**Department Computer Science** **Distributed Systems**
**VU University** **08.01.2008**

*1a* Some distributed systems can simply not scale, no matter which techniques are applied. Give an example of such a system and explain why scalability is (close to) impossible. *5pt*

*Think of a centralized database system that needs to handle many concurrent updates (transaction systems often have this). In order to improve scalability, one could consider replicating the database, but that would instantly lead to a serious synchronization problem when updates need to take place.*

*1b* Explain how code shipping as is done with Java applets in the case of Web services, can help improve scalability. Which scalability problem is actually being tackled? *5pt*

*With code shipping, a client will be allowed to prepare, for example, the input for a remote database. As such, we reduce the communication between client and server, and as such are overcoming latency problems that hinder scalability.*

*2a* Explain the difference between a process virtual machine and a virtual machine monitor. *5pt*

*A PVM is essentially an interpreter or emulator that takes the executable of an application and runs it on an operating system. An example of a PVM is the one used in Java. A VMM is a subsystem that mimics a specific instruction set, thereby hiding the underlying hardware architecture. Typically, a VMM is capable of running a complete operating system along with several of its applications.*

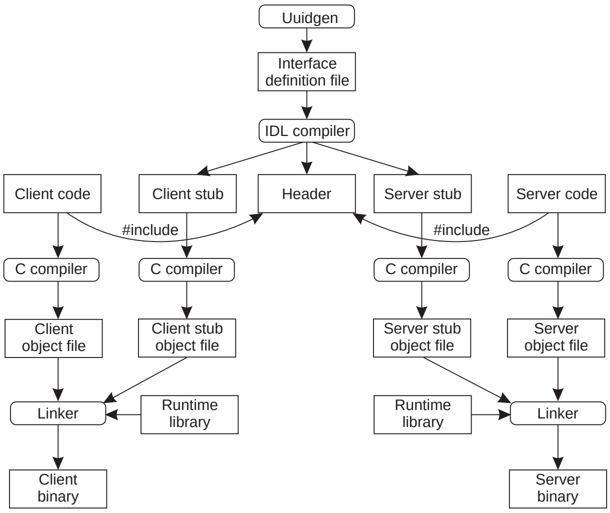
*2b* Explain how virtual machine technology is used in the PlanetLab system, and how it helps to support multiple applications. *5pt*

*PlanetLab makes use of so-called vservers, which is essentially a specific and isolated Linux environment. Applications can have their own private environment, implemented by means of a chroot equivalent. This technology helps in setting up parallel vservers across multiple PlanetLab nodes. An application in that case will be allocated a vserver on different machines, after which it can make use of the resources available to those respective machines. In this sense, vservers help to run distributed applications concurrently, while keeping them isolated from each other.*

*2c* Virtual machines help in migrating applications between different servers. Why is this easier than general solutions for strong mobility? *5pt*

*The real issue here is that strong mobility requires that a complete execution environment (consisting of the process stack and such) is migrated to a different machine. That can only be done easily if the target machine and operating system is exactly the same as the source. With virtual machines this problem goes away: the execution environment as a whole, including the parts on which it depends, are migrated, effectively reducing the migration to a special case of weak mobility.*

*3a* Explain which functions are implemented in the runtime library of an RPC system. See also the following figure. *5pt*



*The library typically consists of (configurable) send and receive primitives, such as parameterised primitives for socket communication. In addition, we should expect to see all kinds of routines for converting machine-dependent data structures into network- and machine neutral representations.*

*3b* Executing an RPC requires that a client can contact a server. How does it find the contact point for a server, and what does that contact point consist of? *5pt*

*Looking up a contact point is typically done by contacting a name server at a well-known address and passing it an identifier of the RPC server that the client wants to contact. The name server, such as a port mapper, returns a transport-level address, consisting of an IP address and port number.*

*3c* RPC systems cannot support local references (such as pointers), as these refer to objects only locally accessible. Instead, global object references should be used, if possible. Outline an implementation of such a reference. *5pt*

*One possible implementation is that it refers to an RPC call running on the machine wanting to provide a local reference, such as a get or put statement. In Java, this problem is solved by implementing the reference as a complete client-side stub, with the contact address hard-coded in the stub. This solution is possible because Java stubs can be migrated across the network and directly executed at the receiving side.*

*4a* Explain the principle of an epidemic protocol. *5pt*

*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.*

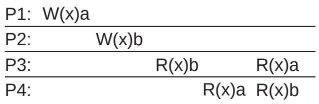
*4b* Sketch how the nodes in a distributed system can each compute the size of the system (i.e., the total number of nodes) using an epidemic protocol. *5pt*

*Let each process i maintain a variable $x_i$. Every time a process i contacts another process j, $x_i, x_j \leftarrow (x_i + x_j)/2$. In this case $x_i$ will converge to $\bar{x}$. If we initialize $x_1$ to 1 and all the others to 0, each process will compute $1/n$, where n is the number of processes. Your answer may also include the concurrent computation of max, so that we do not have to determine who is allowed to start with 1.*

*4c* What is the problem with removing a data item in an epidemic system, and how can this problem be solved? *5pt*

*The problem is that if there is only a single process left that still holds the item to be removed, it will almost instantly start to spread that item as something new: the others don't have it. The standard solution is to spread an update on the data item stating that is officially declared dead, which is the equivalent of stating that it should be considered obsolete, and thus as have being removed.*

*5a* Why is the following data store not sequentially consistent? Is it causally consistent? Be sure to explain your answer. *5pt*



*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.*

*5b* Consider a system that combines read-your-writes consistency with writes-follow-reads consistency. Is this system also sequentially consistent? Explain your answer. *5pt*

*No, it is not sequentially consistent. Although the combination effectively provides location-independent consistent behavior for a single process, it does not guarantee that when there are two concurrent write operations at different locations, that the effect of those writes will be seen everywhere in the same order.* As a side note: if we combine all client-centric consistencies, it turns out that you will have a sequentially consistent system.

*5c* Consistency can also be formulated in terms of numerical deviations. Give an example of such a form of consistency, and sketch how that consistency can be enforced. *5pt*

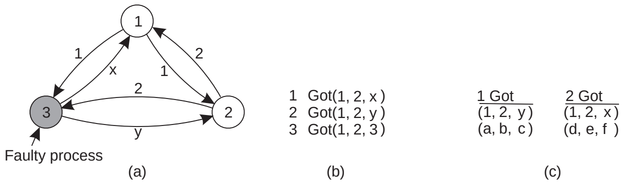
*Typically, one may want to specify for a stock exchange system that the values of copies of the same share that are replicated at different locations, deviate no more than 1% from each other. One way* *to enforce such a consistency is to gossip update information between nodes, such that each node maintains information on how far others are deviating from the value of the freshest value. You can easily obtain points if you make this more concrete by mentioning the vectors $TW[i,j]$ and $TW_k[i,j]$ that record what k knows about how far i is in processing updates that were initiated at j and showing that $TW[k,k] - TW_k[i,k]$ exceeds a limit, it's time to propagate updates again.*

*6a* What is a $k$ fault-tolerant group, and how does $k$ depend on failure semantics? *5pt*

*A k fault-tolerant group of processes is a group that continues to operate as expected even in the presence of k failing processes. In the case of crash/performance failures, you need $k+1$ processes; when dealing with Byzantine failures but processes do not communicate with each other, you need $2k+1$ members so that you can perform majority voting. With communication between processes, $3k+1$ members are needed.*

*6b* Show that having three processes of which one is faulty, is not enough to guarantee agreement between the two nonfaulty ones in a Byzantine setting. *5pt*

*The easiest thing to do is to prove this by contradiction: let the three processes pass their information in two rounds, and see whether you can get a majority. Essentially, this means drawing Figure 8-6:*



1 Got(1, 2, x )
2 Got(1, 2, y )
3 Got(1, 2, 3 )

| 1 Got | 2 Got |
|---|---|
| (1, 2, y ) | (1, 2, x ) |
| (a, b, c ) | (d, e, f ) |

*7c* Explain what a flash crowd is, and why it is so difficult to develop general-purpose predictors. *5pt*

*A flash crowd is a sudden burst in requests for a specific Web document. Although predictors can be made, they are very specific to Web request traffic, making it virtually impossible to come up with solutions that will automatically configure them for an unknown trace of requests.*

*7b* What is the best measure against flash crowds? Explain briefly how it works. *5pt*

*The best measure is simply to keep a number of replicas available that will return the responses to client requests. However, we still need to be able to process those requests. This means that the original site will accept incoming requests and subsequently redirect them to the resepctive replicas.*

**Grading:** *The final grade is calculated by accumulating the scores per question (maximum: 90 points), and adding 10 bonus points. The maximum total is therefore 100 points.*