DS2201 – Distributed Systems
Exam Questions

The following example questions were officially handed out during the autumn term 2011. Most of them were used in previous exams. Despite the hard work and controversial discussions amongst the students, the answers are not 100% error-free. Please report mistakes to: thomas_galliker@bluewin.ch

Section A

A:1 1p. What is meant by access transparency?
A remote resources are accessed using location independent names
B local and remote resources are accessed using the same operations
C a replicated resource is accessed exactly as if it was a single object
D a resource will handle all requests equally independent of location of client

A:2 1p. What is meant by location transparency?
A remote resources are accessed using location independent names
B local and remote resources are access using the same operation
C a replicated resources is accessed exactly as if it was a single object
D a resource will handle all request equal independent of location of client

A:3 1p. What is meant by concurrency transparency?
A threads are allowed to access shared data structures
B processes can access resources without interfering with each other
C a replicated resources is accessed exactly as if it was a single object
D new nodes can be added to a system without changing the application

A:4 1p. What is meant by failure transparency?
A failures are concealed for the users of a resource
B resource errors will raise exception that can be handled by the user
C resources can fail but only by crashing
D a robust system where resources will not fail

A:5 1p. What is meant by replication transparency?
A processes have knowledge of replication scheme and can take advantage of it
B users of a resource access it as if it was not replicated
C data is immutable and can be serialized on disk
D replies to queries are copied and logged to avoid dirty reads

A:6 1p. What is significant for a client server architecture?
A the client is the active part
B servers have more execution power
C several clients but only one server
D the server is the active part

A:7 1p. What would we call a system where one node is always reacting on requests and other nodes only communicate with this node?
A an asynchronous system
B a client server system
C a peer-to-peer system
D a synchronous system

A:8 1p. What is significant for a peer-to-peer architecture?
A no single node may initiate an operation
B nodes are structured in a hierarchy of client and servers
C all nodes can communicate directly with all other nodes in the network[1]
D all nodes are active and can be the initiator of operations

A:9 1p. What do we know in a synchronous system?
A that all operations will take equal amount of time
B exactly how long time it takes to deliver a message
C that all messages will be delivered
D the upper bound of the time to perform an operation

A:10 1p. What is significant for an asynchronous system?
A that all operations will take equal amount of time
B that our internal clock has a bounded drift
C that the time to perform an operation does not have an upper bound
D that messages might not be delivered

A:11 1p. What do we call a system where the maximum times for operations and message delivery are known?
A a fault tolerant system
B a real time system
C an asynchronous system
D a synchronous system

A:12 1p. What is provided by UDP?
A acknowledged delivery of messages
B a one-way byte stream between two processes
C a best effort delivery of messages to a process
D ordered delivery of messages

A:13 1p. What is provided by TCP?
A a guarantee that messages will always reach its destination
B a full-duplex stream between two processes
C a best effort delivery of messages
D transactional control and persistence

A:14 1p. Which address(es) can be found in the TCP header?
A there is no address field in the TCP header
B the network IP address
C a process identifier
D the port number(s)

A:15 1p. What is significant for synchronous communication?
A the send operation blocks and waits for the receive operation
B a sent message will always be received
C send operations can only be performed with certain intervals
D sender and receiver must have synchronized clocks

A:16 1p. What is significant for asynchronous communication?
A the receiver blocks and waits until a message arrives
B a sent message will always be received
C implemented over UDP
D a sender can continue without waiting for the receive operation

A:17 1p. Can a synchronous communication interface be used in an asynchronous way?
A yes, by performing the send operation in a separate thread
B no, the send operation will block and stop the whole execution
C no, synchronizing clocks will always take time

[1] Is not valid for all kinds of peer-to-peer networks (e.g. ring-based p2p networks).

D yes, by minimizing the latency for the receive operation

A:18 1p. What is the difference between the two nodes that are communicating over a stream socket?
A the node that connected to the server-socket is the node that should close the socket
B the node that created the server-socket should close the socket
C nothing, they have equal rights and responsibilities
D the node that connected to the server-socket must send the first message

A:19 1p. The Erlang call gen_tcp:recv(Socket, 0) will return:
A the complete message as sent to the Socket
B the first part (possibly the whole) message sent to Socket [2]
C list of 64K characters
D all messages sent to Socket before closed by the sender

A:20 1p. What is the purpose of the marshaling procedure?
A consistency checking of type information
B to compress data structures
C to protect an application from unauthorized requests
D to encode application layer structures in an external form

A:21 1p. How are arguments passed in Java RMI?
A call by reference only
B call by copy only
C remote objects as reference, all other as copies
D serialized objects as reference, all other as copy

A:22 1p. Which invocation semantic is provided by Java RMI?
A no guarantees
B at least once
C exactly once
D at most once

A:23 1p. What level of transparency is provided by method invocation in Java RMI?
A access transparency only since we need to explicitly describe the location
B location transparency only since we need to capture remote exceptions
C access and location transparency
D neither access nor location transparency

A:24 1p. How do Erlang processes communicate?
A only through global storage
B synchronous message passing
C asynchronous message passing
D modifying shared data structures

A:25 1p. Does Erlang provide a form of location transparency?
A No - you always need to know the node address of a process.
B Yes - there are no explicit node addresses in the system.
C Yes - a process can use a process identifier without having to know the address of the node where the process lives.
D No - since the address of a registered process contains the IP address of the node, there is no transparency in the system.

A:26 1p. How is the destination defined in an Erlang send operation?
A as a process identifier
B as a process identifier or a local or remote registered name
C a registered name
D an identifier and the binder to contact

[2] As by the Erlang documentation (http://www.erlang.org/doc/man/gen_tcp.html#recv-2) , "all available bytes are returned" if `Length=0`. But the "all" depends on the configured value for the buffer size. By default this value is configured to 8192 bytes (check that with function `inet:getopts(Socket, [recbuf])`. and set new values with `inet:setopts(Socket, [{recbuf,16384}])`.).

A:34 1p. What is a soft link in a file system?
A a mapping of a name to a file
B a link from a file to a path position
C a path that is resolved to another path [3]
D a link between two files

A:35 1p. What is a hard link in a file system?
A a mapping of a name to a file identifier [4]
B a link from a file descriptor to a path position
C a path that is resolved to another path
D a link between two files

A:36 1p. What is the purpose of the Unix lseek operation?
A find the position of string in file
B set the read/write pointer of an opened file
C find file descriptor matching name
D search for name mapped to given file descriptor

A:37 1p. How is a NFS client-side cache entry validated?
A if the call-back promise is not older than t seconds
B if the difference between the server modification time and the client modification time is less that t seconds
C if the server modification time is equal to the client modification time
D if the validity was checked less than t seconds ago or if the server modification time is equal to the client modification time

A:38 1p. How is authentication control handled in Sun NFS?
A server keeps a log and only commit changes when client does final authentication
B authentication is provided by RPC in each operation [5]
C authentication is done when file is opened
D authentication left to client, the server will trust all client operations

A:39 1p. How does a NFS server know at what position to read and write an opened file?
A it keeps a file table entry with a read/write position
B NFS only allow read operations and therefore does not need position information
C not needed since all read and write operations are on a whole file
D each read and write operation holds the position [6]

A:40 1p. How is AFS client side caching implemented?
A the server promise to notify the client if a file is modified by another client
B the client will check the server status with each write operation
C the client will check the last modified time of the server before each read
D the consistency is checked only when a file is opened

A:41 1p. Can two client have an inconsistent view of a file using AFS?
A no, updates are not performed unless all call-backs have been confirmed
B yes, since consistency is only checked when the file is opened
C no, clients will check server status with each read and write operation
D yes, if a call-back message is lost, a cached copy can be used although the original has been modified [7]

More information about NFS, AFS
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.739&rep=rep1&type=pdf

[3] See http://en.wikipedia.org/wiki/Soft_link
[4] See http://en.wikipedia.org/wiki/Hard_link
[5] NFS server is stateless server, so the user's identity and access rights must be checked by the server on each request.
[6] http://www.alacritech.com/nfs/nfs-tutorial-part-2-nfs-reads-writes-caching-matters
[7] When callback message is lost due to communication link failure, an old version of a file may be opened after it has been updated by another client. Limited by time *T* after which a client validates callback promises (T is typically a few minutes).
See http://tele.informatik.uni-freiburg.de/lehre/ws01/dsys/Lectures/Lecture19-1.pdf

A:42 1p. What is the difference between a URL and a URN? [8]
A URN refer to a location dependent resource
B URL resolves into any replica of a resource matching the URI
C URN resolves into any replica of a resource matching the URI
D URLs are a subset of URNs

A:43 1p. How are inconsistencies of the resolver cache handled in the DNS architecture?
A each entry has a time-to-live [9]
B changes in DNS servers will be pushed to resolvers with cached values
C the resolver will check if an entry has changed since requested
D a server will redirect requests to the new location

A:44 1p. What is the advantage of using recursive navigation for DNS queries?
A servers can hide internal DNS hierarchy [10]
B less burden placed on servers, most work done by client
C improves resolver caching
D server does not need to hold a request state

A:45 1p. How are inconsistent cached entries removed from a DNS resolver?
A a resolver will not cache entries that could become inconsistent
B authorized DNS server will actively revoke previous replies
C updated entries are pushed from servers to resolvers
D all entries have an expiration time set when cached

A:46 1p. How can we make two computer clocks perfectly synchronized?
A use atomic clocks
B use the Stanford network synchronization protocol
C send a message containing a time stamp every microsecond
D we can not

A:47 1p. What accuracy can be provided using Christian's algorithm?
A ±(Tround × 2 − minimum latency)
B ±(Tround)
C ±(Tround ÷ 2)
D ±(Tround ÷ 2 − minimum latency) [11]

A:48 1p. What is the purpose of the Berkeley algorithm?
A to synchronize a node with a stratum-1 source
B to perform internal synchronization
C to compensate for wide area network delays
D to divide a network into a hierarchy of synchronization nodes

A:49 1p. What does the reply from a NTP server contain?
A send and receive time of request and send time of reply [12]
B the delta of the client time-stamp and the server time
C the send time of the reply
D the latency of the request and the send time of the reply

A:50 1p. What is true if A happened before B?
A B is a consequence of A
B A and B occurred in the same process
C A must have occurred in real-time before B
D it is unknown if A occurred in real-time before B

---

[8] A Uniform Resource Name (URN) is a Uniform Resource Identifier (URI) that uses the urn scheme.
http://de.wikipedia.org/wiki/Uniform_Resource_Identifier#URIs.2C_URLs_und_URNs
[9] http://de.wikipedia.org/wiki/Domain_Name_System
[10] Not "improve resolver caching". It is true that the servers can improve by caching the replies but the resolvers will not improve.
[11] http://en.wikipedia.org/wiki/Cristian's_algorithm
[12] http://en.wikipedia.org/wiki/Network_Time_Protocol#Clock_synchronization_algorithm

A:51 1p. What is true if A happened real-time before B?
A B is a consequence of A
B A and B occurred in the same process
C A could have happened before B
D B could have happened before A

A:52 1p. What is true for events A and B?
A if A caused B then A happened before B [13]
B if A happened before B then A caused B
C if A occurred in real-time before B then A happened before B
D if A occurred in real-time before B then A caused B

A:53 1p. What can we know if we use Lamport clocks?
A if L(a) < L(b) then a happened before b
B if L(a) < L(b) then a could have caused b
C if L(a) < L(b) then a must have caused b
D if L(a) < L(b) then a occured in real-time before b

A:54 1p. What can we know if we use Lamport clocks?
A if, and only if, L(a) < L(b) then a happened before b
B if, but not only if, L(a) < L(b) then a happened before b
C if, but not only if, a happened before b then L(a) < L(b)
D if L(a) = L(b) then a = b

A:55 1p. What can we conclude looking at the Lamport clock timestamps of two events?
A if L(a) < L(b) then a happened after b
B if L(a) > L(b) then b happened before a
C if L(b) = L(a) then we cannot conclude anything about a and b
D if L(a) $\not<$ L(b) then a did not happened before b [14]

A:56 1p. What is the most that we know if we use vector clocks?
A V (a) < V (b) if and only if a happened before b
B if V (a) < V (b) then a happened before b
C if a happened before b then V (a) < V (b)
D if V (a) = V (b) then a and b are unordered

A:57 1p. What is the definition of a consistent cut?
A all events in the cut are strictly ordered in a happened-before order
B if e happened-before f and e is in the cut then f is in the cut
C if e and f are in the cut then e and f have a causal order
D if e is in the cut and f happened-before e then f is in the cut

A:58 1p. What is the definition of a stable global state predicate?
A if a system enters a state where the predicate holds true it will remain true in all future states
B the predicate holds true in all consistent global states
C the predicate will eventually hold true in all linearizations
D the predicate will never hold true in any consistent state reachable from the original state

A:59 1p. What is the definition of a unstable global state predicate?
A if a system enters a state where the predicate holds true it will remain true in all future states
B the predicate holds true in all consistent global states
C the predicate could hold true in a state but then be false in future states
D the predicate will never hold true in any consistent state reachable from the original state

[13] Happened-before is a logical relationship between two parallel processes. “Real-time before” does not automatically mean happened-before if there is no relationship (like A:send, B:recv) between processes. Vice versa we could say, that if A happened before B, then A must also have been happened “real-time” before B.
[14] More implications can be found on http://en.wikipedia.org/wiki/Lamport_timestamps

A:60 1p. Give an example of a stable global state predicate.
A none of the alternatives are stable
B deadlock
C the cargo door is open at 10.000 meters
D A is waiting on a message from B and B is waiting for a message from A

A:61 1p. What are the requirements for using the Bully algorithm?
A we must have reliable failure detectors [15]
B nodes are not allowed to crash ☺
C a total-order multicast must be available
D we need a persistent coordinator

A:62 1p. What is the benefit of a reliable multicast?
A messages are delivered within a upper bound time
B the sender will not crash during an operation
C messages are guaranteed to be delivered to all correct processes
D messages are delivered in total-order

A:63 1p. Can we implement a reliable multicast using only basic multicast?
A no, we need a reliable failure detector
B yes, but only if we have network supported multicast
C no, we need a synchronous communication network
D yes, by resending each received message to all other nodes

A:64 1p. What is the definition of total-order multicast? [16]
A messages are delivered in FIFO order
B messages are delivered in real time order
C messages are delivered in happen-before order
D messages are delivered in the same sequence

A:65 1p. What is a dirty-read during a transaction?
A reading a value that has been written by the same transaction
B reading an old value that will be over written
C reading a value that has not been committed
D reading ahead of a write operation

A:66 1p. What is two-phase locking in a transaction?
A not taking any locks once a lock has been released [17]
B taking a read lock that is strengthened to a write lock
C reserving all locks before taking the first
D taking locks in a strict order

A:67 1p. What does it mean that a transaction meets the atomicity property?
A intermediate results must not be visible to other transactions
B either all or no operations in the transaction are performed
C only one datum is allowed to be updated in each transaction
D the transaction can safely be duplicated resulting in the same server state

A:68 1p. What does it mean that a transaction meets the isolation property?
A intermediate results must not be visible to other transactions
B either all or no operations in the transaction are performed
C effects of the transaction are isolated from server failures
D the transaction can safely be duplicated resulting in the same server state

[15] http://en.wikipedia.org/wiki/Bully_algorithm
[16] See book „Distributed Systems – Concepts and Design“, page 669.
[17] http://en.wikipedia.org/wiki/Two-phase_locking

A:69 1p. What does it mean that a transaction meets the durability property?
A either all or no operations in the transaction are performed
B effects of the transaction are isolated from server failures
C the effects of the transaction will remain even if we have a server crash
D the transaction can safely be duplicated resulting in the same server state

A:70 1p. What is two-phase commit?
A a protocol that uses a global lock that is taken by each server that wants to commit
B a protocol where sub-transactions are allowed to commit independently of each other
C a protocol that ensures atomicity in a distributed transaction [18]
D a protocol where other transactions are allowed to see temporary value before final commit

A:71 1p. What is a phantom deadlock?
A one that affects more than two nodes
B one that is not cyclic
C one that is detected but not stable
D one that requires to abort more than one transaction

A:72 1p. What is a view, created by a group membership service
A the set of processes belonging to a group
B a total order of delivered messages in a group
C an address expansion of a multicast group
D an access point for a client front-end

A:73 1p. In a view-synchronous group membership protocol, a process that enters the group, and is included in the delivered view, will be guaranteed to be delivered:
A all messages in the history of the group
B all messages starting from the view when it enters
C all messages starting from the next view after it enters
D only the messages delivered before it joined

A:74 1p. Assume we have a server that has a p percent up-time and use n replicated servers to increase reliability, what will the up-time be for the system as a whole?
A $(p - 1)^n$ percent
B $1 - (1 - p)^n$ percent [19]
C $1 - p^n$ percent
D $p^n - 1$ percent

A:75 1p. Why is view-synchronous communication different from reliable multicast?
A messages are guaranteed not to be duplicated
B we will have a lower bound on message delay
C a message is delivered by all non-crashed processes
D a message is sent and delivered only by processes belonging to the same view

A:76 1p. Distributed hash tables are designed to limit the number of hops to reach any key to:
A exactly log(N)
B at most N2
C at most log(N)
D at most N/2

[18] http://www.jguru.com/faq/view.jsp?EID=20929
[19] http://eventhelix.com/RealtimeMantra/FaultHandling/system_reliability_availability.htm

A:2011:2 1p What is the most significant difference between a a synchronous and asynchrounous distributed system?
The answer that I was looking for was "possibility of perfect failure detectors" but this was only selected by less than half of the students.

A:2011:4 1p When is the TCP window size a limiting factor for high capacity communication?
The answer is "over long fat communication links". If the link is "fat" it means that we can quickly send a window of bytes, if the link also is "long" we must wait untill we can send the next window.

A:2011:11 1p Can two Unix processes simultaneous write to different positions in a single file?
This is of course the case and there is no magic involved. The two processes will of course have independent file table entries that wil have different positions in the file.

Section B

B:1 1p. What is a good reason for choosing UDP rather than TCP?
A you need to know that a message is handled by the remote application
B you have large messages or a sequence of messages
C you have a small message that should be sent with little delay
D UDP will guarantee the delivery of a message

B:2 1p. What is a good reason for choosing TCP rather than UDP?
A you need to know that a message is handled by the remote application
B you have large messages or a sequence of messages
C you have a small message that should be sent with little delay
D TCP will guarantee the delivery of a message

B:3 1p. What is the maximum TCP capacity in a 100 Mbps link with 250 ms round trip latency using a 64 Kbyte window size?
A 40 Mbps
B 2 Mbps [20]
C 10 Mbps
D 512 Kbps

B:4 1p. What is the maximum TCP capacity in a 100 Mbps link with 25ms round trip latency using a 64 Kbyte window size?
A 20 Mbps [21]
B 2 Mbps
C 80 Mbps
D 512 Kbps

B:5 1p. Why is a remote object passed as a reference and not as a copy in Java RMI?
A more efficient since it requires less bytes to be sent
B wrong, they are copied
C the object contains a mutable state that should not be duplicated
D objects are always passed as reference

B:6 1p. Can we have a circular construction in Erlang?
A no, data structures are always copied
B yes, simply create a circular data structure: X = {foo, X}
C no, Erlang is functional language without any circular structures
D yes, processes can refer to each other in a circular way [22]

B:7 1p. Can we implement a RPC system with exactly-once semantics in an asynchronous system with non-failing nodes but unreliable networks?
A no, since messages can be lost a reply is not guaranteed to reach the client
B yes, if the client keeps resending a uniquely tagged request until a reply is received and a server keeps track of all handled request in order not to duplicate a request
C yes, simply resend the request until an acknowledgment is received
D no, we can only achieve at-most-once or at-least-once but not both

B:8 1p. What is the difference when an error is reported for an at-least-once and at-most-once remote procedure call?
A in the at-most-once case, the remote call will not have been executed
B nothing, in either case we don't know if the remote call has been executed
C in the at-least-once case, the remote call will not have been executed at all
D in the at-least-once case, the call could have been executed more than once

[20] 250ms delay for each 64Kbyte package means, we can send 4x 64Kbyte within 1 second. 4x 64Kbyte/s = 256Kbyte/s = 2048Mbit/s
[21] Basically the same calculation as above, only that we have a throughput of 40x 64Kbyte/s.
[22] This is only valid for Erlang processes – not for data structures!

B:9 1p. When is it better to use the Berkeley algorithm rather than NTP for clock synchronization?
A never, NTP is always more accurate and more efficient
B when a node needs to be well synchronized with a stratum-1 server
C if we have a synchronous network
D when implementing internal synchronization [23]

B:10 1p. Assume that a NTP server cannot respond to a received request in less than 40 ms, how does this influence the accuracy of synchronizing clients?
A synchronization is limited by the larger of network delay and server delay
B not at all
C synchronization is limited by average network delay plus 40 ms
D synchronization is limited by average network delay plus 20 ms

B:11 1p. At time 117 you receive a NTP reply with the following information: request sent at 82, received at 111, reply sent at 120. How should you adjust your time?
A advance 16 steps [24]
B advance 21 steps
C advance 9 steps
D advance 32 steps

B:12 1p. What is the difference between a Lamport clock and a vector clock?
A nothing, two names for the same thing
B only difference is that Lamport clocks are more efficient since they require smaller messages
C only the vector clock gives a complete description of the ”happened before order”
D only the Lamport clock gives a complete description of the ”happened before order”

B:13 1p. When is it problematic to use vectors clocks?
A when we have a dynamic set of processes [25]
B if nodes are not synchronized using for example NTP
C if we have an asynchronous system
D when we do not have an elected leader process

B:14 1p. An alternative way of implementing a vector clock would be to keep a set of the highest counters seen from each process (including own), send it along with any message, update own counter and merging the own set and received set when a message is received. This would have the following advantage:
A new processes can easily be added
B the set is easier to represent
C nothing, it is identical to vector clocks
D events would become totally ordered

B:15 1p. If events in a given set are to be ordered in a total order that respect the happen-before order what is the advantage of using vector clocks?
A only vector clocks will give us the happen-before order
B only Lamport clocks will give us the happen-before order
C if two events are unordered only vector clocks can order them
D no advantage at all

B:16 1p. What do we know if we record a snapshot using the algorithm by Chandy and Lamport?
A the execution passed through the state described by the snapshot
B a non-stable predicate that is true for the snapshot has also been true during the execution
C if the algorithm terminates then the execution will also terminate
D there is a linearization from the original state to the final state that passes through the snapshot

---

[23] http://en.wikipedia.org/wiki/Berkeley_algorithm

[24] Client sends at 82, Server receives at 111 (means, 29 later). Server responds at 120, Client receives at 117 (means, -3 later). Thus, the clients clock lacks obviously behind. The average network delay is ((29+(-3))/2)=13. Now, if the clocks were the same on client and server, the receive time on the client would be (120+13)=133. We thus have a clock difference of (133-117=)16.

[25] Because of VC add/remove operation

B:17 1p. Assuming that we collect all state transitions of nodes and have them tagged with vector clocks what can we then do?
A for any unstable predicates determine if it possibly was true during the execution
B for any unstable predicates determine if it was true during the execution
C determine all stable, but no unstable, predicates
D determine termination but not deadlock

B:18 1p. How can we detect that a non-stable predicate definitely was true during an execution?
A let each node evaluate the predicate based on local state
B generate all consistent runs and show that the predicate is true at one point in all
C collect a snap-shot and examine if the predicate is true
D since the predicate is non-stable it cannot be determined

B:19 1p. What is the advantage of a ring-based election algorithm compared to the bully algorithm?
A nodes can easily change priority in-between elections
B better worst case turnaround time [26]
C better best case turnaround time
D more reliable if nodes fail

B:20 1p. What is the advantage of a bully algorithm compared to a ring-based algorithm?
A nodes can easily change priority in-between elections
B better worst case number of messages
C better turnaround time [27]
D two nodes will never be elected even if we have an unreliable failure detector

B:21 1p. What does Lamport clocks give us when implementing distributed mutual-exclusion?
A the system is prevented to dead-lock [28]
B requests will be granted in real time order
C request are granted in happen-before order
D at most one process may enter the critical section at a time (safety) [29]

B:22 1p. When does Ricart and Agrawalas mutual exclusion algorithm perform better than a central solution?
A when there is a risk of nodes crashing
B when the number of nodes is four or less
C under low congestion when hardly any conflict will occur
D under high congestion since only one message is needed to release and obtain the lock

B:23 1p. How are FIFO, causal and total order multicast related?
A a total order is also a FIFO order
B a FIFO order is also a total order
C a causal order is also a FIFO order [30]
D a causal order is also a total order

[26] Worst case election effort in ring-based algorithms: 3(N-1) which means a complexity of Big-O(N).
Worst case election effort in bully-based algorithms: N^2 which means a complexity of Big-O(N^2).
See http://cswilliams.ncat.edu/comp750/LeaderElection.pdf

[27] Best case election effort in ring-based algorithms: 2N messages.
Best case election effort in bully-based algorithms: N-2 messages. → Irrelevant difference for large N.

[28] Could eventually be the correct solution. Lamport time stamps help nodes to decide, which one's allowed to enter the critical section.
See http://www.cs.vu.nl/~ast/books/ds1/05.pdf

[29] The safety requirement should already be given by distributed mutual-exclusion.

[30] Book S.667: "Causal ordering implies FIFO ordering, since any two multicasts by the same process are related by happened-before".
Coordination -> Slide 36: Total ordered: delivered in the same order by all processes (I think it doesn't matter if the order is happened before or FIFO)

B:24 1p. How can we guarantee that processes, that can crash, communicating over an asynchronous network can reach a non-trivial consensus? [31]
A we cannot, we can only hope for the best [32]
B by implementing a two phase commit protocol
C by implementing a three phase commit protocol
D do a leader election and let the leader decide

B:25 1p. What is the problem of using optimistic concurrency control when implementing a transactional server?
A you could end up in a dead-lock situation
B hard to provide liveness
C large overhead if no conflict occurs
D if a conflict occurs, both transactions must be aborted

B:26 1p. What problem could we still have even if two transactions are serially equivalent?
A lost updates if both transactions read the same object
B dirty read if one transaction reads a value that is not yet committed
C inconsistent retrieval if both transactions write to the same object
D no problems since all conflicting operations are performed in the same order

B:27 1p. If the coordinator dies, what information is not enough for a set of servers to complete a two-phase-commit operation?
A a server has received a commit message
B a majority of servers have sent commit messages [33]
C a server has received a abort message
D a server has sent an abort message

B:28 1p. In an active replicated server, how do we know that the replicas are in a consistent state?
A they do not change state and are thus by definition consistent
B a coordinator uses two-phase commit for each request
C the primary replica sends update messages to all other
D they reliably receive all requests in a total order

B:29 1p. Why is it important that a primary server in a passive replicated system uses view-synchronous group communication?
A so that requests from front-ends will be serialized
B so that all or none of the backup servers have received an update before a new primary is elected
C to make sure backups receive updates in a total order
D it is not necessary, its sufficient to use a reliable multicast

B:30 1p. Will an active replicated server with multiple front ends, that uses a total-order multicaster, handle requests in causal order?
A yes, since requests are processed in total order
B not if clients can communicate with each other in between sending request and receiving reply
C never, since total-order multicast does not guarantee causal ordering
D no, an active replicated system can never guarantee causal order

B:31 1p. What is the main function of the leaf set in a DHT?
A ensure a ceiling in the number of hops to reach any key
B ensure that copies of certain keys are spread on diverse locations
C shortcut the look-up when is close to the destination key [34]
D ensure the content is replicated

---

[31] http://www.scs.stanford.edu/08wi-cs240/notes/chubby.txt

[32] It is not possible to get distributed consensus in an asynchronous system where nodes can crash.
See http://en.wikipedia.org/wiki/Consensus_(computer_science)#Impossibility

[33] If not yet all servers have sent an ok-commit to the coordinator before it dies, no server will have received the commit from the coordinator. Thus we cannot commit the whole transaction. (As soon as one server receives the do-commit command, all involved servers will ask one another whether they have received something from the coordinator).

[34] "Whenever a peer receives a packet to route or wants to send a packet it first examines its leaf set and routes directly to the correct node if one is found." (From http://en.wikipedia.org/wiki/Pastry_(DHT))

Section C

C:1 2p. How is NFS client side caching implemented? Assume no notion of real time and describe a situation that does not abide to the one-copy semantics of a regular Unix file systems? Describe why this situation would not occur in a regular Unix file system.
→ Compare local timer T has not yet been exceeded. If T is still valid, our local copy can be updated. If T has exceeded, the client must get a new copy from the server until he is allowed to update the file.
→ Clients may have out-of-date cache entries for brief periods of time when files are shared. This can lead to invalid writes at the server.
→ Regular Unix file systems use a daemon called lockd to preserve the one-copy semantics.

C:2 2p. Give four reasons why we are interested in examining a global state of a distributed system. For each of the reasons describe a problem that can be determined by looking at the global state but not by looking at each local node independently?

The four reasons are following:
→ Deadlock Detection: If we do not compare the global state with the local state of the nodes, we could determine a deadlock which isn't really one ("phantom deadlock").

→ Garbage Collection: In distributed garbage collection it could happen, that we remove an object whose reference is still on the way to the receiver.

→ Termination Detection: ???

→ Lost Token Detection: The liveness of ring-based networks mainly depends on the availability of the token. The token is passed further by each node in the ring. If we capture a global state while the token is on the way from one node to the other, we could mistakenly determine the loss of the token.

→ Debugging: Distributed debugging is a further case where the global state is required. But is the difference between the global state and the node's local state in this case relevant?

C:3 4p. Describe two different ways how we can implement total-order multicast using only b-multicast with no link-layer multicast support. Compare the two strategies.

→ Distributed ISIS:
- Multicast a message and request a sequence nr.
- When receiving a message, propose a sequence nr. and place the message in an ordered hold-back queue
- After collecting all proposals, select the highest sequence nr. and multicast the agreement
- When receiving an agreement tag msg "agreed" and reorder hold-back queue

Pros: No single point of failure. Nodes can crash.
Cons: Many messages needed for coordination (=overhead)

→ Using a sequencer:
- Message are sent to the sequencer as well as to the members
- Sequencer maintains a group specific sequence nr.

Pros: Sequencer-based solutions allow combinations of total order, FIFO order and causal order multicast.
Cons: Sequencer may become a bottleneck. Sequencer is single point of failure.

→ Retransmit messages:
- Every node in the group retransmits (multicasts) the message that it received.
- Duplicated messages are discarded with the help of message sequence numbers.

C:4 2p. In an active replication implementation of a fault tolerant server the group service will multicast messages in total-order. Why is this important? How could we relax the rules, is total-order always required. Discuss and show with examples.

Please report a solution to thomas_galliker@bluewin.ch

C:5 4p. A friend of yours has implemented a distributed mutual exclusion lock for a known, fixed number of processes. A process must acquire the lock before entering the critical section. The protocol is very simple; a process sends request and enters a state waiting for ok messages from all other processes. If it receives other requests while in the waiting state it will simply put them on hold until it has executed the critical section.

Your friend has some problems with deadlocks, something he detects and handles with timeouts. Describe to him what the problem is and a simple way to at least handle the deadlock problem while preserving the distributed architecture. Also warn him that although your simple solution will prevent deadlocks from occurring it might have other problems. Describe these problems and give your friend some hints on how to handle them.

The algorithm used in this example seems to work similar as the Ricart and Agrawala algorithm. To make it working deadlock-free, each process needs to keep track of a Lamport clock. Requests to enter the critical section are of the form <Timestamp, Processidentifier>. All requests are queue in a sorted list. As soon as one process leaves the critical section it replies to the next in the list.

Timeouts are a very simple but also dangerous method to resolve deadlocks. It is hard to calculate a fair timeout period. Overloaded systems would cause lot of transactions to time out.

C:6 4p. You have been assigned to implement a logging process in a distributed system. The process should receive messages from all processes the system and write these to a file. The system should run continuously and the log file is used to monitor the execution and determine the order of events.
Describe a solution and its limitations.

The problem here is of course to first of all capture the happened-before order. This could be done using Lamport clocks but the problem we then have is that we do not know when it is safe to write messages to the terminal; there might always be a message with a lower Lamport time stamp arriving late.
Vector clocks would solve this problem since we could determine if a message is depending on an event that has not been reported yet. One problem we would have when using Vector clocks is that we would have to settle for a fixed number
of processes.

C:7 4p. Assume we have a non-deterministic implementation of a server (uses a multi-threaded implementation that could deliver different replies depending on scheduling of threads) and we want to build a replicated server using either passive or active replication architecture. Would this be possible and what would the problems be with each strategy?

The main problem that must be addressed is in the active replication architecture. Active replication relies on the fact that servers are deterministic state machines and that they of course will have the same state if they receive the same requests. To use the existing implementation one would have to make it deterministic or at least make sure that the servers were left in a consistent state after each request even if the replies differed. The passive replication strategy does not rely on a deterministic server since the primary server will update the backup servers with the new state that it computed.

C:8 4p. We have implemented a distributed service where nodes communicate only through multicast messages. Since it is important to preserve total order a central sequencer is used that handles all multicast messages. Multicast messages are not very frequent but they need to be delivered quickly. To minimize delay they are sent one by one using UDP.
What must be taken care of, if we want the sequencer to implement a reliable multicast protocol that provides total and FIFO order? Assume that the nodes are correct i.e. they will not crash. How can your implementation be changed if we drop the requirement on FIFO order?

→ The question is: In which combination do you want to implement total order resp. FIFO order?
→ In total order multicast, a sequencer maintains a group specific sequence number (consecutive and increasing). Each process maintains a Lamport clock locally. Processes send messages to all nodes including the sequencer. All nodes store messages in a buffer queue. The sequencer stores messages in hold-back queues (one per process). The sequencer tags incoming messages with an unique sequence number (cross-process wise; unique id's).
→ What happens if messages are taking over one another on the way from one process to the sequencer? In this case, the sequencer must only tag messages in a consecutive order! If it tagged one process' messages with L=1,2,3 and then receives L=6, it must wait for the arrival of L=5 and 6 (of this specific process).
→ With the arrival of consecutive messages, the sequencer sends out order commands to all processes, indicating that they are allowed to deliver the messages to the application layer.
→ If we skip the requirement of FIFO ordering, the sequencer is not needed to keep track of process specific hold-back queues.

C:9 4p. A friend of yours wants some advice on how to implement a server based application to keep track of a "tennis ladder". The tennis ladder is a competition at the local tennis club where all players are ranked in a list ordered by whom they have beaten. Any one can schedule a game by challenging a player that is at most two positions higher on the list. A player that is being challenged can not deny the opponent a game. all games are played on Saturday afternoons.
The interface should be web based so all players can schedule a game from home. Some simple authentication scheme is needed to prevent someone from schedule games for others. Your friends first attempt was chaotic. He knew that all members were gentlemen and that they would not challenge someone who was already scheduled but, if one player challenged a second player and a third player, at more or less the same time, challenged the first player, confirmed games would be lost. The second attempt used a locking scheme but this was even worse; first it deadlocked (a consultant fixed the deadlock problem for 3000 euro) but even then, no games were played. It turned out that all players tried to fool the system by logging in early on Sunday morning and then do nothing, preventing anyone from challenging them (so much for gentlemen). So this is your chance. First explain why the first solution does not work. Then explain why the implementation with locks caused deadlocks and why 3000 euro is a bit too much to pay for a simple fix. Last, and most important, describe a better solution to the problem so that players actually can challenge each other and no one can prevent others from challenging them.

Please have a look at this page: http://www.kth.se/en/student/program-kurser/kurshemsidor/kurser-ict/ID2201/HT08-1/examination/part-c-1.50085

C:10 4p. A distributed hash table (DHT) uses a hashing of object names as the key for each object. Assuming we have names on objects that could be used as keys, for example phone numbers of subscribers, what would the problem be if the phone numbers were used instead of the hash of the numbers?
A related problem occurs when we are adding new servers to an existing distributed table, even if hashing is used. How can we limit this problem when adding new servers to the table.

The point is that hash values are better distributed than phone numbers. The performance of a distributed hash table is better, the more evenly distributed the keys are. Imagine a DHT ring consists of 8 nodes. If the key attribute is the phone number instead of a hash value, it was possible that all phone numbers are stored on few nodes. (In the worst case they're all stored on one single node!). Hash values help distributing key-value pairs across the whole ring.

The same problem consists if we want to add a new node to the DHT. How can we get a balanced ring? New servers are probably best placed where nodes serve the highest amount of key-value pairs.

C:11 4p. You're building a transactional database and have some options in how to implement the concurrency control. What would the pros and cons be to implement an optimistic concurrency control scheme? When would it be most useful?

→ Optimistic concurrency control can cause problems with starvation. (It is hard to provide liveness since some transactions may never get a chance to commit their changes).
→ Optimistic concurrency control is used if a high read-to-write ratio is expected.

→ Pros: No overhead due to lock maintenance. No risk of deadlocks. Better concurrency since long transactions do not avoid other transactions from running (as it is with lock-based transactions)
→ Cons: Backward validation needs to keep track of (possibly large) write sets that happened in predeceasing (+overlapping) transactions. Forward validation needs to keep track of (possibly large) read sets. This is of course in both cases an overhead. In case of conflicts between optimistic transactions, there can possibly be a high overhead due to rollback/restart of transactions.
Validation Deadlock is possible if to transactions validating parallel.

C:12 4p. Assume we have a service that consists of a very large media database containing several millions of key indexed images (think Google Earth). We have a limited set of producers that inserts new images to the database but this does not happen very often. We also have millions of concurrent clients that only read information from the database. The producers and clients do not communicate with each other in any other way. How would you use distribution to increase the performance of this service. Is it important that the producers and clients do not communicate outside of the database?
How would your implementation have to change if the producers also read from the data-base and their actions dependent on the information in the database? This could for example be only add an image if it contains less cloud than the one we have or add a day light image if the image to the east is day light,
to create a view of a rising sun?

Optimistic concurrency control with forward validation could be an applicable approach. Backward validation would compare millions of read operations in the current transaction with a few write operations of the last (+overlapping) transaction(s). If we use forward validation, we compare the (few) write operations of the current transaction with the millions of read transactions of the later transactions. This is by far more efficient!

C:13 4p. A large scale key value store is implemented as a distributed hash table with entries replicated on three nodes for high availability. A read operation must be confirmed by R nodes and a write operation by W nodes. Depending on the requirements of the system one can choose to R and W so that the desired properties are met.
What are the pros and cons of different R and W values? Outline a system that uses W equal to 1 and R equal to 3. What will the problems be? Can we detect and possibly resolve problems that occur?
What if R is 2?

Quorum-based replica control can be used to ensure that no two copies of a data item are read or written by two transactions concurrently.

Different weightings for R and W provide different performance and reliability characteristics. It's a trade-off between concurrency and risk.

1. $V_r + V_w > V$
2. $V_w > V/2$

The first rule ensures that a data item is not read and written by two transactions concurrently. The second rule ensures that two write operations from two transactions cannot occur concurrently on the same data item. The two rules ensure that one-copy serializability is maintained.

3-Node Example with Quorum W:3, R:1 → This system is configured for a very high read-to-write ratio. Read operations are much faster since each node only needs to get 1 read vote while write operations need to get the vote of all participating nodes.

3-Node Example with Quorum W:3, R:2 → This system is configured for a moderate read-to-write ratio. One of the three nodes has now 2 votes to get V=4 (rule 1). If this node fails, the resource remains read-only since none of the other nodes can get the required write quorum.

C:14 4p. Jules Verne wrote about a world deep inside the earth with trees, lakes and animals. He forgot to mention the inhabitants of this world called Lamponians. The Lamponians lived in isolated villages and rarely traveled. The problem with living in this world deep inside the earth was that there is no sun or stars to track and thus very hard to keep track of time. The interesting thing was how the Lamponians had organized their postal system. The mails was carried by pigeons between post offices (pigeons never got lost but they could take some time to deliver mail). Despite the problem of keeping time, the postal system could provide some sort of time. When you handed in a letter to be delivered at the local post office you would get a slip back, stating at what time the letter was sent. You could collect letters whenever you wanted but had to confirm that you had picked them up. Each letter was stamped with a code that that could be used to determine when it was sent. As an example of how this could be used we can look at the football matches.
Every now and then (more or less a week, but no one knows since it is hard to keep track of time), the Lamponians went to one of four football matches. There was limited number of seats on each match and the Lamponians had to send in an application to go to one of the matches. They all had good friends in other villages whom they wanted to see a game with, so they did send a lot of mails before deciding what game to go to. The Lamponians did want to watch a game with friends but easily became upset. If a Lamponian sent in an application and then told her friends that this was the best game, she would be very upset if she did not get a ticket while one of her friends, who applied only after being told that it was the best game, did get a ticket. In order not to upset anyone the Lamponian Football Association would wait until all Lamponians had sent in their applications and only then sort things out using the time codes on the letters. In some cases they didn't really know who sent their letters first and had to toss a coin to decide who should get the ticket. Everything worked, or at least no one complained. The only problem was that they had to wait for applications from all Lamponians before allocating tickets.
Explain how the postal system was constructed and how it could keep track of time. Suggest a solution so that the Lamponian Football Association could distribute tickets without waiting for all applications (you should not upset anyone). You might wonder how Lamponians could turn up for a game in time if they had no clocks and this is a mystery, don't worry about this.

Please have a look at this page: http://www.kth.se/en/student/program-kurser/kurshemsidor/kurser-ict/ID2201/HT08-1/examination/part-c-1.50085

C:15 4p. Assume that you have four uniquely named processes communicating in an asynchronous system. Assume also that they will behave correctly, that no messages are lost and that we have all the time in the world; how can you then implement a consensus protocol? Assume that messages can get lost, how does this change the situation? Assume that we have a deadline to meet and that all processes have to decide what to do before the deadline, how does this change the situation?

In the first scenario all processes simply send their estimate and they all follow the one with the first name in some agreed order. If messages can get lost nothing changes but we have to implement a protocol that acknowledge messages and resend them if lost. In the last scenario the problem cannot be solved since we cannot tell for sure that all other processes know about the decision.

C:16 4p. A historian wants to investigate the origin of a brilliant idea: who actually was first and how the idea evolved. As material to this investigation he has a set of articles, unfortunately without dates but of course with very good references (still no dates). Several papers describe the idea but they could of course be written independently from each other. How will he be able to trace the idea and determine the relationship? Assuming he is given two papers that describe the idea, how can he determine who was first without looking at the other papers? How does this relate to vector clocks?
Assuming that an author wrote articles one by one, and numbered them; could we show that two articles from two authors did happened in real-time order although we don't have a reference dependencies between the two articles? Assume we want to reward papers presenting original ideas. Design a very simple scheme that will allow us to reward a paper from a set of submitted papers so that we will never give a reward to a paper if there is another submitted paper with the same idea that the first paper depends on.

The articles can be placed in a partial order based on the references. If we are given two articles that do not reference each other then their relationship can not be established just by looking at the two papers. Only by looking at all papers can we determine if two articles are related. The system is not identical to vector clocks since it only keeps a vector of its immediate dependencies. If a reference list would include the reference lists (recursively) of all its references then it would be similar to a vector clock. Assume author A writes article a1 and a2, and that author B writes a paper with a reference to a2. Then we know that a1 was written in real-time before b1 but there is no reference relationship.