CSCE 438 HW #5 (50 points)

1. *[50 points (0.65pt each)]* Choose the statements below that are **TRUE**:

   1.1. Strict, Sequential and Causal Consistencies are Data-Centric consistency models.

   1.2. Eventual Consistency, Monotonic Reads and Monotonic Writes are Client-Centric consistency models.

   1.3. Strict Consistency assumes absolute global time.

   1.4. In Strict Consistency a "read is expected to return the value resulting from the most recent write"

   1.5. Strict Consistency can be achieved if we have GPS.

   1.6. In Sequential Consistency "the result of any execution is the same as if the (read and write) operations by all processes on the data were executed in an arbitrary order."

   1.7. In Causal Consistency, if an update U1 causes another update U2 to occur, then, U1 should be executed before U2 at each copy.

   1.8. Causal Consistency is stronger than sequential consistency.

   1.9. In Eventual Consistency, if no updates take place for a long time, all replicas will gradually become consistent.

   1.10. In a Monotonic Read, "if a process reads the value of a data item x, then any successive read operation on x by that process will always return that same value or a more recent value."

   1.11. In a Write Monotonic consistent store, a write operation by a process on a data item x is completed before any successive write operation on x by the same process.

   1.12. One implementation of Sequential Consistency is to use a centralized process, called sequencer.

   1.13. Write-through will impact caching in a distributed file system.

   1.14. The read-ahead in distributed file systems, requests chunks of data when they are needed.

   1.15. The first version of NFS ran over UDP, using Sun RPC.

   1.16. NFS uses caching at the client (caching data, file attributes and pathname bindings).

   1.17. All NFS writes are write-through to disk.

   1.18. Inconsistencies between local caches and server in NFS are solved by comparing time-stamps.

   1.19. NFS always invalidated data after some time (3 seconds for data in open files).

1.20. NFS version 2 extends the client-side buffer caching to disk, in 64KB chunks.

1.21. NFS version 3 continued to use UDP, because its simplicity.

1.22. NFS version 3 started to support 64bit file sizes.

1.23. GFS was designed with Google's application workload specifically in mind.

1.24. In GFS, during a write operation, the master for a chunk sends data to replicas in a daisy chain.

1.25. In GFS, if the master reboots and then finds a chunck server has a newer version number for a chunk, it adopts that version number.

1.26. In GFS, if a chunk server dies, then the master decrements the count of replicas for all chunks on that chunk server.

1.27. GFS was designed for small streaming reads.

1.28. GFS was designed for large sequential writes that append.

1.29. In GFS, the performance of operations is very good for all apps.

1.30. GFS architecture contains one master server.

1.31. In the GFS architecture there is one chunk server for each chunk.

1.32. The master server in GFS holds metadata and most frequently accessed data files.

1.33. The master server in GFS holds all metadata in RAM.

1.34. A read/write operation with a chunk server in GFS specifies the chunk handle and the byte range.

1.35. In GFS, the client issues control/metadata requests to the chunk servers.

1.36. A client in GFS uses no caching.

1.37. The GFS consistency model defines "consistent" as: file region all clients see as same, regardless of replicas they read from."

1.38. In GFS, the serial success result of a write operation is DEFINED.

1.39. In GFS, a successful record append data mutation is DEFINED.

1.40. In GFS, a successful concurrent write is CONSISTENT but DEFINED.

1.41. In GFS, a failure of a write or record append operation is INCONSISTENT.

1.42. BigTable is a sparse, centralized persistent multi-dimensional sorted map.

1.43. The map in BigTable is indexed by row key, column key and timestamp.

1.44. A row range (partition) in BigTable is also called tablet.

1.45. The items in a BigTable cell are stored in decreasing timestamp order.

1.46. BigTable is an alternate way for storing data than GFS and it implements its own replication.

1.47. BigTable processes share the same machines with MapReduce and GFS machines.

1.48. In the BigTable, there is a master server and many tablet servers.

1.49. In the BigTable, a client communicates directly with tablet servers for reads/writes.

1.50. The master server in BigTable maintains the set of live tablet servers and the current assignment of tablets to tablet servers.

1.51. The memtable is an in RAM storage for storing the committed writes that arrived at a tablet server.

1.52. When memtable size increases and reaches a threshold, it is frozen and committed to an SSTable in GFS.

1.53. The execution of a MapReduce program creates a Master process and several Worker processes.

1.54. The master process of a MapReduce program assigns map and reduce tasks to worker processes.

1.55. The intermediate results of a MapReduce program are stored in GFS.

1.56. If the master process of a MapReduce program crashes a new master is elected through Chubby.

1.57. A Partition Function in a MapReduce program is used for ensuring that records with the same intermediate key end up at the same worker.

1.58. A MapReduce program will use information from GFS (location of replicas) to decide which file blocks will be processed by which worker process.

1.59. When a worker process in MapReduce segfaults, the information about the record it processed is lost.

1.60. If the master process in MapReduce sees two failures for the same record, then it tells the next worker to skip the record.

1.61. In HDFS, a DataNode is the same as a ChunkServer in GFS.

1.62. In HDFS, a NameNode is the same as the Master Node in GFS.

1.63. A block in GFS is the same as a chunk in HDFS.

1.64. If a system can provide strong consistency, the programming model (how the programmer uses the API) is greatly simplified.

1.65. A Mobile Cloud consists of Distributed Storage and Distributed Data Processing

1.66. A k-out-of-n system is an n-component system that works if and only if k or less components work

1.67. Radio transmission is one of the major source of energy consumption on mobile devices.

1.68. In “Resource allocation For edge computing” lecture data is allocated in a way that the overall data retrieval energy is minimized.

1.69. In “Resource allocation For edge computing” lecture, the failure probability estimation includes failures due to disconnection from network.

1.70. In “Resource allocation For edge computing” lecture, Importance Sampling was used for approximating the expected distance between two nodes.

1.71. The Read-ahead (prefetch) policy in a distributed file system, minimizes the wait when it actually is needed.

1.72. The Write-on-close policy in a distributed file system is synonym with session semantics.

1.73. The VFS layer in NFS stands for Vectored File System.

1.74. In MapReduce, the arguments for a mapping function is a key and a value.

1.75. In MapReduce, the arguments for a mapping function is a key and a value.

1.76. The Partition Function in MapReduce ensures that records with the same intermediate key end up at the same worked.