一、概念题（共40分）

1. Availability, reliability (4pt) (chap8)
   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。

2. Stateless server, soft state (4pt) (chap3)
   A stateless server does not keep information on the state of its clients, and can change its own state without having to inform any client.
   A particular form of a stateless design is where the server maintains what is known as soft state. The server promises to maintain state on behalf of the client, but only for a limited time. After that time has expired, the server falls back to default behavior.

3. Recovery line (2pt) (chap8)
   A recovery line corresponds to the most recent consistent collection of checkpoints.

4. Scalability (2pt) (chap1)
   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.

5. Stable storage (2pt) (chap8)
   Replicate all data on at least two disks, and keep one copy “correct” at all times.
   After a crash:
   a) If both disks are identical: you’re in good shape.
   b) If one is bad, but the other is okay (checksums): choose the good one.
   c) If both seem okay, but are different: choose the main disk.
   d) If both aren’t good: you’re not in a good shape.

6. Content replication (2pt) (chap7)
   Model: We consider objects (and don’t worry whether they contain just data or code, or both)
Distinguish different processes: A process is capable of hosting a replica of an object or data:
- Permanent replicas: Process/machine always having a replica
- Server-initiated replica: Process that can dynamically host a replica on request of another server in the data store
- Client-initiated replica: Process that can dynamically host a replica on request of a client (client cache)

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.
Happen-before is a transitive relation, so if a→b and b→c, then a→c.

8. MapReduce (2pt)
MapReduce is a programming model and engine for processing and generating large datasets. Users specify a map function to process a key/value pair and produce intermediate key/value pairs, and a reduce function to merge all values associated with the same intermediate key. The resulting program runs automatically in parallel and can run on a large cluster of machines. The runtime system handles the details of data partitioning, scheduling execution on a set of machines, and managing the necessary internal communications. This makes it easy for any programmer to leverage the resources of a large distributed system.

9. Goals of distributed systems (2pt) (chap1)
A distributed system should make resources easily accessible; it should reasonably hide the fact that resources are distributed across a network; it should be open; and it should be scalable.

10. Transparency, different types of transparency (4pt) (chap1)
A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.
The concept of transparency can be applied to several aspects of a distributed system, the most important ones include access transparency, location transparency, migration transparency, relocation transparency, replication transparency, concurrency transparency, failure transparency, persistence transparency, and security transparency.

11. A finite machine (2pt) (chap 3)
A state machine consists of state variables, which encode its state, and commands, which transform its state. Each command is implemented by a deterministic program; execution of the command is atomic with respect to other commands and modifies the state variables and/or produces some output.
一个process接收消息，依照状态图来对指定server进行操作，并更新状态，以此循环。

12. A virtual machine (2pt) (chap 3)
A virtual machine can support individual processes or a complete system depending on the abstraction level where virtualization occurs. Some VMs support flexible hardware usage and software isolation, while others translate from one instruction set to another.

13. Cloud computing (3pt)
A popular phrase that is shorthand for applications that were developed to be rich Internet applications that run on the Internet (or "cloud"). In the cloud computing paradigm, software that is traditionally installed on personal computers is shifted or extended to be accessible via the Internet. These "cloud applications" or "cloud apps" utilize massive data centers and powerful servers that host web applications and web services. They can be accessed by anyone with a suitable Internet connection and a standard web browser.

14. Isochronous transmission mode (2pt) (chap 4)
It is necessary that data units are transferred on time. This means that data transfer is subject to a maximum and minimum end-to-end delay, also referred to as bounded (delay) jitter.

15. Data-centric consistency, client-centric consistency (2pt) (chap 7)
Data-Centric Consistency Models, the general organization of a logical data store, physically distributed and replicated across multiple processes. In particular, each process that can access data from the store is assumed to have a local (or nearby) copy available of the entire store. Write operations are propagated to the other copies. 比较考虑对数据的读写一致。Client-centric consistency, the principle of a mobile user accessing different replicas of a distributed database. The data stores are characterized by the lack of simultaneous updates,
or when such updates happen, they can easily be resolved. Most operations involve reading
data. 是考虑每次所连的副本不同，从用户角度考察每一个副本修改的一致性。

16. Virtually synchronous, atomic multicast (4pt) (chap8)
Essence: Consider views $V = RCV(c) \cup SND(c)$
Processes are added or deleted from a view $V$ through view changes to $V^*$; a view change is
to be executed locally by each $P \in V \cap V^*$

(1) For each consistent state, there is a unique view on which all its members agree.
   Note: implies that all non-faulty processes see all view changes in the same order
(2) If message $m$ is sent to $V$ before a view change $vc$ to $V^*$, then either all $P \in V$ that
   execute $vc$ receive $m$, or no processes $P \in V$ that execute $vc$ receive $m$.
   Note: all non-faulty members in the same view get to see the same set of multicast
   messages.
(3) A message sent to view $V$ can be delivered only to processes in $V$, and is discarded by
   the following views.
A reliable multicast algorithm satisfying (1)–(3) is virtually synchronous

Model: a multicast channel $c$ with two (possibly overlapping) groups: The sender group
$SND(c)$ of processes that submit messages to channel $c$. The receiver group $RCV(c)$ of processes
that receive messages from channel $c$.

Atomic multicast: All active processes receive the same thing. Ensure that a message $m$
submitted to channel $c$ is delivered to process $P \in RCV(c)$ only if $m$ is delivered to all members
of $RCV(c)$

Guarantee: A message is delivered only to the non-faulty members of the current group. All
members should agree on the current group membership.

17. Basic paxos algorithm (4pt)
   http://en.wikipedia.org/wiki/Paxos_(computer_science)

This protocol is the most basic of the Paxos family. Each instance of the Basic Paxos protocol decides on a single
output value. The protocol proceeds over several rounds. A successful round has two phases. A Proposer should not
initiate Paxos if it cannot communicate with at least a Quorum of Acceptors:

**Phase 1a: Prepare**
A Proposer (the leader) creates a proposal identified with a number N. This number must be greater than any previous proposal number used by this Proposer. Then, it sends a Prepare message containing this proposal to a Quorum of Acceptors.

**Phase 1b: Promise**

If the proposal's number N is higher than any previous proposal number received from any Proposer by the Acceptor, then the Acceptor must return a promise to ignore all future proposals having a number less than N. If the Acceptor accepted a proposal at some point in the past, it must include the previous proposal number and previous value in its response to the Proposer.

Otherwise, the Acceptor can ignore the received proposal. It does not have to answer in this case for Paxos to work. However, for the sake of optimization, sending a denial (Nack) response would tell the Proposer that it can stop its attempt to create consensus with proposal N.

**Phase 2a: Accept Request**

If a Proposer receives enough promises from a Quorum of Acceptors, it needs to set a value to its proposal. If any Acceptors had previously accepted any proposals, then they'll have sent their values to the Proposer, who now must set the value of its proposal to the value associated with the highest proposal number reported by the Acceptors. If none of the Acceptors had accepted a proposal up to this point, then the Proposer may choose any value for its proposal.\(^{[17]}\)

The Proposer sends an Accept Request message to a Quorum of Acceptors with the chosen value for its proposal.

**Phase 2b: Accepted**

If an Acceptor receives an Accept Request message for a proposal N, it must accept it if and only if it has not already promised to only consider proposals having an identifier greater than N. In this case, it should register the corresponding value v and send an Accepted message to the Proposer and every Learner. Else, it can ignore the Accept Request.

Rounds fail when multiple Proposers send conflicting Prepare messages, or when the Proposer does not receive a Quorum of responses (Promise or Accepted). In these cases, another round must be started with a higher proposal number.

Notice that when Acceptors accept a request, they also acknowledge the leadership of the Proposer. Hence, Paxos can be used to select a leader in a cluster of nodes.

Here is a graphic representation of the Basic Paxos protocol. Note that the values returned in the Promise message are null the first time a proposal is made, since no Acceptor has accepted a value before in this round.

**Message flow: Basic Paxos**

(first round is successful)

<table>
<thead>
<tr>
<th>Client</th>
<th>Proposer</th>
<th>Acceptor</th>
<th>Learner</th>
</tr>
</thead>
</table>

5
18. CAP theorem (3pt)
   
   [Link to Wikipedia CAP theorem page]

   In theoretical computer science, the CAP theorem, also known as Brewer's theorem, states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:[1][2]

   - **Consistency** (all nodes see the same data at the same time)
   - **Availability** (a guarantee that every request receives a response about whether it was successful or failed)
   - **Partition tolerance** (the system continues to operate despite arbitrary message loss or failure of part of the system)

   According to the theorem, a distributed system can satisfy any two of these guarantees at the same time, but not all three.

19. At least once semantics, at most once semantics (4pt) (chap8)

   [Link to University of North Carolina webpage on IPC]

   **At-least-once:** The call executes at least once as long as the server machine does not fail. These semantics require very little overhead and are easy to implement. The client machine continues to send call requests to the server machine until it gets an acknowledgement. If one or more acknowledgements are lost, the server may execute the call multiple times. This approach works only if the requested operation is **idempotent**, that is, multiple invocations of it return the same result. Servers that implement only idempotent operations must be **stateless**, that is, must not change global state in response to client requests. Thus, RPC systems that support these semantics rely on the design of stateless servers.

   **At-most-once:** The call executes at most once - either it does not execute at all or it executes exactly once depending on whether the server machine goes down. Unlike the previous semantics, these semantics require the detection of duplicate packets, but work for non-idempotent operations.

20. Coordinated Checkpointing (2pt) chap8

   **Coordinated Checkpointing**

   **Essence:** Each process takes a checkpoint after a globally coordinated action

   **Question:** What advantages are there to coordinated checkpointing?
**Simple solution:** Use a two-phase blocking protocol:
- A coordinator multicasts a *checkpoint request* message
- When a participant receives such a message, it takes a checkpoint, stops sending (application) messages, and reports back that it has taken a checkpoint
- When all checkpoints have been confirmed at the coordinator, the latter broadcasts a *checkpoint done* message to allow all processes to continue

**Observation:** It is possible to consider only those processes that depend on the recovery of the coordinator, and ignore the rest

21. **Data store (2pt) (chap7)**
   Traditionally, consistency has always been discussed in the context of read and write operations on shared data, available by means of (distributed) shared memory, a (distributed) shared database, or a (distributed) file system. In this section, we use the broader term **data store**.
   The access to a replicated entity is typically uniform with access to a single, non-replicated entity. The replication itself should be transparent to an external user. Also, in a failure scenario, a failover of replicas is hidden as much as possible.

22. **Consistency unit (2pt) (chap7)**
   specifies the **data unit** over which consistency is to be measured.

23. **Primary-Based Protocols, Replicated-Write Protocols (4pt) (chap7)**
   Primary-Based protocols that allow processes to perform read operations on a locally available copy, but should forward write operations to a (fixed) primary copy.
   In replicated-write protocols, write operations can be carried out at multiple replicas instead of only one, as in the case of primary-based replicas.

24. **Universal Coordinated Time (2pt) (chap6)**
   UTC is an astronomical time standard that is the basis for the time on the "wall clock". In 1972 it was internationally agreed that the duration of the second should conform to the TAI stand, but that the number of seconds in an hour will have to be occasionally modified by inserting a leap second into UTC to maintain synchrony between the wall clock time and the astronomical phenomena, like day and night.

二、简答题（共38分）
1. Q: In a two-phase commit process, a participant is blocked in state READY, waiting for the global vote as sent by the coordinator. If that message is not received within a given time, what decisions should the participant should take?. (5pt) (chap8)

Chap8, page 357
we wish to check with others:
If Q is in COMMIT, then commit. If Q is in ABORT, then ABORT.
If Q in INIT, then can safely ABORT.
If all in READY, nothing can be done.

2. Q: 1) Resolve the following key lookups for the shown Chord-based P2P system: (5pt) (chap5)

<table>
<thead>
<tr>
<th>source</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>30</td>
</tr>
<tr>
<td>21</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>18</td>
</tr>
</tbody>
</table>

15@4: 14–18; 22@4: 20–21–28; 30@21: 28–1; 27@21: 28; 18@20: 4–14–18

2) Adjust the finger tables of nodes 18 and 14 when a node with ID 24 enters the ring. Also give the finger table of node 24. (5pt) (chap5)

Node 18: [20,20,24,28,4]; Node 14: [18,18,18,24,1]; Node 24: [28,28,28,1,9].

3) Chord allows keys to be looked up recursively or iteratively. Explain the differences, as well as the main advantage of iterative over recursive lookup. (5pt) (chap5)

With recursive lookups, a message is forwarded from peer to peer until it reaches its destination. In contrast, with an iterative lookup, the requester is returned the next peer it should ask for the key. One can argue that in the case of Chord, iterative lookups are much better: recursive lookups do not have the advantage of proximity-awareness. Also, note that iterative lookups have the advantage of letting the client handle failures more easily.
3. 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? (3pt) (chap3)

\[
\frac{1000}{(15*2/3 + 90*1/3)} = 25 \\
\frac{1000}{15} = 66.7
\]

4. Q: Explain the principle of an epidemic protocol. (5pt) (chap4)
Your answer should at least include that a process P randomly selects another process Q to exchange new data items, following either a pull, push, or push-pull protocol.

5. Q: Scalability can be achieved by applying different techniques. What are these techniques? (3pt) (chap1)
A: Scaling can be achieved through distribution, replication, and caching.

6. Q: List pitfalls when developing distributed systems. (3pt) (chap1)
The network is reliable. The network is secure. The network is homogeneous. The topology does not change. Latency is zero. Bandwidth is infinite. Transport cost is zero. There is one administrator.

7. Q: List four kinds of client-centric consistency models. (4pt) (chap7)
Monotonic reads, monotonic writes, read your writes, writes follow reads.

8. Q: Why is the following data store not sequentially consistent? Is it causally consistent? Be sure to explain your answer. (5pt) (chap7)

```
P1: W(x)a
---------------------------------------------
P2:   W(x)b                                    
---------------------------------------------
P3:    R(x)b R(x)a                             
---------------------------------------------
P4:     R(x)a R(x)b                            
```

It is not sequentially consistent because P3 and P4 are reading the effects of concurrent writes (by respectively P1 and P2) in different orders. It is causally consistent because there are no causal relationships that need to be obeyed.