1-

1. **Client-server**: There is clear adminstrator, single point of failure, secure (relatively), easy to locate files, server might be a bottleneck. Client is active with server being a slave.  
   **Peer-to-peer**: Hard to administrate, files are everywhere and need to locate them, challenging to secure, clients can be servers and vice versa, truly distributed, low cost.
2. In distributed computing, network is unreliable and not secure, we never know if our messages are being delivered or if they’re genuine messages between our systems. Same problem with Byzantine generals problem where they are never sure if their message actually reached the other general, and that message reached them without being tampered with.
3. A mechanism initiated by a coordinator, asking every participating process if it can commit, and if it can it sends a yes, if all process reach an agreement the coordinator sends a global commit to all processes, otherwise it sends a global abort. In case of coordinator failure other processes go into the process of electing new one.
4. Having to depend on external vendor which will incur extra costs, and might not have exact setup one might need. (E.G. our university server doesn’t let anything past php 3 run on it, gl asking them to install nodejs or anything of year +2003)
5. Strict consistency requires 0 latency in copying memory values from one process to another. This is impossible as distributed system introduce latency over network. Plus the many other issues related with how unreliable the network is, sometime we get faster result othertime slower, causing read inconsistency and what not, also it kills concurrency. Also requires a an absolute global time (what is most recent?)
6. A name server takes a name and returns one or more attributes of the named object.  
   A directory server takes attribute value, and returns sets of attributes of objects with those attribute values.
7. Pointers refer to a memory location that is local to the caller only and it is completely meaningless to the recipient. Instead we use Stubs to copy to and from what’s in the memory location pointed at by the pointer.
8. They provide a backward recovery; in case of a failure or entering a deadlock the system detects that and restore the last saved checkpoint to ensure the system remains intact and not left in corrupted state.
9. Physical redundancy: having more than one server storing same information.  
   Time redundancy: Doing the same action again and again.   
   Information redundancy: Sending extra bits with information to allow recovery.
10. Send unlimited requests utilising different threads and computers to the server querying for unneeded information, keeping the server busy processing my useless requests and denying the resources to other people who wants to get marked.

2-

1. Amdahl’s law is used to find maximum expected improvement to an overall system when a portion of the system is improved, used in parallel computing to predict the speedup from using multiple processors.  
   Not all applications are parallelizable since paralleling introduce some sort of latency. If the proportion of an application that can be parallelised is x (0 < x < 1), then the maximum speed boost is: parallelisation overhead + 1/1-x.
2. Atomic: All or nothing; either all operation execute correctly or none at all. To ensure we don’t leave the system in a corrupted state.  
   Consistence: An application doesn’t violate the Database integrity constraints, consistent with predefined rules.  
   Isolated: Concurrent processes won't interfere with one another.   
   Durable: Any updates are persistence after committing
   1. Availability = 1 - all (p(failure))  
      P(failure) = p(repair time / repair time + mean time between failures)   
      Availability = 1 - ((0.5/2.5) \* (0.5/4) \* (0.5/12.5))  
      Availability = 0.999   
      The service is highly available.
   2. Availability = 1 - ((0.5/2.5) \* (0.5/4))  
      Availability = 0.975   
      Still highly available.
   3. I would work to choose whichever computer that maximise the mean time between failure and lower the fix time.
3. 1. It is not feasible under sequential consistency model. By following the sequential consistency model it provides constraints on the read.   
      To get a y = 7, the process execution will be like this:  
      P2 P3 P3 P1 P3. But this will make it impossible for z to get the value 14.   
      And vice versa, to get z = 14, we need to do:  
      P1 P1 P4 P4 P2 P4
   2. It is possible. By doing the following:  
      P2 - P2(x) = 2  
      P3 - pass while  
      P3 - y = P2(x) // y = 2   
      P1 - P1(x) = 1  
      P3 - y = 5 \* P1(x) + y // y = 5 \* 1 + 2  
      P1 - P1(x) = 4  
      P4 - pass while  
      P4 - z = P1(x) // z = 4  
      P4 - z = 5 \* P2(X) + z // 5 \* 2 + 4   
      P2 - P2(x) = 3 // finishing executing

3-

1. To invoke a RPC a client need to pass to the server a relevant arguments.  
   The client wrap the arguments inside a Stub, send the Stub to the server, the server unwrap the Stub, invoke the relevant procedure with the arguments passed by, get results, put results in the Stub, send Stub back to client, client unpack the stub and return to application call.   
   The stub work as an abstraction and provide common ground that both server and client can agree on, this way the client can run different operating system/versions than the server.
2. One mechanism of controlling concurrent transactions is by using locks, those locks prevent other process from accessing until current process finish and release the lock.   
   Which might cause what is known to be a deadlock, when two processes are waiting for each other to release their locks. In other words there is a cycle in the wait-for graph of transactions for locks.   
   Distributed deadlock is when the locks are held in different physical places, and the cycle isn’t apparent for a single server.   
     
   One solution is to have a coordinator whose job is to forward to each server the wait-for graph.  
   Or we can use edge chasing, by sending a probe to the server the holds lock, which in turn forward the message if it is also waiting for a lock, if the transaction receive the message again and it is in it, that means a deadlock has happened.
3. We run a greedy heuristic to find a site with minimum total cost, we place the first server there.  
   Then we repeat same step however taking into account the previous servers already setuped. (by updating the distance matrix with min(value, value to closest server)
4. 1. When 1 enters it initiate an election  
      Send one message to each of the other computers.   
      Which in turn replies saying they are bigger, and each start their own election.  
      So   
      5 (1’s election) + 5 (responses back to 1) + 4 (2’s election) + 4 (responses back to 2) + 3 (3’s election) + 3 + 2 (4’s election) + 2 + 1 + 1 (5’s election) + 0 (6’s election)   
      Each send a response back as well.   
      When node sends a response message. But then 6 relies it is the boss, so it has to tell everyone else that it is the boss, so it also send 5 messages  
      So total messages: (5 + 4 + 3 + 2 + 1) + (5 + 4 + 3 + 2 + 1) + 5 = 35
   2. Now we have 5 entering the ring rather than 1.   
      5 initialise the election, but only 6 will also call for election.   
      Totaling: 1 (5’s election) + 1(6’s reply) + 0 (6’s election) + 5 (6’s telling everyone it is the boss)  
      Total messages: 7

4-

1. - Msg id: 96 will be processed.   
   Jerry will get: ‘fail: too many reservations’   
   Effect on database: no change to the database   
   Outqueue: enqueue the message that Jerry will receive  
     
   - Msg id: 90 will be processed after  
   Tom will get: ‘slot reserved’  
   Effect on database: Database entry 542 will be updated to have name Tom  
   Update reservations of Tom to 2   
   Outqueue: enqueue the message that Tom will receive  
     
   - Msg id: 86   
   Tom will get: ‘fail: too many reservations’   
   Effect on database: no change to the database   
   Outqueue: enqueue the message that Tom will receive
2. Msg 96 won’t change.  
   Msg 90 won’t change.  
   Msg 86 also won’t change.
3. It would be possible for Tom, but not for Jerry.  
   Tom:  
   Msg 86 in one thread, Msg 90 in another, both accessing the database, both have reservation of 1, finish the check, reserve 540 and 542, then commit both messages.  
     
   It is not possible for Jerry, since Jerry already holds 2 messages and the max\_reservation check will prevent any thread to make any extra reservations.
4. 1. Will receive outdated or corrupted availability. If another process is doing a booking the same time we might even get their booking
   2. Might get too many booking backs due to bookings and and cancellation not respecting the database integrity.
   3. Cancelling a slot might not fully commit and fail somewhere, allowing user to book as many as they want really (depend on how the check for max booking is done, if done on code level this is possible)
5. If request is cancel:  
    Dequeue a request from the in-queue  
    If requested slot is ‘free’:  
    Send ‘fail: slot is free and not occupied by you’ response message;  
    Else if request.User == requested slot.reservedBy  
    Remove reservation from that slot (marking it as free);  
    Reduce the number of reservation held by user by 1  
    Send ‘slot freed’ response message;  
    Endif  
   Commit; // transaction commits
6. it is avg queue size = arrival rate \* time to serve.