 Write pseudocode for dependency checks and merge procedures (as used in Bayou) suitable for a simple room-booking application.

*18.15 Ans.*

```
Operation: room.book(booking).
    let timeSlot = booking.getPreferredTimeSlot();
    Dependency check:
    existingBooking = room.getBooking(timeSlot);
    if (existingBooking != null) return "conflict" else return "no conflict";
    Merge procedure:
    existingBooking = room.getBooking(timeSlot);
    // Choose the booking that should take precedence over the other
    if (greatestPriority(existingBooking, booking) == booking)
        then { room.setBooking(timeSlot, booking); existingBooking.setStatus("failed");}
        else {booking.setStatus("failed");}
```

– in a more sophisticated version of this scheme, bookings have alternative time slots. When a booking cannot be made at the preferred time slot, the merge procedure runs through the alternative time slots and only reports failure if none is available. Similarly, alternative rooms could be tried.

---

18.16 In the Coda file system, what are the advantages of the replication of some or all file volumes on multiple servers?

*18.16 Ans.*

- The files in a replicated volume remain accessible to any client that can access at least one of the replicas.
- The performance of the system can be improved by sharing some of the load of servicing client requests on a replicated volume between all of the servers that hold replicas.

---

18.17 Devise a scheme for integrating two replicas of a file system directory that underwent separate updates during disconnected operation. Use either Bayou's operational transformation approach, or supply a solution for Coda.

*18.17 Ans.*

The updates possible on a directory are (a) changing protection settings on existing entries or the directory itself, (b) adding entries and (c) deleting entries.

Many updates may be automatically reconciled, e.g. if two entries with different names were added in different partitions then both are added; if an entry was removed in one partition and not updated in the other then the removal is confirmed; if an entry's permissions were updated in one partition and it was not updated (including deletion) in the other, then the permissions-update is confirmed.

Otherwise, two updates, in partitions A and B, respectively, may conflict in such a way that automatic reconciliation is not possible. e.g. an entry was removed in A and the same entry in B had its permissions changed; entries were created with the same name (but referring to a different file) in A and B; an entry was added in A but in B the directory's write permissions were removed.

We leave the details to the reader.

---

18.18 Available copies replication is applied to data items $A$ and $B$ with replicas $A_x$, $A_y$ and $B_m$, $B_n$. The transactions $T$ and $U$ are defined as:

$T$: *Read*($A$); *Write*($B$, 44). $U$: *Read*($B$); *Write*($A$, 55).

Show an interleaving of $T$ and $U$, assuming that two-phase locks are applied to the replicas. Explain why locks alone cannot ensure one copy serializability if one of the replicas fails during the progress of *T* and *U*. Explain with reference to this example, how local validation ensures one copy serializability.

*18.18 Ans.*

An interleaving of T and U at the replicas assuming that two-phase locks are applied to the replicas:

| T | | U | |
|---|---|---|---|
| x:= Read (Ax) | lock Ax | | |
| Write(Bm, 44) | lock Bm | | |
| | | x:= Read (Bm) | Wait |
| Write(Bn, 44) | lock Bn | • | |
| Commit unlock | Ax,Bm,Bn | • | |
| | | Write(Ax, 55) | lock Ax |
| | | Write(Ay, 55) | lock Ay |

Suppose Bm fails before T locks it, then U will not be delayed. (It will get a lost update). The problem arises because Read can use one of the copies before it fails and then Write can use the other copy. Local validation ensures one copy serializability by checking before it commits that any copy that failed has not yet been recovered. In the case of T, which observed the failure of Bm, Bm should not yet have been recovered, but it has, so T is aborted.

---

18.19 Gifford's quorum consensus replication is in use at servers *X*, *Y* and *Z* which all hold replicas of data items *A* and *B*. The initial values of all replicas of *A* and *B* are 100 and the votes for *A* and *B* are 1 at each of *X*, *Y* and *Z*. Also *R* = *W* = 2 for both *A* and *B*. A client reads the value of *A* and then writes it to *B*.

(i) At the time the client performs these operations, a partition separates servers *X* and *Y* from server Z. Describe the quora obtained and the operations that take place if the client can access servers *X* and *Y*.

(ii) Describe the quora obtained and the operations that take place if the client can access only server *Z*.

(iii) The partition is repaired and then another partition occurs so that *X* and *Z* are separated from *Y*. Describe the quora obtained and the operations that take place if the client can access servers X and Z.

*18.19 Ans.*

i) Partition separates X and Y from Z when all data items have version v0 say:

| X | Y | Z |
|---|---|---|
| A= 100 (vo) | A= 100(vo) | A= 100(vo) |
| B= 100(vo) | B= 100(vo) | B= 100(vo) |

A client reads the value of A and then writes it to B:

read quorum = 1+1 for A and B - client Reads A from X or Y

write quorum = 1+1 for B client Writes B at X and Y

ii) Client can access only server Z: read quorum = 1, so client cannot read, write quorum = 1 so client cannot write, therefore neither operation takes place.

iii) After the partition is repaired, the values of A and B at Z may be out of date, due to clients having written new values at servers X and Y. e.g. versions v1:

| X | Y | Z |
|---|---|---|
| A= 200(v1) | A= 200(v1) | A= 100(vo) |
| B= 300(v1) | B= 300(v1) | B= 100(vo) |

Then another partition occurs so that X and Z are separated from Y.

The client *Read* request causes an attempt to obtain a read quorum from X and Z. This notes that the versions (v0) at Z are out of date and then Z gets up-to-date versions of A and B from X.

Now the read quorum = 1+1 and the read operation can be done. Similarly the write operation can be done.