## SOLUTIONS TO CHAPTER 1 PROBLEMS

1. **Q:** An alternative definition for a distributed system is that of a collection of independent computers providing the view of being a *single system*, that is, it is completely hidden from users that there even multiple computers. Give an example where this view would come in very handy.

   **A:** What immediately comes to mind is parallel computing. If one could design programs that run without any serious modifications on distributed systems that appear to be the same as nondistributed systems, life would be so much easier. Achieving a single-system view is by now considered virtually impossible when performance is in play.

2. **Q:** What is the role of middleware in a distributed system?

   **A:** To enhance the distribution transparency that is missing in network operating systems. In other words, middleware aims at improving the single-system view that a distributed system should have.

3. **Q:** Many networked systems are organized in terms of a back office and a front office. How does organizations match with the coherent view we demand for a distributed system?

   **A:** A mistake easily made is to assume that a distributed system as operating in an organization, should be spread across the entire organization. In practice, we see distributed systems being installed along the way that an organization is split up. In this sense, we could have a distributed system supporting back-office procedures and processes, as well as a separate front-office system. Of course, the two may be coupled, but there is no reason for letting this coupling be fully transparent.

4. **Q:** Explain what is meant by (distribution) transparency, and give examples of different types of transparency.

   **A:** Distribution transparency is the phenomenon by which distribution aspects in a system are hidden from users and applications. Examples include access transparency, location transparency, migration transparency, relocation transparency, replication transparency, concurrency transparency, failure transparency, and persistence transparency.

5. **Q:** Why is it sometimes so hard to hide the occurrence and recovery from failures in a distributed system?

   **A:** It is generally impossible to detect whether a server is actually down, or that it is simply slow in responding. Consequently, a system may have to report that a service is not available, although, in fact, the server is just slow.

6. **Q:** Why is it not always a good idea to aim at implementing the highest degree of transparency possible?

   **A:** Aiming at the highest degree of transparency may lead to a considerable loss of performance that users are not willing to accept.

7. **Q:** What is an open distributed system and what benefits does openness provide?

   **A:** An open distributed system offers services according to clearly defined rules. An open system is capable of easily interoperating with other open systems but also allows applications to be easily ported between different implementations of the same system.

8. **Q:** Describe precisely what is meant by a scalable system.

   **A:** A system is scalable with respect to either its number of components, geographical size, or number and size of administrative domains, if it can grow in one or more of these dimensions without an unacceptable loss of performance.

9. **Q:** Scalability can be achieved by applying different techniques. What are these techniques?

   **A:** Scaling can be achieved through distribution, replication, and caching.

10. **Q:** Explain what is meant by a virtual organization and give a hint on how such organizations could be implemented.

    **A:** A virtual organization (VO) defines a group of users/applications that have access to a specified group of resources, which may be distributed across many different computers, owned by many different organizations. In effect, a VO defines who has access to what. This also suggests that the resources should keep an account of foreign users along with their access rights. This can often be done using standard access control mechanisms (like the rwx bits in UNIX), although foreign users may need to have a special account. The latter complicates matters considerably.

11. **Q:** When a transaction is aborted, we have said that 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.

12. **Q:** Executing nested transactions requires some form of coordination. Explain what a coordinator should actually do.

    **A:** A coordinator need simply ensure that if one of the nested transactions aborts, that all other subtransactions abort as well. Likewise, it should coordinate that all of them commit when each of them can. To this end, a nested transaction should wait to commit until it is told to do so by the coordinator.

13. **Q:** We argued that distribution transparancy may not be in place for pervasice systems. This statement is not true for all types of transparencies. Give an example.

    **A:** Think of migration transparency. In mnay pervasive systems, components are mobile and will need to re-establish connections when moving from one access point to another. Preferably, such handovers should be completely transparent to the user. Likewise, it can be argued that many other types of transparencies should be supported as well. However, what should not be hidden is a user is possibly accessing resources that are directly coupled to the user's current environment.

14. **Q:** We already gave some examples of distributed pervasive systems: home systems, electronic health-care systems, and sensor networks. Extend this list with more examples.

    **A:** There are quite a few other examples of pervasive systems. Think of large-scale wireless mesh networks in cities or neighborhoods that provide services such as Internet access, but also form the basis for other services like a news system. There are systems for habitat monitoring (as in wildlife resorts), electronic jails by which offenders are continuously monitored, large-scale integrated sports systems, office systems deploying active badges to know about the whereabouts of their employees, and so on.

15. **Q:** Sketch a design for a home system consisting of a separate media server that will allow for the attachment of a wireless client. The latter is connected to (analog) audio/video equipment and transforms the digital media streams to analog output. The server runs on a separate machine, possibly connected to the Internet, but has no keyboard and/or monitor connected.

## SOLUTIONS TO CHAPTER 2 PROBLEMS

1. **Q:** If a client and a server are placed far apart, we may see network latency dominating overall performance. How can we tackle this problem?

   **A:** It really depends on how the client is organized. It may be possible to divide the client-side code into smaller parts that can run separately. In that case, when one part is waiting for the server to respond, we can schedule another part. Alternatively, we may be able to rearrange the client so that it can do other work after having sent a request to the server. This last solution effectively replaces the synchronous client-server communication with asynchronous one-way communication.

2. **Q:** What is a three-tiered client-server architecture?

   **A:** A three-tiered client-server architecture consists of three logical layers, where each layer is, in principle, implemented at a separate machine. The highest layer consists of a client user interface, the middle layer contains the actual application, and the lowest layer implements the data that are being used.

3. **Q:** What is the difference between a vertical distribution and a horizontal distribution?

   **A:** Vertical distribution refers to the distribution of the different layers in a multitiered architectures across multiple machines. In principle, each layer is implemented on a different machine. Horizontal distribution deals with the distribution of a single layer across multiple machines, such as distributing a single database.

4. **Q:** Consider a chain of processes $P_1$, $P_2$, ..., $P_n$ implementing a multitiered client-server architecture. Process $P_i$ is client of process $P_{i+1}$, and $P_i$ will return a reply to $P_{i-1}$ only after receiving a reply from $P_{i+1}$. What are the main problems with this organization when taking a look at the request-reply performance at process $P_1$?

   **A:** Performance can be expected to be bad for large $n$. The problem is that each communication between two successive layers is, in principle, between two different machines. Consequently, the performance between $P_1$ and $P_2$ may also be determined by $n-2$ request-reply interactions between the other layers. Another problem is that if one machine in the chain performs badly or is even temporarily unreachable, then this will immediately degrade the performance at the highest level.

5. **Q:** In a structured overlay network, messages are routed according to the topology of the overlay. What is an important disadvantage of this approach?

   **A:** The problem is that we are dealing only with *logical* paths. It may very well be the case that two nodes $A$ and $B$ which are neighbors in the overlay network are physically placed far apart. As a consequence, the logically short path between $A$ and $B$ may require routing a message along a very long path in the underlying physical network.

6. **Q:** Consider the CAN network from Fig. 2-0. How would you route a message from the node with coordinates (0.2,0.3) to the one with coordinates (0.9,0.6)?

   **A:** There are several possibilities, but if we want to follow the shortest path according to a Euclidean distance, we should follow the route (0.2,0.3) $\rightarrow$ (0.6,0.7) $\rightarrow$ (0.9,0.6), which has a distance of 0.882. The alternative route (0.2,0.3) $\rightarrow$ (0.7,0.2) $\rightarrow$ (0.9,0.6) has a distance of 0.957.

7. **Q:** Considering that a node in CAN knows the coordinates of its immediate neighbors, a reasonable routing policy would be to forward a message to the closest node toward the destination. How good is this policy?

   **A:** In our example from the previous question, it can already be seen that it need not lead to the best route. If node (0.2,0.3) follows this policy for the message destined for node (0.9,0.6), it would send it off to node (0.7,0.2).

8. **Q:** Consider an unstructured overlay network in which each node randomly chooses $c$ neighbors. If $P$ and $Q$ are both neighbors of $R$, what is the probability that they are also neighbors of each other?

   **A:** Consider a network of $N$ nodes. If each node chooses $c$ neighbors at random, then the probability that $P$ will choose $Q$, or $Q$ chooses $P$ is roughly $2c\ /\ (N-1)$.

9. **Q:** Consider again an unstructured overlay network in which every node randomly chooses $c$ neighbors. To search for a file, a node floods a request to its neighbors and requests those to flood the request once more. How many nodes will be reached?

   **A:** An easy upper bound can be computed as $c \times (c-1)$, but in that case we ignore the fact that neighbors of node $P$ can be each other's neighbor as well. The probability $q$ that a neighbor of $P$ will flood a message only to nonneighbors of $P$ is 1 minus the probability of sending it to at least one neighbor of $P$:

$$q = 1 - \sum_{k=1}^{c-1} \binom{c-1}{k} \left(\frac{c}{N-1}\right)^{k} \left(1 - \frac{c}{N-1}\right)^{c-1-k}$$

   In that case, this flooding strategy will reach $c \times q\ (c-1)$ nodes. For example, with $c = 20$ and $N = 10,000$, a query will be flooded to 365.817 nodes.

10. **Q:** Not every node in a peer-to-peer network should become superpeer. What are reasonable requirements that a superpeer should meet?

    **A:** In the first place, the node should be highly available, as many other nodes rely on it. Also, it should have enough capacity to process requests. Most important perhaps is that fact that it can be trusted to do its job well.

11. **Q:** Consider a BitTorrent system in which each node has an outgoing link with a bandwidth capacity $B_{out}$ and an incoming link with bandwidth capacity $B_{in}$. Some of these nodes (called seeds) voluntarily offer files to be downloaded by others. What is the maximum download capacity of a BitTorrent client if we assume that it can contact at most one seed at a time?

    **A:** We need to take into account that the outgoing capacity of seeding nodes needs to be shared between clients. Let us assume that there are $S$ seeders and $N$ clients, and that each client randomly picks one of the seeders. The joint outgoing capacity of the seeders is $S \times B_{out}$, giving each of the clients $S \times B_{out}\ /\ N$ immediate download capacity. In addition, if clients help each other, each one of them will be able to download chunks at a rate of $B_{out}$, assuming that $B_{in} > B_{out}$. Note that because of the tit-for-tat policy, the download capacity of a BitTorrent client is mainly dictated by its *outgoing* capacity. In conclusion, the total download capacity will be $S \times B_{out} / N + B_{out}$.

**12. Q:** Give a compelling (technical) argument why the tit-for-tat policy as used in BitTorrent is far from optimal for file sharing in the Internet.

**A:** The reasoning is relatively simple. Most BitTorrent clients are operated behind asymmetric links such as provided by ADSL or cable modems. In general, clients are offered a high incoming bandwidth capacity, but no one really expects that clients have services to offer. BitTorrent does not make this assumption, and turns clients into collaborative servers. Having symmetric connections is then a much better match for the tit-for-tat policy.

**13. Q:** We gave two examples of using interceptors in adaptive middleware. What other examples come to mind?

**A:** There are several. For example, we could use an interceptor to support mobility. In that case, a request-level interceptor would first look up the current location of a referenced object before the call is forwarded. Likewise, an interceptor can be used to transparently encrypt messages when security is at stake. Another example is when logging is needed. Instead of letting this be handled by the application, we could simply insert a method-specific interceptor that would record specific events before passing a call to the referenced object. More of such example will easily come to mind.

**14. Q:** To what extent are interceptors dependent on the middleware where they are deployed?

**A:** In general, interceptors will be highly middleware-dependent. If we consider Fig. 2-0, it is easy to see why: the client stub will most likely be tightly bound to the lower level interfaces offered by the middleware, just as message-level interceptors will be highly dependent on the interaction between middleware and the local operating system. Nevertheless, it is possible to standardize these interfaces, opening the road to developing portable interceptors, albeit often for a specific class of middleware. This last approach has been followed for CORBA.

**15. Q:** Modern cars are stuffed with electronic devices. Give some examples of feed-back control systems in cars.

**A:** One obvious one is cruise control. On the one hand this subsystem measures current speed, and when it changes from the required setting, the car is slowed down or speeded up. The anti-lock braking systems (ABS) is another example. By pulsating the brakes of a car, while at the same time regulating the pressure that each wheel is exerting, it is possible to continue steering without losing control because the wheels are blocked. A last example is formed by the closed circuit of sensors that monitor engine condition. As soon as a dangerous state is reached, a car may come to an automatic halt to prevent the worst.

**16. Q:** Give an example of a self-managing system in which the analysis component is completely distributed or even hidden.

**A:** We already came across this type of system: in unstructured peer-to-peer systems where nodes exchange membership information, we saw how a topology could be generated. The analysis component consists of dropping certain links that will not help converge to the intended topology. Similar examples can be found in other such systems as we referred to as well.

**17. Q:** Sketch a solution to automatically determine the best trace length for predicting replication policies in Globule.

**A:** An origin server would need to use the traces from $T_i$ to $T_{i+1}$ to check its prediction of policy for that period. It can simply see whether the policy that would have been chosen on the actual access patterns is the same as the one chosen based on the requests in the period $T_{i-1}$ to $T_i$. This would allow the server to compute the prediction error. By varying the trace length, the origin server would be able find the length for which the prediction is minimal. In this way, we get an automatic determination of the optimal trace length, effectively contributing to the self-managing nature of Globule.

**18. Q:** Using existing software, design and implement a BitTorrent-based system for distributing files to many clients from a single, powerful server. Matters are simplified by using a standard Web server that can operate as tracker.

## SOLUTIONS TO CHAPTER 3 PROBLEMS

**1. Q:** In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 15 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in a cache in main memory. If a disk operation is needed, as is the case one-third of the time, an additional 75 msec is required, during which time the thread sleeps. How many requests/sec can the server handle if it is single threaded? If it is multithreaded?

**A:** In the single-threaded case, the cache hits take 15 msec and cache misses take 90 msec. The weighted average is $2/3 \times 15 + 1/3 \times 90$. Thus the mean request takes 40 msec and the server can do 25 per second. For a multithreaded server, all the waiting for the disk is overlapped, so every request takes 15 msec, and the server can handle 66 2/3 requests per second.

2. **Q:** Would it make sense to limit the number of threads in a server process?

   **A:** Yes, for two reasons. First, threads require memory for setting up their own private stack. Consequently, having many threads may consume too much memory for the server to work properly. Another, more serious reason, is that, to an operating system, independent threads tend to operate in a chaotic manner. In a virtual memory system it may be difficult to build a relatively stable working set, resulting in many page faults and thus I/O. Having many threads may thus lead to a performance degradation resulting from page thrashing. Even in those cases where everything fits into memory, we may easily see that memory is accessed following a chaotic pattern rendering caches useless. Again, performance may degrade in comparison to the single-threaded case.

3. **Q:** In the text, we described a multithreaded file server, showing why it is better than a single-threaded server and a finite-state machine server. Are there any circumstances in which a single-threaded server might be better? Give an example.

   **A:** Yes. If the server is entirely CPU bound, there is no need to have multiple threads. It may just add unnecessary complexity. As an example, consider a telephone directory assistance number for an area with 1 million people. If each (name, telephone number) record is, say, 64 characters, the entire database takes 64 megabytes, and can easily be kept in the server's memory to provide fast lookup.

4. **Q:** Statically associating only a single thread with a lightweight process is not such a good idea. Why not?

   **A:** Such an association effectively reduces to having only kernel-level threads, implying that much of the performance gain of having threads in the first place, is lost.

5. **Q:** Having only a single lightweight process per process is also not such a good idea. Why not?

   **A:** In this scheme, we effectively have only user-level threads, meaning that any blocking system call will block the entire process.

6. **Q:** Describe a simple scheme in which there are as many lightweight processes as there are runnable threads.

   **A:** Start with only a single LWP and let it select a runnable thread. When a runnable thread has been found, the LWP creates another LWP to look for a next thread to execute. If no runnable thread is found, the LWP destroys itself.

7. **Q:** X designates a user's terminal as hosting the server, while the application is referred to as the client. Does this make sense?

   **A:** Yes, although it may seem a bit confusing. The whole idea is that the server controls the hardware and the application can send requests to manipulate that hardware. From this perspective the X Window server should indeed reside on the user's machine, having the application acting as its client.

**8.** **Q:** The X protocol suffers from scalability problems. How can these problems be tackled?

**A:** There are essentially two scalability problems. First, numerical scalability is problematic in the sense that too much bandwidth is needed. By using compression techniques, bandwidth can be considerably reduced. Second, there is a geographical scalability problem as an application and the display generally need to synchronize too much. By using cachinng techniques by which effectively state of the display is maintained at the application side, much synchronization traffic can be avoided as the application can inspect the local cache to fins out what the state of the display is.

**9.** **Q:** Proxies can support replication transparency by invoking each replica, as explained in the text. Can (the server side of) an application be subject to a replicated calls?

**A:** Yes: consider a replicated object *A* invoking another (nonreplicated) object *B*. If *A* consists of *k* replicas, an invocation of *B* will be done by each replica. However, *B* should normally be invoked only once. Special measures are needed to handle such replicated invocations.

**10.** **Q:** Constructing a concurrent server by spawning a process has some advantages and disadvantages compared to multithreaded servers. Mention a few.

**A:** An important advantage is that separate processes are protected against each other, which may prove to be necessary as in the case of a superserver handling completely independent services. On the other hand, process spawning is a relatively costly operation that can be saved when using multithreaded servers. Also, if processes do need to communicate, then using threads is much cheaper as in many cases we can avoid having the kernel implement the communication.

**11.** **Q:** Sketch the design of a multithreaded server that supports multiple protocols using sockets as its transport-level interface to the underlying operating system.

**A:** A relatively simple design is to have a single thread *T* waiting for incoming transport messages (TPDUs). If we assume the header of each TPDU contains a number identifying the higher-level protocol, the tread can take the payload and pass it to the module for that protocol. Each such module has a separate thread waiting for this payload, which it treats as an incoming request. After handling the request, a response message is passed to *T*, which, in turn, wraps it in a transport-level message and sends it to tthe proper destination.

**12. Q:** How can we prevent an application from circumventing a window manager, and thus being able to completely mess up a screen?

**A:** Use a microkernel approach by which the windowing system including the window manager are run in such a way that all window operations are required to go through the kernel. In effect, this is the essence of transferring the client-server model to a single computer.

**13. Q:** Is a server that maintains a TCP/IP connection to a client stateful or stateless?

**A:** Assuming the server maintains no other information on that client, one could justifiably argue that the server is stateless. The issue is that not the server, but the transport layer at the server maintains state on the client. What the local operating systems keep track of is, in principle, of no concern to the server.

**14. Q:** Imagine a Web server that maintains a table in which client IP addresses are mapped to the most recently accessed Web pages. When a client connects to the server, the server looks up the client in its table, and if found, returns the registered page. Is this server stateful or stateless?

**A:** It can be strongly argued that this is a stateless server. The important issue with stateless designs is not if *any* information is maintained by the server on its clients, but instead whether that information is needed for correctness. In this example, if the table is lost for what ever reason, the client and server can still properly interact as if nothing happened. In a stateful design, such an interaction would be possible only after the server had recovered from a possible fault.

**15. Q:** Strong mobility in UNIX systems could be supported by allowing a process to fork a child on a remote machine. Explain how this would work.

**A:** Forking in UNIX means that a complete image of the parent is copied to the child, meaning that the child continues just after the call to fork. A similar approach could be used for remote cloning, provided the target platform is exactly the same as where the parent is executing. The first step is to have the target operating system reserve resources and create the appropriate process and memory map for the new child process. After this is done, the parent's image (in memory) can be copied, and the child can be activated. (It should be clear that we are ignoring several important details here.)

**16. Q:** In Fig. 3-0 it is suggested that strong mobility cannot be combined with executing migrated code in a target process. Give a counterexample.

**A:** If strong mobility takes place through thread migration, it should be possible to have a migrated thread be executed in the context of the target process.

17. **Q:** Consider a process $P$ that requires access to file $F$ which is locally available on the machine where $P$ is currently running. When $P$ moves to another machine, it still requires access to $F$. If the file-to-machine binding is fixed, how could the systemwide reference to $F$ be implemented?

    **A:** The simplest solution is to create a separate process $Q$ that handles remote requests for $F$. Process $P$ is offered the same interface to $F$ as before, for example in the form of a proxy. Effectively, process $Q$ operates as a file server.

18. **Q:** Describe in detail how TCP packets flow in the case of TCP handoff, along with the information on source and destination addresses in the various headers.

    **A:** There are various ways of doing this, but a simple one is to let a front end to execute a three-way handshake and from there on forward packets to a selected server. That server sends TCP PDUs in which the source address corresponds to that of the front end. An alternative is to forward the first packet to a server. Note, however, that in this case the front end will continue to stay in the loop. The advantage of this scheme is that the selected server builds up the required TCP state (such as the sequence numbers to be used) instead of obtaining this information from the front end as in the first scenario.

## SOLUTIONS TO CHAPTER 4 PROBLEMS

1. **Q:** In many layered protocols, each layer has its own header. Surely it would be more efficient to have a single header at the front of each message with all the control in it than all these separate headers. Why is this not done?

   **A:** Each layer must be independent of the other ones. The data passed from layer $k+1$ down to layer $k$ contains both header and data, but layer $k$ cannot tell which is which. Having a single big header that all the layers could read and write would destroy this transparency and make changes in the protocol of one layer visible to other layers. This is undesirable.

2. **Q:** Why are transport-level communication services often inappropriate for building distributed applications?

   **A:** They hardly offer distribution transparency meaning that application developers are required to pay significant attention to implementing communication, often leading to proprietary solutions. The effect is that distributed applications, for example, built directly on top of sockets are difficult to port and to interoperate with other applications.

3. **Q:** A reliable multicast service allows a sender to reliably pass messages to a collection of receivers. Does such a service belong to a middleware layer, or should it be part of a lower-level layer?

   **A:** In principle, a reliable multicast service could easily be part of the transport layer, or even the network layer. As an example, the unreliable IP multicasting service is implemented in the network layer. However, because such services are currently not readily available, they are generally implemented using transport-level services, which automatically places them in the middleware. However, when taking scalability into account, it turns out that reliability can be guaranteed only if application requirements are considered. This is a strong argument for implementing such services at higher, less general layers.

**4.** **Q:** Consider a procedure *incr* with two integer parameters. The procedure adds one to each parameter. Now suppose that it is called with the same variable twice, for example, as *incr*(*i*, *i*). If *i* is initially 0, what value will it have afterward if call-by-reference is used? How about if copy/restore is used?

**A:** If call by reference is used, a pointer to *i* is passed to *incr*. It will be incremented two times, so the final result will be two. However, with copy/restore, *i* will be passed by value twice, each value initially 0. Both will be incremented, so both will now be 1. Now both will be copied back, with the second copy overwriting the first one. The final value will be 1, not 2.

**5.** **Q:** C has a construction called a union, in which a field of a record (called a struct in C) can hold any one of several alternatives. At run time, there is no sure-fire way to tell which one is in there. Does this feature of C have any implications for remote procedure call? Explain your answer.

**A:** If the runtime system cannot tell what type value is in the field, it cannot marshal it correctly. Thus unions cannot be tolerated in an RPC system unless there is a tag field that unambiguously tells what the variant field holds. The tag field must not be under user control.

**6.** **Q:** One way to handle parameter conversion in RPC systems is to have each machine send parameters in its native representation, with the other one doing the translation, if need be. The native system could be indicated by a code in the first byte. However, since locating the first byte in the first word is precisely the problem, can this actually work?

**A:** First of all, when one computer sends byte 0, it always arrives in byte 0. Thus the destination computer can simply access byte 0 (using a byte instruction) and the code will be in it. It does not matter whether this is the low-order byte or the high-order byte. An alternative scheme is to put the code in all the bytes of the first word. Then no matter which byte is examined, the code will be there.

**7.** **Q:** Assume a client calls an asynchronous RPC to a server, and subsequently waits until the server returns a result using another asynchronous RPC. Is this approach the same as letting the client execute a normal RPC? What if we replace the asynchronous RPCs with asynchronous RPCs?

**A:** No, this is not the same. An asynchronous RPC returns an acknowledgment to the caller, meaning that after the first call by the client, an additional message is sent across the network. Likewise, the server is acknowledged that its response has been delivered to the client. Two asynchronous RPCs may be the same, provided reliable communication is guaranteed. This is generally not the case.

8. **Q:** Instead of letting a server register itself with a daemon as in DCE, we could also choose to always assign it the same endpoint. That endpoint can then be used in references to objects in the server's address space. What is the main drawback of this scheme?

   **A:** The main drawback is that it becomes much harder to dynamically allocate objects to servers. In addition, many endpoints need to be fixed, instead of just one (i.e., the one for the daemon). For machines possibly having a large number of servers, static assignment of endpoints is not a good idea.

9. **Q:** Would it be useful to also make a distinction between static and dynamic RPCs?

   **A:** Yes, for the same reason it is useful with remote object invocations: it simply introduces more flexibility. The drawback, however, is that much of the distribution transparency is lost for which RPCs were introduced in the first place.

10. **Q:** Describe how connectionless communication between a client and a server proceeds when using sockets.

    **A:** Both the client and the server create a socket, but only the server binds the socket to a local endpoint. The server can then subsequently do a blocking read call in which it waits for incoming data from any client. Likewise, after creating the socket, the client simply does a blocking call to write data to the server. There is no need to close a connection.

11. **Q:** Explain the difference between the primitives mpi_bsend and mpi_isend in MPI.

    **A:** The primitive mpi_bsend uses buffered communication by which the caller passes an entire buffer containing the messages to be sent, to the local MPI runtime system. When the call completes, the messages have either been transferred, or copied to a local buffer. In contrast, with mpi_isend, the caller passes only a pointer to the message to the local MPI runtime system after which it immediately continues. The caller is responsible for not overwriting the message that is pointed to until it has been copied or transferred.

12. **Q:** Suppose that you could make use of only transient asynchronous communication primitives, including only an asynchronous receive primitive. How would you implement primitives for transient *synchronous* communication?

    **A:** Consider a synchronous send primitive. A simple implementation is to send a message to the server using asynchronous communication, and subsequently let the caller continuously poll for an incoming acknowledgment or response from the server. If we assume that the local operating system stores incoming messages into a local buffer, then an alternative implementation is to block the caller until it receives a signal from the operating system that a message has arrived, after which the caller does an asynchronous receive.

**13.** **Q:** Suppose that you could make use of only transient synchronous communication primitives. How would you implement primitives for transient *asynchronous* communication?

**A:** This situation is actually simpler. An asynchronous send is implemented by having a caller append its message to a buffer that is shared with a process that handles the actual message transfer. Each time a client appends a message to the buffer, it wakes up the send process, which subsequently removes the message from the buffer and sends it its destination using a blocking call to the original send primitive. The receiver is implemented in a similar fashion by offering a buffer that can be checked for incoming messages by an application.

**14.** **Q:** Does it make sense to implement persistent asynchronous communication by means of RPCs?

**A:** Yes, but only on a hop-to-hop basis in which a process managing a queue passes a message to a next queue manager by means of an RPC. Effectively, the service offered by a queue manager to another is the storage of a message. The calling queue manager is offered a proxy implementation of the interface to the remote queue, possibly receiving a status indicating the success or failure of each operation. In this way, even queue managers see only queues and no further communication.

**15.** **Q:** In the text we stated that in order to automatically start a process to fetch messages from an input queue, a daemon is often used that monitors the input queue. Give an alternative implementation that does not make use of a daemon.

**A:** A simple scheme is to let a process on the receiver side check for any incoming messages each time that process puts a message in its own queue.

**16.** **Q:** Routing tables in IBM WebSphere, and in many other message-queuing systems, are configured manually. Describe a simple way to do this automatically.

**A:** The simplest implementation is to have a centralized component in which the topology of the queuing network is maintained. That component simply calculates all best routes between pairs of queue managers using a known routing algorithm, and subsequently generates routing tables for each queue manager. These tables can be downloaded by each manager separately. This approach works in queuing networks where there are only relatively few, but possibly widely dispersed, queue managers.

A more sophisticated approach is to decentralize the routing algorithm, by having each queue manager discover the network topology, and calculate its own best routes to other managers. Such solutions are widely applied in computer networks. There is no principle objection for applying them to message-queuing networks.

**17.** **Q:** With persistent communication, a receiver generally has its own local buffer where messages can be stored when the receiver is not executing. To create such a buffer, we may need to specify its size. Give an argument why this is preferable, as well as one against specification of the size.

**A:** Having the user specify the size makes its implementation easier. The system creates a buffer of the specified size and is done. Buffer management becomes easy. However, if the buffer fills up, messages may be lost. The alternative is to have the communication system manage buffer size, starting with some default size, but then growing (or shrinking) buffers as need be. This method reduces the chance of having to discard messages for lack of room, but requires much more work of the system.

**18.** **Q:** Explain why transient synchronous communication has inherent scalability problems, and how these could be solved.

**A:** The problem is the limited geographical scalability. Because synchronous communication requires that the caller is blocked until its message is received, it may take a long time before a caller can continue when the receiver is far away. The only way to solve this problem is to design the calling application so that it has other useful work to do while communication takes place, effectively establishing a form of asynchronous communication.

**19.** **Q:** Give an example where multicasting is also useful for discrete data streams.

**A:** Passing a large file to many users as is the case, for example, when updating mirror sites for Web services or software distributions.

**20.** **Q:** Suppose that in a sensor network measured temperatures are not timestamped by the sensor, but are immediately sent to the operator. Would it be enough to guarantee only a maximum end-to-end delay?

**A:** Not really if we assume that the operator would still need to know when the measurement took place. In this case, a timestamp can be attached when the measurement is received, but this would mean that we should also have guarantees for minimum end-to-end delays.

**21.** **Q:** How could you guarantee a maximum end-to-end delay when a collection of computers is organized in a (logical or physical) ring?

**A:** We let a token circulate the ring. Each computer is permitted to send data across the ring (in the same direction as the token) only when holding the token. Moreover, no computer is allowed to hold the token for more than $T$

seconds. Effectively, if we assume that communication between two adjacent computers is bounded, then the token will have a maximum circulation time, which corresponds to a maximum end-to-end delay for each packet sent.

**22. Q:** How could you guarantee a minimum end-to-end delay when a collection of computers is organized in a (logical or physical) ring?

**A:** Strangely enough, this is much harder than guaranteeing a maximum delay. The problem is that the receiving computer should, in principle, not receive data before some elapsed time. The only solution is to buffer packets as long as necessary. Buffering can take place either at the sender, the receiver, or somewhere in between, for example, at intermediate stations. The best place to temporarily buffer data is at the receiver, because at that point there are no more unforeseen obstacles that may delay data delivery. The receiver need merely remove data from its buffer and pass it to the application using a simple timing mechanism. The drawback is that enough buffering capacity needs to be provided.

**23. Q:** Despite that multicasting is technically feasible, there is very little support to deploy it in the Internet. The answer to this problem is to be sought in down-to-earth business models: no one really knows how to make money out of multicasting. What scheme can you invent?

**A:** The problem is mainly caused by ISPs, as they see no reason to save on bandwidth (their clients are paying anyway). However, matters may change in scenarios such as the following. An Internet broadcasting service pays for a certain quality-of-service as promised by various ISPs. Each of these ISPs will see a drop in their income when they cannot meet these QoS requirements. At this point, they may now have an incentive to start deploying multicasting as they can offer better (and guaranteed) service.

**24. Q:** Normally, application-level multicast trees are optimized with respect stretch, which is measured in terms of delay or hop counts. Give an example where this metric could lead to very poor trees.

**A:** The underlying assumption with stretch is that communication delays predominate performance. However, in the case of, for example, video broadcasting, it is the available which counts. In that case, we would like to construct trees that *maximize* costs (measured in terms of bandwidth).

**25. Q:** When searching for files in an unstructured peer-to-peer system, it may help to restrict the search to nodes that have similar files as yourself. Explain how gossiping can help to find those nodes.

**A:** The idea is very simple: if, during gossiping, nodes exchange membership information, every node will eventually get to know about all other nodes in the system. Each time it discovers a new node, it can be evaluated with respect to its semantic proximity, for example, by counting the number of files in common. The semantically nearest nodes are then selected for submitting a search query.

## SOLUTIONS TO CHAPTER 5 PROBLEMS

**1. Q:** Give an example of where an address of an entity $E$ needs to be further resolved into another address to actually access $E$.

**A:** IP addresses in the Internet are used to address hosts. However, to access a host, its IP address needs to be resolved to, for example, an Ethernet address.

**2. Q:** Would you consider a URL such as *http://www.acme.org/index.html* to be location independent? What about *http://www.acme.nl/index.html*?

**A:** Both names can be location independent, although the first one gives fewer hints on the location of the named entity. Location independent means that the name of the entity is independent of its address. By just considering a name, nothing can be said about the address of the associated entity.

**3. Q:** Give some examples of true identifiers.

**A:** Examples are ISBN numbers for books, identification numbers for software and hardware products, employee numbers within a single organization, and Ethernet addresses (although some addresses are used to identify a machine instead of just the Ethernet board).

**4. Q:** Is an identifier allowed to contain information on the entity it refers to?

**A:** Yes, but that information is not allowed to change, because that would imply changing the identifier. The old identifier should remain valid, so that changing it would imply that an entity has two identifiers, violating the second property of identifiers.

**5. Q:** Outline an efficient implementation of globally unique identifiers.

**A:** Such identifiers can be generated locally in the following way. Take the network address of the machine where the identifier is generated, append the local time to that address, along with a generated pseudo-random number. Although, in theory, it is possible that another machine in the world can generate the same number, chances that this happens are negligible.

**6. Q:** Consider the Chord system as shown in Fig. 5-0 and assume that node 7 has just joined the network. What would its finger table be and would there be any changes to other finger tables?

**A:** Let us first consider the finger table for node 7. Using the same method as we introduced when discussing Chord, it can be seen that this table will be [9, 9, 11, 18, 28]. For example, entry #2 is computed as $succ(7+21) = succ(9) = 9$. More tables will need to change, however, in particular those of node 4 (which becomes [7,7,9,14,28]), node 21 ([28,28,28,1,7]) and node 1 ([4,4,7,9,18]).

7. **Q:** Consider a Chord DHT-based system for which $k$ bits of an $m$-bit identifier space have been reserved for assigning to superpeers. If identifiers are randomly assigned, how many superpeers can one expect to have in an $N$-node system?

   **A:** The total number of superpeers in an overlay of $N$ nodes will be equal to $\min\{2^{k-m}N, 2^k\}$.

8. **Q:** If we insert a node into a Chord system, do we need to instantly update all the finger tables?

   **A:** Fortunately not. Consider the previous question and assume that only the finger table of node #7 is installed and that the rest are kept as they are. The worst that can happen is that a request to look up, say, key 5, is routed to node #9 instead of #7. However, node #9 knows that node #7 has joined the system and can therefore take corrective action.

9. **Q:** What is a major drawback of recursive lookups when resolving a key in a DHT-based system?

   **A:** A problem is that the requesting client will never be able to discover what went wrong when no answer is returned. Somewhere along the route that corresponds to resolving the key, a message may have been lost or a node may have failed. For this reason, an iterative lookup is sometimes preferred: the client will know exactly which part of the lookup did not come through and may be able to choose an alternative node to help out.

10. **Q:** A special form of locating an entity is called anycasting, by which a service is identified by means of an IP address (see, for example, Sending a request to an anycast address, returns a response from a server implementing the service identified by that anycast address. Outline the implementation of an anycast service based on the hierarchical location service described in Sec. 5.2.4.

    **A:** Each service has a unique identifier associated with it. Any server implementing that service, inserts its network-level address into the location service at the directory node of the leaf domain in which the server resides. Lookup requests use the identifier of the service, and will automatically return the nearest server implementing that service.

11. **Q:** Considering that a two-tiered home-based approach is a specialization of a hierarchical location service, where is the root?

    **A:** The root is formed jointly by all home locations, but is partitioned in such a way that each mobile entity has its own root server.

12. **Q:** Suppose that it is known that a specific mobile entity will almost never move outside domain $D$, and if it does, it can be expected to return soon. How can this information be used to speed up the lookup operation in a hierarchical location service?

    **A:** Simply encode the domain $D$ in the identifier for the entity that is used for the lookup operation. The operation can then be immediately forwarded to the directory node *dir(D)*, from where the search continues.

13. **Q:** In a hierarchical location service with a depth of $k$, how many location records need to be updated at most when a mobile entity changes its location?

    **A:** Changing location can be described as the combination of an insert and a delete operation. An insert operation requires that at worst $k+1$ location records are to be changed. Likewise, a delete operation also requires changing $k+1$ records, where the record in the root is shared between the two operations. This leads to a total of $2k+1$ records.

14. **Q:** Consider an entity moving from location $A$ to $B$, while passing several intermediate locations where it will reside for only a relatively short time. When arriving at $B$, it settles down for a while. Changing an address in a hierarchical location service may still take a relatively long time to complete, and should therefore be avoided when visiting an intermediate location. How can the entity be located at an intermediate location?

    **A:** Combine the hierarchical location service with forwarding pointers. When the entity starts to move, it leaves behind a forwarding pointer at $A$ to its next (intermediate) location. Each time it moves again, a forwarding pointer is left behind. Upon arrival in $B$, the entity inserts its new address into the hierarchical location service. The chain of pointers is subsequently cleaned up, and the address in $A$ is deleted.

15. **Q:** The root node in hierarchical location services may become a potential bottleneck. How can this problem be effectively circumvented?

    **A:** An important observation is that we are using only random bit strings as identifiers. As a result, we can easily partition the identifier space and install a separate root node for each part. In addition, the partitioned root node should be spread across the network so that accesses to it will also be spread.

16. **Q:** Give an example of how the closure mechanism for a URL could work.

    **A:** Assuming a process knows it is dealing with a URL, it first extracts the scheme identifier from the URL, such as the string *ftp:*. It can subsequently look up this string in a table to find an interface to a local implementation of the FTP protocol. The next part of the closure mechanism consists of extracting the host name from the URL, such as *www.cs.vu.nl*, and passing that to the local DNS name server. Knowing where and how to contact the DNS server is an important part of the closure mechanism. It is often hard-coded into the

URL name resolver that the process is executing. Finally, the last part of the URL, which refers to a file that is to be looked up, is passed to the identified host. The latter uses its own local closure mechanism to start the name resolution of the file name.

**17.** **Q:** Explain the difference between a hard link and a soft link in UNIX systems. Are there things that can be done with a hard link that cannot be done with a soft link or vice versa?

**A:** A hard link is a named entry in a directory file pointing to the same file descriptor as another named entry (in possibly a different directory). A symbolic link is a file containing the (character string) name of another file. With a soft link you can link to a different disk partition or even to a different machine.

**18.** **Q:** High-level name servers in DNS, that is, name servers implementing nodes in the DNS name space that are close to the root, generally do not support recursive name resolution. Can we expect much performance improvement if they did?

**A:** Probably not: because the high-level name servers constitute the global layer of the DNS name space, it can be expected that changes to that part of the name space do not occur often. Consequently, caching will be highly effective, and much long-haul communication will be avoided anyway. Note that recursive name resolution for low-level name servers is important, because in that case, name resolution can be kept local at the lower-level domain in which the resolution is taking place.

**19.** **Q:** Explain how DNS can be used to implement a home-based approach to locating mobile hosts.

**A:** The DNS name of a mobile host would be used as (rather poor) identifier for that host. Each time the name is resolved, it should return the current IP address of the host. This implies that the DNS server responsible for providing that IP address will act as the host's name server. Each time the host moves, it contacts this home server and provides it with its current address. Note that a mechanism should be available to avoid caching of the address. In other words, other name servers should be told *not* to cache the address found.

**20.** **Q:** How is a mounting point looked up in most UNIX systems?

**A:** By means of a mount table that contains an entry pointing to the mount point. This means that when a mounting point is to be looked up, we need to go through the mount table to see which entry matches a given mount point.

**21.** **Q:** Consider a distributed file system that uses per-user name spaces. In other words, each user has his own, private name space. Can names from such name spaces be used to share resources between two different users?

**A:** Yes, provided names in the per-user name spaces can be resolved to names in a shared, global name space. For example, two identical names in different name spaces are, in principle, completely independent and may refer to different entities. To share entities, it is necessary to refer to them by names from a shared name space. For example, Jade relies on DNS names and IP addresses that can be used to refer to shared entities such as FTP sites.

**22. Q:** Consider DNS. To refer to a node *N* in a subdomain implemented as a different zone than the current domain, a name server for that zone needs to be specified. Is it always necessary to include a resource record for that server's address, or is it sometimes sufficient to provide only its domain name?

**A:** When the name server is represented by a node *NS* in a domain other than the one in which *N* is contained, it is enough to give only its domain name. In that case, the name can be looked up by a separate DNS query. This is not possible when *NS* lies in the same subdomain as *N*, for in that case, you would need to contact the name server to find out its address.

**23. Q:** Counting common files is a rather naive way of defining semantic proximity. Assume you were to build semantic overlay networks based on text documents, what other semantic proximity function can you think of?

**A:** One intriguing one is to have a look at actual content when possible. In the case of documents, one could look at similarity functions derived from information retrieval, such as the Vector Space Model (VSM).

**24. Q:** Set up your own DNS server. Install BIND on either a Windows or UNIX machine and configure it for a few simple names. Test your configuration using tools such as the Domain Information Groper (DIG). Make sure your DNS database includes records for name servers, mail servers, and standard servers. Note that if you are running BIND on a machine with host name *HOSTNAME*, you should be able to resolve names of the form *RESOURCE-NAME.HOSTNAME*.

## SOLUTIONS TO CHAPTER 6 PROBLEMS

**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 $0.01 \times 60$ sec, or 600 msec.

3. **Q:** One of the modern devices that have (silently) crept into distributed systems are GPS receivers. Give examples of distributed applications that can make use of GPS information.

   **A:** One typical example that comes to mind is sports and health care. There are now GPS-based body-area networks that allow a person to keep track of his pace while exercising an outdoors sports. These networks are often augmented with heart rate monitors and can be hooked up to a computer to downlaod the sensored data for further analysis. Another example is formed by car-navigation equipment, which is generally based on GPS receivers. Hooked up to a (portable) personal computer that can be connected to the Internet, maps and such can be continously updated. Related are GPS-based distributed systems for tracking the movement of cars and trucks.

4. **Q:** When a node synchronizes its clock to that of another node, it is generally a good idea to take previous measurements into account as well. Why? Also, give an example of how such past readings could be taken into account.

   **A:** The obvious reason is that there may be an error in the current reading. Assuming that clocks need only be gradually adjusted, one possibility is to consider the last $N$ values and compute a median or average. If the measured value falls outside a current interval, it is not taken into account (but is added to the list). Likewise, a new value can be computed by taking a weighted average, or an aging algorithm.

5. **Q:** Add a new message to Fig. 6-0 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 $A$.

6. **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.

7. **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.

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. 6-0), upon receiving a message from a process releasing its exclusive access to the resources 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. 6-0 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 such that every request is answered immediately, either with permission or with denial. If there are no processes accessing resources 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 accessed resources and the queue by reading the 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 access a resources, 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 that 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. 6-0 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 point-to-point message is as expensive as doing a broadcast, we need only send one broadcast message to all processes requesting access to the resource. Likewise, only one release broadcast message is needed. The delay becomes $1 + (n - 1)$: one delay coming from the broadcast request, and an additional $n - 1$ as we still need to receive a message from each other process before being allowed to access the resource.

**13. Q:** A distributed system may have multiple, independent resources. Imagine that process 0 wants to access resource $A$ and process 1 wants to access resource $B$. Can Ricart and Agrawala's algorithm lead to deadlocks? Explain your answer.

**A:** It depends on the ground rules. If processes access resources strictly sequentially, that is, a process holding a resource may not attempt to access another one, then there is no way that it can block while holding a resource that some other process wants. The system is then deadlock free. On the other hand, if process 0 may hold resource $A$ and then try to access resource $B$, 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 resource is handled independently of all the others.

**14. 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.

**15. Q:** In Fig. 6-0 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.

**16. Q:** UNIX systems provide many facilities to keep computers in synch. Notably the combination of the *crontab* tool (which allows to automatically schedule operations) and various synchronization commands are powerful. Configure a UNIX that keeps the local time accurate with in the range of a single second. Likewise, configure an automatic backup facility by which a number of crucial files are automatically transferred to a remote machine once every 5 minutes. Your solution should be efficient when it comes to bandwidth usage.

## SOLUTIONS TO CHAPTER 7 PROBLEMS

**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:** 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 consistency constraints.

**3. 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.