

# Message Queue Evaluation Notes

From Second Life Wiki

Second Life Wiki > Message Queue Evaluation Notes

## Contents

- 1 Summary and Overview
  - 1.1 Criteria
    - 1.1.1 Group Chat Use Case
- 2 Scaling
- 3 Currently Available Software
  - 3.1 AMQ Protocol
    - 3.1.1 RabbitMQ
    - 3.1.2 Apache QPID/Red Hat MRG
    - 3.1.3 OpenAMQ
  - 3.2 XMPP
    - 3.2.1 ejabberd
      - 3.2.1.1 AMP Extension
      - 3.2.1.2 PubSub Extension
  - 3.3 Stomp
    - 3.3.1 ActiveMQ
    - 3.3.2 Sprinkle
  - 3.4 JMS
    - 3.4.1 OpenMQ
  - 3.5 Other
    - 3.5.1 IRC
    - 3.5.2 PSYC
    - 3.5.3 Gearman
    - 3.5.4 Amazon SQS
    - 3.5.5 beanstalkd
    - 3.5.6 Peafowl
    - 3.5.7 Starling
    - 3.5.8 SMTP
    - 3.5.9 Zero MQ
- 4 Reading/Related Links

## Summary and Overview

One of the infrastructure tools that we've identified for the future internal architecture of Second Life is messaging. Message queuing systems allow systems that send messages to not have to worry about how they will be delivered, and allow consumers of messages to gather whichever ones interest them, at their own pace.

Ideally we'd have a completely scaleable system that clients could treat as singular black box. It would act as a well-known cluster to which senders or receivers of messages could connect, and be able to communicate asynchronously to or from anywhere else on the grid. Unfortunately it seems as though this dream, like so many others, is unattainable by any currently available software. We investigated around 15 open source systems that were explicitly designed for message queueing and found that none of them achieved this ideal.

Our use cases mostly involve very large numbers of queues; the smallest number we're even considering is double the number of concurrent users. Our largest use case would be of the same order of magnitude as registered users. This means that we have to plan for millions, and probably tens or hundreds of millions, of message queues, since we want whatever system we choose to last us until then. Pretty much all of the message queue systems we investigated are intended to maximize message throughput rather than number of consumers. The clustering that they implement mainly serves as additional horsepower to deliver more message throughput. In particular the clustering we've seen replicates all state to every machine in the cluster, meaning that the cluster cannot add queues beyond the capacity of an individual node. In order to have a solution that scales in terms of number of queues, each node will have to be able to contain a subset of the global state -- it actually seems as though such a message system would want to be coupled with a distributed storage system.

In any case, given that we expect that we'd have to develop our own queue scaling solution that involves partitioning, which may or may not be a task we are interested in taking on, the strongest candidates are RabbitMQ and Apache QPID. Both are mature products that support AMQP (though they support different versions). Both have strong vendor support, and both have good single-host performance numbers. There is some more investigation to be done -- we'd like to know their true maximum capacity when clustered, to evaluate whether it's worth clustering at all, given that we have to implement partitioning ourselves anyway.

We're unfortunately pretty far from having closed the case on which technology to choose, or even if we can use any of these at all. Investigation will continue!

## Criteria

### Questions

1. Support for consuming messages in our two main languages, Python and C++
2. Support for producing them in all of our main languages (or, e.g., an HTTP gateway)
3. What's the raw message rate that a single server can handle at different message sizes (e.g. 1 byte, 128 bytes, 1K, 128K)?
4. What are the latency figures for these message rates?
5. What are the server's clustering characteristics?
  - HA/failover

- scalability for message quantity
  - scalability for number of clients/queues
6. What about cross-colo traffic? Does it do SSL, does it need tunneling, is it designed to deal with interrupted communications?
  7. How do the health of the protocol standard, the software, and the community look?
  8. Can we engage people on support and development contracts if necessary?
  9. Is it possible to set up persistent queues, where undelivered messages will survive server restarts? For some interesting perspectives, see "What Matters in an Asynchronous Job Queue" (<http://www.rockstarprogrammer.org/post/2008/oct/04/what-matters-asynchronous-job-queue/>) and this discussion on persistence and failure scenarios ([http://groups.google.com/group/beanstalk-talk/browse\\_thread/thread/e8d1a6ea80f0d57a](http://groups.google.com/group/beanstalk-talk/browse_thread/thread/e8d1a6ea80f0d57a)).

The major use cases that we believe could be implemented with message queues are these:

- IM
- blue box notifications in the viewer
- group notices
- group chat
- group votes
- presence notifications
- anything else that currently is an ImprovedInstantMessage (<http://wiki.secondlife.com/wiki/ImprovedInstantMessage>)
- friendship
- details of registration such as initial inventory
- requests to regenerate map tiles

Many other uses that we haven't put up there would arise when developing various applications. Note that we **do not** intend to expose the message queue system directly to the viewer; we would use it instead for behind-the-scenes implementation of the above use cases. Discussing the protocol by which the viewer sends and receives messages with the server is far outside the scope of this research.

## Group Chat Use Case

The primary use case that we examined was that of group chat. This is the most obvious use that we'd put message queues to, and it's also one of the only use cases up there that is a current pain point for Residents. Here are some numbers describing group chat, which were captured at 11am one day:

- online residents: ~68,000
- number of group chats: 131953
- number of memberships: 629637
- avg number of groups/resident: 9.3
- average group size: 4.8
- most online members in a group: 785
- stdev group size: 13.1
- group churn/second: 23.2
- membership churn/second: 260
- messages sent/second: 8.6

**Please note:** this data was measured some time ago, and every value you see has increased by today, and will continue to grow for the foreseeable future. It was not enough for us to pick a technology that could just barely handle the current load; we have to pessimistically assume that whatever we pick will be used for many years hence, and will therefore have to deal with at least an order of magnitude more of everything.

Our minimum hurdle for a platform is 4x the above values, and ideally could horizontally scale nearly infinitely across the most important axes to us: number of group chats, number of memberships, and messages/second.

Group churn represents the number of groups added or removed from the system every second. In an AMQP-based system a "group" would be added when someone binds the first queue to the a key representing the group's id, and the group would be dropped when the last binding to the key is removed. In any routing system of this nature, there will be two data structures that are the inverse of each other: one is queueName->keys table and the other is the keys->queueNames table. Group churn would primarily affect the keys->queueNames table.

Membership churn comes from agents joining and leaving groups, and is indirectly related to group churn. There is vastly more membership churn because each agent is in roughly 9 groups -- one agent logs in for 15 seconds and then logs out, that's 18 points of "churn". In an AMQP-based system this churn would be visible as bindings being added and dropped, which would thus affect the queueName->keys table.

Both types of churn are substantially higher than messaging rate. Even if we successfully implement a feature that will allow residents to opt out of joining group chat on login, I'd guess that we'd see at most a 75% reduction in churn as a result, which would still leave membership churn nearly an order of magnitude larger than messages sent.

The group size metrics are interesting because it implies that most groups are pretty small. The downside is we have roughly 2 unique groups for every concurrent resident, so our group system has to scale along that axis very well.

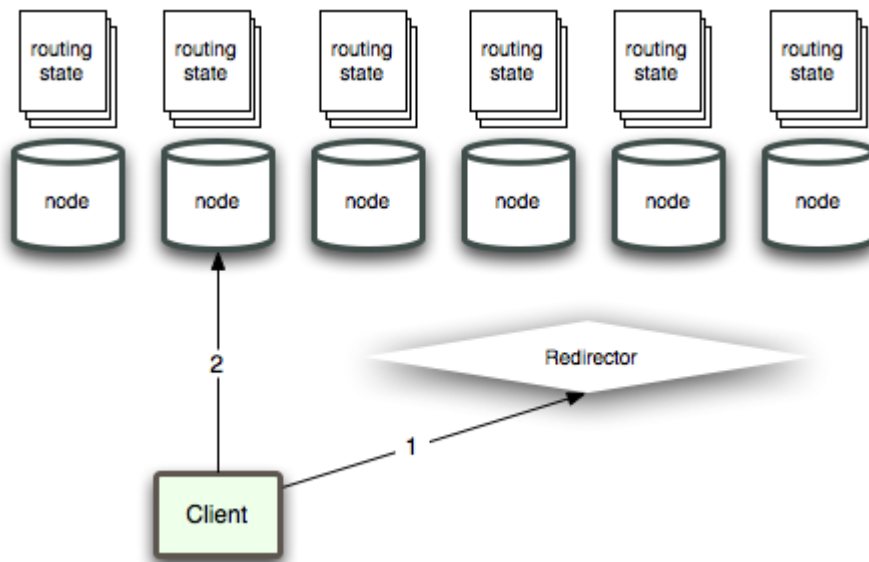
Overall, these numbers are a real whack upside the head because they reveal that for our particular use case, we worry far more about number of queues, and ability to handle high churn, than we do about message delivery rate. Most of the benchmarks regarding the software we evaluated optimize for message rate rather than churn or quantity of queues.

Note that Twitter has a similar set of requirements (<http://gojko.net/2009/03/16/qcon-london-2009-upgrading-twitter-without-service-disruptions/>) (complete with startlingly low message rate), though they don't have such tight latency requirements as we do.

## Scaling

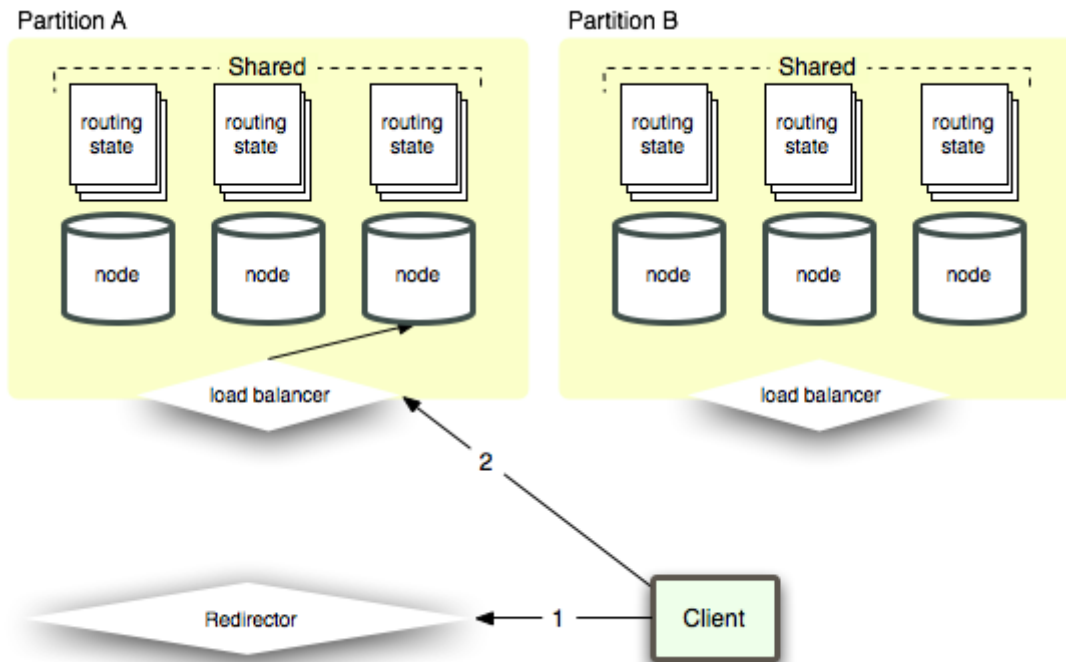
Given that we haven't found any system that provides seamless scaling along the axis of queues, we have come up with some designs for how the system could be partitioned. Any partition scheme is a bit of a downer because then the clients have to know about the partitioning.

The best way to partition is probably to partition on routing key. When a client wants to receive messages for a routing key, or when a sender wants to send to a routing key, the first thing it does is ask a redirector which slice to talk to. The system would look like:



This system has the property that each client will be talking to as many slices as routing keys of interest (proportional, anyhow). This may be acceptable for an application where there's only a finite number of routing keys that any particular client may be interested in (such as group chat, where there's an upper limit on the number of groups a resident may be in).

We could cut down on the sheer quantity of connections by using clustering to reduce the number of slices. The messaging technologies we looked at support clustering in a way that would allow us to create clusters of 2-3 nodes that have roughly 1.5-2 times the quantity of queues as a single node. This would be a more complex system than not using clustering at all, but it would reduce the number of slices, and thus connections, by a factor of 1.5-2, which may be important.



The clustered partition scheme isn't fundamentally different than the unclustered partition scheme, it's just a constant-factor optimization. However, if the cluster added high-availability properties (such as being able to continue running even when one node fails), then it would be worth it.

## Currently Available Software

### AMQ Protocol

AMQP is a developing standard that is aimed pretty squarely at our use cases: high-performance, flexible, message routing and delivery. The standard is being iterated upon and is not finalized yet. This is a risk factor because it's a moving target and all current editions are mutually incompatible -- the 0-10 edition is nontrivially different from 0-8, and I've read some rumors that the next edition removes or changes some of the core concepts in a dramatic way.

We believe that AMQP is the appropriate standard for our use cases; therefore we investigated the AMQP servers much more heavily. Here's a decent example (<http://blogs.digitar.com/jjww/2009/01/rabbits-and-warrens/>) of how to apply AMQP to a particular use case (with emphasis on Rabbit).

- Twisted AMQP (<https://launchpad.net/txamqp>), a 0-8 and 0-9 Twisted Python client
- Build your Own AMQP Client (<http://hopper.squarespace.com/blog/2008/6/21/build-your-own-amqp-client.html>)

### RabbitMQ

Rabbit Home (<http://www.rabbitmq.com/>)

RabbitMQ is already installed in our image and is currently being deployed by the data warehousing team (Ivan in particular) to handle syslog message delivery.

1. Support for consuming messages in our two main languages, Python and C++?
  - Yes, many python clients and carrot-rabbit (<http://code.google.com/p/carrot-rabbit/>)
2. Support for producing them in all of our main languages (or e.g. an HTTP gateway)
  - Yes, they have HTTP and Stomp gateways.
3. What's the raw message rate that a single server can handle at different message sizes (e.g. 1 byte, 128 bytes, 1K, 128K)?
  - Not investigated because widespread reports indicate that this number is very high; this will not be a bottleneck.
4. What are the latency figures for these message rates?
  - Latency is in tens of microseconds.
5. What are the server's clustering characteristics?
  - HA/failover
    - Weak. If clustered, the cluster behaves poorly when nodes go away, until their data is restored; see [1] (<http://lists.rabbitmq.com/pipermail/rabbitmq-discuss/2009-February/003338.html>). The developers suggested setting up two queues on two hosts and doing client-side duplicate removal, but this seems like we'd be doing work that the message queue system should be doing for us.
  - scalability for message quantity
    - Untested but theoretically clustering improves this.
  - scalability for number of clients/queues
    - Not supported. Currently a single Rabbit node will support up to 250,000 queues before consuming all available memory. If clustered with a second host, the second host will have 800 MB of its memory consumed as well even though it's not doing any work. This is because all hosts in a cluster contain a complete copy of the routing tables. The data structures required for queues themselves are stored on the first host, and consume the other 1200 MB available. This implies that if queues were evenly added to both hosts, we could probably get up to around 400,000 queues total. We'd like to test that hypothesis.
6. What about cross-colo traffic?
  - Not if hosts in a cluster are cross-colo from each other. There's a plugin called shovel (<http://hopper.squarespace.com/blog/2008/6/22/introducing-shovel-an-amqp-relay.html>) that is explicitly designed for cross-colo situations, though. However, if clustering doesn't support the number of queues we want, there's little reason to worry about clusters that span colos. The wire protocol is believed to be robust against flaky connections; it's hard to test that though.
7. Does it do SSL, does it need tunneling, is it designed to deal with interrupted communications?
  - Supports SSL (<http://www.rabbitmq.com/api-guide.html#ssl>), however this appears to be both outside of the spec and a bit of a hack. AMQP as a whole doesn't yet have an ssl story, so we'd have to go outside the spec to achieve security.
8. How do the health of the protocol standard, the software, and the community look?
  - Strong. Rabbit in particular has a relatively strong community -- ~500 messages per month on mailing list and key developers respond frequently.
9. Can we engage people on support and development contracts if necessary?
  - Yes, we already have contacts with them and have already gotten enhancements.
10. Is it possible to set up persistent queues, where undelivered messages will survive server restarts?
  - Yes. These are not replicated.

#### Notes:

- ~5000 lines of code (i.e. not many places to hide bugs)
- Queues can fill up available memory if clients can't consume non-immediate messages quickly enough (immediate messages get discarded if a client can't consume them).
- RabbitMQ Performance Testing and Troubles (<http://forum.trapexit.org/maillinglists/viewtopic.php?p=42890&sid=cb78a92ab50adefbf60e3770c0ba7e9b>)
- Even in clustered configuration, queues only reside on a single machine.
- Blog oriented towards Rabbit (<http://hopper.squarespace.com/>)
- <http://everburning.com/news/tag/amqp/> Ruby-oriented but examples are pretty clear
- <http://www.somic.org/d/samovskiy-amqp-rabbitmq-cohesiveft.pdf>
- <http://somic.org/2008/11/11/using-rabbitmq-beyond-queueing/>
- <http://www.lshift.net/blog/category/lshift-sw/rabbitmq/>
- In our tests, if you create a bunch of auto-delete queues (~100), the rabbit server becomes unresponsive for 2 minutes after you remove the clients (presumably because the server starts deleting the queues when you remove the clients). We generally have to restart the server. This may be problematic in production; we should generate a test case that repros this.
- py-amqplib requires Python >2.4
- Collected data: [http://spreadsheets.google.com/a/lindenlab.com/ccc?key=p0\\_PzEgG1YZeKbg\\_y4n1nFg](http://spreadsheets.google.com/a/lindenlab.com/ccc?key=p0_PzEgG1YZeKbg_y4n1nFg)
- Restarting individual hosts in a cluster is seamless -- it totally comes back together without any extra work, yay.
- Rabbit (and mnesia) is very sensitive to DHCP/DNS failures. We had a DHCP failure on our host, and it corrupted everything, leading to rogue beam processes and messed up mnesia files. Had to nuke everything.
- HA by creating multiple queues on multiple hosts: <http://www.nabble.com/Re:-Q-replication-anxiety-p21953124.html>
- Discussing huge numbers of queues: <http://lists.rabbitmq.com/pipermail/rabbitmq-discuss/2009-February/003431.html>
- Discussion about what happens when one node in a cluster goes away, and how it's less than ideal: <http://lists.rabbitmq.com/pipermail/rabbitmq-discuss/2009-February/003338.html>

## Apache QPID/Red Hat MRG

<http://cwiki.apache.org/confluence/display/qpid/Index> <http://www.redhat.com/mrg/>

QPID is sort of a compilation project from Apache. They have a pile of clients, and two brokers that each support different versions of the AMQP standard. The C++ broker appears to be where the good stuff is at. Installing it was not difficult -- it only required a few additional Boost libraries to be installed.

1. Support for consuming messages in our two main languages, Python and C++
  - Yes, comes bundled with clients in both.
2. Support for producing them in all of our main languages (or e.g. an HTTP gateway)
  - No, only python and C++.
3. What's the raw message rate that a single server can handle at different message sizes (e.g. 1 byte, 128 bytes, 1K, 128K)?
  - Untested; again, not likely to be a bottleneck.
4. What are the latency figures for these message rates?
  - Untested, it comes with a perftest tool that we should run.
5. What are the server's clustering characteristics?
  - HA/failover
    - Queue Replication (<http://cwiki.apache.org/confluence/display/qpid/queue+state+replication>) could be used in conjunction with client failover (<http://qpid.apache.org/connection-url-format.html>) to implement HA queues. Untested.
  - scalability for message quantity
    - Unknown.
  - scalability for number of clients/queues
    - The C++ broker tops out at around 500,000 queues per host. We haven't yet determined whether this number can be increased by clustering/federation (<http://cwiki.apache.org/confluence/display/qpid/Using+Broker+Federation>).
6. What about cross-colo traffic? Does it do SSL, does it need tunneling, is it designed to deal with interrupted communications?
  - Same as rabbit; these questions are properties of AMQP.
7. How do the health of the protocol standard, the software, and the community look?
  - Relatively small community, though active. 721 messages to qpid-dev list in December 2008, though 90% seem to be JIRA spam; only 71 to qpid-users.
8. Can we engage people on support and development contracts if necessary?
  - Red Hat supports a version called MRG -- unknown whether we could get them to help us use it on Debian. ;-)
9. Is it possible to set up persistent queues, where undelivered messages will survive server restarts?
  - Yes.

#### Notes:

- MRG is based on QPID, per this page ([http://www.redhat.com/docs/en-US/Red\\_Hat\\_Enterprise\\_MRG/1.0/html/Messaging\\_Installation\\_Guide/sect-Messaging\\_Installation\\_Guide-Installing\\_RHM-Installing\\_RHM\\_on\\_RHEL5.html](http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_MRG/1.0/html/Messaging_Installation_Guide/sect-Messaging_Installation_Guide-Installing_RHM-Installing_RHM_on_RHEL5.html)).
- Client libs: C++, Java, Ruby, Python, C#. Only the C++ client appears to have any documentation.
- <http://cwiki.apache.org/confluence/display/qpid/Documentation> <-- this link was surprisingly hard to find
- Python client is annoying to use. Before even getting started you have to set an environment variable pointing at the spec (`export AMQP_SPEC=/tmp/qpid/qpid-M4/specs/amqp.0-10.xml`) and modify the spec file itself to point at the DTD because relative paths don't work for some reason. The code is written poorly; no comments, run-on functions, uses long-deprecated things like `raise` "string" everywhere. Also, it's slow as hell starting up and at runtime. Finding another 0-10 compatible python client would be important.

## OpenAMQ

<http://www.openamq.org/>

! This product has been eliminated from the race

- <http://www.openamq.org/doc:user-3-advanced>
- Supports HA via primary/secondary failover, EXCEPT "What we do not attempt to do includes: ... The handling of persistent messages or transactions in any way.....Replication of exchanges, queues, bindings, or messages between servers. All server-side state must be recreated by applications when they fail over."
- HA support is mutually exclusive from federation support.
- Supports federation; we'd want to use the "locator" federation type.
- Seems immature; few docs, not a strong community -- however, it's actually pretty old.
- Implements a subset of AMQP 0.9
- <http://amqp.wdfiles.com/local--files/deleted:library:cluster/clustering.pdf>
- Java centric, no bindings for other languages we love to use.

## XMPP

### ejabberd

<http://www.ejabberd.im/>

! This product has NOT been eliminated, but we did not evaluate it for want of time.

Clients: <http://www.jabber.org/web/Clients>

- Widely deployed and super-popular.
- XMPP seems to not be the best fit for generic applications -- it seems that it could at best solve the group-chat and person-to-person IM cases, if at all.

## AMP Extension

<http://xmpp.org/extensions/xep-0079.html> Message queue extension, available for ejabberd via this patch (<https://support.process-one.net/browse/EJAB-449>).

This extension adds:

- Reliable data transport -- the sender requires notification (positive and/or negative) of message delivery.
- Time-sensitive messages -- the message is valid only until a certain date and time.
- Transient messages -- the message should not be stored offline for later delivery.

The AMP extension is immature and has no client support -- we'd be going it ourselves if we wanted to use that. We might decide that we don't need the AMP extension's semantics.

## PubSub Extension

[http://www.ejabberd.im/mod\\_pubsub-usage](http://www.ejabberd.im/mod_pubsub-usage)

<http://ppolv.wordpress.com/2007/11/28/managing-a-jabber-pubsub-service/>

Doesn't seem to be client support for this, but it's probably better than AMP extension client support and I haven't looked especially hard yet.

## Stomp

Products that support primarily the Stomp protocol.

<http://stomp.codehaus.org/Protocol>

Stomp is a message passing scheme that is tremendously simple. It kinda looks like HTTP, but it's actually even simpler. The simplicity is both a strength and a weakness.


- Human-readable
- Requires null-termination of message bodies; the spec implies that it's OK to include nulls in the message body if a content-length header is included, but if the header is not supplied nulls are not acceptable. This article (<http://cometdaily.com/2008/10/08/scalable-real-time-web-architecture-part-1-stomp-comet-and-message-queues/>) and subsequent discussion is interesting.
- Underspecified header syntax means implementations might differ w.r.t. all-dashes, spaces between : and header body, capitalization, etc.

There are stomp clients written in nearly every language; since the protocol is so simple it seems as if people write new clients whenever they encounter even the slightest problem with someone else's client.

- stomp.py -- nicely written, but seems to be just barely maintained by a guy who doesn't really use it any more. Its error reporting sucks (prints java exceptions to stdout, doesn't give any feedback to python program). But it works!

## ActiveMQ

<http://activemq.apache.org/>


 This product has been eliminated from the race

Related Jira : <jira>DEV-19958</jira>

- HA via master/slave failover built-in: <http://activemq.apache.org/masterslave.html> It's not great, but it's better than "you lost all your shit!"
- Does not support AMQP (how did we miss this earlier?)
- Clustering built-in with autodiscovery. It's very nice to start up activemq on two machines and see them find each other automatically.
- Broker fails after opening 700 queues on a node due to too many open files -- apparently it has a tempfile for each queue, and an open file for each JAR (of which there are 124!), and 60 miscellaneous filehandles. The number of open files does not decrease when the client quits. Some googling implies that there are numerous bugs in ActiveMQ relating to leaked filehandles.
- Even if you cluster two nodes together, each created subscription opens a new filehandle on both nodes in the cluster! This means that the total number of queues creatable in a clustered system is very small.

## Sprinkle

<http://www.thuswise.org/sprinkle/index.html>

 This product has been eliminated from the race

- Consumes 100% of CPU when idle.
- Requires an external process to create a new directory for each queue -- no way to create queues by talking to the server (unless you hook up a demon listening to a special meta-queue for messages to create new queues...an approach that is appealing in its purity, but...no)

## JMS

JMS is a Java API for asynchronous messaging. In a charming bit of java-centrism it specifies the API that a java program sees, but not a wire protocol or anything like that; I can only assume that each client is intended to develop its own custom wire protocol for talking to the broker, but there isn't much information about this out there. Some of the clients that support Stomp or AMQP also support some subset of JMS, and

presumably the JMS API simply uses Stomp or AMQP as the wire protocol. The development of the wire protocols have been influenced somewhat by the needs of JMS.

## OpenMQ

<https://mq.dev.java.net/about.html>

! This product has been eliminated from the race

- Very java-specific.
- This is the free version of Java System Message Queue ([http://www.sun.com/software/products/message\\_queue/index.xml](http://www.sun.com/software/products/message_queue/index.xml))
- JMS only.

## Other

Products that have their own protocol. These are generally less interesting to us because they are already at a disadvantage relative to implementations of a particular protocol in terms of community size, possibility of paid support, and being able to replace them with a different implementation if the need arises. However, we still take these into consideration in the hopes of finding a diamond in the rough.

## IRC

- Top 100 IRC Rooms (<http://searchirc.com/top100.php>). Looks like the largest tops out near 3400 members.
- Carlo Wood's notes on IRC (<http://www.xs4all.nl/~carlo17/irc/run-irc.htm>). Of note is the fact that IRC does not guarantee message order.
- Interesting and discouraging overview of chat protocol scalability (<http://irc.pages.de/research.html>)
- Not designed for anything other than group chat or P2P IM; we'd have to come up with something else for our other use cases.
- No offline storage of messages.
- Doesn't have a great character set story ([http://www.irchelp.org/irchelp/rfc/chapter2.html#c2\\_2](http://www.irchelp.org/irchelp/rfc/chapter2.html#c2_2)); content would have to be encoded.

## PSYC

<http://about.psyc.eu/>

- Just gauging off the roadmap (<http://about.psyc.eu/Roadmap>), doesn't seem to have "arrived".

## Gearman

<http://www.danga.com/gearman/>

! This product has been eliminated from the race

- "Gearman is a system to farm out work to other machines, dispatching function calls to machines that are better suited to do work, to do work in parallel, to load balance lots of function calls, or to call functions between languages."
- Not really what we're looking for.

## Amazon SQS

<http://aws.amazon.com/sqs/>

! This product has been eliminated from the race

- Performance 40x slower than rabbitmq (<http://notes.variogr.am/post/67710296/replacing-amazon-sqs-with-something-faster-and-cheaper>)
- External hosting raises troubling questions about security/privacy, and the ability of our operations team to handle failure cases (i.e. if SQS goes down we are SOL unless we separately maintain our own compatible farm of servers, in which case we gain little in operational savings)
- External hosting adds non-negligible latency

## beanstalkd

<http://xph.us/software/beanstalkd/>

! This product has been eliminated from the race.

- Client libs in Python (2.5), PHP, Perl, but not C++.
- No persistent queues, but one of the touted features is preservation of queue state between server restarts/crashes.
- Here's the protocol doc: <http://github.com/kr/beanstalkd/tree/v1.1/doc/protocol.txt?raw=true>
- Immature
- Works only with Python-2.6, which is two versions above the installed version on etch.

## Peafowl

<http://code.google.com/p/peafowl/> (A light weight server for reliable distributed message passing.)

! This product has been eliminated prima facie from the race.

- Uses memcache protocol, which is not a message queueing protocol. The argument that we should be using a message queueing protocol rather than a key-value protocol is convincing to us. The impedance mismatch between what the memcache protocol is trying to do and what we want out of a message queue system is quite large and it seems our effort is better spent trying to find something that is designed for the purpose.

## Starling

<http://rubyforge.org/projects/starling/>

Persistent Message Queue, in Ruby though

! This product has been eliminated prima facie from the race.

## SMTP

<http://www.ietf.org/rfc/rfc2821.txt>

! Ha ha we're just kidding.

- Very mature protocol
- Many vendors
- Huge community
- Persistent queues only - no support for transient queues
- No built-in CreateQueue/DeleteQueue operations
- No support for transactional messages
- Many failure modes will result in multiple delivery of messages

## Zero MQ

<http://www.zeromq.org/>

! This product has been eliminated from the race. Read "NOTE"

- From above page: ØMQ is already very fast. We're getting 13.4 microseconds end-to-end latencies and up to 4,100,000 messages a second today.
- Papers on Tests - <http://www.zeromq.org/area:results>
- Not based on AMQP (<http://amqp.org>) and has an AMQP plugin.
- Chat example: <http://www.zeromq.org/code:examples-chat>
- Python module's API is currently much simpler when compared to original C++ API. The difference is that Python module doesn't allow for full control of ØMQ threading as C++ does. Instead, the Python module extension creates single I/O thread that can be accessed from a single application thread. This doesn't allow for seamless scaling on multicore boxes. However, it is our intent to expose full ØMQ API via Python in the future.
- **NOTE:** It looks like the python Ømq extension is broken. The sending and receiving functions work only when they are performed within the same Python process (probably has something to do with the above note about the module API in Python)
- **UPDATE to NOTE:** Python extension works as advertised using the trunk version. Using this, it was possible to generate 100000 queues, for 4 'clients', and the zmq\_server did not bog down the CPU (in fact it hovered around 2%)
- Java wrapper does not work as it fails at `System.loadLibrary()`

## Reading/Related Links

- Open Source Queueing and Messaging Systems? (<http://jeremy.zawodny.com/blog/archives/010511.html>)
- MQP Benchmarking ([http://www.cse.ohio-state.edu/~narravul/papers/subramoni\\_whpcf08.pdf](http://www.cse.ohio-state.edu/~narravul/papers/subramoni_whpcf08.pdf))
- AMQP vs XMPP/RabbitMQ vs ejabberd (<http://lists.rabbitmq.com/pipermail/rabbitmq-discuss/2008-June/001290.html>)

This page was last modified on 12 January 2010, at 17:27. This page has been accessed 99,464 times.  
 Copyright © 2017 Linden Research, Inc. Licensed under Creative Commons Attribution-Share Alike 3.0 unless otherwise noted.  
[Privacy policy](#) [About Second Life Wiki](#) [Terms of Use](#)