CS6650 - Scalable Distributed Systems

Name - Mahvash Maghrabi

Homework Set #4

Late days left : 4

Answer the following questions using explanations and diagrams as needed. No implementation needed.

1. The TaskBag is a service whose functionality is to provide a repository for 'task descriptions'. It enables clients running in several computers to carry out parts of a computation in parallel. A master process places descriptions of subtasks of a computation in the TaskBag, and worker processes select tasks from the TaskBag and carry them out, returning descriptions of their results to the TaskBag. The master then collects the results and combines them to produce the final result.
The TaskBag service provides the following operations:setTask allows clients to add task descriptions to the bag; takeTask allows clients to take task descriptions out of the bag. A client makes the request takeTask, when a task is not available but may be available soon. Discuss the advantages and drawbacks of the following alternatives:
i) The server can reply immediately, telling the client to try again later.
ii) The server operation (and therefore the client) must wait until a task becomes available.
iii) Callbacks are used.

i) When the server replies to the client mentioning to try again later it is unsatisfactory as the client will have to poll the server in carrying out extra requests which may lead to unnecessary network traffic. Also other clients might have requested for a task before the client that was told to try again later which will result in the waiting client being disadvantaged.

ii) The server operation has to wait until a task becomes available. The reasons could be that the operation requested by one client cannot be completed till an operation requested by another client has been performed. It happens when some clients are producers while the others are consumers. Another reason could be the client needing the resource may have to wait for other clients to release it. The wait and notify methods are a solution to this problem. The thread calls wait on an object so as to suspend itself and to allow another thread to execute a method. A thread calls notify to inform any thread waiting on that object that it has changed some of its data. If the server and the client wait until a task becomes available, the waiting client can immediately respond when a task is available. Doing this will avoid the overhead that will occur with polling and ensure that clients receive tasks in the same order in which they were requested.

iii) Callbacks are programming constructs they allow a method to call another method after completing the execution. They notify clients when a requested operation is completed or when a resource becomes available. Using callbacks will involve the client providing a method which will be called by the server when the task is available, this will get the task description and also execute the task. It will prevent overhead of polling but it requires more complex programming and implementing it in distributed systems will be a challenge.

2. A server manages the objects a1, a2, ..., an. The server provides two operations for its clients:

read (i) returns the value of ai; write(i, Value) assigns Value to ai.

The transactions T and U are defined as follows:

T: x = read(j); y = read (i); write(j, 44); write(i, 33); U: x = read(k); write(i, 55); y = read (j); write(k, 66).

Give three serially equivalent interleavings of the transactions T and U.

The interleavings of T and U are told to be serially equivalent if the outputs produced are same.
When T runs before U - xT = aj0 ; yT = ai0 ; xU = ak0 ; yU= 44; ai =55; aj =44; ak = 66.
When U runs before T - xT = aj0 ; yT = 55; xU = ak0 ; yU= aj0 ; ai =33; aj =44; ak = 66,

Interleaving 1

T	U
x:=read(j)	
	x:= read(k)
	write(i, 55)
y = read(i)	
	y = read(j)
	write(k, 66)
write(j, 44)	
write(i, 33)	

Interleaving 2

T	U
x:=read(j)	
y:=read (i)	
	x= read(k)
write(j, 44)	
write(i, 33)	
	write(i, 55)
	write(k, 66)

Interleaving 1 is equivalent to U before T and Interleaving 2 is equivalent to T before U

3. Concurrency Control and Java Apps

a. **Explain the 3 different concurrency control methods (Ch 16 Coulouris Book – Locks based, Optimistic and Time-stamp Ordering) briefly. Compare and contrast how they achieve concurrency control.**

Concurrency control allows users to access a database in a multiprogramming fashion while hiding the fact that each user is executing alone on its own system. The main task to achieve this is to prevent the updates or changes done by one user from interfering with retrievals and updates done by another user. It ensures the consistency and the correctness of data access by multiple concurrent transactions. Here we are discussing the three different concurrency control methods in distributed systems

Lock based Concurrency Control

In lock based concurrency control the transactions obtain locks on the data items they access so that other transactions cannot modify the same data simultaneously. The locks can read or write locks and are released only after the transaction has been done. They can be local, global and the distributed locks are managed by a centralized lock manager. This concurrency control has data consistency but it might lead to deadlocks and reduced concurrency. Deadlocks occur when two or more processes are waiting for each other to get a resource.

Optimistic Concurrency Control

In optimistic concurrency control the transactions assume the conflicts to be very rare and so they carry on the transactions without locking. It does not lock every record instead the system looks for indications that two users tried to update the same record at the same time. When the information is confirmed then one of the user's updates is discarded and the user is informed about it . Every transaction reads the data items it requires and then writes back the updated data when the transaction ends. Locking overhead is reduced but the concurrency is increased. This type of concurrency might lead to frequent rollbacks which might decrease the performance.

Timestamp Ordering Concurrency Control

In timestamp concurrency control every transaction is given a unique timestamp and the transactions are executed according to the order of timestamps given to every transaction. Timestamp is a unique identifier to identify a transaction. They are assigned in the order in which they are submitted to the system. Every transaction will read the data items and write back the updated data making sure that the transaction's timestamp is prior to the timestamps of any transactions that have already modified the same data item. This concurrency guarantees serializability and prevents conflicts. This might lead to increased communication overhead.

In conclusion,
Lock-based concurrency control provides strong data consistency and prevents conflicts, but it can lead to network traffic and increased communication overhead.
Optimistic concurrency control increases concurrency by reducing locking overhead but can lead to frequent rollbacks and decreased performance
Timestamp ordering concurrency control guarantees serializability and prevents conflicts, but it can lead to increased communication overhead.
Therefore which concurrency control has to be chosen depends on the system requirements, the workload of the system. Distributed systems also require distributed transaction management to ensure consistency and fault tolerance.

b. Java Concurrency In Practice by Brian Goetz is the practice manual for Dist Systems. Study Chapter 6 (Task Execution) which is a small chapter. Identify 5 different programming devices (e. g. Tasks, Exec Framework) used in Java for Task Execution. Then, briefly explain how you might use these to implement a basic locks based concurrency control in a simple Web App.

The five different programming devices used in Java for Task Execution are as follows :

1) **Executor Framework -** Java has its own own multithreading framework which is called the the Java Executor Framework.It manages threads and executes tasks in a thread pool. Using the framework lets us submit the task for execution and managing all the threads is taken care of by the executor. It is used to run the runnable objects with no need of creating new threads each and every time and mostly reusing the threads that have already been created. Different types of executors are :
    - SingleThreadExecutor
    - FixedThreadPool(n)+
    - CachedThreadPool
    - ScheduledExecutor
2) **Execution Policies -** An execution policy is mainly responsible for answering the questions like What, Where, When, and How of the task execution. It answers questions like :
    - What are the threads in which the tasks will be executed?
    - What is the order in which tasks will be executed?
    - How many tasks will be running concurrently?
    - How many tasks will be held in the queue?
    - Which task should be neglected if the system is facing overloading?
    - What actions should be taken post execution of a task?

The execution policy giving optimal results depends on the basis of the computing resources and the quality of service.

3) **Thread Pools -** They manage a pool of worker threads. The thread pool is bound to a work queue which holds all the tasks that are waiting to be executed. A worker thread works as such that it requests for the next task from the work queue then it executes it, and then waits for another task. Some of the advantages of this is reusing the thread that already exists instead of creating a new thread brings down the cost over multiple requests. It also improves responsiveness. If the size of the thread pool is properly tuned there will be enough threads to keep the processors busy and not have so many that your application runs out of memory or crashes due to competition among threads for resources.

   The thread pool can be created by calling the methods in Executors:
   newFixedThreadPool - This method creates a fixed-size thread pool. It creates threads as tasks are submitted and keeps the pool size constant.

   newCachedThreadPool - It can reap idle threads when the pool size exceeds and can add new threads when demand increases

   newSingleThreadExecutor - It creates a single worker thread to process tasks.Tasks are processed sequentially in a task queue (FIFO, LIFO, priority order)

   newScheduledThreadPool - It is fixed size and it supports delayed task execution

4) **Executor Lifecycle -** As an Executor processes tasks asynchronously we cannot determine the state of previously submitted tasks . While Some might have been completed there might be some that might be currently running while others may be waiting in the queue for execution. For shutting down an application there are two ways one is a graceful shutdown and the other is an abrupt shutdown. Executors can shut the application in both ways and also send the status of tasks that were affected by the shutdown. The ExecutorService interface extends Executor, adding a number of methods for lifecycle management . It has three states that are running, shutting down, and terminated. Initially it is created in the running state. The shutdown method here is a graceful shutdown. The shutdownNow method initiates an abrupt shutdown. Any task that is given to the ExecutorService after it has shut down is handled by the rejected execution handler. When all tasks are completed, the ExecutorService transitions to the terminated state.

5) **Delayed Periodic Tasks -** There are two facilities that manage deferred and periodic tasks. The ScheduledThreadPoolExecutor can be created through its constructor or through the newScheduledThreadPool factory. The Timer creates only a single thread for executing timer tasks and if the timer task takes too long to run it will affect the accuracy of other tasks. Scheduled thread pools provide multiple threads for executing both deferred and periodic tasks. Timer is that it behaves poorly if a timed task throws an unchecked exception. ScheduledThreadPoolExecutor deals properly with such tasks.

Locks-based concurrency control is a technique that is used to manage access to shared resources. It is achieved using locks. Locks are synchronization mechanisms that prevent multiple threads from accessing a shared resource simultaneously. Whenever a thread acquires a lock, it has access to the shared resource and all the other threads are supposed to wait until the lock is released before they can access it. Using locks ensures that only one thread at a time can modify the shared resource and this will eventually prevent race conditions and ensure data consistency. Implementing a basic lock-based concurrency control in a simple web app will require the use of the above mentioned tools in task execution.

Thread pools - We can create a thread pool with a fixed number of threads which will ensure that only a limited number of requests are processed concurrently. Whenever a request is made a thread pool can be submitted for processing. The lock can be acquired before processing the data so that no other thread is modifying the shared data that the request is accessing. The lock can be released once the request is processed.

Executor Framework -This will handle the thread management and ensure that all requests are being processed.

Execution Policies - This can be used to control the number of tasks that are running concurrently and the size of the work queue. A maximum number of threads can be set so as to prevent resource contention and a maximum size of work queue this will prevent memory overflow.

Executor Lifecycle - It can be used to gracefully shut down the application. When shutting down we can wait for all running tasks to complete and notify any waiting threads to release any locks they are holding.

Delayed Periodic Tasks- A web application may fetch its advertisements from an external ad server and in case advertisement is not available within the given time frame. It instead displays a default advertisement so that advertisement unavailability does not undermine the site’s responsiveness requirements.

4) Transaction Processing in a Distributed System
What are distributed transactions in Oracle DB? How are they different from Remote Transactions? Give examples. How does Oracle DB use Naming service and 2-Phase Atomic COmit Protocols to manage distributed transactions?

A transaction that has one or more than one statements that might individually or as a group update the data on two or more distinct nodes is called a distributed transaction. If all statements of a transaction refer only to a single remote node then that transaction is remote and not distributed. There are two types of operations in distributed systems :

- DML and DDL Transactions - CREATE, DELETE, INSERT, LOCK, SELECT
- Transaction Control Statements - COMMIT, ROLLBACK, SAVEPOINT

An Example :

```
UPDATE company.employee
  SET loc = 'BOSTON'
  WHERE empno = 100;
UPDATE company.cubicle
  SET cubno = 20
  WHERE cubno = 30;
COMMIT;
```

A remote transaction contains only statements that access a single remote node. A distributed transaction has statements that access multiple nodes.
Remote transactions perform a single transaction in this transaction the client starts a transaction with the system and then waits for a response. The system completes the transaction and sends the response back to the client.
Distributed transactions perform multiple transactions the client starts the transaction that involves multiple distributed systems. The client sends a request to all the distributed systems and waits for a response from each one of them. When all the responses are successful the transaction is committed else it rolls back.

A distributed transaction involves altering data on multiple databases. The distributed transaction processing is more difficult as the database has to coordinate the commit or roll back of the changes in a transaction as a self-contained unit. It ensures the data integrity using the two-phase commit mechanism. It also uses a naming service to manage distributed transactions.

Naming Service: This helps in locating the distributed database resources that participate in a distributed transaction. This mainly resolves the names of the resources to their network addresses which enables communication between the distributed database resources.
Two Phase Commit Protocol: This protocol ensures that all the distributed database resources participating in the transaction commit or roll back together in an atomic manner. The protocol consists of the following two phases:

Prepare Phase -The global coordinator asks participating nodes other than the commit point site to promise to commit or roll back the transaction when there is a failure. If there is any node that cannot prepare then the the transaction is rolled back
Commit Phase - If all participants reply with a "ready to commit" message, the global coordinator sends a commit request to every participant to commit the transaction. If there is any participant who replies that it is not ready to commit it sends a rollback request. There is also a forget phase where the global coordinator forgets about the transaction.

5. Using Chandy-Lamport algorithm, and the below diagram show when each process records its local state (you can annotate the figure) and list the channel states for each process captured in the snapshot. Black dotted lines are marker messages. Red lines are messages (A to F)

The channel states after visualizing and understanding the given snapshot are as follows :

P1

C_21 = records -> stops record -> Received message from event A (P2)

C_31 = records -> stops record -> Received message from event D (P3)

P2

Circle event = [A]

C_12 = empty

C_32 = records -> stops record -> Received message from event F (P3)

P3

Circle event = [B,D,F]

C_23 = empty

C_13 = record -> stops record -> Received message from event E (P1)