Nested transactions

Q1.

1. Executing nested transactions requires some form of coordination. Explain what a coordinator should actually do?
2. In order to make the answer to this question more solid and clear let me start wit a brief explanation on what actually is a nested transaction.

Anested transaction is a new transaction begun within the scope of another transaction. Several transactions can begin from the scope of one transaction thus the transaction that starts the nested transaction is called theparentof the nested transaction.

The features of nested transactions as to why they exist are listed below.

- Nested transactions enable an application to isolate errors in certain operations.
- Nested transactions allow an application to treat several related operations as a single operation.
- Nested transactions can function concurrently.

Now coming on to answer the exact question; the function of a coordinator is that it should take the primary request automatically in the order in which it receives. It should check the unique identifier in case it has already received and executed the request and if it identifies, it should resend the response back

Servers which perform requests in distributed transaction needs to communicate with each other to coordinate their actions, therefore there are a few process that involves when the coordinator is in play and they are;

- In order to keep track of the participants and their information the coordinator keeps a list of references whenever they are involved as this is will be helpful at the time of aborting it.
- When the client sends a request it first reaches the coordinator which then resends a unique ID to the client which ensures that the coordinator is now responsible for the exchange of transactions.
- At some instance if there is a new participant who joins the transaction, the coordinator should be informed and it is then the coordinator updates its list of participants and this is where the joint method in the coordinator interface is used.

3. We argued that distribution transparency may not be in place for pervasive systems. This statement is not true for all types of transparencies. Explain what you understand by pervasive system. Give an example?
4. In general Pervasive systems which is also well known as Ubiquitous computing, can be easily derived by the term ubiquitous which means being everywhere at the same time, When applying this logic to technology, the term ubiquitous implies that technology is everywhere and we can use it irrespective of the location and time.

It is important to note that pervasive systems are built by a number of different distributed components integrated and tagged together that can be invisible and also visible at times which in general terms is known as transparency.

The following points will make it clear to why pervasive systems are important in the current context.

Pervasive systems are changing our day to day activities in a various ways. When it comes to using today’s digitalized equipments users tend to

- communicate in different ways
- be more active
- conceive and use geographical spaces differently
- In addition, pervasive systems are
- global and local practically everywhere
- social and personal
- public and private
- invisible and visible

From my understanding, reading and gathering its is true that Distribution transparency may not be in place for pervasive systems but arguably there are rare instances which it can be, because the backend of pervasive system is can be made invisible as the actual user need not know how the process takes place behind the scene.

Here is a typical example on how a pervasive system can involve in a humans day to day life.

Assume a lecturer is preparing himself for a lecture presentation. The lecture room is in a different campus which is a 15 minute walk from his campus. Its time to leave and he is not quiet ready. He takes his HTC palmtop with him which is a Wi-Fi enabled handheld equipment and walks out. The pervasive system transfers his undone work from his Laptop to his HTC Palmtop, so

that he can make his editing’s during his walk through voice commands. The system knows where the lecturer is heading towards by the campus location tracker. It downloads the presentation to the projection computer in which he is going to present and keeps it prepared for the lecture to begin. Now by the time the lecturer reaches his class he has done the final changes. As the presentation proceeds, he is about to display a slide with a diagram with numerical information regard to forecasts and budgets. The system immediately realises that there might be a mistake in this and warns the lecturer, he realizing this at the right time skips the slide and moves on to other topics to make the presentation smooth leaving the students impressed by his quality presentation.

Q2. Consider a chain of processes P1, P2… Pn implementing a multitiered client-server architecture. Process Pi is client of process P i+1, and P i will return a reply to Pi-1 only after receiving a replyfrom P i+1. What are the main problems with this organization when takinga look at the request-reply performance at process P1?

From my understanding a Multitiered client-server Architecture basically refers to where more components in terms of hardware and more importantly softwares are added and tied up to build or in other words construct a complete architecture which facilitates the process of presentation, application processing, and data management to be logically processed separately.

In relation to the question the limitations and the problems this organization would face is that if the processes are too large that is referring to Pn according to the example there will be bottle neck kind of situation arising

and this can make the whole process slow and there will be a chain of processes un processed.

A Multitier architecture does not run on its own there are other hardware and software components involved in it and if any of these components drop in performance the whole architecture will see a drop in performance.

Another problem is that it would more difficult to program and test than in normal architectures because more devices have to communicate in order to complete a client’s request.

Q3. Strong mobility in UNIX Systems could be supported by allowing a process to fork a child on a remote machine. Explain how this would work?

It is easy to get the initial understanding if the logic behind the term forking a child is made clear. Forking in UNIX refers to the process which the parents image is completely copied to the child. This start when UNIX starts a new process.

Basically, how it works is that: the main parent process which already exists forks a child process which is the new process created. Then as the next step the newly created child process gets a duplicate copy of the parent’s data., and now it has 2 processes with the same data and the child process can now be activated

To create a child process there are 2 basic steps to be followed.

- The System creates an exact copy of parent process by the process of forking

- The process in UNIX are built with different codes therefore the code of the parent process should be substituted within the code of the child process.
- We must also have the system reserved with ample resources to create the child process and memory map for it.

As a result of this it can also be said that the child process inherits all the system variables of the parent process.

The only issue in this would that using the forking process consumes more time and memory to duplicate the parent’s environment, and to create a unique structure for the child.

Q4. Describe how Connectionless Communications between a client and a server proceeds when using sockets?

Let me step into answering the question straightaway where the following paragraph will explain how the connectionless communication is taking place between the client and a server using the help of programmed sockets.

It is clear that the connection uses UDP to connect and program where the server receives connectionless datagrams from many clients and prints them.

Initially, a socket is constructed while it is in unconnected state, which means the socket is in it’s own and is not associated with any other destination beyond its boundary. The subroutines that needs to be connected binds a destinations i. e. the IP address of the server and the port number to which it listens the requests which is a permanent one to the

socket and now puts it in connected state. Once this process is completed behind the scene an application program will call the subroutine to establish a connection before it prepares it self to transfer data through a socket. More importantly all sockets that are used with connectionless datagram i. e. UDP services does not need be connected before they are used but connecting them makes a more efficient and effective way to transfer data between the client and the sever without specifying the destination each an every time.

Note: The processes cannot share ports during any time of the process as it is specified permanently to the desired connection itself having said that UDP multicast has the ability to share port numbers which uses a slightly different concept which will not be discussed in this answer.

The diagram below illustrates the example in a clear view

Q5. The Request-Reply Protocol is underlying most implementations of remote procedure calls and remote method invocations. In the Request-Reply Protocol, the request messages carry a request ID so that the sender can match answer messages to the requests it sent out.

1. Task: Describe a scenario in which a client could receive a reply from an earlier request.
2. Before stepping into answering the questions straightaway let me first briefly explain what the Request-Reply protocol is and why it is used for. The Request-reply protocol is an effective special-purpose protocol for distributed systems based on UDP datagrams

The functions of the RRP are listed below

- When the RRP is in play the reply message from the server forms an acknowledgement for the message requested by the client => avoiding overhead
- There is no guarantee that if a requested message is sent that it will result in a method being executed
- Re-transmission and identification of messages can increase reliability
- RRP helps to keep history of messages to avoid re-execution and repetition in the method during a request when transmitting reply messages.

Now coming onto answer the question, assume that a client requests the server and is waiting for a reply message, accordingly the client should get the requested reply within a certain period of time if it doesn't the client sends another request which in other words is known as idempotent operations i. e. operations that can be performed repeatedly with the same effect as if it had been performed exactly once: re-execute the operation. If the server receives the second request it then provides a conditional acknowledgement message this depicts that the server guarantees a reply for the client without letting the client to make any more requests for the same message which it has already made.

The diagram below has also explained the same as said above.

The Request-Reply-Acknowledge (RRA) protocol is a variant of the Request-Reply (RR) protocol, where the client has to acknowledge the server's reply. Assume that the operations requested by the client are not idempotent, that is, their outcome is different if they are executed a second time.

3. Task: For each of the two protocols, RR and RRA, describe which information the server has to store in order to reliably execute the requests of the client and return information about the outcome. Discuss as well when the server can delete which piece of information under the two protocols

Basically the main difference between Request-Reply (RR) and Request-Reply Acknowledge (RRA) is that In the Request-Reply Protocol, the requested messages carry a request ID so that the sender can match answer messages to the requests it sent out but where as this is not the case in Request-Reply-Acknowledgement (RRA) protocol, here the client acknowledges the server's reply messages, and the acknowledgement message contains the ID in the reply message being acknowledged.

If we are specifically talking about transmitting requests in the transport layer the Request-Reply protocol is the most effective one to be used because:

- No acknowledgments are necessary at the transport layer.
- Since it is often built by UDP datagrams connection establishment overheads can be avoided.
- There is no necessity for flow control as there are only small amount of data being transferred.

In order to reliably execute the requests made by the clients the server has to importantly store the information that is in the request ID so that it makes the server identify the client and respond to its request immediately.

The Request ID contains the following information which the server has to store.

- Sending process identifier
- IP address of the client
- Port number through which the request has come
- Integer sequence number incremented by sender with every request

Arguably this can also be the most efficient protocol compared with the Request-Reply Acknowledge protocol because this provides Non-idempotent operations i. e. re-send result stored from previous request but the exception here is that it requires maintenance of a history of replies so that it can make use whenever it receive a request.

It is clearly said that the non-idempotent operations do have their limitations therefore to limit the size of history and make the connection more reliable and efficient we use Request-Reply Acknowledge protocol.

REFERENCES

- Distributed Systems – Concepts and Design, 3rd Ed. G Coulouris, Jean Dollimore, Tim Kindberg: Books
- Distributed Systems: Principles and Paradigms by Andrew S. Tanenbaum (Author), Maarten van Steen (Author)
- Other Internet sources
- Websites and Forums
- Lecture slides and notes