Design Document of:

CloudKon clone with Amazon EC2, S3, SQS, and DynamoDB

This experiment is about implementing a cloudkon with various services provides by amazon AWS, the experiment is divided into two modules 1) CloudKon with Local queue (in-memory) and 2) Cloudkon with remote queue (SQS) with DynamoDB to keep track of duplicates.

Programming language used for implementation:

LocalQueue- Java

RemoteQueue- Java

Test Bed used:-

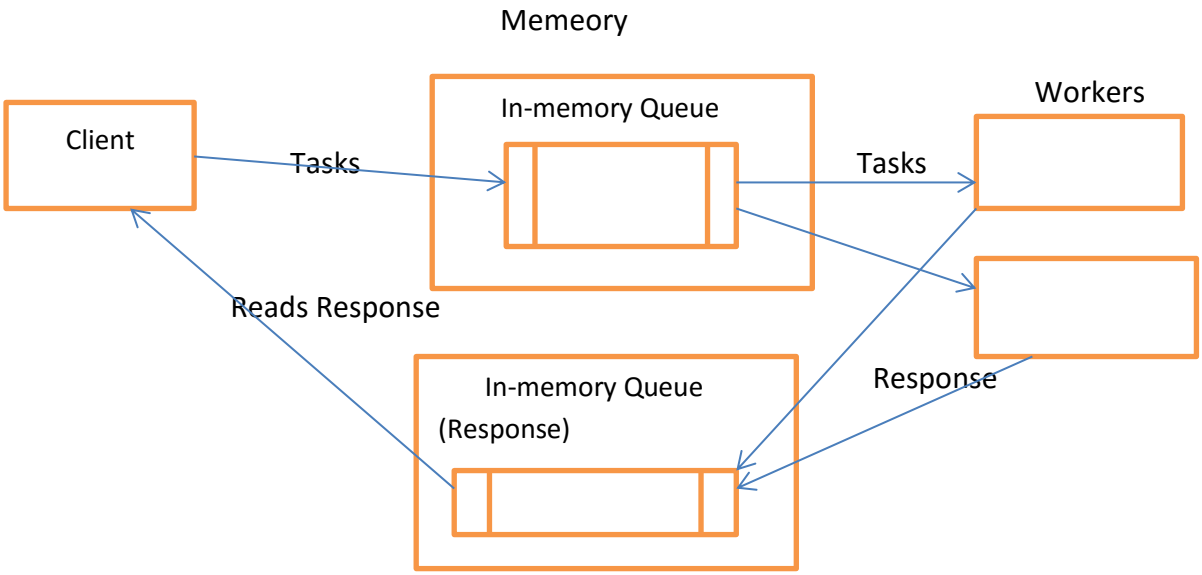
Amazon Free Tier EC2.T2 Micro

Processor: Intel Xeon Processor speed up to 3.3GHz

Client and Local Back End Workers:

This is about implementing the local in-memory queue, the implementation language used for this is java, where one Client is created responsible to create the in memory blocking queue. First it reads a command which will give a queue name, along with the workload file name. Then client creates a blocking queue, along with this it reads a sleep tasks from the workload file and put it into the queue. The worker which polls the queue for the task will get notified about existence of a task, worker will take task from the queue and executes the sleep task along with it generates a response as 0 for successful execution and 1 for failure during execution. Client on the other side uses this response to check whether the response is correct or not. The implementation doesn't require any kind of network so there will not be any overhead of network.

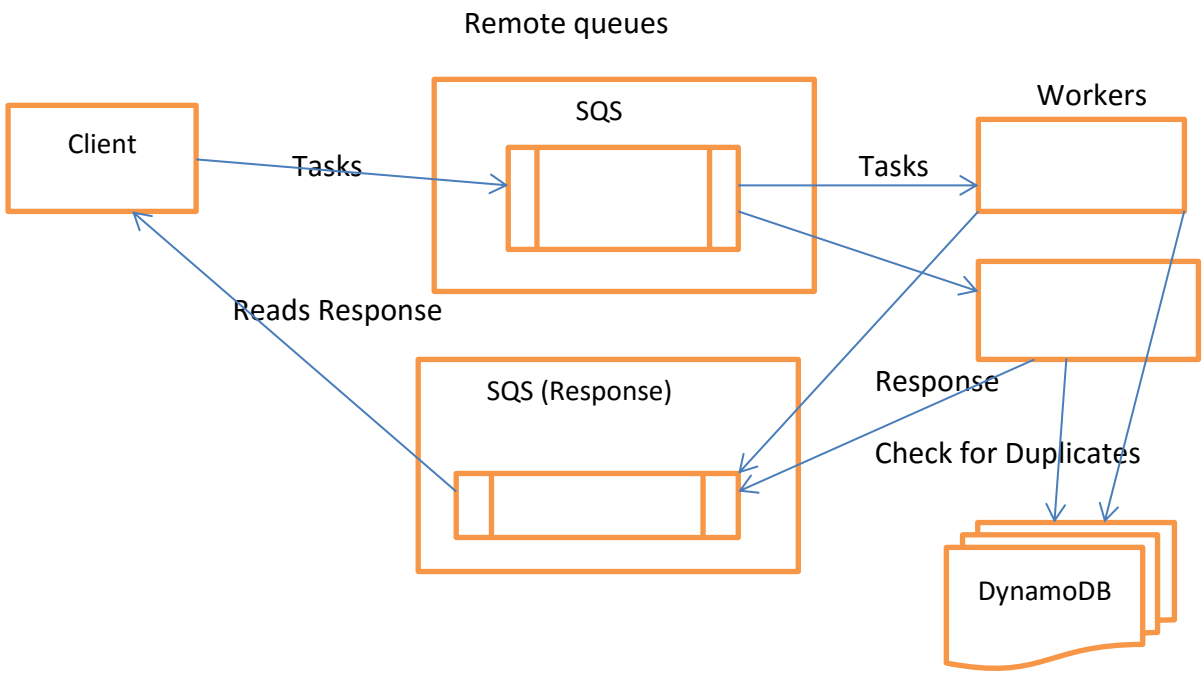
Typical architecture for this scenario:



Client and Remote Back End Workers

This implementation of an experiment, considers a queue at the network level, now the SQS by AWS is used as remote queue, where Client running on one instance responsible to create to read the command line argument for workload file and creates queues for job scheduling task for workers. Client reads the tasks from workload file and put that into SQS, now workers which continuously polling the remote queue will read a task from the queue and process it. One problem with SQS is that, while reading the tasks from SQS it doesn't guarantee to deliver the task exactly once, so in order to track the duplicates, DynamoDB is used, so before executing the tasks, the worker will scan the DynamoDB with message ID, if the message Id is already present then worker will ignore the duplicate tasks else it will execute it. After execution of tasks the worker will send a response to the client about the completion of task execution.

Typical architecture for this scenario:



Conclusion

Through the designing of CloudKon, learned that how the tasks is scheduled when the workers are running within the system and also learned that how jobs are being scheduled when workers are spread across the network, which requires independent job scheduling.