GROUP:		**Grade**	**C**	**I**	**N**
NAME:		SURNAMES:			
SIGNATURE:		ID# (DNI):			

This exam consists of 40 multiple choice questions. In every case only one of the answers is the correct one. You should indicate your answer by writing an "X" within the corresponding cell to the left. All questions have the same value. If correctly answered, they contribute 0,25 points to the final grade. If incorrectly answered, the contribution is negative, equivalent to 1/5th the correct value, which is -0,05 points. So, think carefully your answers.

If you have reasonable doubts, write an "*" (total doubt) or a number "3", at the end of the question's statement, and use the margins to explain further. The explanation must be brief.

The exam can be completed within 1 ½ hours, but you have up to 2 hours to finish up.

:

1. Distributed systems

	If they are scalable, they cannot be concurrent.
	Use only messages as a communication mechanism.
	Can be formed by only one process.
	Currently they only follow the cloud computing model.
	All the above.
	None of the above.

2. The roles of developer, service provider, system administrator and user

	Are clearly defined in a SaaS, but they are not always carried out by different agents.
	On personal computers, they are carried out by the PC owner.
	On *mainframes* the user also took occasionally the role of system admin.
	In enterprise data centers it is easy and cheap to manage the service provider and system admin roles.
	All the above.
	None of the above.

3. Examples of SaaS

	Linux
	Google Drive
	ZeroMQ.
	Microsoft Word
	All the above.
	None of the above.

4. About the levels of service in cloud computing:

	In an IaaS the provider offers networks and virtual machines to let the user deploy her distributed applications.
	In a SaaS the provider offers networks and virtual to let the user deploy there her distributed applications.
	In a PaaS the provider offers distributed applications, guaranteeing their scalability and manageability, so that users can directly access them.
	An IaaS needs an underlying PaaS provider to enable the IaaS to provide its services to the users.
	All the above.
	None of the above.

5. We have seen various levels of service in the cloud...:

	Three: IaaS, PaaS and SaaS.
	Five: Reliable, Available, Safe, Maintainable and Secure.
	Five: Sequential, Causal, Processor, FIFO, and Cache.
	Two: Active and Passive.
	All the above.
	None of the above.

6. Necessary mechanisms in cloud computing systems…:

	Virtualization.
	Sequential management (no concurrency).
	Scalability.
	Strong consistency.
	All the above.
	None of the above.

7. The deployment of a distributed application…:

	Poses some practical problems: component upgrading, maintainability, availability of the service...
	Has been automated in current IaaS systems.
	Is managed by the user within current SaaS.
	This is a solved problem since the *mainframe* days.
	All the above.
	None of the above.

8. A simple theoretical model for a distributed system usually assumes that…:

	Processes are sequential agents.
	There are no failures.
	Processes do not need to communicate among themselves.
	Relevant events are external in all cases, and caused by the flow of time.
	All the above.
	None of the above.

9. Concerning synchrony in a simple theoretical model of a distributed system…:

	Processes will be synchronous if all of them are able to carry out exactly one event on each algorithm step.
	Channels will be synchronous when the propagation and delivery times for each message can be bounded.
	Communication will be synchronous when the sender blocks waiting for an answer from the receiver.
	Clocks will be synchronous if all process clocks are synchronized with a global real time clock.
	All the above.
	None of the above.

10. When comparing multi-threaded servers with asynchronous servers in a scalable, distributed system...:

	Asynchronous servers block on each received request
	Asynchronous servers are usually event-oriented, and fit better with a simple model for a distributed system.
	Multi-threaded servers always offer better performance, as they can carry out multiple actions at the same time
	Multi-threaded servers offer a simpler state management approach, as there are no *guards*, nor *actions* associated to *events*.
	All the above.
	None of the above.

11. Multi-threaded servers...:

	Provide an asynchronous programming model.
	Do not require synchronization mechanisms to access shared resources.
	Minimize message usage among agents, increasing the scalability of the implemented services.
	May increase the scalability of the implemented service, if most operations use shared resources.
	All the above.
	None of the above.

12. Asynchronous servers...:

	Use an event-driven model.
	Can be implemented using nodejs and ZeroMQ.
	Their code is structured around *callbacks*, which are the actions associated with concrete events.
	Avoid the problems introduced by critical sections.
	All the above.
	None of the above.

13. A middleware...:

	Generally keeps components from being inter-operable
	When managing inter-communications, it will sit beneath the transport layer.
	It usually satisfies one or more standards, facilitating the development of distributed applications.
	Forces using the client/server interaction pattern among the components.
	All the above.
	None of the above.

14. Examples of middleware types and their provided functionality:

	URL. Provides location services in a distributed system.
	Javascript. Allows creation of distributed applications.
	RPC. Provides a client/server interaction mechanism with location transparency.
	SaaS. Manages virtualization of resources, providing tools to develop scalable web services.
	All the above.
	None of the above.

15. **The Remote Procedure Calls:**

	Behave exactly like local procedure calls.
	Use specific processor characteristics to pass arguments to the server.
	Show failure modes different from those shown by local procedure calls.
	Require a broadcast mechanisms form the transport layer.
	All the above.
	None of the above.

16. **Standards in distributed computing...:**

	Provide reasonable approaches to resolve concrete problems.
	Facilitate interoperability.
	Provide high level functionality.
	Allow developers to become familiar with the techniques they need to use.
	All the above.
	None of the above.

17. **A RPC Library...:**

	Is a middleware example.
	Is used by Java RMI.
	Uses the SOAP protocol in Web Services.
	Is used by the REST architectural style, taking the HTTP protocols as a base.
	All the above.
	None of the above.

18. **Distributed system objects...:**

	Are usually scalable, as one can create new objects within the application when required, without limitations.
	Offer an asynchronous programming model. Consequently they can scale easily.
	Facilitate highly cohesive designs, with easily reusable components with low coupling.
	Minimize contention and generation of critical sections. This also helps scalability.
	All the above.
	None of the above.

19. **Messaging middleware...:**

	Provides in some cases an asynchronous, and highly scalable interaction model.
	Let programmers develop their applications without needing to share resources. This increases scalability.
	Can be used to implement replicated servers.
	They include ZeroMQ, as an example which is *persistent* and *brokerless*.
	All the above.
	None of the above.

20. **Messaging middleware is considered persistent when...:**

	It does not block message senders. This way scalability of applications is improved.
	Uses a name service to find the server processes that must receive the messages. This way replication transparency is achieved.
	Messages are kept temporarily in buffers, managed by the channel. The receiver does not need to be active when the sender sends the message.
	It uses the network and transport layers to perform routing and management of the communication channel.
	All the above.
	None of the above.

21. Middleware examples:

	Java RMI.
	ZeroMQ.
	RabbitMQ
	JINI.
	All the above.
	None of the above.

22. Horizontal scalability…:

	Is about replacing a component by another, with a higher capacity.
	Implies that the system be scalable and adaptable. That scalability must be dynamic (reacting to load variations) and autonomous (without human intervention).
	Guarantees robustness of the system.
	Requires a monitoring subsystem (reporting load and performance) as well as a reconfiguration action automation subsystem.
	All the above.
	None of the above.

23. We can use … as a mechanism to increase the size scalability of a service:

	Reduce or eliminate synchronization needs among the agents implementing the service.
	Use decentralized algorithms.
	Forward as much computation as possible to the client agents.
	Replicate the server components, using the weakest consistency model fitting the service's needs.
	All the above.
	None of the above.

24. … can cause contention:

	To use an asynchronous programming model.
	To use decentralized algorithms.
	To use a passive replication approach.
	To use synchronization mechanisms.
	All the above.
	None of the above.

25. Elasticity...:

	Is a property for any kind of application (both distributed and non-distributed).
	Requires scalability and adaptability.
	It makes sense only on IaaS systems.
	Is obtained when using an active replication model.
	All the above.
	None of the above.

26. The three dimensions of scalability we have seen are...:

	Performance, Persistence and Availability.
	Consistency, Availability and network partition tolerance.
	Size, Distance, and Administration.
	Vertical, Horizontal, Oblique.
	All the above.
	None of the above.

27. Vertical scalability...:

	Increases service capacity by replacing its components.
	Improves availability of services.
	Increases service capacity adding new nodes, resources or components.
	Improves service security.
	All the above.
	None of the above.

28. A scalable system...:

	Needs to remove its contention points.
	Needs to be robust.
	Can increase its service capacity when needed.
	May not always show a strong consistency.
	All the above.
	None of the above.

29. *sharding*, that is, the partition of a distributed data base (e.g. as used in MongoDB)...:

	Is an implementation of the data distribution approach, improving size scalability.
	Is an implementation of the task distribution approach that improves horizontal scalability.
	Can increase the concurrency degree, without using synchronization primitives.
	It does not always require keeping multiple replicas for any of the elements of the database.
	All the above.
	None of the above.

30. Within the active replication model...:

	Service robustness is guaranteed.
	General omission failures can be supported.
	There is no risk of inconsistencies when executin non-deterministic operations.
	There is no need for any kind of synchronization among the replicas to obtain sequential consistency.
	All the above.
	None of the above.

31. Within the passive replication model...:

	All replicas in a service play the same role.
	Arbitrary (byzantine) failures can be supported.
	Service security and safety are guaranteed.
	Less computational power is consumed compared to the active replication model, when operations take a long time to process, but modify a small amount of state.
	All the above.
	None of the above.

32. Select the alternative offering the longest consistency models compatible with the following execution. The selection should only contain models compatible with the execution:

P1:W(x)1, P2:W(x)2, P3:R(x)1, P3:W(x)3, P2:W(x)4, P4:R(x)2, P4:R(x)3, P4:R(x)1, P5:R(x)2, P4:R(x)4, P5:R(x)3, P5:R(x)1, P5:R(x)4

	Sequential, Cache, Processor, Causal, FIFO.
	Cache.
	Causal, FIFO.
	Processor, FIFO.
	Processor, Cache, FIFO.
	FIFO.

33. **Select the alternative offering the longest consistency models compatible with the following execution. The selection should only contain models compatible with the execution:**
P1:W(x)1, P2:W(x)2, P3:R(x)1, P3:W(x)3, P2:W(x)4, P4:R(x)1, P4:R(x)3, P4:R(x)2, P5:R(x)2, P4:R(x)4, P5:R(x)1, P5:R(x)3, P5:R(x)4

	FIFO.
	Processor, FIFO, Cache.
	Causal, FIFO.
	Causal, Cache.
	Processor, Causal.
	Processor, Causal, Cache, FIFO.

34. **Select the alternative offering the longest consistency models compatible with the following execution. The selection should only contain models compatible with the execution:**
P1:W(x)1, P2:W(x)2, P3:W(x)3, P2:W(x)4, P4:R(x)3, P4:R(x)1, P4:R(x)2, P5:R(x)3, P4:R(x)4, P5:R(x)1, P5:R(x)2, P5:R(x)4

	Cache.
	FIFO.
	Processor, FIFO, Cache.
	Causal, FIFO.
	Sequential, Causal, Processor, FIFO, Cache.
	Causal, Cache.

35. **Relation among faults, errors, and failures:**

	There may be failures not caused by any fault.
	A failure will be produced if there are errors, and there is no redundancy in the component where the errors appeared.
	Replication solves all error situations, to keep them from becoming failures.
	With a proper diagnostic mechanisms, and a correct remedial action, it is possible to avoid failures from producing errors.
	All the above.
	None of the above.

36. **… is an example of a failure model:**

	Active, passive.
	Channel and link, network , transport.
	Stop, crash, general omission.
	Maintainability , primary partition, sequential consistency.
	All the above.
	None of the above.

37. **Properties of a dependable system:**

	Reliability, maintainability, safety, availability, security.
	Consistency, Atomicity, isolation, durability.
	Scalability, elasticity, adaptability, efficiency.
	Correctness, efficiency, ease of use, liveness.
	All the above.
	None of the above.

Consider the following nodejs module, `shared.js`.

```
// shared memory module
//
var zmq = require('zmq');
var id = "";
var pb = zmq.socket('pub');
var sb = zmq.socket('sub');
var local = {};

sb.subscribe("");
pb.bind('tcp://*:8888');

function join(nodes) {
    nodes.forEach(function (ep) {
      sb.connect(ep);
    });
}
function myID(nid) {
    id = nid;
}

sb.on('message', function (name, value) {
    local[name] = value;
});

// initialization
exports.init = function(nodes, id) {
    join(nodes);
    myID(id);
}
// Write function
exports.W = function (name, value) {
    pb.send([name, value]);
    local[name] = value;
    console.log("W" + id + "(" + name + ")" + value + ":");
};
// Read function
exports.R = function (name) {
    console.log("R" + id + "(" + name + ")" + local[name] + ":");
    return local[name];
};
```

This module implements operations *Read* (**R)** and *Write* (**W**) on a set of variables shared by a group of processes.

When a process wants to join a sharing group of processes, it includes in its program `require('shared.js')`, initializing it with the URLs of the endpoints of the other processes, and with and identifier for itself.

`shared.js` prints a log of the **R**/**W** operations performed, including the values involved. This log represents the sequence of R/W events of the process in the order they happen.

Assume we launch four processes, with ids 1,2,3,4, sharing variable “X”. Assume, further, they are launched on the same command console. The events from all processes, as they are printed to the console by `shared.js`, are interleaved in the console’s output. Further assume that there is no pub message loss.

Within this context, answer the following two questions.

38. R/W sequence … can be observed at the console:

	R1(X)2:W1(X)4:R2(X)4:W3(X)67:R4(X)undefined:
	R1(X)undefined:W1(X)4:R2(X)4:W2(X)67:R4(X)4:
	R2(X)2:W2(X)4:R1(X)4:R3(X)67:R4(X)undefined:
	R1(X)2:W2(X)2:R1(X)4:W3(X)4:R4(X)5:
	All the above.
	None of the above.

39. Module shared.js implements …: consistency

	Sequential
	Processor
	Causal
	FIFO
	All the above.
	None of the above.

40. The general goals of replication are…:

	Increase availability.
	Increase throughput.
	Decrease latency of requests.
	Implement horizontal scalability.
	All the above.
	None of the above.