Mollom Architecture - Killing Over 373 Million Spams at 100 Requests Per Second

Tuesday, February 8, 2011 at 12:18PM

Todd Hoff in Example

Mollom is one of those cool SaaS companies every developer dreams of creating when they wrack their brains looking for a viable software-as-a-service startup. Mollom profitably runs a useful service—spam filtering—with a small group of geographically distributed developers. Mollom helps protect nearly 40,000 websites from spam, including one of mine, which is where I first learned about Mollom. In a desperate attempt to stop spam on a Drupal site, where every other form of CAPTCHA had failed miserably, I installed Mollom in about 10 minutes and it immediately started working. That's the out of the box experience I was looking for.

From the time Mollom opened its digital inspection system they've rejected over 373 million spams and in the process they've learned that a stunning 90% of all messages are spam. This spam torrent is handled by only two geographically distributed machines that handle 100 requests/second, each running a Java application server and Cassandra. So few resources are necessary because they've created a very efficient machine learning system. Isn't that cool? So, how do they do it?

To find out I interviewed Benjamin Schrauwen, cofounder of Mollom, and Johan Vos, Glassfish and Java enterprise expert. Proving software knows no national boundaries, Mollom HQ is located in Belgium (other good things from Belgium: Hercule Poirot, chocolate, waffles).

Statistics

- Serving 40,000 active websites, many of which are very large customers like Sony Music, Warner Brothers, Fox News, and The Economist. A lot of big brands, with big websites, and a lot of comments.
- Find 1/2 million spam messages each day.
- Handle 100 API calls/second.
- A spam check is low latency, taking between 30-50msecs. The slowest connection would be 500msec. The 95th percentile of latency is 250msecs. It's really optimized for speed.
- Spam classification efficiency is at 99.95%. This means that only 5 in 10,000 spam messages were not caught by Mollom.
- Netlog, which is a social networking site in Europe, has their own Mollom setup in their own datacenter. Netlog handles about 4 million messages a day on custom classifiers that are trained on their data.

Platform

- Two production servers run in two different datacenters for failover.
  - One server is on the East coast and one is on the West coast.
  - Each server is an Intel Xeon Quad core, 2.8GHz, 16GB RAM, 4 disks of 300 GB, RAID 10.
- SoftLayer - the machines are hosted by SoftLayer.
- Cassandra - a NoSQL database selected for it's write performance and ability to operate across multiple datacenters.
- Glassfish - open source application server for the Java EE platform. They picked Glassfish for it's enterprise ready features like replication and failover.
- Hudson - provides for continuous testing and deployment of the

backend across all their servers.

- Java - From the start Mollom was written in Java.
- Munin - used to measure and plot metrics concerning server health.
- MySQL - JPA (The Java Persistence API) is used for regular data sets and Cassandra is used for large data sets.
- Pingdom - used for uptime monitoring.
- Zendesk - used for support.
- Drupal - used for the main website with a custom E-commerce module.
- Unfuddle - Subversion hosting used for source code control by their distributed development team.

What is Mollom?

Mollom is a web service for filtering out various types of spam from user generated content: comments, forum posts, blog posts, polls, contact forms, registration forms, and password request forms. Spam determination is not only based on the posted content, but also on the past activity and reputation of the poster. Mollom's machine learning algorithms act as your 24x7 digital moderator, so you don't have to.

How is it used?

Applications like Drupal, for example, integrate Mollom using a Module that installs itself into content editing integration points so content can be checked for spam before being written to the database. The process looks like:

- When a user submit comments to a website an API call is made to the backend servers.
- The content is analyzed, if it is spam the site will be told to block it, or if the backend is unsure it will advise the site to show a

CAPTCHA, which they also serve.
- When the CAPTCHA is filled in correctly the content will be accepted. In most cases a human will not see a CAPTCHA, the content will directly be accepted as *ham*, ham being good content and *spam* being bad content.
- CAPTCHA is only displayed with the machine learning algorithms are not 100 percent sure, so for the most part humans are not inconvenienced.

Dashboard

Mollom includes a pretty nifty dashboard for each account that shows you how much ham has been accepted and how much spam has been rejected. The amount of spam that one sees in the graph is really depressing.

Operations Process

Installation. Installation is quite easy for Drupal. Install it like any other module. Create an account on the Mollom website. Get a pair of security keys, configure these keys into the module, and select which parts of the system you want to protect with Mollum. That's about it.

Daily. I check regularly to see if spam has gotten through. It's not 100% so some spam does get through, but very little. If spam does get through there's a way to tell Mollom that this post was really spam and should be deleted. This is what you would have to do anyway, but in the process you are helping train Mollom's machine learning algorithms about what is spam.

Allows for anonymous user interaction. With a good spam checker it's possible to have a site where people can interact anonymously, which is what a lot of people using certain types of sites really prefer. Once you require registration engagement goes way down and registration doesn't stop spammers anyway.

Not Everything is Rosy

Dealing with false positives is Mollom's biggest downside. Spam detection is a difficult balancing act between rejecting ham and accepting spam. Mollom's machine learning algorithms seems to work quite well, but there is a problem sometimes with good posts being rejected, you get the dreaded: *Your submission has triggered the spam filter and will not be accepted*. Currently there is no recourse. Few things piss off a user more than having their glorious comment rejected as spam when it's obvious, to a human, that it's not. A user will only try a few times to get around the problem and then they will simply give up and walk away.

The problem is there is no way to fix the problem. To protect the machine learning algorithms from being gamed, Mollom does not allow you to present an example of a wrongly rejected chunk of content that should be accepted, though they are working on adding this in the future.

It's a tough decision. Static CAPTCHA systems, that is systems that only require a user pass a test to submit content, simply do not work once a site has been targeted for serious attack. User registration doesn't work. Moderating every post requires a very high burden, especially for "hobby" sites, given a site can have thousands of spams a day. And spam completely kills a site, so the risk of angering some users has to be balanced against having no users in the end because of a bombed-out site.

Business Model

- What makes Mollom a no-brainer is it is free to try. You only pay once your site starts accepting over 100 ham messages a day. For a small site this may never happen.

- Once you pass the free threshold there is Mollom Plus (1 EUR a day) and Mollom Premium (3600 EUR/year/site) pricing tiers, which seemed pretty reasonable.
- The free websites are not a resource drain as you might expect, they are are actually a source of critical training data. All websites using Mollom are constantly feeding back data to the backend classifiers. The more people that using Mollom the better is the training through all the feedback it is getting from users. Without free sites Mollom wouldn't be as accurate as it is.

Architecture

- Mollom is very engineering driven. A major emphasis has been put on making Mollom as efficient as possible both in code and server resource usage.
- Physically each server can handle all requests, but they have complete failover. Work is distributed between the machines. If one goes down then work moves to the other machine.
    - They used to have 3 servers, but since they improved the performance a lot they can use the third server as a staging server.
    - Each server can handle a full 100 connections per second and each connection runs the entire pipeline: full text analysis, calculating the reputations of the authors, and serving CAPTCHAS.
- Really optimized for low latency. Since the spam detection is part of the process of content being submitted to a site, if it took a long time it would be really annoying to users.
- Mollom has went through several phases of evolution:
    1. Initially a small team of two people worked part time on the algorithms, the classifiers, the real business problems they

were trying to solve. To build out the infrastructure on the backend they used their own implementation of thread-pool, connection-pool, resource management. They found they were spending too much time on supporting all this stuff and making it scale. They then switched to Glassfish, a Java application server, so they would have worry a lot less about memory management, REST handling, XML parsing, and database connection pooling.

2. The main problem in the past was disk bandwidth. They need to track reputations for all IP addresses and for all URLs on the Internet, so this was a massive datastore with a lot of random access.
3. In the early days they used a cheap virtualized machine and everything was in MySQL, which didn't scale.
4. They then moved to solid state disk and stored everything in files. Solid state solved the write problem, but there were issues:
    1. It's really expensive.
    2. It's very sensitive to the type of file system installed.
    3. Writes are fast, but iterating over the data, which they did quite often, to clean up the data or to train a new classifier over millions of millions of small objects, was still very slow.
5. They then moved away from solid state and moved to Cassandra.

- Cassandra is now used as their database for write heavy loads and as a caching layer:
    - Runs on a RAID 10 disk configuration (striped and mirrored), which are very good for heavy read/writes.
    - Cassandra is optimized for writes and Mollom has a lot more writes than reads.
    - Designed to be distributed inside a datacenter and across

datacenters.
    - A downside is there is no standard NoSQL interface which makes it difficult to write applications.
    - Cassandra's row caching saves them from having to add another caching layer in their system, which removes a lot of application code.
    - Cassandra has an aging feature which will automatically deletes data after a period of time. Europe has strict privacy laws which require certain data be removed after a period of time. This feature is a huge win. It's an extremely expensive operation and Cassandra handling it removes a lot of application code.
- The path a blog comment takes through the system:
    - A request can come from any client. Clients load balance requests across servers. This part is explained later. The typical client is a Drupal system. Requests can be XML-RPC or REST.
    - Requests are handled by a Glassfish application server and follow a typical application server workflow: requests are handled by servlets and are delegated to session beans.
    - Paying customers are serviced first, free customers may experience longer delays.
    - The request is parsed and analyzed. More on this later. A spam score is determined and returned to the user. So there are different functional parts of Mollom: spam checking and CAPTCHA handling. CAPTCHA includes generating, serving, and processing the responses. Different sessions beans are responsible for the parts of the Mollom functionality.
    - Classifiers are fully in RAM. A small piece of content is blown up and broken into thousands and thousands of tiny tokens that could identify spam. These classifiers have in the area of a few million tokens in RAM. The classifiers need to

run really quickly so they have to be in RAM.
    - What is kept in Cassandra are reputation scores, frequencies, URLs, and IP addresses. Cassandra's new row caching feature now acts as their caching layer. Previously they implemented an internal cache, but that was removed.
    - Both machines in both datacenters run Cassandra and the Glassfish application server. Cassandra is constantly replicating data between datacenters.
    - The in-memory datastructures are not replicated directly. They write through to Cassandra which then replicates. The cache on the other side has a timeout so it will go to Cassandra and pick up the new data. It's eventually consistent. There's a small window of inconsistency, but the models aren't negatively impacted over such a short period of time.
    - Consistency is managed on a case-by-case basis. For reputation and IP addresses eventual consistency is fine. Session data, including CAPTCHA sessions, is kept strictly consistent so it will follow correctly when a machine fails over.
- Client Side Load Balancing
    - Mollom uses client side load balancing that distributes load based on latency, etc. As a startup they didn't have the money to buy a large load balancer. They also had a goal to be able to do global load balancing over multiple datacenters, which would have required an expensive and complicated setup.
    - Management of the client list is through an API. Each client has a list of available servers it can use.
    - Each client can get a different list of servers to use. Paying customers can be provided a list of closer servers to reduce latency.
    - When a server fails the clients will try the next server in the list.

- Client side load balancing helped with migration from the old system to the new Glassfish based system. New users were given the address of the migrated machines and old users still worked on the old machines and could be migrated in orderly fashion by updating their server lists. This allowed testing so they could test functionality and then scaling and performance. They looked at responses times, how many connections were in the connection queue, and how long connections would stay in the queue. They could test what would happen if they increased threads in the thread pool, changed the number of JDBC connections, and other configurations. Once everyone migrated the old servers were shutdown. Very little downtime was experienced during the transition, which is key for a highly available system. While the system is down spams are getting through.
- A downside of the client side approach is that if third party clients are written poorly then there will be problems. The client could get a server list, for example, and iterate over it in reverse order, which is wrong. They now work closely with client developers and provide quality reference code examples so developers can learn best practices.

- Machine Learning
  - Mollom is a set of learning systems. A stand-alone CAPTCHA solution, which neither considers user behavior nor point of origin, can never achieve this level of informed protection, and generally requires users to solve a CAPTCHA on every post. Using Mollom's text analysis, users must only solve CAPTCHAs when Mollom is unsure about a post.
  - The average message length is about 500 characters and this is blown up into 3,000 features. The spaminess of a post is determined by looking at the reputation of the IP address or Open ID, they look at the user ID, sentiment, language,

profanity, specific words and combinations of words, they look at how well the text has been written, etc. All these are based on classifiers. Some of the classifiers are statistical in nature that can learn in automatically. Some classifiers that are rule based to ensure they can never go wrong. It's a combination of all these tests in the end that determine the spam score.
  - They learn from this process and the classifiers and internal metrics are updated in real-time.
- Glassfish handles work scheduling and was designed to handle multicore workloads.
  - The key is to create a design the works in parallel as much as possible and has as small a lock window as possible.
  - The number concurrent HTTP connections is tuned so they have an appropriately sized pool of available connections.
  - They uses 16 threads per server.
  - Most of the calls are handled by stateless session beans which work well for concurrency management.
  - They keep a number of session beans in a pool, but let Glassfish decide how many should be in a pool. At peak load there will be more session beans in the pool so requests can be handled efficiently. 32 session beans can be running in parallel at any given moment.
  - All the classifiers are actually session beans that are being reused by different threads.
  - Each session bean has their own client connection to Cassandra so they will not block each other.
  - When a user doesn't respond to a CAPTCHA that session is cleaned up and the Mollom learns that was probably spam.
  - There is one instance of each classifier per server.
  - There is a very short period of lock contention on session cleanup where classifiers are being updated.

    - Updated classifiers are written back to Cassandra every 1/2 hour.
- Application Integration
    - Mollom uses an open API which can be integrated into any system.
    - Libraries: Java, PHP, Ruby, and more.
    - Integrated solutions: Drupal, Joomla, Wordpress, and other content management systems.
    - Third parties generate new bindings based on example code produced by Mollom.
- To monitor the health of a server they continually monitor, using Munin:
    - What is the size of the heap after garbage collection?
    - What is the number of available connections?
    - What is the number of threads available in the thread pool? Make sure there's no one thread waiting for a long time for a lock.
- When you look at their architecture, Mollom is trying to build an architecture that can transparently work out of multiple datacenters a set themselves up to scale-out when they've outgrown a single server system:
    - Client side load balancing is used to select and failover servers.
    - Glassfish clustering will be used to failover at the application tier and to make is easy to add and remove machines.
    - Cassandra will be used manage the data tier across datacenters.
- Netlog's installation of Mollom has some interesting characteristics. It process more than the main Mollom.com servers, but the distribution of spam is completely different because their people communicating are part of social network. The spam distribution on Netlog 90% ham and 10% spam, where it's exactly the reverse out in the cruel world of blogs. The interesting implication is that it takes less resources to process ham so they can actually perform

more work on the same servers.

- They initially tried virtualized servers, and thought of using Amazon, but they found IO was the main bottleneck using shared virtualized servers. IO latency and bandwidth were real problems so they decided to go scale-up and use bigger machines and bigger disks.
    - Surprisingly they are not CPU bound. Only two of the 8 cores are computing. The others are just doing IO.
    - Mollom's traffic is fairly constant, so having a dedicated servers is more cost efficient. They see Amazon more of a way to handle peak loads.
- Development Process
    - They are distributed team. Three people are in Belgium, someone in Texas, Boston, and Germany.
    - Scrum is used as the development process and they are very happy with it. The scrum meeting happens over Skype at 2PM. They've found as they've grown they need more process.
    - Developers develop locally and submit code into Unfuddle.
    - Hudson is used as their continuous integration environment. Hudson made their migration from the old to the new Glassfish system easier because tests had to be passed before deployment. They didn't lose much time because problems were found before production deployment.
    - They have many tests: unit tests, system tests, Drupal tests. Only when these tests are passed by Hudson can the system be deployed.
    - Deployment is still done manually to reduce potential downtime.
    - Whenever they found a problem with garbage collection times it was always due to a memory leak in their application. In case of a memory leak they take a core dump. To analyze a core dump from a 16GB machine is not that easy, you

probably can't analyze it on your local machine, so what they do is rent a large memory instance on Amazon to analyze the dump. It takes about 2 hours to process the heap dump. They compare two dumps, one after 10 hours of execution time and one after 20 hours of execution time. If there are major differences then it's probably because of a memory leak.

Future Directions

- Mollom API uses XML-RPC, they are now testing a REST implementation to make it easier for services to mashup with Mollom.
- Now that they've transitioned to Cassandra it will be easier for them to scale-out when growth dictates.
- Soon to be released are enterprise features that make it possible to manage hundreds of websites as a unit. It will be easy to moderate across a set of websites by sentiment, spam score, or delete all the comments from certain IP addresses.
- They've had some talks about getting into the streaming data business like Twitter, but they are restricted by Europe's stricter privacy policies.
- They will experiment using Glassfish for load balancing in each datacenter.
- If the load goes up 10x they will have to add more Cassandra nodes. Disk IO is the bottleneck. Only when they have to grow more than 10x will they need to add more application servers.

Lessons Learned

- **Efficiency leads to happiness.** Mollom takes high performance engineering very seriously. They are proud that Mollom is extremely cost effective. It can handle many many requests on a single server, with low latency, which makes customers happy, which makes them happy because they don't have to maintain lots of machines, and costs are low. They've made this a priority from the start and have chosen the right technologies to achieve their goals. This enabled them to take the profit they've made and invest in marketing, building a user base, and building new products on top of Mollom.

- **Free for breadth, pay for depth**. Machine learning requires a lot of example data to be able to detect spam successfully. To get that data Mollom offers a free service to customers, who provide the breadth of data needed to better train the learning algorithms, they are a constant source of intelligence and feedback. Larger customers provide the revenue and also benefit from the data learned from the free clients. This model seems peculiar to big data and machine learning, which as we all know is the future of everything.
- **Remove non domain-specific obstacles**. Big systems take a lot of infrastructure work. The infrastructure effort often takes work away from work on a product's true value producing domain related features (classifiers, reputation systems, client libraries). Mollom consciously tried to get out of the infrastructure business as much as possible with their selections of Cassandra and Glassfish.
- **Be careful with client side code**. Code on the client side is attractive because it uses other people's resources instead of yours. The problem is that code can be poorly written which will make your system look bad. Work closely with client developers and provide quality reference code examples so developers can learn best practices.
- **Prioritize paying customers**. Paying customers get a better quality of service. They are handled first in the queues and experience less delay through the system. Paying customers have access to failover server and free customers only have one server.
- **Reduce code by letting the stack do the heavy lifting**. In the early

days the Mollom code base was a lot bigger than it is now. Cassandra removed a lot of complex code by handling replication and row caching and Glassfish removed a lot application code and will handle clustering. Simplify over time.
- **Minimize lock contention**. Mollom spent a lot of time working on minimizing lock contention in the Glassfish server as this became the major bottleneck. Lock as little is possible to maintain full parallelism.

Related Articles

- Mollom API
- Mollom Drupal Module
- Mollom Technical Whitepaper
- Mollom's Twitter Account
- Episode #072 - Mollom.com's GlassFish backend with Dries and Johan
- Mollom gets a new backend
- Fighting spam with Mollom on Glassfish
- To learn more about big data and machine learning take a look at some materials from the Strata Conference.

Article originally appeared on High Scalability (http://highscalability.com/).

See website for complete article licensing information.