

# Cloud Performance Benchmark Series

## Amazon Simple Queue Service (SQS)

Md. Borhan Uddin

Bo He

Radu Sion

ver. 0.5



## 1. Overview

“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 simply 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 makes it easy to build an automated workflow, working in close conjunction with the Amazon Elastic Compute Cloud (Amazon EC2) and the other AWS infrastructure web services.

Amazon SQS works by exposing Amazon’s web-scale messaging infrastructure as a web service. Any computer on the Internet can add or read messages without any installed software or special firewall configurations. Components of applications using Amazon SQS can run independently, and do not need to be on the same network, developed with the same technologies, or running at the same time.” (amazon.com)

Experiments were performed to benchmark SQS. Two main different aspects were experimented with: (i) number of simultaneously supported users, and (ii) message size and throughput rate characteristics.

## 2. Setup

SQS can be created in 5 regions – US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore) and Asia Pacific (Tokyo). The message body can contain up to 64 KB of text in any format (default is 8KB). However, we have noticed in our experiments, that **any message body that contains more than 9.5 KB of text cannot be sent to SQS** even after reconfiguring the maximum message size to be 64 KB. Messages can be retained in queues for up to 14 days (default is 4 days), and can be sent and read simultaneously. We performed tests under two settings: loopback (Figure 1a), and end-to-end (Sender-SQS-Receiver) (Figure 1b).

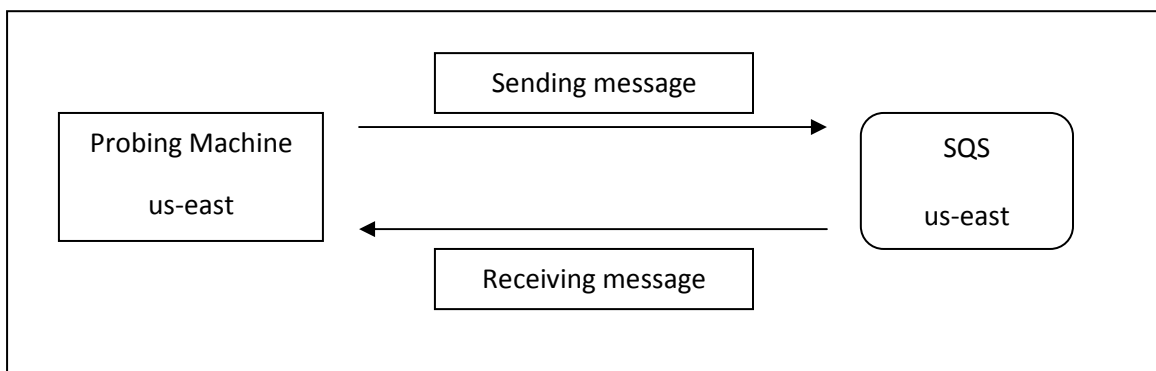
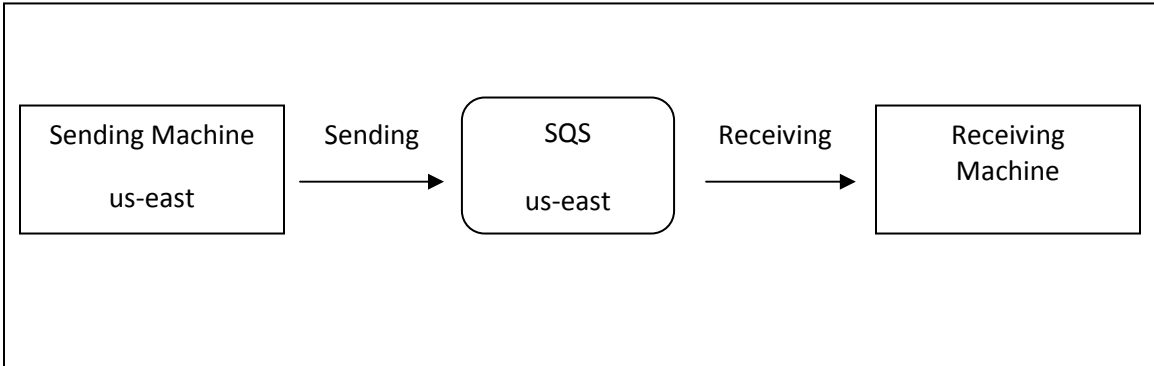


Figure 1a: Experimental setup for loopback setting

In the loopback setup (Figure 1a), SQS was probed from an XLarge EC2 instance with Amazon Web Service (AWS) API tools and PERL scripts installed on it. This instance was constantly sending and receiving the messages simultaneously.



**Figure 1b: Experimental setup for end-to-end (Sender-SQS-Receiver) setting**

In the end-to-end setup (Figure 1b) two XLarge EC2 instances were used -- one acted as a sender (sending messages to SQS), while the second acted as a receiver (receiving messages from SQS).

No significant difference (throughput, delay etc.) has been observed between these two settings. For conciseness we therefore decided to present here mainly the loopback setting results but note they apply to both scenarios.

All deployed probing machines' configurations were identical in terms of allocated memory (RAM) in Gigabytes (GB), Platform, and EC2 Compute Units [ECU; 1 ECU aims to approximate an Intel 1.0-1.2 GHz 2007 Opteron or 2007 Xeon CPU] etc. The processing power is structured as a set of multiple virtual cores (VC). A VC in turn is considered as a collection of ECUs (and has an associated number of ECUs/VC). Thus, the total number of ECU compute units can be computed by knowing the number of virtual cores and the number of ECUs per virtual core. Table 1 shows the configurations of the probing machine:

Instance Size	Memory (GB)	Total ECU (=VC* ECU/VC)	Platform	AMI Id
Extra Large	15	8 (=4*2)	Fedora 14 64-bit	ami-e291668b

**Table 1:** Probing machines' configurations (ECU=EC2 Compute Unit, VC=Virtual Core)

There are some setup parameters (options) in SQS, but the SQS server configuration does not seem to be open to subscribers (i.e., no console/terminal access and no way to

customize CPU, IO, memory etc. by users). The following are the queue attributes of SQS (which cannot be altered):

- **VisibilityTimeout** - length of time (seconds) for which a message received from a queue will be invisible to other receivers. An integer from 0 to 43200 (12 hours). If message is not deleted from queue, it will become visible to others after this interval.
- **MaximumMessageSize** – allowed bytes per message. An integer from 1024 to 65536. The default for this attribute is 8192 (8KB).
- **MessageRetentionPeriod** – time (seconds) SQS retains a message. An integer from 3600 (1 hour) to 1209600 (14 days). The default for this attribute is 345600 (4 days).

The configuration details of the SQS are illustrated in Table 2.

Name	Visibility Timeout	Maximum Message Size	Message Retention Period
SQS	0 second	65536 (64 KB)	345600 (4 days)

**Table 2:** Configurations of Simple Queue Service (SQS)

To test SQS behavior for multiple users sending and receiving messages simultaneously the following setup was deployed: One XLarge probing machine in the us-east region was set up to run from 1 to 128 sender threads simultaneously which pushed messages to an SQS instance, also in the us-east region. Simultaneously, the same machine was running from 1 to 128 polling threads to receive (pull) the messages.

*a. Throughput*

SQS throughput was measured as follows: One XLarge probing machine in the us-east region, was configured to repeatedly send from 0.5KB to 9.5KB sized plain text messages (in 0.5KB increments) to an SQS instance, also in the us-east region. Average achieved throughput per message size was calculated. Messages (and their corresponding threads) were identified by cross-matching cryptographic hash values at both sender and receiver side.

*b. Cost*

The dollar cost per message was also computed using the Amazon pricing model at the time of writing (included here for reference). Two types of pricing need to be considered: requests cost and cost of data transfer (in and out). Figure 2a shows an example Data Transfer pricing chart of Amazon SQS. Figure 2b shows an example Requests pricing chart of Amazon SQS.

### Data Transfer\*\*

The pricing below is based on data transferred "in" and "out" of Amazon SQS.

Region: <input type="text" value="US East (Virginia)"/>	
Pricing	
<b>Data Transfer IN</b>	
All data transfer in	\$0.100 per GB
<b>Data Transfer OUT***</b>	
First 1 GB / month	\$0.000 per GB
Up to 10 TB / month	\$0.150 per GB
Next 40 TB / month	\$0.110 per GB
Next 100 TB / month	\$0.090 per GB
Greater than 150 TB / month	\$0.080 per GB

**Figure 2a:** Pricing (Data Transfer) of Amazon Simple Queue Service (amazon.com)

### Requests

- \$0.01 per 10,000 Amazon SQS Requests (\$0.000001 per Request)

**Figure 2b:** Pricing (Requests) of Amazon Simple Queue Service (amazon.com)

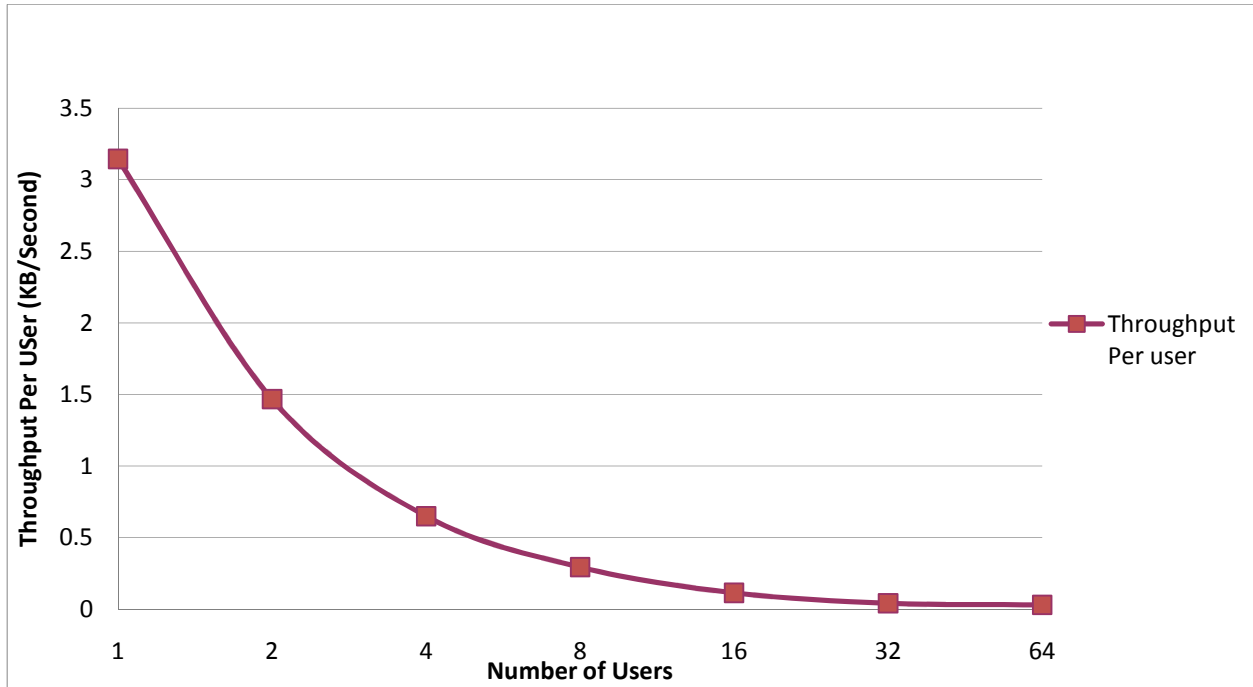
The cost per message was calculated in US microCents (1 microCent =  $10^{-6}$  cent) as follows:

$$\text{Cost per message} = \text{SQS Message Request message price} + \text{SQS Data Transfer price} \times \text{Amount of Data Transferred per message}$$

## 3. Results

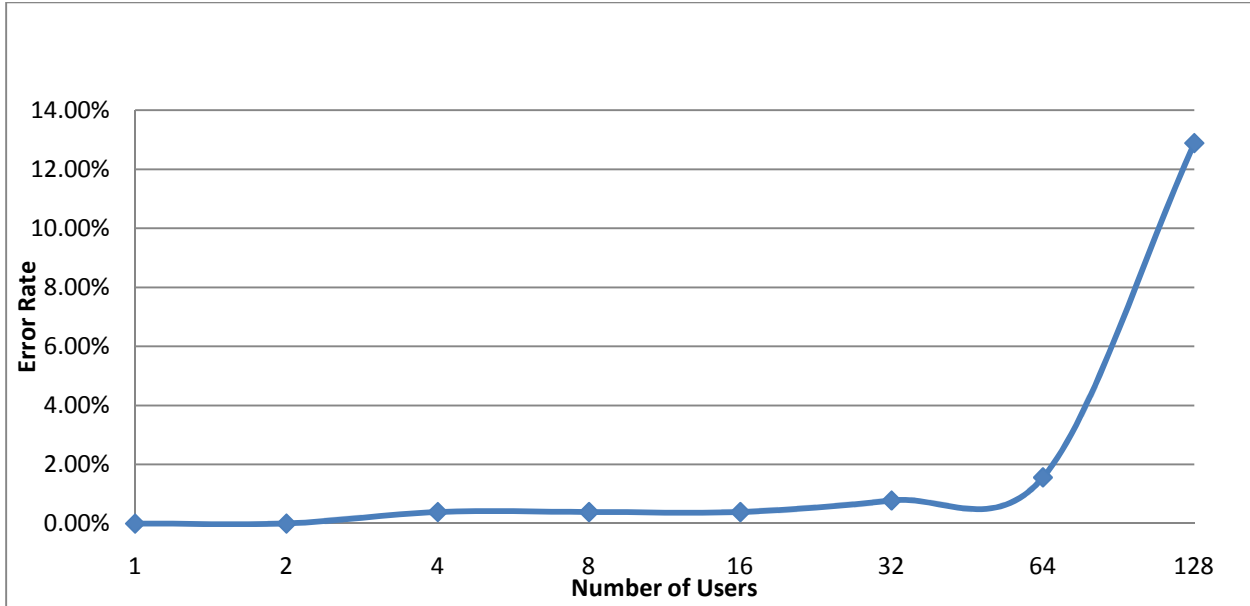
### 3.1. Behavior as a function of number of users:

With increasing number of users SQS throughput seems to drop significantly, and the error percentage increases (Figure 3a).



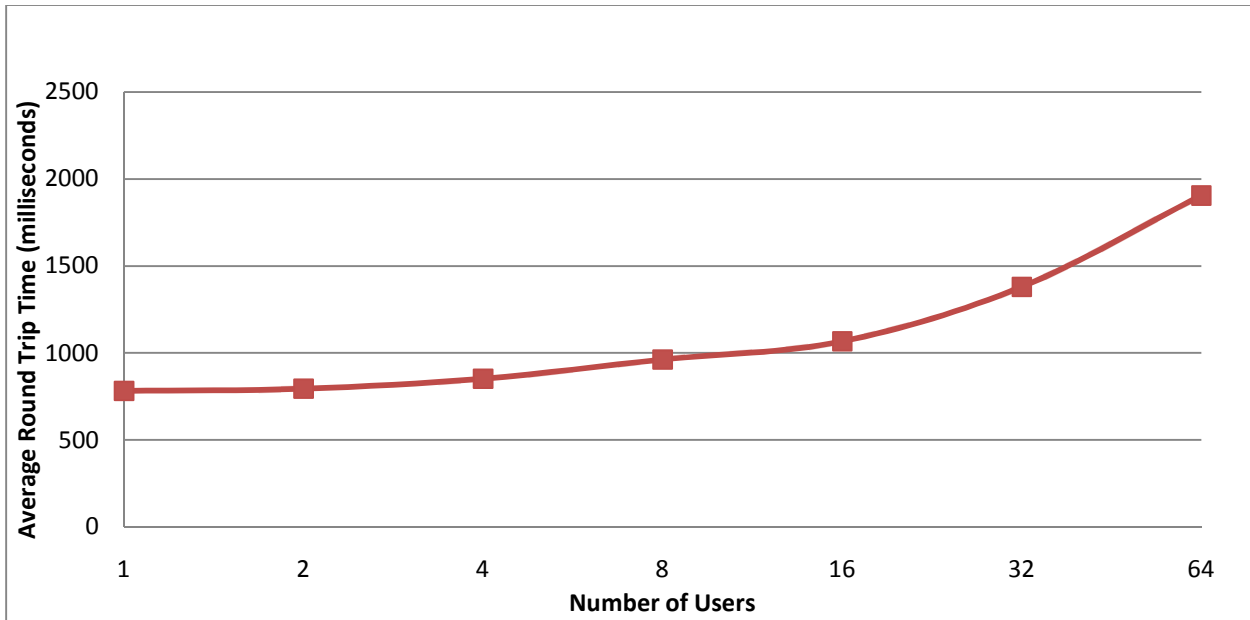
**Figure 3a:** Observed Throughput per User vs. Number of users in Multi-user Setup

The results for a 128 user setting were dropped due to too many errors (message loss or explicit SQS error message). This is a clear indication that SQS suffers from significant scalability issues, its throughput is low (maximum 3-4 KB/Second at low load) and it doesn't scale with an increasing number of concurrent users.



**Figure 3b:** Error Rate vs. Number of users in multi-user setup

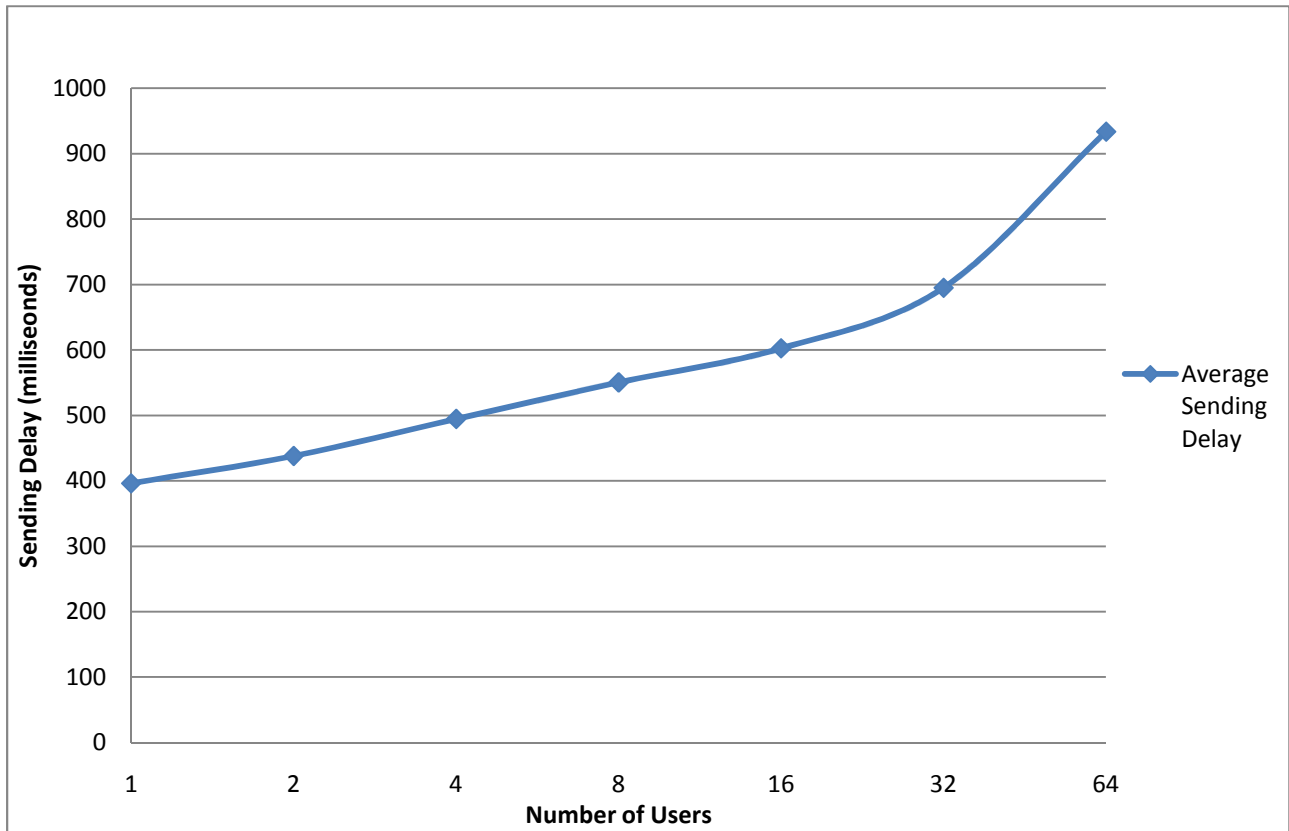
Once we detected such high error rates we believed it may be interesting to understand the error behavior as a function of number of simultaneous users. Up to 256 send requests were run for an increasing number of users (from 1 to 128). Figure 3b depicts the behavior. As can be seen, error rates are relatively low up to 48-64 users. Beyond that, however a significant spike is observed that we could not immediately explain.



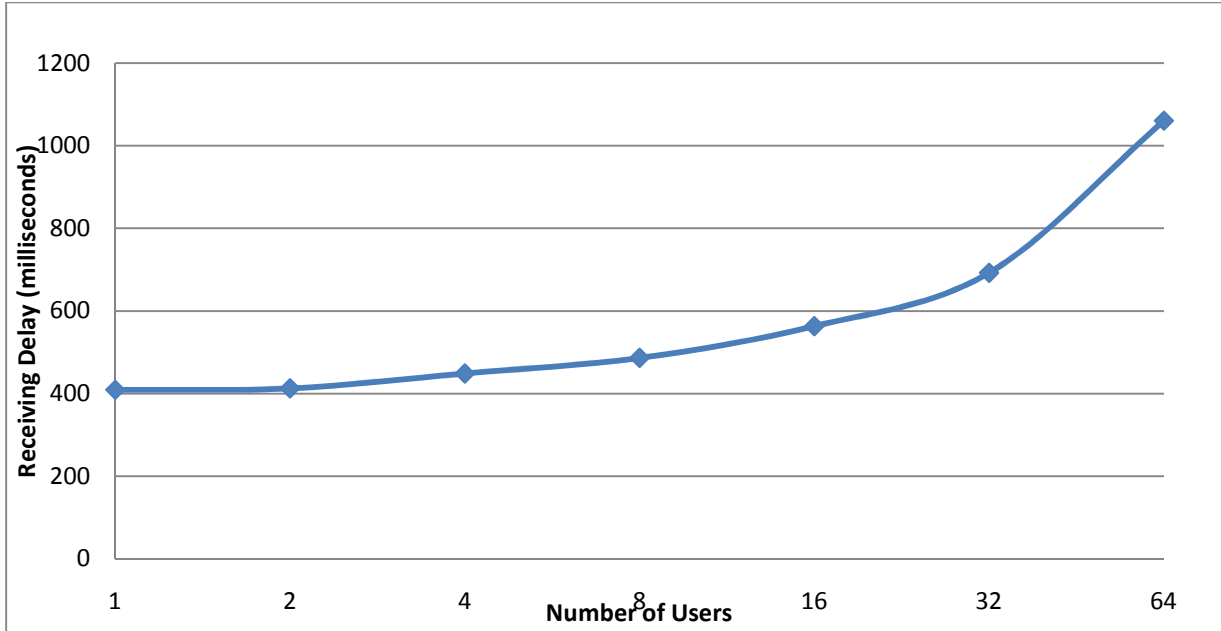
**Figure 3c:** Average Round Trip Time vs. Number of users in multi-user setup

Figure 3c, depicts the average message Round Trip Time (RTT) as a function of increasing number of users. A positively correlated behavior can be observed. This may be the result of increasing message processing latencies inside SQS.

Sending and receiving delays also seem to illustrate a similar phenomenon. With increasing number of users, both delays increase. Due to the SQS's inability to handle high request volumes, individual user experience (specifically throughput) suffers significantly.



**Figure 3d:** Average Sending Delay vs. Number of Users in Multiple Users Setup

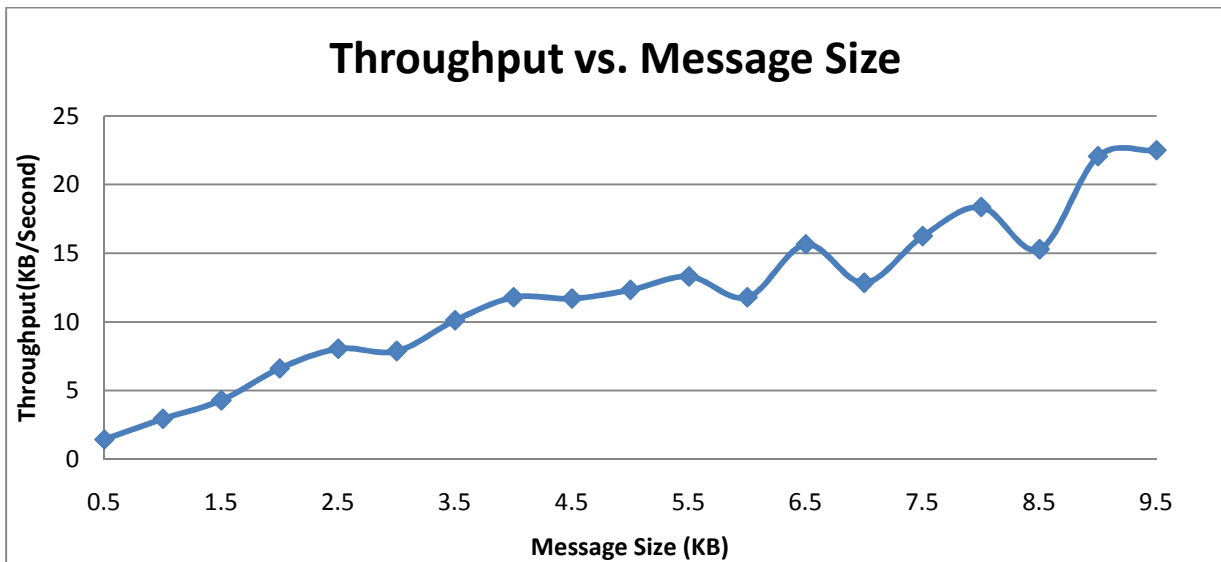


**Figure 3e:** Average Receiving Delay vs. Number of Users in Multiple Users Setup

### 3.2. Behavior as a function of Message Size

#### a. Throughput

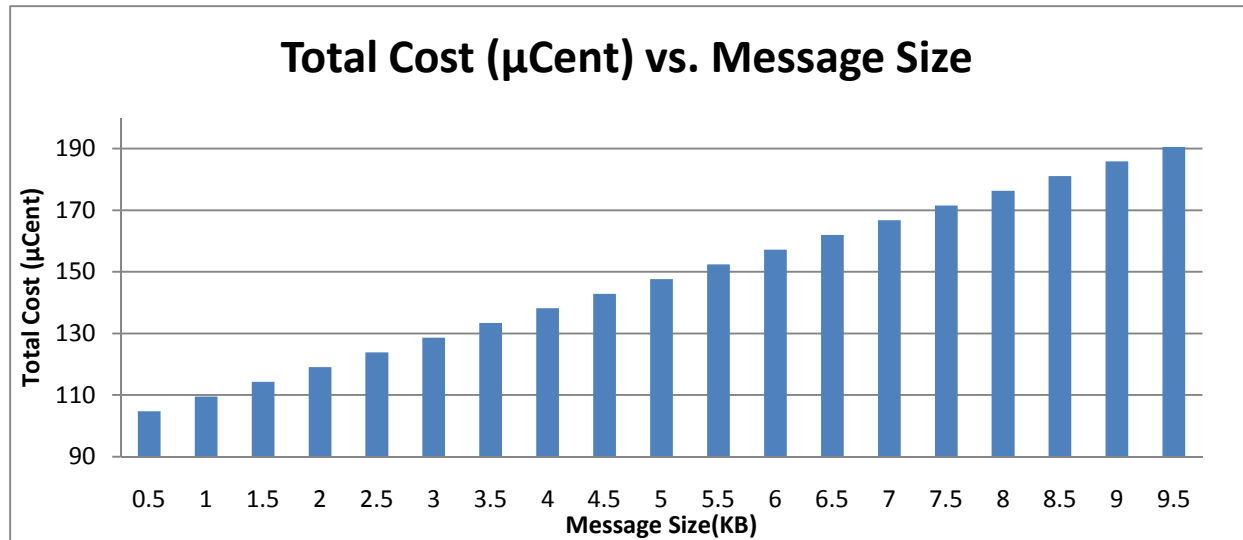
With increasing messages sizes (0.5-9.5KB, 0.5KB increments) in a single-user setup, SQS throughput naturally increases (Figure 4a). Overall throughput increases from 1.5 KB/second to 22.5 KB/second as the message size increases from 0.5 KB to 9.5 KB. Messages larger than 9.5KB could not be sent.



**Figure 4a:** Throughput vs. Message Size

### b. Cost

The individual message cost was also calculated for each message size that we could send. As expected, with increasing message sizes, the cost of sending each individual message also increases. Figure 4b shows the cost per message increases from about 105  $\mu$ Cent to 190  $\mu$ Cent as the message size increases from 0.5 KB to 9.5 KB.



**Figure 4b:** Cost vs. Message Size in Different Message Size Experiment

## 4. Conclusion

Unfortunately, the performed experiments only led to one conclusion. SQS featured poor scalability and does not seem to handle enough end-to-end messages to saturate even a small percentage of the available network bandwidth. This may be the result of inefficient message queue service design or unexplained SQS networking or latency bottlenecks.

SQS may be ok for small message sizes and relatively small numbers of concurrent users. For providing a reliable, scalable, and simple message queue service, it incurs a relatively low cost.

However, for applications with a large number of concurrent users and/or messages larger than 10KB, SQS may be (significantly) less than optimal.