DataSift Architecture: Realtime Datamining at 120,000 Tweets Per Second

Tuesday, November 29, 2011 at 9:12AM

Todd Hoff in BigData, Example, Twitter

I remember the excitement of when Twitter first opened up their firehose. As an early adopter of the Twitter API I could easily imagine some of the cool things you could do with all that data. I also remember the disappointment of learning that in the land of BigData, data has a price, and that price would be too high for little fish like me. It was like learning for the first time there would be no BigData Santa Clause.

For a while though I had the pleasure of pondering just how I would handle all that data. It's a fascinating problem. You have to be able to reliably consume it, normalize it, merge it with other data, apply functions on it, store it, query it, distribute it, and oh yah, monetize it. Most of that in realish-time. And if you are trying to create a platform for allowing the entire Internet do to the same thing to the firehose, the challenge is exponentially harder.

DataSift is in the exciting position of creating just such a firehose eating, data chomping machine. You see, DataSift has bought multi-year re-syndication rights from Twitter, which grants them access to the full Twitter firehose with the ability resell subsets of it to other parties, which could be anyone, but the primary target is of course businesses. Gnip is the only other company to have these rights.

DataSift was created out of Nick Halstead's, Founder and CTO of DataSift, experience with TweetMeme, a popular real-time Twitter news aggregator, which at one time handled 1.1 billion page views per day. TweetMeme is famous for inventing the social signaling mechanism, better known as the retweet, with their retweet button, an idea that came out of an even earlier startup called fav.or.it (favorite). Imagine if you will a time before *like* buttons were plastered all over the virtual place.

So processing the TweetMeme at scale is nothing new for the folks at DataSift, what has been the challenge is turning that experience into an Internet-scale platform so that everyone else can do the same thing. That has been a multi-year odyssey.

DataSift is position themselves as a realtime datamining platform. The platform angle here is really the key take home message. They are pursuing a true platform strategy for processing real-time streams. TweetMeme while successful, could not be a billion dollar company, but a BigData platform could grow that large, so that's the direction they are headed. A money quote by Nick highlights the logic in neon: "There's no money in buttons, there's money in data."

Part of the strategy behind a platform play is to become the incumbent player by building a giant technological moat around your core value proposition. When others come a knockin they can't cross over your moat because of your towering technological barrier to entry. That's what DataSift is trying to do. The drawbridge on the moat is favored access to Twitter's firehose, but the real power is in the Google quality real-time data processing platform infrastructure that they are trying to create.

DataSift's real innovation is in creating an Internet scale filtering system that can quickly evaluate very large filters (think Lady Gaga follower size) combined with the virtuous economics of virtualization, where the more customers you have the more money you make because they are sharing resources.

How are they making all this magic happen? Let's see...

Site: http://DataSift.com

Information Sources

- The articles and videos listed at the end of this article.
- The primary source is an interview with: Nick Halstead, Founder and CTO of DataSift; Lorenzo Alberton, Chief Architect at DataSift.

Stats

- You can't get the entire Twitter firehose. If you did it would cost 10 cents per 1000 tweets. The firehose is at 250 million tweets a day. That's 25K a day in licensing costs.
- 936 CPU Cores
- Supports 16,000 Data Streams Per Server

- Processes Whole Twitter Firehose 250+ million Tweets per day. As a comparison, Visa says in 2007 they processed 27.612 billion transactions, which this paper estimates is 2100 transactions per second during the busiest 8 hours of the day. Those are complete transactions however.
- Current Peak Delivery of 120,000 Tweets Per Second (260Mbit bandwidth)
- Performs 250+ million sentiment analysis with sub 100ms latency
- 1TB of augmented (includes gender, sentiment, etc) data transits the platform daily
- Data Filtering Nodes Can process up to 10,000 unique streams (with peaks of 8000+ tweets running through them per second)
- Can do data-lookup's on 10,000,000+ username lists in real-time
- Links Augmentation Performs 27 million link resolves + lookups plus 15+ million full web page aggregations per day.
- Staff of 30 people
- 4 years of development

Platform

Languages Used

- C++ for the performance-critical components, like the core filtering engine (custom virtual machine)
- PHP for the site, external API server, most of the internal web services, and a custom-built, high performance job queue manager.
- Java/Scala for the communication with HBase, for Map/Reduce jobs and for processing data from Kafka
- Ruby for our Chef recipes

Data Stores

- MySQL (Percona server) on SSD drives
- HBase cluster (currently, ~30 hadoop nodes, 400TB of storage)
- Memcached (cache)
- Redis (still used for some internal queues, but probably going to be dismissed soon)

Message Queues

- 0mq (custom build from latest alpha branch, with some stability fixes, to use publisher-side filtering), used in different configurations:
  - PUB-SUB for replication / message broadcasting;
  - PUSH-PULL for round-robin workload distribution;
  - REQ-REP for health checks of different components.
- Kafka (LinkedIN's persistent and distributed message queue) for high-performance persistent queues.
- In both cases they're working with the developers and contributing bug reports / traces / fixes / client libraries.

CI / Deployments

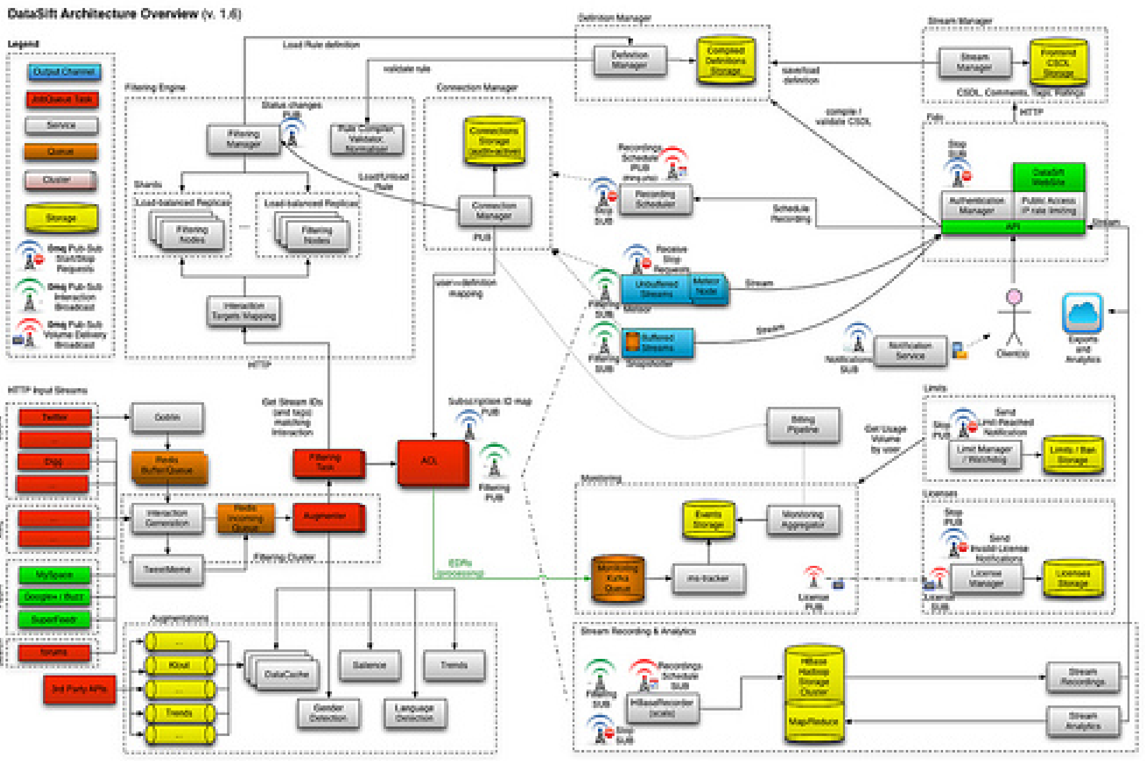
- All code (regardless the language) is pulled from the repo from Jenkins every 5 mins (if there's a change) and automatically tested and verified with several QA tools.
- Each project is built and packaged as an RPM and moved to the dev package repo.
- Several environments (dev, integration, QA, staging, production) or different policies and levels of testing.
- Chef to automate deployments and manage configuration.

Monitoring

- All services emit StatsD events, which are combined with other system-level checks, added to Zenoss and displayed with Graphite.

Architecture in a Picture

DataSift has created an awesome picture of their overall architecture. Here's a small version, for the full sized version please go here.



- The diagram has two halves: everything on the left is data processing and everything on the right is data delivery. 40+ services run in the system, these include: license service, monitoring service, limit service, etc.
- The system as a whole has a number of different scaling challenges that must be solved nearly simultaneously: handling the firehose, low latency natural language processing and entity extraction on tweets, low latency in-line augmentation of tweets, low latency handling very large individual filters, low latency evaluation of a large number of complex filters from a large population of customers, link resolution and caching, keeping a history of the firehose by persisting the 1TB of data it sends each day, allowing analytics to be run on the history of the firehose, real-time billing, real-time authentication and authorization, a dashboard to let customers know the status of their streams, streaming filter results to 1000s of clients, monitoring every machine every filter and every subsystem, load and performance testing, handling high network traffic, and messaging between services in a low-latency fault tolerant manner.
- We won't cover every box or line of what they do, but we will hit the highlights.

Basic Ideas

- **The point of it all**:
  - **Democratization of data access**. Consumers can do their own data

processing and analytics. An individual should, if they wished, be able to determine which breakfast cereal gets the most tweets, process the data, make the charts, and sell it to the brands. With a platform in place that's possible whereas if you had to set up your own tweet processing infrastructure it would be nearly impossible.

  - **Software as a service to data**. Put in a credit card, by an hour or months worth of data. Amazon EC2 model. Charge for what you use. No huge sign up period or fee. Can be a small or large player.
  - **You don’t really need big data, you need insight**. Now what we do with the data? Tools to create insights yourself. Data is worthless. Data must be used in context.
  - The idea is to **create a whole new slew** of applications that you couldn't with Twitter’s API.

- ETL (extract, transform, load). It might help to think of DataSift as a kind of ETL style system with real-time data streams as the input and your application as the output.
- **Extract**
  - Data is read from multiple sources, but licensed access to Twitter's firehose is the major draw at the moment. A firehose is all of Twitter's public tweets. You also have access to Digg and MySpace data.
  - Identity is not established between different data sources, but they normalize all the data so you know where the user names are located so you can match identities.
- **Transform**
  - Data is extracted out of the firehose and normalized. Twitter data is highly dimensional, it has 30 plus attributes and you get access to them all. These attributes include geolocation, name, profile data, the tweet itself, timestamp, number of followers, number of retweets, verified user status, client type, etc.
  - Entity extraction and natural language processing is applied to the tweet. Language is determined, for example, and that result is made available in the meta-data .
  - When a tweet comes in they have to query 5 different services to augment a tweet with more data. The include sentiment analysis, gender detection, and Klout score.
  - Each link is resolved and the contents fetched so filters can be applied to content and the links themselves.

  - They plan on adding more services as time goes on.
  - Data is fully elaborated before the filtering process occurs.
- **Filtering**
  - You can't pay for the entire firehose so a relatively simple declarative language called CSDL is used to filter out the Tweets you don't want and keep the Tweets you do want.
  - A filter looks like: *interaction.content contains “apple” AND interaction. content contains “orange”*
  - All tweets matching a filter form a stream. Each stream is assigned an identifier that looks something like “4e8e6772337d0b993391ee6417171b79”
  - Streams pop in conversation everywhere. A stream is the output of a CSDL filter applied to input. Through filters you are creating the stream of events you are interested in.
  - Stream identifiers are included in rules to further qualify which tweets should be in a stream. Rules build on filters and other rules: *rule “4e8e6772337d0b993391ee6417171b79” and language.tag == “en”*
  - Scalable filtering is the major innovation by DataSift. They use a compiler to schedule filters efficiently across a cluster. They spent 3 years developing a scalable rule evaluation solution.
  - Rules are made from a combination of filters and can have 1000s potentially millions of terms. Millions and millions of rules can be evaluated at once, scheduled across hundreds of machines. Rules can be reused and hierarchically ordered.
  - Filters include regular expressions, you can filter out tweets based on text in the profile, for example. There are many different targets and operators.
- **Load**
  - Tweets that match the filter are accessible to external applications via a REST API, HTTP Streaming, or Websockets.
  - A client library exists for: PHP, Ruby, C#, Java, Node.js. Saves writing HTTP requests yourself.
  - What you receive are JSON objects containing all the data in a normalized, fully augmented format. You’ll get the gender, geolocation, point-of-interest, country, author, the tweet, follower count, Klout score, etc in the stream. There up to 80 fields for each message.
- **Applications**
  - An application receives a fully elaborated JSON object over one of the

egress strategies. DataSift just sifts, it doesn't bake. So if you want something done with the tweets beyond the default transformations, then you'll have to write an application to do it.
    - Tweets are not guaranteed to arrive in any order and you are not guaranteed to see all the tweets ever sent. Applications must work on a aggregation/ sampling basis.
- **Billing**
    - Billing is in terms of magical units called **DPU**s - data processing units. Each filter is assigned a DPU. It costs 20 cents per DPU hour. The cheapest you can run is 1 DPU which would cost $150 to run one stream for an entire month.
    - The idea is the more resources you use the more you should pay.
- **Development**
    - Register. Look at example streams.
    - Learn CSDL. Write your filter.
    - Preview the filter on-line to see the quality of the results you get. They don’t just return raw data. They have a stream browser that has a map view, a word cloud view, and diagrams showing sentiment and gender, and more.
    - Fine tune. Get the results you want and take a look at the DPUs to see if you can afford it.
    - Collect data using the API of your choice and do something with it. What you do with the data is up to you. You can process it inline in real-time. Maybe display some graphs. Store it in your own database for post processing.

Key Ideas

Use Cases

Given how silly most tweets seem individually, it's almost hard to imagine they have value collectively, so it's sobering to take a look what people may want to use these systems for:

- A broad trend is the merging of datasets together. Bringing silo’d datasets together to get better insights. Merge social media data with your customer data at a grand scale and you can start to see a correlation between behaviour patterns and insights between the two data sets.
- Curation, monitoring, alerting.

- Tracking: TV show, politics, weather, colds, etc.
- Finance. Looking at the reactions people have towards a company.
- What is every one buying? Looking for broad trends over a month-to-month scale.
- Search for tweets with Google in text from people with over 500 followers near all Starbucks in the world.
- Disaster mitigation. Knowing where the needs are.
- Put all Best Buy locations in a rule so you can see tweets from within Best Buy without having to know a tweet was from a Best Buy using a hashtag. Works for any event or location.
- Curate news. Look at source of information with a certain number of retweets. Look at subject to say if technology. All in real-time, delivered in milliseconds.

The usual mix of the venal plus profound. How this power is used is up to you grasshopper.

Real-time Only has Far Reaching Consequences

The nature of DataSift as a real-time filtering system has far reaching consequences. Keep in mind that it has no memory. It has no history. You are not seeing an ordered stream that you can iterate over in any direction. You are getting data live from the firehose. The tweets get directed to filters, pass through the filters, and then they are gone, like the night.

This is a sampling model. You can’t think you are consuming the entire stream. Any application that expects to see every tweet from an account, in order, will not work. The type of applications this is targeted at are applications that can be accurate sampling highly targeted data.

For example, if you wanted to build a real-time control system with DataSift using Twitter as a message bus, it would not work. You could not guarantee that you would see every command over an account or that those commands would be ordered properly.

The type of problem you can solve is to create a filter that fires when there's 100 mentions of the word "earthquake" in 10 seconds. Then you could use the geolocation information to figure out where an earthquake has occurred. There's no tweet order required and it doesn't matter if you see every tweet. It's a different mindset.

Filter Only has Far Reaching Consequences

A surprise for me was that DataSift is a real-time filtering engine only (hence the word "sift" in their name). Higher level services are to be provided by partners.

I was expecting DataSift to be more of a stateful segmentation platform, enabling the answering of questions like "What is the count of males, 18-24, that live in USA?" DataSift doesn't count, so it can't have the sliding window counting features of a segmentation platform.

That's a consequence of it being **stateless**. And technically speaking DataSift can't identify an age segment either, mostly because Twitter has a pretty anemic profile feature. DataSift is storing the firehose in HBase/Hadoop for offline analytics, but that is not real-time.

So you have to wrap your head around the filter model. No application logic can be executed on DataSifts's platform. There are no stored procedures. Tweets can be augmented in-line via value added services, like sentiment analysis, but that's a highly structured and constrained process, that's not your application. And if you are expecting a sort of pipe model, nope, that's not it either. Streams can't be piped and weaved through a series of transducers.

What DataSift is, it is for a reason. DataSift is a highly sophisticated and scalable filter system. It filters out tweets from the firehose to create a highly targeted stream of tweets for you to process in a separate application. Their job is to get the data down to the cross section of tweets that are needed to perform analysis. Data comes in and is matched against rules, if it doesn't match it's tossed, if it does match, it's put in your bucket. Filters are like a gauntlet, only the most deserving pass.

The drivers for a filter only model are: **firehose scale and low latency**. The firehose produces events at a very high rate. How would you make an Internet scale platform that folds in the application logic for potentially millions of customers? How do you do that and build an end-to-end system that maintains a low latency guarantee? You can't.

What can you do? Evaluate large filters. Fast. More on this in the next section.

Filtering Engine

- When people think of filters they may have in mind SQL select statements where large "where" clauses are discouraged for performance reasons. DataSift takes the opposite approach, they assume very large sets of filter criteria and have made the evaluation scalable and efficient. Their target examples include monitoring every tweet from every Starbucks in the world or loading Lady Gaga's follower list into a filter. That could be millions terms in a filter, evaluated in real-time.
- Filtering at this scale requires a different approach. They started with work they did at TweetMeme. The core filter engine is in C++ and is called the Pickle Matrix.
- Over three years they've developed a compiler and their own virtual machine. We don't know what their technology is exactly, but it might be something like Distributed Complex Event Processing with Query Rewriting.
- A compiler takes filters and targets a cluster with a Manager and Node servers. The Manager’s job is to determine that if a rule is loaded anywhere else and if can be put somewhere that is highly optimized, like close to someone else that is running a similar stream. Node's job is throw tweets at rules, get a list of matches, and push the matches down the pipeline.
- They have clever algorithms for deduping workloads so rules are shared as much as possible. Each filter is not run in isolation. The filters form a shared space. If there's a filter that filters only for Lady Gaga references then, to the greatest degree possible, that filter is run on each tweet once and the result is shared between all rules using the same filter. It takes a very clever compiler and scheduling algorithm to make this work. The payoff is amazing. Instead of continually running duplicate filters they are only run once.
- Rules can be a hierarchical and the compiler is clever about trying to put them together so that they share rule evaluations. If a node already has a rule then it will attract filters that use that rule. That rule is run just once for all the filters on the node.
- This is a type of virtualization play. By combining workloads of multiple customers together they can exploit any commonality in the filters. DataSift is paid for every operation even though they only run it once. If a regular expression is shared, for example, it is only run once. It can’t always work this way, perhaps a node is loaded so can’t take another client, so it must run on another machine.
- The entire configuration dynamically changeable at runtime. Their scheduling algorithm is constantly looking at monitoring data. If latency crosses a threshold

then then the system will be reconfigured.

- Filters are processed immediately in memory. Each server, for example, can run 10,000 streams.
- Nodes have a rule cache so they can be shared.
- Compiler supports short circuiting to optimize filters.
- If a regular expression is bigger than an entire machine, for example, then they will automatically load balance across Nodes. Remember large filters are the expected norm. If you want a filter on all of Lady Gaga's followers, then filter scalability must be a core capability.
    - The filtering engine is sharded. Each shard receives the full firehose and every "interaction" (i.e. a "tweet" or "FB status" or "blog post") is processed by all shards, the results of which are collated before being sent downstream.
    - Each of the N nodes in a single shard (which are replicas of each other) receives 1/N of the hose on a round robin basis.
    - So, supposing there are 2 shards, with 3 nodes each, 50% of the filters would be found on each shard. Each node in a shard would receive 1/3 of the hose (and would have 50% of the filters on it). And it would be a replica of the other nodes in that shard.
    - Thus, if a very heavy filter is loaded, that will be balanced by adding more nodes to the shard the heavy rule is loaded into, thus reducing the amount of firehose a single node has to process.
    - So far, no single filter has been too heavy for a single node, but they are considering splitting the filter processing so sub-predicates are processed on mass first, then the resultant filters separately. They already do this for embedded rules (e.g., given a filter like "Get me all tweets containing 'apple' AND matching rule XYZ", filter XYZ is processed separately).
- A big push is for customers to create reusable public rules to encourage the reusability of data streams. People can use any of the public streams in their own rules. For example, someone could make a dirty word filter that creates a stream. You can build off that stream by using it. Everyone shares that same stream so if a 1000 users are using the dirty word filter the filter is not getting evaluated 1000 times. The filter is getting evaluated once for everyone, which is highly efficient.

Augmentation Pipeline for Each Tweet

Tweets are augmented with 3rd party datasets. Making these augmentations low latency was a major pain point.

- When a tweet comes in they have to query 5 different services to augment a tweet with more data. Klout, for example, has 100 million twitter profiles, so they perform a database request internally against a local copy of the Klout database. They bring datasets into their system instead of making API service calls across the Internet.
- To bring in datasets into their system they need to form close partnerships. Their sentiment analysis engine was licensed, made fast, made clusterable, and made suitable to handle 500 million hits a day with low latency.
- Each service must have < 100 msecs response time. After 500 msecs it's thrown away.
- One in 10 tweets is a link. So they'll go get the content and let you filter against that content too. So if a brand is mentioned it can be resolved against the actual content to figure out what it all means.

No Cloud for Them

- AWS is used to generate test traffic, but for a distributed application they thought AWS was too limited, especially in the network. AWS doesn't do well when nodes are connected together and they need to talk to each other. Not low enough latency network. Their customers care about latency.
- They had to pay attention to fine details like how the switches are setup and how they are routed through the master load balancer.
- Learned about using custom packet sizes from Twitter. They are using Jumbo frames.
- Twitter may be segmenting the firehose in the future due to the size.
- Their connection to Twitter flows through the Internet, there's no peering relationship.

Platform Means Open

- An API was built first and then the web site was built on the API. You could make your own import tools with a GUI and they are cool with that.
- The goal is to make DataSift invisible by having developers embed it in their applications. It is a platform and the work of the platform is to take away difficult bit of dealing with mass data. Others can build bridges into other systems.

DataSift wants to concentrate on filtering and data handling. Let others provide the GUIs etc.

Billing

- Affordable real-time billing products don't exist so they built their own. The challenge was to make people happy with real-time billing solution.
- Customers are billed in real-time. You can look at your dashboard to see what things are costing you. Experience with Amazon and Google App Engine has shown this is a very important and trick part of the system. Money is on the line here.
- They charge using a defined unit called the DPU - data processing unit. The more resources you make them use the more your filters will cost.
- Every filter is assigned a DPU.
- It costs 20 cents per DPU hour. A 10 DPU filter will cost $2/hr.
- The cheapest you can run is 1 DPU which would cost $150 to run one stream for an entire month.
- Filtering on 10K geolocations is going to be expensive. Looking up one regular expression is going to be cheap.
- Regular expressions are cheap, but the longer they are the longer the processing takes which means they are more expensive. Geopolygons are fairly complex to do. Long lists of key words are very cheap because they use a hash. Phrase matching and near word matching are slightly more expensive.
- The idea is that it’s cheaper for you to filter to the smallest possible dataset than pay licensing for data you don’t need.
- If you run 10,000 streams and it ends up being 50% of the firehose that’s fine, but they don’t just sell 50% of the firehose without data processing as it makes their platform irrelevant.
- They want customers to describe in as much details possible the data they want out of the filter.
- All males would be 50% of the firehose or 125 million tweets as there’s a 50-50 male female split on Twitter. Creating a filter for all tweets made by males would not be bright. It would be very expensive. What you want to do is look at use cases. Are you a bank, are you a pharma, and figure out what you are interested in specifically. Be as specific as possible. Some companies get only one tweet a day. Out of 250 million tweets a day the question is how do you create a filter to get the two or three that you want.

- Customers can set thresholds so they can decide how much to spend per day. You don’t want to run a stream that can eat up your budget in a few seconds.
- For billing to be accurate the system has extensive monitoring. Dashboards show the health of the platform as whole. Can see the flow between any points in the system, latency between each point. Health of all the servers, throughput, variations in the flow that could highlight problem.

The Output Side

- A real pain point was the output side. How do you deliver tweets from a central place to just the right server the user is connected to? The ended up using 0mq. Using Publish and Subscribe reduced internal network traffic, latency, and software complexity dramatically. They use it almost everywhere now. It’s very flexible. For high availability they broadcast to several listeners.
- Tried several technologies to move data around. Initially used HTTP everywhere, but that was too slow and had high code complexity. Then they tried Redis as a queue, which was faster, but it couldn’t handle the scale. They send billions of messages a day through 0mq.
- Node.js is used on their front-end, for endpoints for HTTP Streaming and Web Socket connections. Pleasantly surprised with the power of Node.js to do network data delivery. This layer is simple, it just has pass data straight through, socket in socket out. They built their own multithreaded model for Node.
- A problem with event systems is knowing why you received an event so it can be dispatched to the correct handling logic. The same tweet could match a gender rule and dirty word rule, for example, and for each case the tweet must be handled differently. DataSift has several ways to make this work, but they have a tagging feature that is quite smart. Once a subfilter matches that fact can be tagged and passed in the meta-data. The engine allows you to create any hierarchy of rules so you can join the rules together and then use tagging to create categories.

Testing

- EC2 is used to generate traffic. The firehose is 60 Mbps.
- To test their system they run it with the equivalent of 11 firehoses all at once. 1000 streams each 1% of the firehouse delivering it to 1000 connections. The bottleneck they hit was the network bandwidth from the hosting provider, not their platform.

Pooling the Stream into a Lake

DataSift also supports non Real-time processing. There are two major collections of tweets stored in HBase:

- Filters can have listeners so tweets are directly stored in HBase allowing the data to be downloaded later. It's possible to schedule the recording of a stream and then browse/use it later.
- A big technical challenge was recording the entire firehose. In two months they will let people run their filters over a persistent version of the firehose stored in HBase/Hadoop. The idea is to be able to run analytics over whatever slice of the firehose interests you. Once all the data is augmented it adds up to 1 TB each day that goes through the platform, so that's a lot of data to store.

How Do You Make Money?

As a developer I'm always interested in how platforms and services can be used to make money for developers, not just the creators. My angle is if you build a service on top of a service that is making money by charging you enough to make money, can you build a profitable service, or will your costs be too high, or will you really have to target high value high margin products to justify the cost? In other words, could TweetMeme be profitably built on DataSift, for example? The build or buy question is always a tough one.

- **Internal use**. If you are a business using DataSift to gain insight about your own products then the cost is justified by whatever value you receive. That's an easy one.
- **Services**. DataSift thinks of themselves as offering commoditized data delivery and that their cost is a fraction of the infrastructure cost you would need to get the data from Twitter and the developer hours to process that data. In their estimation, to build a social monitoring service like Radian6 would take 3 months on top of their system. All you have to concentrate on is the interface. The data collection and processing portion is taken care of. That's a lot of value, if you can find the right market.
- App Store. DataSift plans on having an application store for 3rd partly developers.
- **Augmentation service**. DataSift will pay services to be part of their

augmentation pipeline. The model is revenue share. If someone uses a platform like Klout they'll revenue share back to them. The problem is finding partners who can scale to twitter level.

Lessons Learned

- **The power of platform and ecosystems.** When a platform strategy is executed well it serves as a huge barrier to entry. It also serves as the highest observation post to see what next wave trends are developing. This is the innovate/leverage/ and commoditize model. DataSift already has domain names like LinkSift, so it's part of the grand plan to use their underlying platform to provide higher value services, guided by the intelligence they gather from their platform market.
- **There's no money in buttons, there's money in data.** Follow the money**.** What you are doing now may not be in the money, but it may guide you to where the money can be found.
- **Quality over crap.** The trend has been to release barely viable software and then improve it iteratively using user feedback. DataSift did not take that route. They are releasing a sophisticated scalable product from the start, for two reasons: 1) their base of experience at TweetMeme 2) they didn't feel like they could start small and rearchitect it on the fly.
- **Statelessness is a big win architecturally**. DataSift, because it is real-time only, doesn't have to keep a lot of state in memory or worry about delivering events in order without loss or even being that fault tollerant. Timing is their pain point, keeping end-to-end latency down. That of course pushes the burden of state down to the application developer.
- **Learn from other people's mistakes.** When Twitter licensed the retweet technology from TweetMeme they didn't actually reuse TweetMeme's code. Twitter learned from the code and used TweetMeme as a consultancy service, which helped Twitter get retweet right at launch. In a reverse direction, when DataSift was learning how to handle the firehose, DataSift was able to learn from all the early mistakes that Twitter made, so DataSift could get it right at launch.
- **Services**. It's hard to design using services from the start. They had an architecture from day one, but how you connect those services, how you make those services redundant, how you make those services respond to failures, and how you make everything not fail if one things fails--all these types of issues have been pain points. Dividing components into services is key to solving these distributed programming issues. Services have also allowed each component to

be made scalable in isolation.

- **Message system**. The settled on 0mq as the core message passing system to link servics together and implement reliability. It has worked wonderfully. 1000x better than using HTTP. They like the flexibility of the different communication patterns (publish-subscribe, request-response, etc). HTTP has way too much overhead in both performance and the amount of code needed to use it. Just deliver messages to the code that needs to deal with it. It keeps the interfaces asynchronous, decoupled, and queue based rather than blocking.
- **Connection management**. One of the biggest bottlenecks with real-time streams is how you deal with connects and disconnects. They learned a lot from Twitter. When you make a connection it triggers a lot of activity downstream to activate the connection, like authentication, validate stream requests, determine load, billing services, audit trails, etc. Being able to run 1000s of connects/disconnects per second is a challenge. Rewrote from PHP to C++ and made it multithreaded.
- **Fold common use cases into the platform**. During their People like loading up millions of people in lists. Like all of Lady Gaga’s followers. Point to list and download it automatically into a rule.
- **Metrics are more important than logging**. Logging always gets filled up with stuff you don’t use. It’s more efficient to be able to create alerts for metrics that go out of bounds. When a metric triggers awareness of problems then you can go look at logs.
- **Follow the Salesforce and Amazon model**. They are an on-demand cloud platform and they will build layers on top of what they started with. So DataSift is the engine, and they will make other products like UserSift and LinkSift.
- **Barter**. Don’t take money if you don’t have to. Bootstrap by bartering. They took sponsorships. They sold old domain names to finance purchasing new domain names.
- **Roll the experience of one product into another**. The infrastructure for fav.or.it, which was based on RSS feeds, was used by TweetMeme. This allowed TweetMeme to be developed quickly. Likewise, the experience of building out TweetMeme made the jump to DataSift much easier. From TweetMeme they learned scalability and developed a first rate engineering team.
- **At scale everything matters**. They’ve had problems with frame buffer sizes across their networking gear, for example. They've found Open HTTP stream connections are much more difficult to scale than simple REST calls as it brings out a lot of underlying problems in the network. When they used a REST API they didn't see these problems.

From my interview, reading, and viewing a dozen or so other interviews, I've been very impressed with the quality of thought and effort that has gone into DataSift. I don’t know if it works of course, or if it will succeed, but if it fails my guess is it won't because of a lack of professionalism. Mark Suster, entrepreneur turned VC, explains his investment in DataSift as doubling down on the Twitter ecosystem. I think that's the wrong way to look at it. DataSift can quickly and easily work with any stream. Maybe a better way to look at is doubling down on DataSift.

Related Articles

- This article on Hacker News
- Distributed Complex Event Processing with Query Rewriting by Nicholas Poul Schultz-Møller, Matteo Migliavacca, Peter Pietzuch
- What a Tweet Can Tell You by Marshall Kirkpatrick
- What is the difference between Gnip and DataSift?
- DataSift Link Resolution
- The Art of Scalability - Managing growth by Lorenzo Alberton
- More details on "track on steroids:" DataSift - Interview with Robert Scoble
- Is DataSift's pricing a deterrent for startups?
- How expensive is the infrastructure (bandwidth) of an app on top of Twitter feeds?
- Q&A: Nick Halstead on mining Twitter's firehose with Datasift
- Gist: Records DataSift Twitter interaction stream to Google Fusion Table by Paul M. Watson
- The Future Of Twitter Is In Data, Not Advertising, Says New DataSift CEO
- Twitter is a stream, but it’s also a reservoir by Mathew Ingram
- DataSift API Presentation
- Cassandra vs HBase by Nick Telford (DataSift Platform Architect)
- DataSift development on Twitter
- Fourth DataSift Webinar Video, “Targets”
- MediaSift / DataSift by Genesys Guru
- Why I’m Doubling Down on the Twitter Ecosystem by Mark Suster
- Ways to use Pusher at the Mozilla Festival by Phil Leggetter
- Video of IPTraf stats for 10 high volume streams - This was testing using 10 streams - and throughput was around 3Mb/s on one server.