RARE MILE
TECHNOLOGIES PVT. LTD.

Improving Performance In Amazon SimpleDB and Simple Queue Service(SQS)

In this paper, we try to examine the challenges faced by developers while working with AWS services of SimpleDB and SQS in terms of performance. It includes a brief overview of Amazon Web Services Offerings, what are the benefits they provide and how to make sure that their performance is good.

Manish Singh,

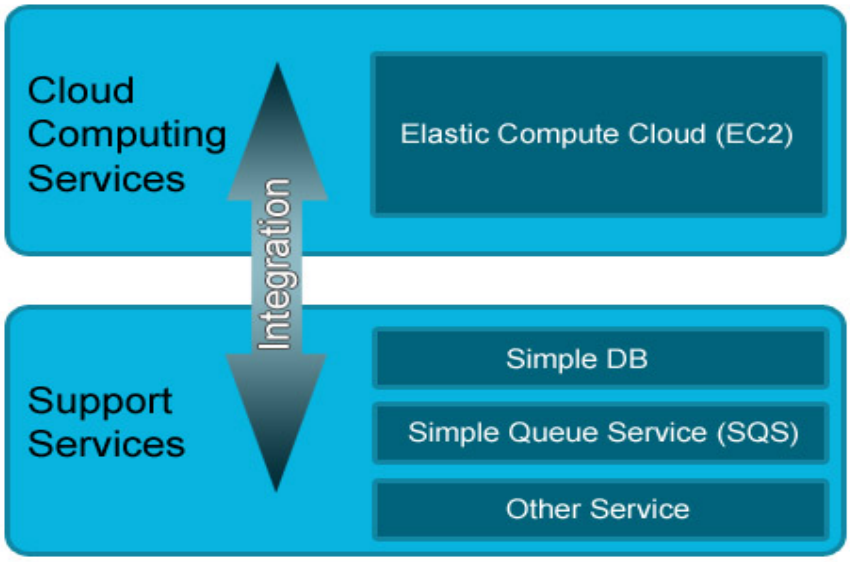
Associate Technical Architect

Rare Mile Technologies

www.raremile.com

Abstract

Amazon Web Services (AWS) is arguably the biggest player in field of cloud services. With approx 1 million registered developers worldwide, around 9.1 billion of quarterly sales (as of July 2011) and more than 30 cloud related services on offering, AWS is one of the most popular Web Services providers. Most of the AWS offerings are cutting edge technologies, have high availability, are supported for variety of platforms, scalable and also low cost which is an important factor in growth of any new business related to IT.



Out of around 30 Service offerings from AWS, Amazon SimpleDB and Amazon SQS are two upcoming services. With time, these are gaining a lot of popularity and acceptance by a wide range of developers. Amazon SimpleDB is a highly available and flexible non-relational data store that can be accessed using Amazon defined web services. Developers simply store and query data items via web services requests and Amazon SimpleDB does the rest.

Amazon Simple Queue Service (Amazon SQS) offers a reliable, highly scalable, hosted queue for storing messages as they travel between computers. By using Amazon SQS, developers can move data between distributed components of their applications that perform different tasks, without losing messages or requiring each component to be always available. Apart from SOAP and REST Web Service APIs, Amazon also offers SDK in different programming languages (Java, .NET, PHP, Ruby etc) using which developers can easily communicate with these services.

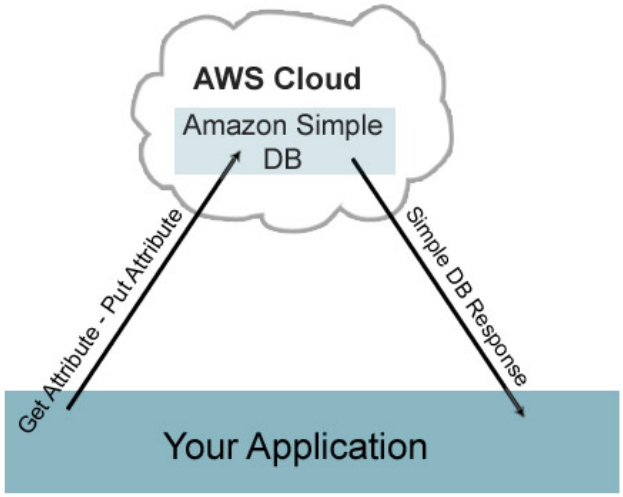
One of the big challenges that most new developers face with these two technologies is Performance! Developers, by nature, focus on implementing the functional features of an application and do not pay much attention to the non functional features such as performance and scalability in the beginning. It is only towards the fag end of the project that they take time out to focus on

issues relating to performance and then spend considerable amount of time to fix performance issues. Anybody who has dealt with a performance issue can vouch for the amount of work, retesting and balancing it takes to solve a performance issue.

Sometimes this can lead to redesigning of many components of an application. Since Amazon Web Services involves dealing with components which reside on remote servers, performance becomes even more critical part of integrating AWS components with the applications. If you think debugging performance issues in local applications is tough, wait till you get to do it on a remote component. In this white paper we will go through the problems related to performance in Amazon SimpleDB and SQS and discuss some tricks to solve these issues. These are mainly based out of our experience of dealing with these components and also based on some guidelines given in documents related to AWS.

SimpleDB brief overview

Before going into performance related complications, let us first get a small overview about SimpleDB.

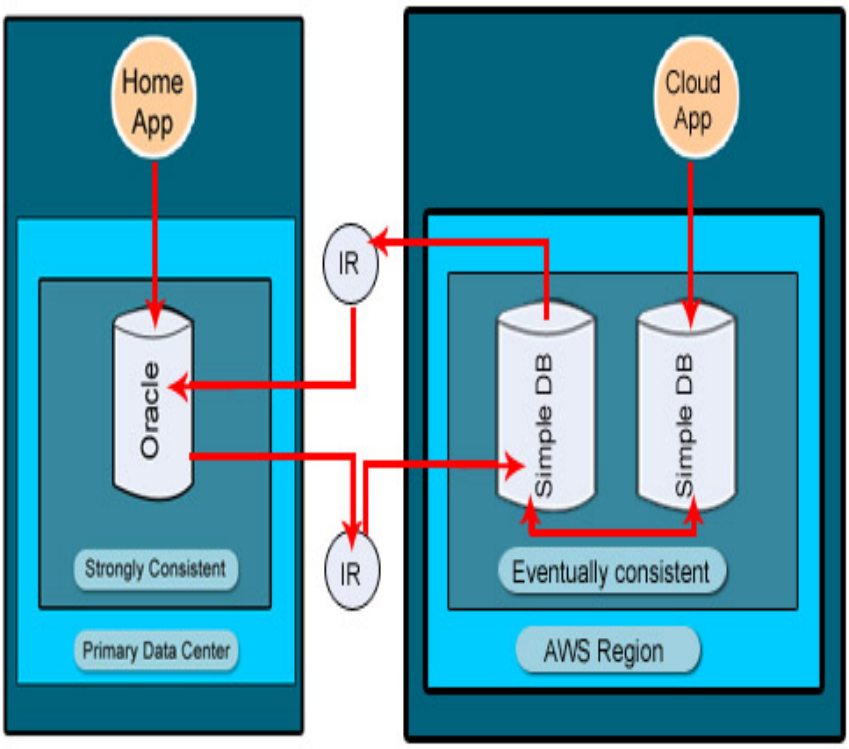


SimpleDB is a data storage service provided by Amazon on its own servers. Anyone who wants to use this needs to create an account with Amazon Web Services. After registration, one can use SimpleDB APIs or language specific SDK (e.g. For Java, .NET etc) provided by AWS for storage and retrieval of data. Here are some key points related to SimpleDB:

- SimpleDB works on the concepts of **domains**, it is similar to the tables on oracle or any other relational DBMS. Each domain contains an **Item** which should be unique (this is equivalent to the primary key in Oracle etc). Each item contains a set of **attributes** (equivalent to column in RDBMS). Every attribute has a name and a value to store.
- SimpleDB does not implement SQL. It has it's own limited set of commands to perform create, read, update and delete operations on domain and items. Similarly there are very limited set of inbuilt functions to operate on data.
- Amazon has pay per use policy. This means the amount you pay to Amazon is dependent on the volume of data that you transfer from your SimpleDB domains. There is no restriction on the number of domains you can create but

there are limits on the size of attribute names and values as well as amount of data you can insert or query at a time.

- When you store data in SimpleDB domain, it replicates the data on many other machines of same region as well. This is done in order to prevent data loss and ensuring high availability. Since replicating the data takes some time, the data stored in SimpleDB is not always consistent. However, it is guaranteed to be consistent after a brief delay. This is called **Eventual Consistency**. Below is an image explaining this phenomenon with comparison to a traditional RDBMS like oracle.



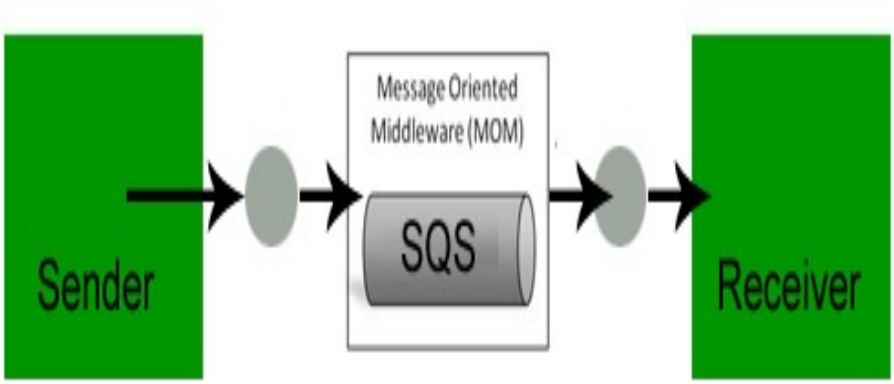
SimpleDB Performance Challenges

With this background, let us look at some performance challenges which are peculiar to SimpleDB. Since SimpleDB stores data on remote machines, the first problem is obviously of latency and response time. Additionally, the eventual consistency behaviour can cause some performance issues as well. Sometimes due to that if you perform a read operation on the data that has just been written then you might not get the updated data for some time. SimpleDB also applies throttling policies per domain. This means after a certain number of requests have been serviced in a second, SimpleDB will start giving service unavailable responses. Moreover, if you increase your data size beyond a certain limit, then the response time increases, which means storing data gets slower. SimpleDB also starts giving 503 service unavailable responses when the data size in any domain becomes very large.

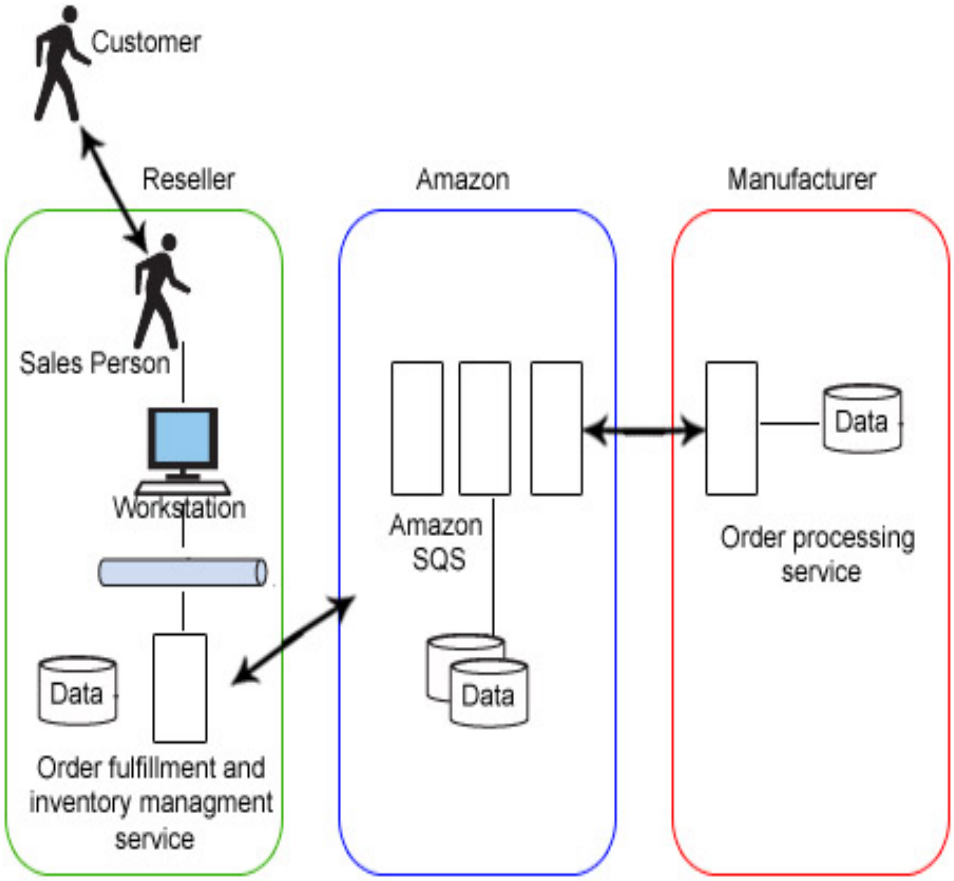
Amazon SQS overview

A queue provides a mechanism to transfer messages between two programs or threads of the same program. Messaging queues promote an asynchronous means of communications, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue

are stored until the recipient retrieves them. Below is one image which explains messaging queue:



Amazon Simple Queue Service (Amazon SQS) offers reliable, highly scalable, hosted queues for storing messages as they travel between computers. By using Amazon SQS, developers can move data between distributed components of their applications that perform different tasks, without losing messages or requiring each component to be always available. Amazon SQS provides simple, standards-based SOAP and REST web services interfaces that are designed to work with any Internet-development toolkit. Here is one example of Amazon SQS usage for order management project:



Here are some key points related to Amazon SQS:

- Amazon SQS provides APIs for creating/deleting queues and sending/receiving data in queues. One can create as many queues as needed.
- When a message is received, it becomes “locked” while being processed. This prevents multiple processes from processing the message simultaneously. If th

message processing fails, the lock expires and the message becomes available again. In case where the application needs more time for processing, the "lock" timeout can be changed dynamically via an API operation.

- Developers can securely share Amazon SQS queues with others. Queues can be shared with other AWS accounts or with anonymous users. Queue sharing can also be restricted by IP address and time-of-day.
- All messages have a globally unique ID that Amazon SQS returns when the message is delivered to the queue. The ID is not required in order to perform any further actions on the message, but it is useful for tracking whether a particular message in the queue has been received. Amazon SQS supports up to 12 hours maximum visibility timeout for the 2009 and 2011 WSDLs.

Amazon SQS Performance Challenges

As a service offered on the cloud, SQS has similar challenges as SimpleDB i.e. latency, response time and consistency. Once you put a message on the queue, it usually takes around one minute for that message to be available for reading. Also there is no guarantee of the sequence in which a message is received after sending to queue. This means that if you have sent three messages in the queue, there is no guarantee that first message will be received first.

Tips to improve performance

Now let us look at how to get around these performance issues. Since both SimpleDB and SQS are cloud based services, there are some common tips of managing performance for both of them. SimpleDB also involves some logic and conditions while storing and retrieving data, so there are some performance tips specific to SimpleDB. Let us have a look at the common things first.

Common tips for both SimpleDB as well as SQS

Below are some of the things which can be done to improve performance for both SimpleDB and SQS:

- **Select the Nearest Region (End Point):** This is the backbone for performance improvement. Amazon provides servers in seven regions worldwide:

  1. US East (Northern Virginia)
  2. US West (Oregon)
  3. US West (Northern California)
  4. EU (Ireland)
  5. Asia Pacific (Singapore)
  6. Asia Pacific (Tokyo) and South America (Sao Paulo).

  These regions are available for both SimpleDB and SQS. So depending upon where your application will be used, you should set the nearest end point of Amazon SimpleDB or SQS region. By default, **sdb.amazonaws.com** or **sqs.amazonaws.com** point to the US East region. All the language SDKs provide methods to set end point. You can utilize these to change end point to the nearest region like **sqs.us-west-1.amazonaws.com or sdb.us-west-1.amazonaws.com** and so on. More details are available on AWS site for these products.

Using the nearest endpoint reduces latency and improves response time to a great extent.

- **Create Multiple Domains /Queues:** As stated earlier, if the data size becomes large in a SimpleDB domain or SQS queue, then it starts taking more time to read or update those or even put new data. So in order to avoid that we need to store those data either in new SimpleDB domain or create a new queue (in terms of messages). In case the stored data is not needed any more, it should be deleted from that domain or moved to an archived domain.

- **Multithreading in Requests:** AWS provide SDK for all major platforms, so if the language supports multithreading then it is better to use that for parallel processing. Multithreaded requests result in better throughput i.e. it will increase number of transactions per Second (TPS).

- **Use Batch Operations:** Amazon also provides facility of batch operations to put/delete data in SimpleDB and send/delete messages in SQS. By using batch operations you can put/delete 25 items at one go in SimpleDB and process 10 messages at once in SQS. In normal circumstances you would have to send 25 and 10 requests respectively. By clubbing multiple requests together, you can cut down on the latency and improve application throughput.

Tips specific to SimpleDB

Here are some performance measurement tips related to SimpleDB:

- In SimpleDB, sorting is lexicographical. This means the data is sorted by alphabets first, then digits. But SimpleDB does not support numerical sorting. So, the number 10 will come before 2 if you try to sort numbers in ascending order. In order to sort data numerically, the numeric attributes should be zero-padded logically. Also, for dates use ISO8601 format, it works better with lexicographical sorting.

- Try to use composite keys as item name wherever possible. Suppose if you have to get data on the base of two attributes then in that case you will have to use AND clause after WHERE clause in query which makes data retrieval slow. So if you give your item name as combination of these two attributes and make it unique, it will improve the performance.

  For example instead of using:

  SELECT * FROM DOMAIN_NAME where FIRST_NAME='FIRST' AND LAST_NAME = 'LAST';

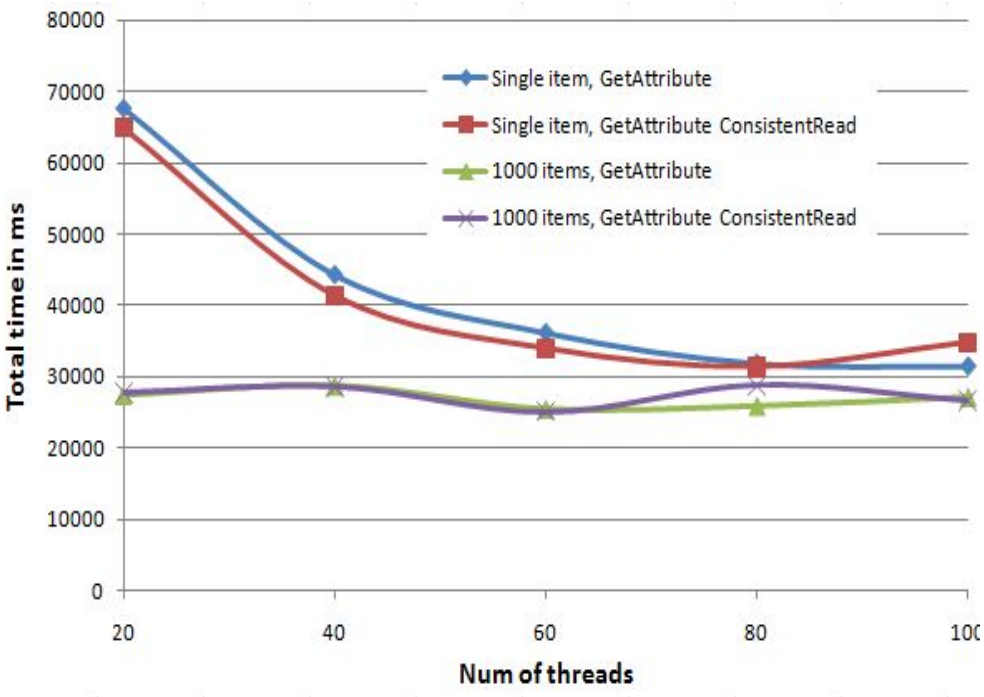
  it's better to use

  SELECT * FROM DOMAIN_NAME WHERE itemName = 'FIRST:LAST'.

  The bottom line is to avoid using conditions in query to improve performance. Use them very rarely.

- It's a known thing that if you use consistent read parameter as true while reading data then it impacts performance. So if you require transactional

guarantees for certain data, use Conditional Puts. You can optionally also use Consistent Reads, but it is not required. In SimpleDB, data consistency comes at the cost of performance. So if you are doing a read just after write, avoid using consistent read = true unless it's critical and necessary. Below is an image that shows the variation in response time while using consistent read for single as well as multiple items.



- By default simple db returns 100 records for one select query. You can increase the limit to maximum of 2500. It is helpful when you know that your query will return more than 100 items. Otherwise you will need to make multiple queries to domain to get complete data.

- We need to avoid non-indexed queries. Avoid queries like:

  select * from MY_DOMAIN where SOME_ATTRIBUTE is NULL.

  These queries are not indexed and can result in an unpredictably long cycle of null results and next-pointer following. If you plan to do this query for customer-facing applications, you must use a pseudo-null i.e. a default value for null. This will allow you to do queries like:

  select * from MY_DOMAIN where SOME_ATTRIBUTE = 'my-pseudo-null'

- Avoid carrying multi-table relationships into the cloud in the form of multi-domain relationships. Try to de-normalize these relationships into single items. Doing joins in the application tier might require multiple round-trips to SDB and open customer-facing functionality to time-outs.

- Since writes are throttled to SimpleDB domains, shard (i.e. break or partition) your domains to scale write traffic if you expect to do more than 70 singleton puts/second at any point in the future. Also, if you ever need more than 1 billion attributes or 10GB of space, shard your domains.

Conclusion

So now we have gone through some details of Amazon SimpleDB and SQS, performance related challenges and steps to improve performance for those. The base for performance improvement is to chose the nearest AWS region, try to send more data in less number of requests (i.e. use batches for operations) and avoid making complex queries to data (in case of SimpleDB. Following these steps will help making you life a bit easier while doing development and save a lot of time of rework which you can utilize spending with your family and friends.

References:

- http://aws.amazon.com/simpledb/
- http://aws.amazon.com/sqs/
- http://practicalcloudcomputing.com/post/712653349/simpledb-essentials-for-high-performance-users-part-1
- Documentation available on Amazon website related to their product suit.