一、概念题（共30分）
1. **k fault tolerant (chap 8)**
   A system is said to be k fault tolerant if it can survive faults in k components and still meet its specifications.

2. **Availability, Reliability (chap 8)**
   **Availability: Readiness for usage.** 说明系统已准备好，马上就可以使用。通常，它指在任何给定的时刻，系统都可以正确地操作，可根据用户的行为来执行它的功能。换句话说，高度可用的系统在任何给定的时刻都能及时地工作。
   **Reliability: Continuity of service delivery.** 指系统可以无故障地持续运行。与可用性相反，可靠性是根据时间间隔而不是任何时刻来进行定义的。

3. **recovery line (chap 8)**
   A recovery line corresponds to the most recent consistent collection of checkpoints.

4. **continuous consistency ranges (chap 7)**
   Deviation in numerical values between replicas, deviation in staleness between replicas, and deviation with respect to the ordering of update operations. These deviations are referred as forming continuous consistency ranges.

5. **eventual consistency (chap 7)**
   If no updates take place for a long time, all replicas will gradually become consistent. This form of consistency is called eventual consistency.

6. **happens-before relation (chap 6)**
   To synchronize logical clocks, Lamport defined a relation called happens-before. The expression a→b is read “a happens before b” and means that all processes agree that first event a occurs, then afterward, event b occurs. The happens-before relation can be observed directly in two situations:
   1) If a and b are events in the same process, and a occurs before b, then a→b is true.
   2) If a is the event of a message being sent by one process, and b is the event of the message being received by another process, the a→b is also true. A message cannot be received before
it is sent, or even at the same time it is sent, since it takes a finite, nonzero amount of time to arrive.

7. finger table (chap 5)

Instead of linear approach toward key lookup, each Chord node maintains a finger table of at most m entries. If FT_p denotes the finger table of node p, then FT_p[i] = succ (p + 2^{i-1})

Put in other words, the i-th entry points to the first node succeeding p by at least 2^{i-1}.

8. out of band data (chap 3)

Data is to be processed by the server before any other data from that client.

9. MapReduce (5pt)

MapReduce is a software framework introduced by Google to support distributed computing on large data sets on clusters of computers. The framework is inspired by map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as their original forms. MapReduce libraries have been written in C++, Java, Python and other programming languages.

10. A scalable system

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.
二、简答题（共70分）

1. Q: What is the difference between a vertical distribution and a horizontal distribution? (chap 2, 5pt)
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.

2. Q: Is a server that maintains a TCP/IP connection to a client stateful or stateless? (chap 3)
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.

3. 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? (chap 4)
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.

4. Q: Routing tables in IBM WebSphere, and in many other message-queuing systems, are configured manually. Describe a simple way to do this automatically. (chap 4)
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.

5. Q: Is an identifier allowed to contain information on the entity it refers to? (chap 5)
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.

6. 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. (chap 6)
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.
7. 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 resource, 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. (chap 6)

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.

8. What kind of consistency would you use to implement an electronic stock market? Explain your answer. (chap 7)

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.

9. When using a lease, is it necessary that the clocks of a client and the server, respectively, are tightly synchronized? (chap 7)

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.

10. 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? (chap 7)

A: No. As soon as the updating process receives an acknowledgment 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.

11. Q: State-based leases are used to offload a server by letting it allow to keep track of as few clients as needed. Will this approach necessarily lead to better performance? (chap 7)

A: No, for the simple reason that for some clients it would still be better to inform them when updates appended. Not maintaining any state may lead to the situation that these clients will often poll the already busy server.

12. With asynchronous RPCs, a client is blocked until its request has been accepted by the server. To what extent do failures affect the semantics of asynchronous RPCs? (chap 8)

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.

13. Q: Despite that GFS scales well, it could be argued that the master is still a potential bottleneck. What would be a reasonable alternative to replace it? (chap 11)
A: Considering that master uses a file name to look up a chunk server, we could also implement the master in the form of a DHT-based system and use a hash of the file name as the key to be looked up. In this way, one would obtain a fully decentralized master.

14. Q: In NFS, attributes are cached using a write-through cache coherence policy. Is it necessary to forward all attributes changes immediately? (chap 11)
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.