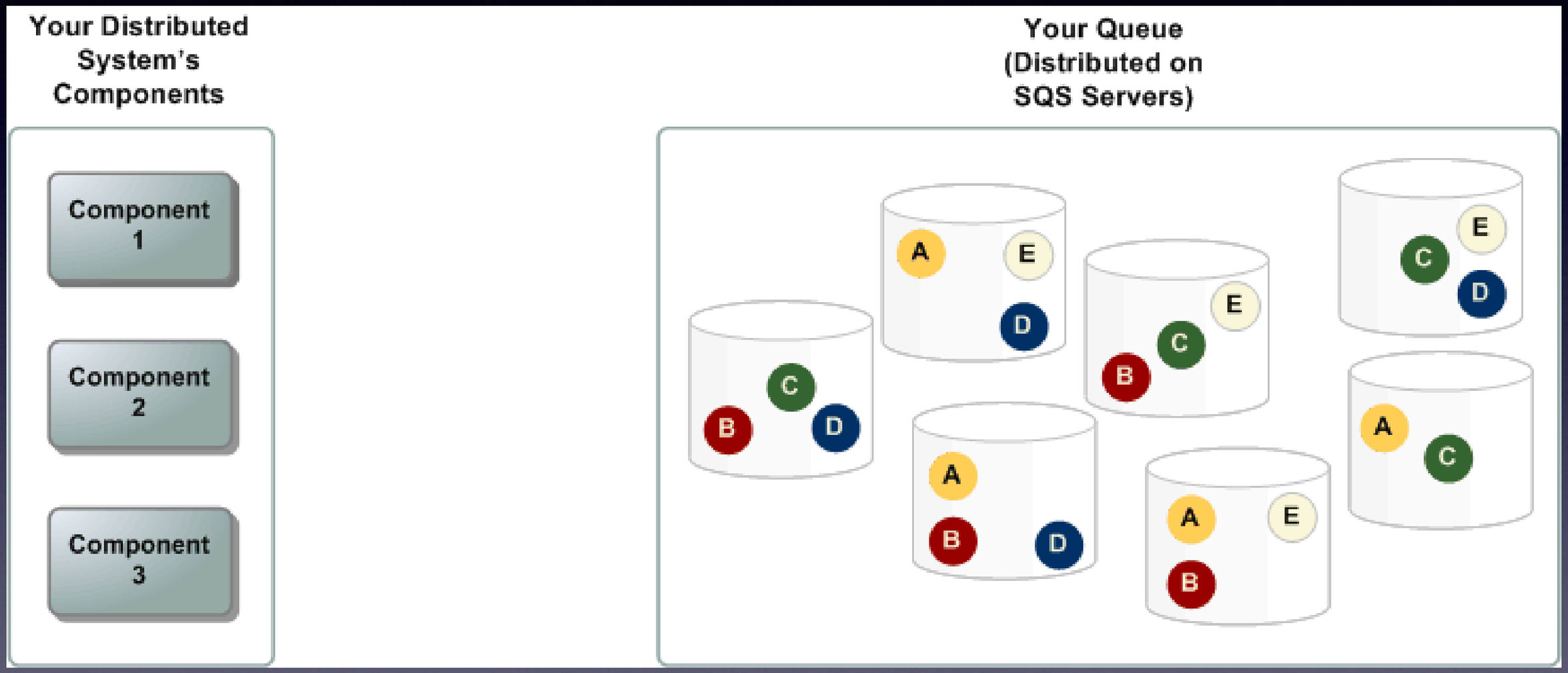
HDMQ :Towards In-Order and Exactly-Once Delivery using **H**ierarchical **D**istributed **M**essage **Q**ueues

Dharmit Patel
Faraj Khasib
Shiva Srivastava

Outline

- What is Distributed Queue Service?
- Major Queue Service
- Amazon SQS, Couch RQS, Apache Hedwig, Apache Kafka, RabbitMQ, ActiveMQ.
- Design and Implementation of HDMQ
- Operation Overview
- Refinements
- Performance Evaluation
- Conclusion
- Future Work
- References

How Distributed Queue Service look like?



In today's world distributed message queues is used in many systems and play different roles such as content delivery, notification system and message delivery tools

Distributed Queue Service

- Multiple Client are connected to this Service.
- Each client is flooding the queue with the messages.
- Each messages is stored on multiple nodes for reliability.
- Each message is at least delivered once.
- It is important for the queue services to be able to deliver messages in larger scales, at the same time it must be highly scalable and provide parallel access.

Major Distributed Queue Services

1. Amazon SQS (Commercial State of the art)
2. Apache Kafka
3. Apache Hedwig
4. Couch RQS
5. ActiveMQ
6. Rabbit MQ

Amazon SQS(Simple Queue Service)

- Amazon SQS is a distributed, message delivery service, which is highly reliable, scalable, simple and secure.
- SQS is Distributed over multiple data center. No Single point of Failure
- Guarantees extremely high availability.
- Deliver Unlimited number of messages at any time.
- Size of the **message <= 256 KB**.
- Ensures **at least 1 delivery** of message.
- Retains messages up to 14 days.
- Allows batching of messages up to 10 messages or 256 KB message size in total.
- Comes with a price tag of $0.50 every 1 Million Request.
- It doesn't guarantee message order.
- Price tag increases for message size > 64 KB.

Apache Kafka

- Highly Scalable as it is designed to allow **single cluster** to **serve** as the **central backbone**.
- Each Kafka fiber maintains a partitioned log.
- Relies heavily on the file system for storing cache messages.
- Kafka nodes perform load balancing.
- Uses Asynchronous Message Sending.
- It replicates its log information for each topic.
- Supports Batch compression of messages.
- Bottleneck is Disk speed and File locking mechanism used.

Couch-RQS

- Based on database system, which is called Couch DB.
- Couch DB is a fast light weigh NOSQL DB.
- Uses database File to store its information which gives bad performance.
- It might be faster than any SQL or No-SQL database.
- Couch RQS cannot run safely in a distributed/replicated environment and cannot scale high, cannot provide high availability.

Apache Hedwig

- Publish-subscribe system designed to carry large amount of data.
- It is designed with the goal to give guaranteed delivery.
- Topic based publisher and subscriber.
- Incremental Scalability and high availability.
- Client publish message associated with a topic, and they subscribe to a topic to receive all messages published with that topic.
- Hedwig Instance(region) consist of number of servers called hubs.
- Hubs partition up topic ownership among themselves.

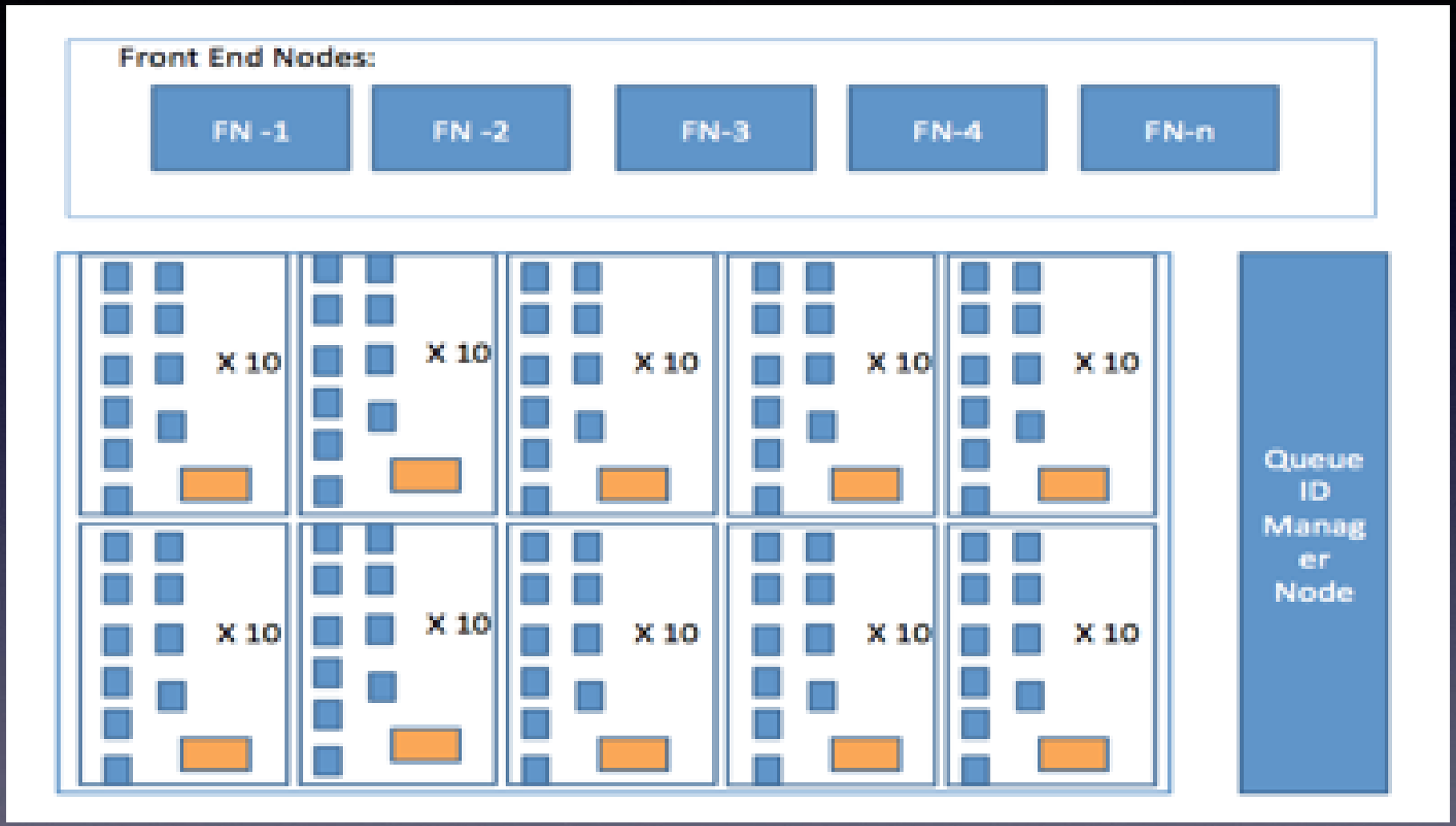
Active MQ

- Message Oriented Library, which ensures reliability between distributed process.
- Optimized to avoid overhead with a P2P or Server Client Model for pushing message to the receiver.
- Uses its own communication Protocol to ensure speed and reliability.
- Communication between server is using simple message.
- With each node launch, node launches the server to listen to any incoming messages and handle them.
- Highly Configurable but its slow and has issue lost/duplicate message.
- Three kind of scaling available in Active MQ like Default Transport, Horizontal scaling and Partitioning.

Rabbit MQ

- Robust messaging system, open platform and supported a large number of client developer platform.
- It uses asynchronous messaging for application to connect and scale.
- Options to tradeoff between performance, reliability, including persistence, delivery acknowledgements, publisher confirms and high availability.
- Uses mirroring for handling hardware failure.
- It offers management UI to monitor and control every aspect of message broker.
- Can report memory usage information for connections, queues, plugins and other processes in memory.
- Ships in the ready to use state, and can be customized in environment Variables, configuration file, runtime parameters and policies.

Design of HDMQ



Details for Design

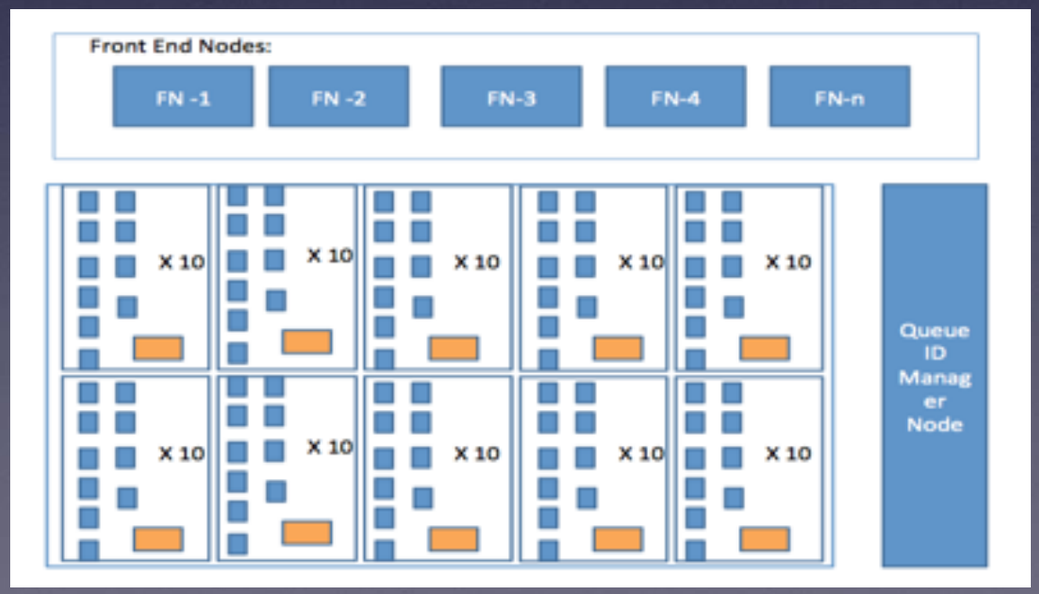
- **Storage Nodes:** All the storage in two hierarchical regions, where a sub region consists of ~10 nodes and a router node, the main region consists of multiple sub regions. All the main regions together make up the storage node system.

- **Front End Nodes:** These are the nodes that clients interact with and make request to. Each front-end node maintains a local hash-table for that contains updates for “Area” for each queue ID. Currently we are using 10:1 ratio for number of storage nodes vs. front-end nodes.

- **Queue ID Manager Node:** We use one queue ID node in the system that determines the storage region for new queues and generate area (queue ID) for the new nodes

Example for Setup

For example assume we have 10,000 total storage nodes and x number of front-end nodes. This system will break down the nodes in regions and sub regions down to where each of lowest hierarchy region contain ~ 10 nodes. In this case we can divide 10,000 nodes in 10 regions of 1000 nodes (1 to 10), then each 1000 node in region of 100 nodes and this 100 node regions in set of 10 nodes. So for example node 2287 will have area – 2, 2, 8



Operation Overview

- **Write Operation:** This are the step for it.

1. Front end node will route the message to the given area.
2. The router in that area will determine which node will be next for insert.
3. This router will follow round robin strategy until all the 10 nodes in the region are full.
4. If region is full, the message will be routers to next available region.

Front end node will maintain hash table and when write operation overflows the current region, it will get updated.

Read Operation

For Read Operation Following are the steps:-

1. Front end nodes use the area to determine the region where message are stored for that queue.
2. They initiate read request to the router for that region to read messages.
3. The reading of message is again done in round robin strategy.
4. This will maintain message order among different storage nodes.
5. If there is overflow of messages to another region, then the queue id is updated in the front end nodes are able to forward the read request to the overflow region.

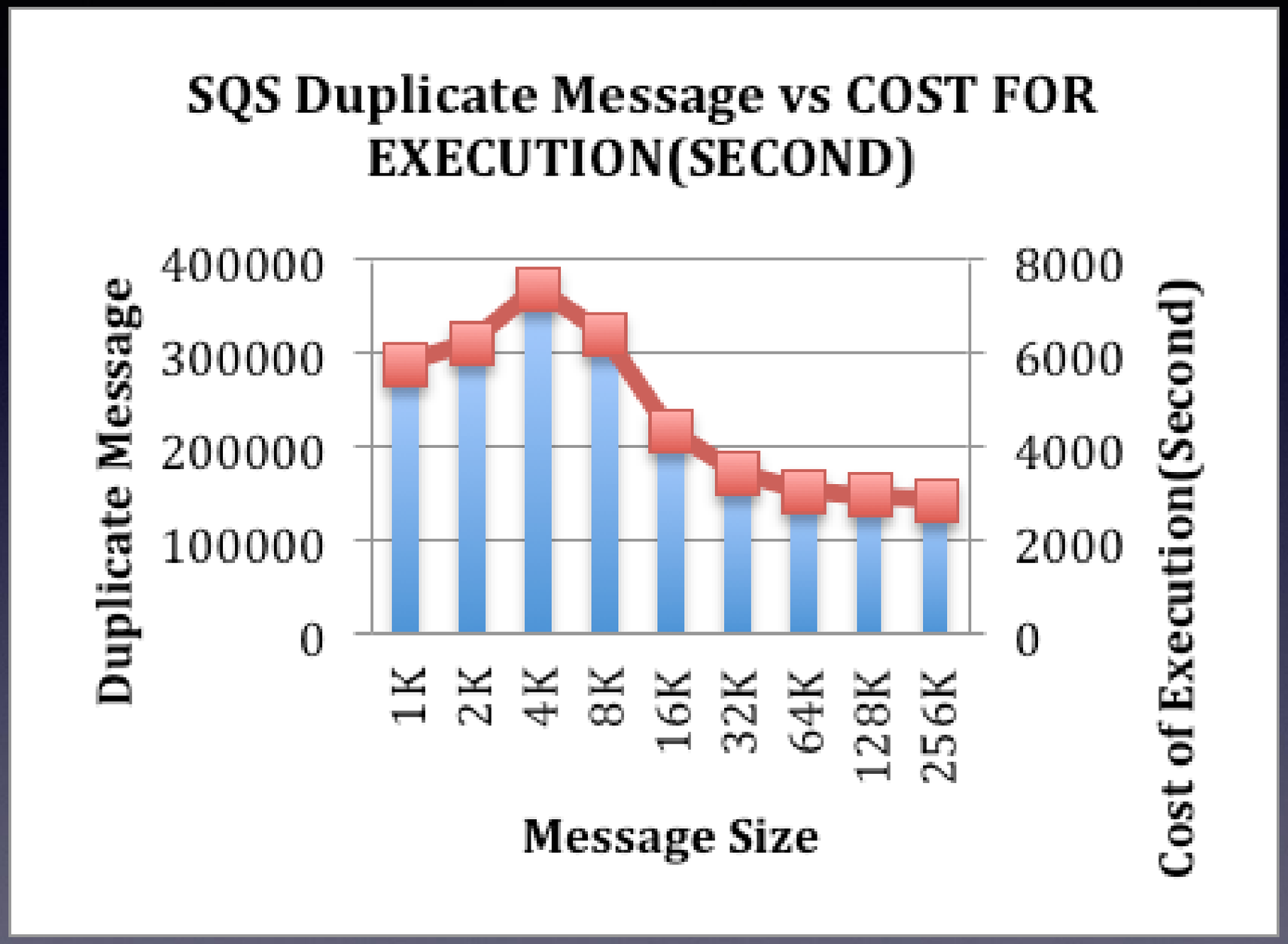
Queue ID Manager Node

- We also have a queue id manager node that will maintain the list of queue ID.
- Its job is to generate new ID based on system load.
- It will also assign the initial area to it.
- This node is low stress node so we need only 1~3 nodes to manage the system.

Refinements

- Exactly One Delivery
- Ordering of Message
- Large Message Size
- Mirrored Section Behavior

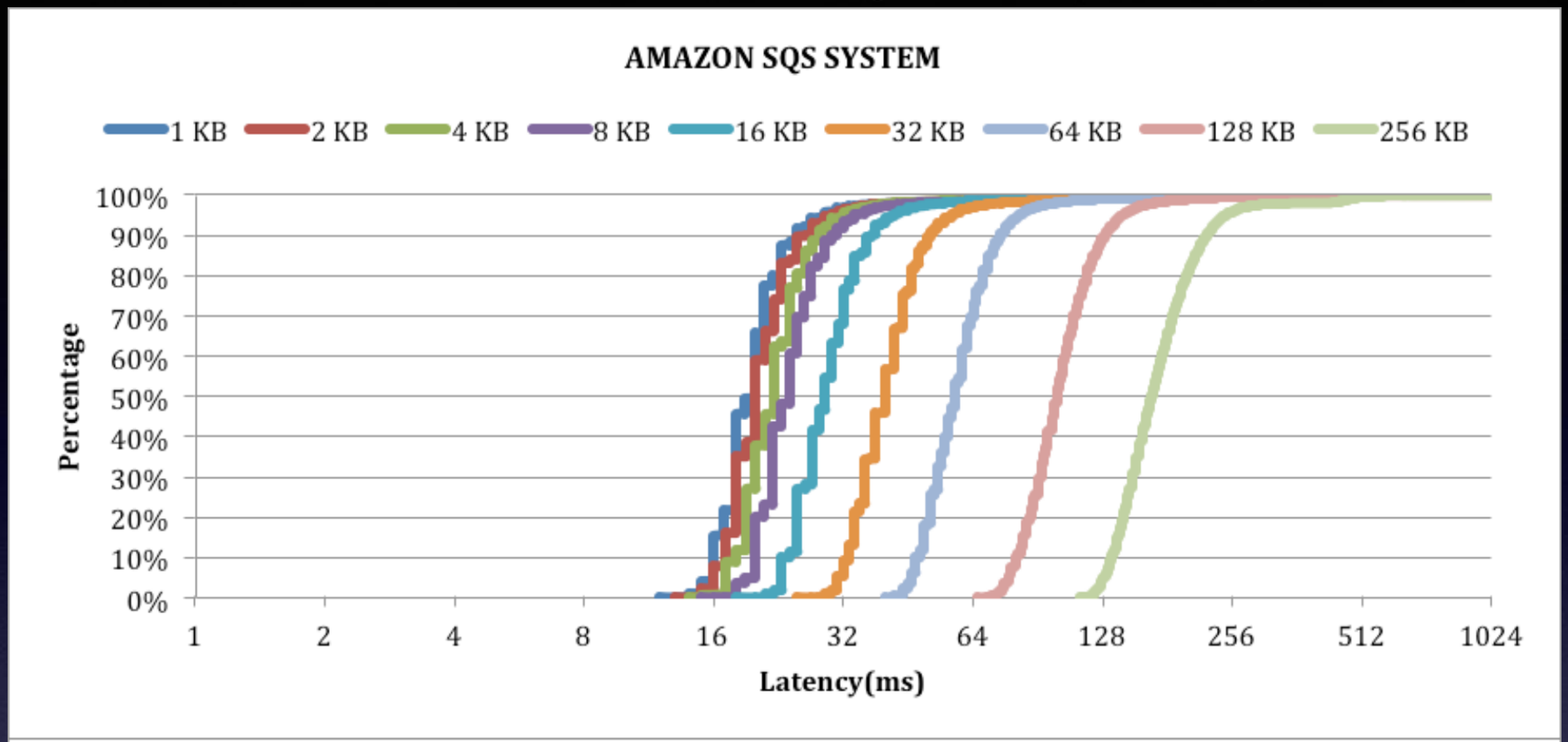
Performance Evaluation



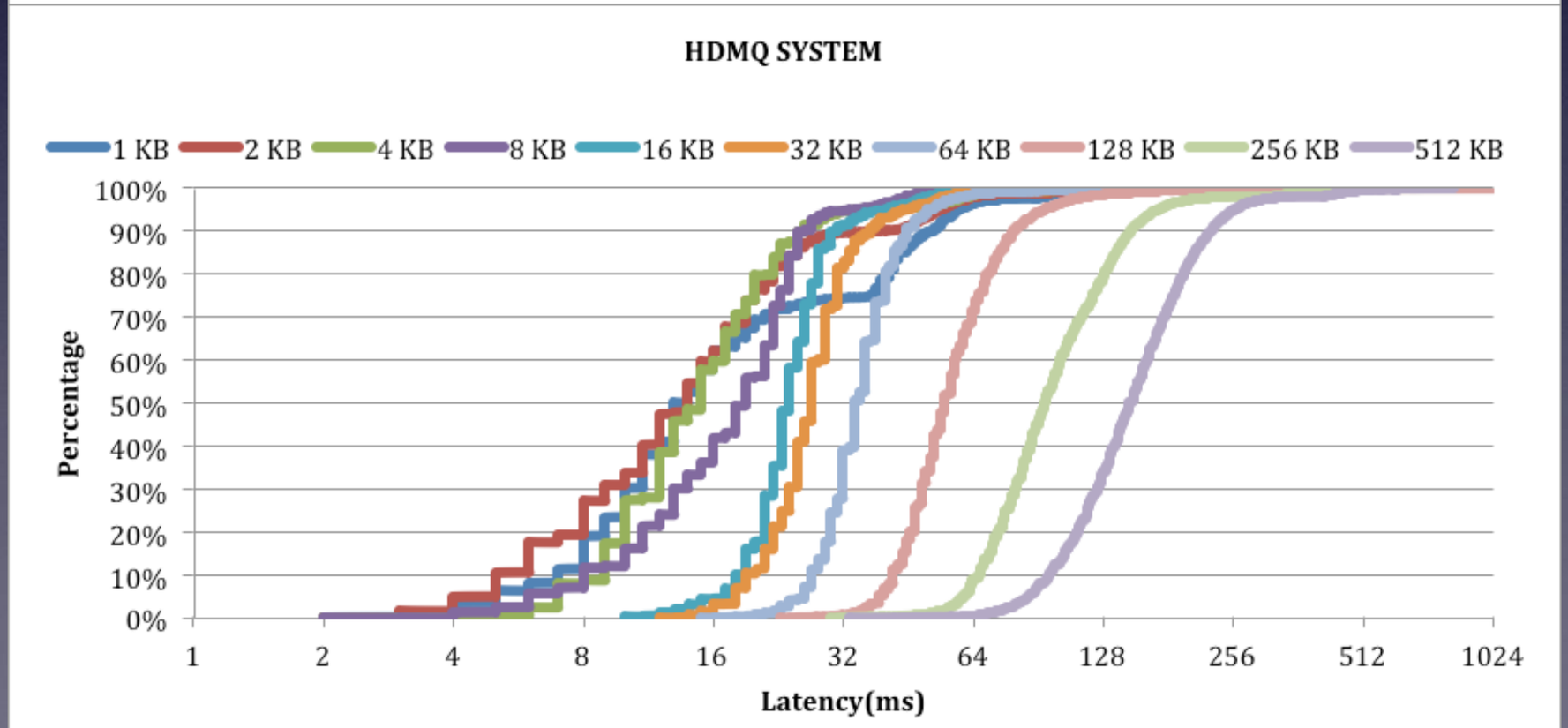
- 20 client
- M1.xlarge
- 1 Million Message

We found that on an average 23.73 % of total messages are found in SQS as repeated messages

SQS VS HDMQ



- 20 client M1.xlarge
- 1 Million Message

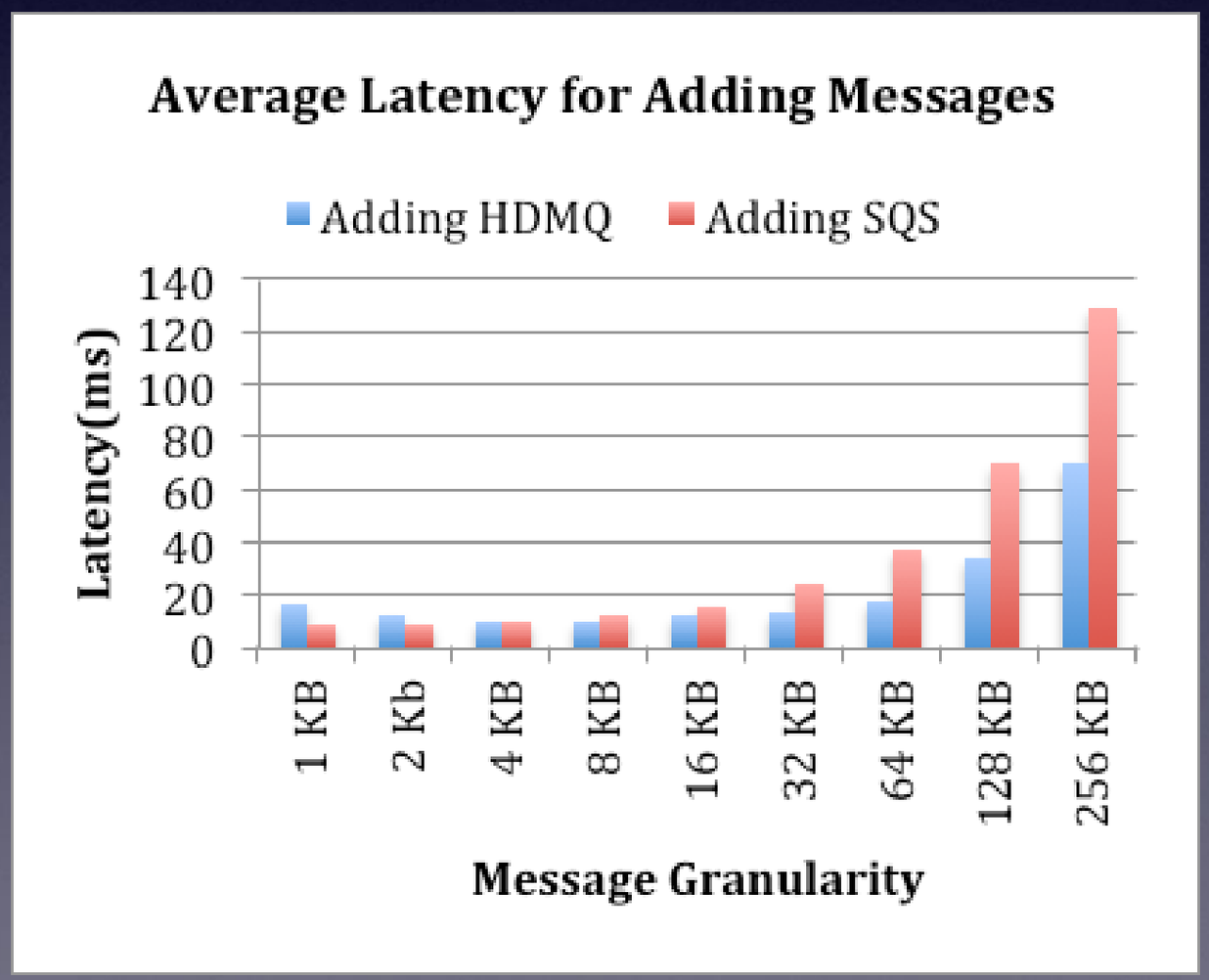


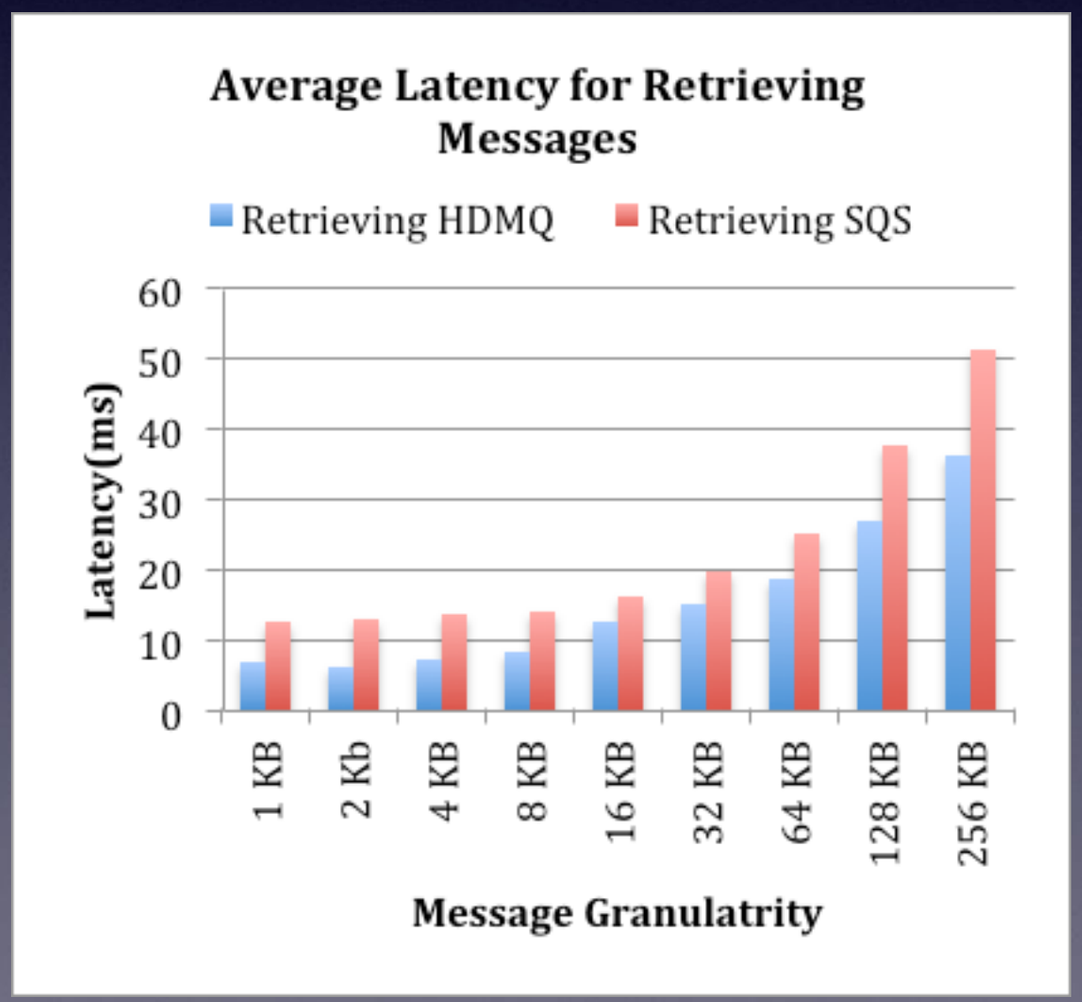
- 20 client m2.4xlarge
- 1 Million Message
- 10 Front-end nodes(m1.xlarge)
- Elastic Load Balancer
- 20 M3 Double Extra Large Instance
- m3.xlarge Load Balancer

Latency

- 20 client M1.xlarge
- 1 Million Message

- 20 client m2.4xlarge
- 1 Million Message
- 10 Front-end nodes
- 20 M3 Double Extra Large Instance
- m3.xlarge Load Balancer

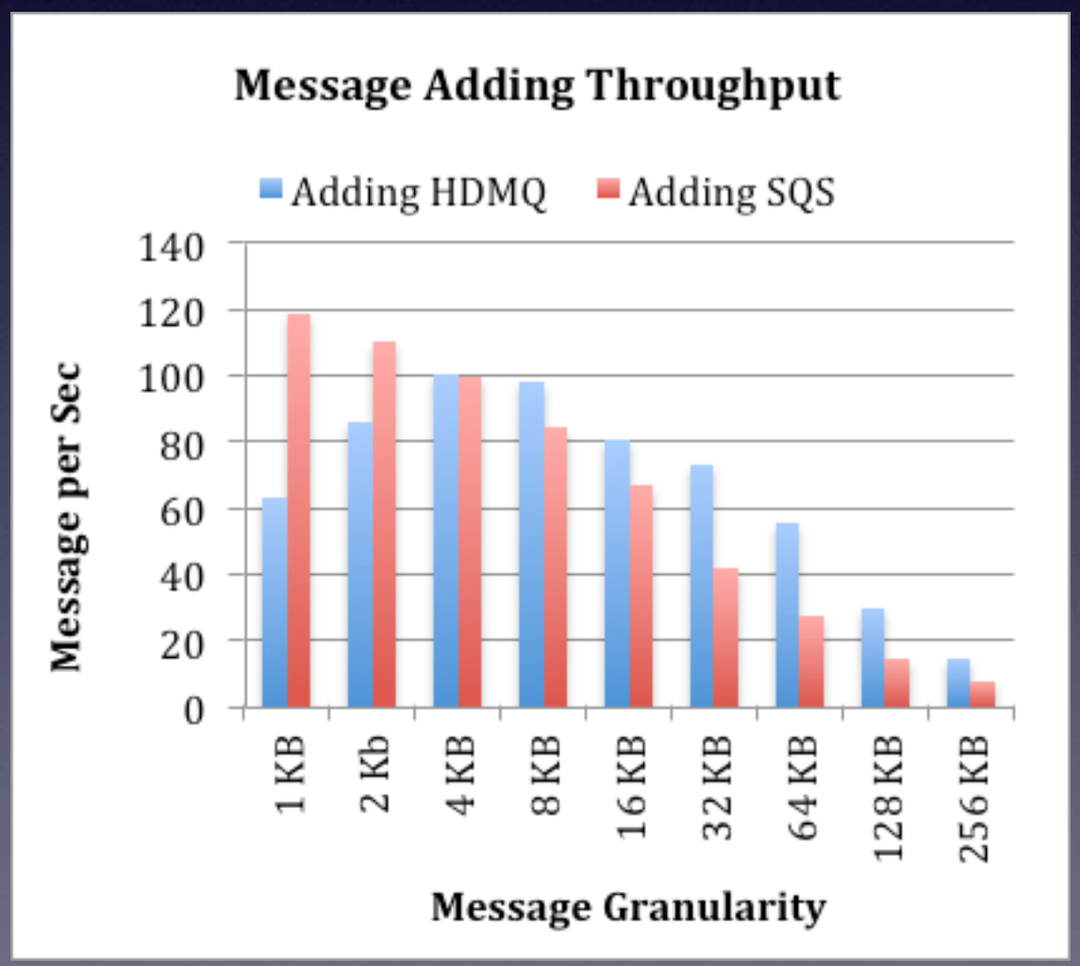


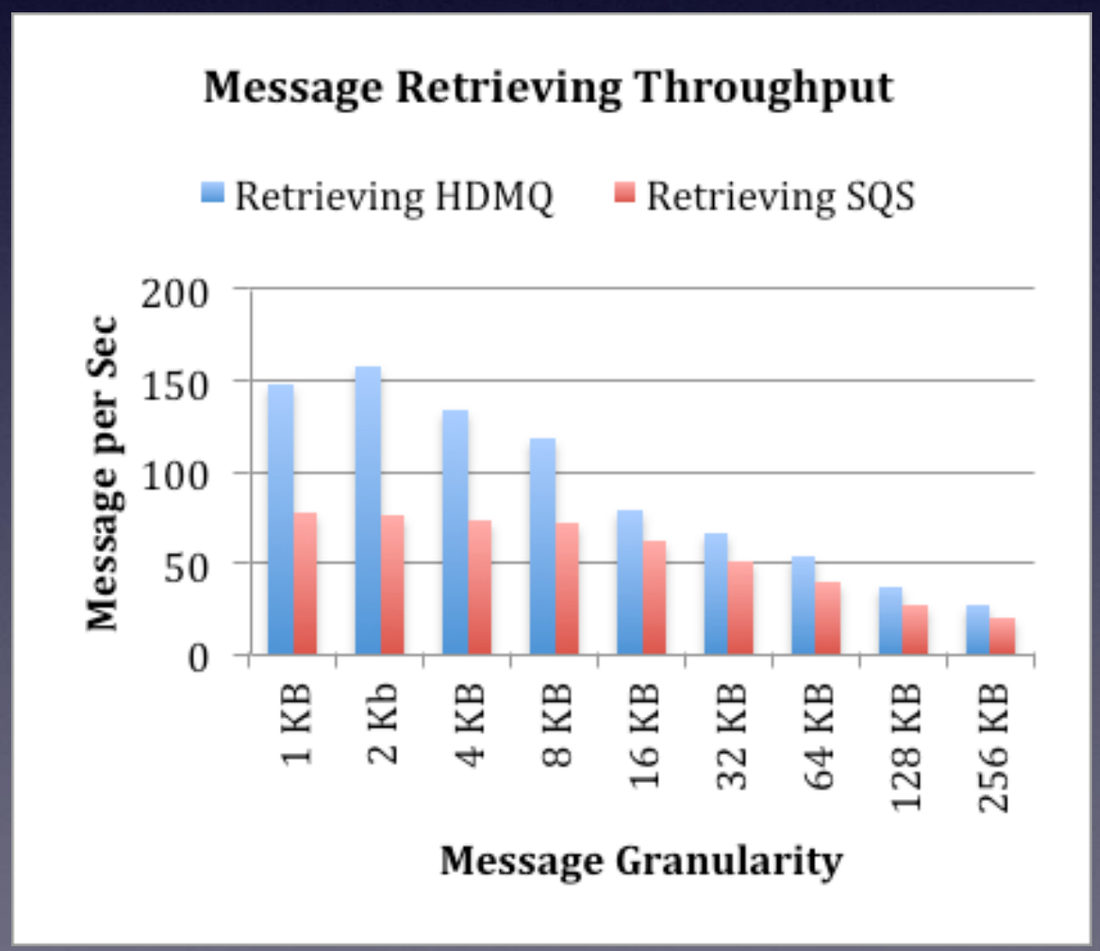


Throughput

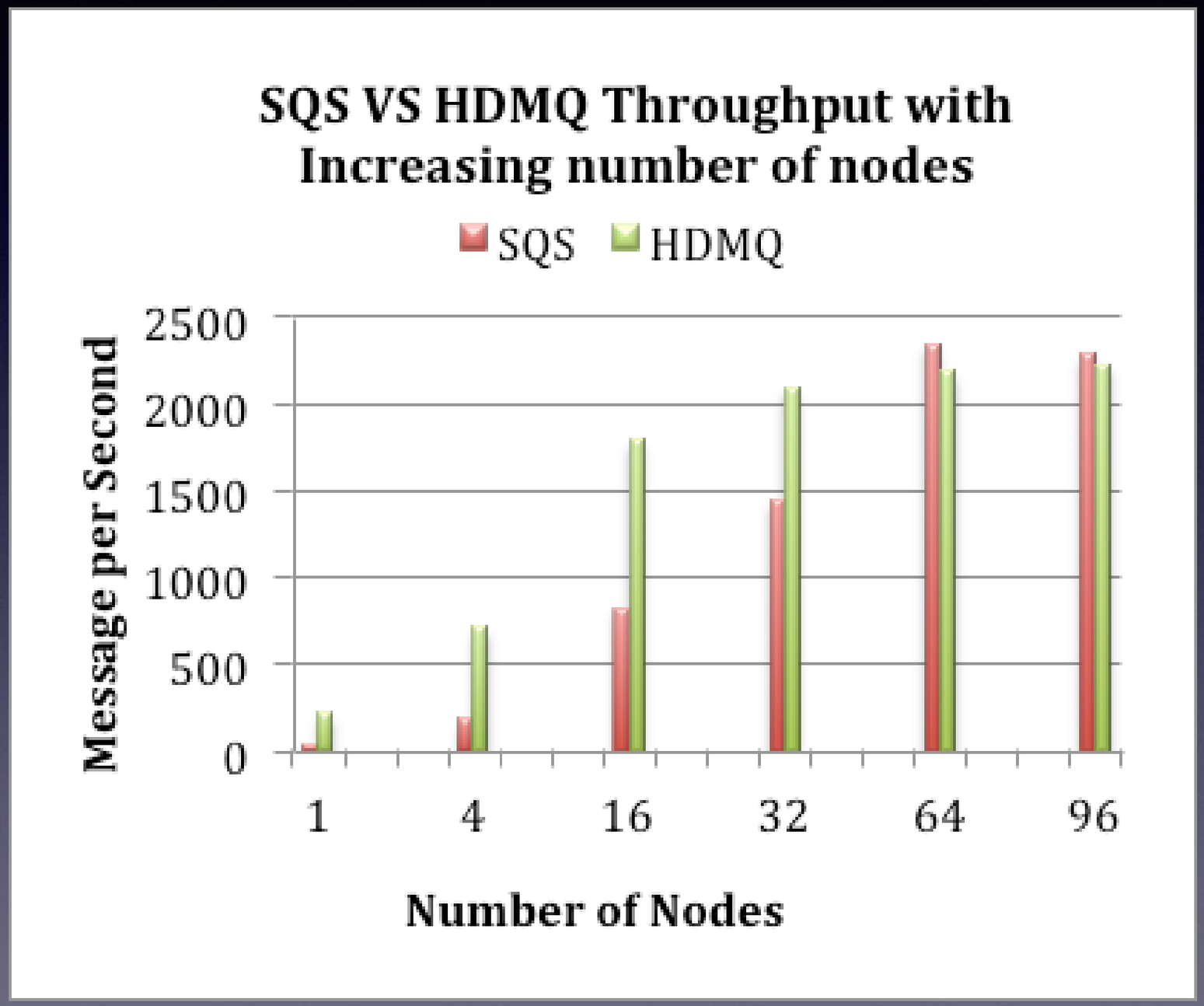
- 20 client M1.xlarge
- 1 Million Message

- 20 client m2.4xlarge
- 1 Million Message
- 10 Front-end nodes
- 20 M3 Double Extra Large Instance
- m3.xlarge Load Balancer





Increasing number of Nodes

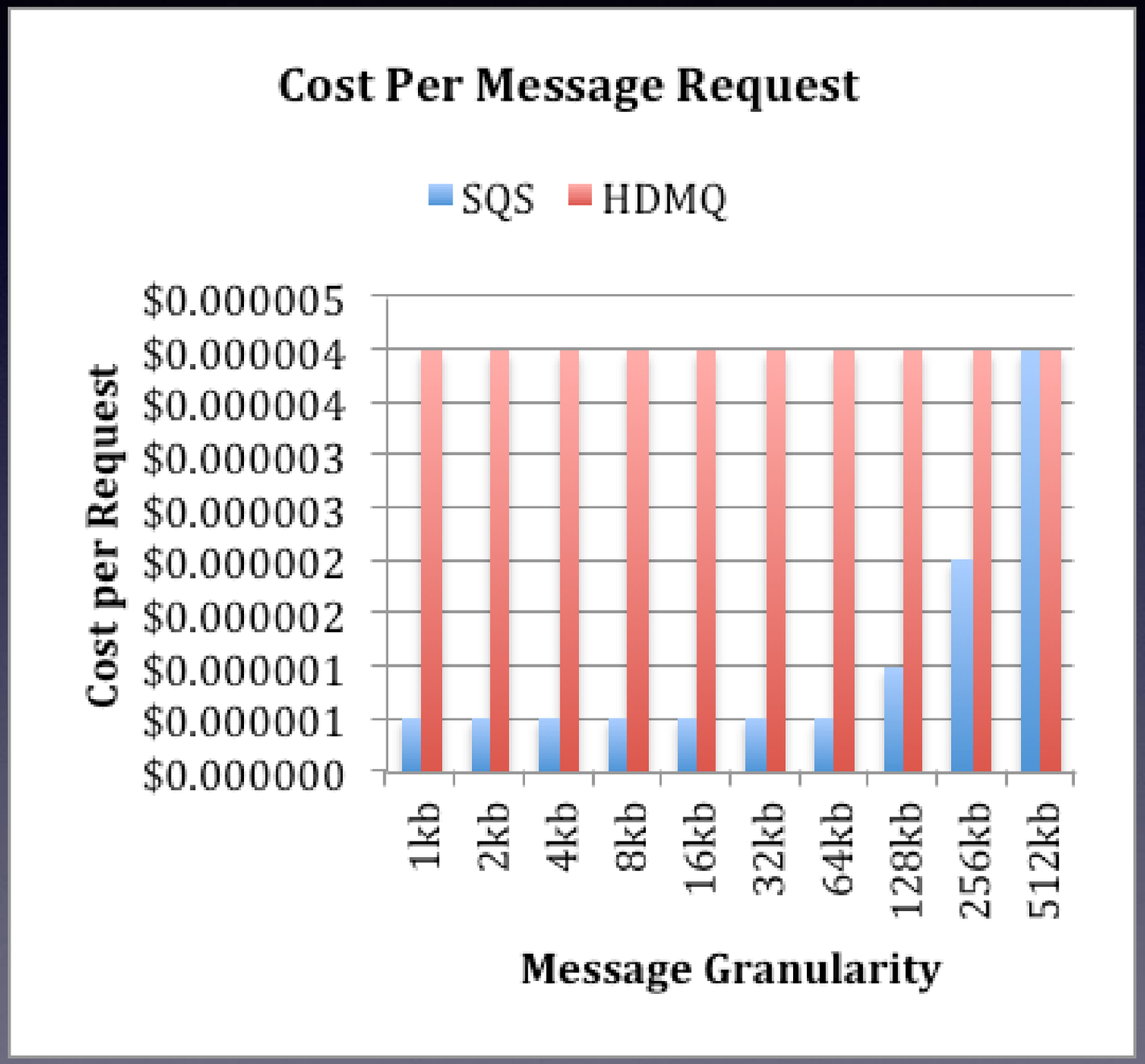


32 KB message

- 20 client M1.xlarge
- 1 Million Message

- 20 client m2.4xlarge
- 1 Million Message
- 10 Front-end nodes
- 20 M3 Double Extra Large Instance
- m3.xlarge Load Balancer

Cost Per Request



Conclusion

- The HDMQ adding and retrieving latency is lower than the SQS latency.
- We also observed that Throughput for adding in HDMQ is little lower than the SQS system but if we implement the router level load balancer then the throughput would be much higher than SQS.
- We also observed that the average receiving throughput of HDMQ is much more higher than the average throughput of Amazon SQS.
- If we combine the average throughput of adding and receiving, HDMQ would be much more faster than Amazon SQS.
- We also observed that the throughput of HDMQ with increasing number of nodes is also higher than the Amazon SQS.
- We also conclude that the cost for implementing the system right now is little higher as we are implementing the system on top of Amazon Web Services using EC2 instance, but if we have message aware queue and our own private cloud, we can reduce that price by a great amount.
- We found HDMQ to outperform SQS by up to 10-20% in throughput, 100% in latency, and 50% less in costs.

Future Work

- We will be implementing our own load balancer in future so that our framework is completely independent from Amazon Web Services.
- We will also implement queue-monitoring service. We will also try to increase the Adding message throughput by implementing local router level load balancer.
- We will also provide more throughputs still maintaining the reliability by providing asynchronous replication.
- We also want to design the framework message aware so that it can scale according to the incoming message size. This will not only reduce the cost of operating per request but will also help us to be aware about the right storage node for the incoming messages size so that the system can scale itself.
- We will also try to configure the number of replicas for the message nodes. As of right now its by default 1 if you start the system with replication.

References

[1] Dongfang Zhao and Ioan Raicu, Supporting Large Scale Data-Intensive Computing with the FusionFS Distributed File System, 2013

[2] Amazon SQS, [online] 2013, http://aws.amazon.com/sqs/

[3] Hedwig, [online] 2013, http://wiki.apache.org/hadoop/HedWig

[4] RabbitMQ in action: distributed messaging for everyone, Videla, Alvaro; Williams, Jason J W, Shelter Island NY : Manning, 2012. - 1288 p.

[5] Jay Kreps, Neha Narkhede and Jun Rao, Kafka: a Distributed Messaging System for Log Processing, 2011.

[6] Snyder, Bruce, Dejan Bosanac, and Rob Davies. "Introduction to Apache ActiveMQ." Active MQ in Action: 6-16.

[7] Couch-RQS, [online] 2013, https://code.google.com/p/couch-rqs/

[8] Iman Sadooghi and Ioan Raicu, CloudKon: a Cloud enabled Distributed tasK executiON framework, 2013

[9] Apache Hedwig [online] 2013, http://zookeeper.apache.org/bookkeeper/docs/r4.0.0/hedwigUser.html

Thank You

?!