Netflix Technology Blog Follow
Learn more about how Netflix designs, builds, and operates our systems and engineering organizations
Jan 10, 2013 · 11 min read

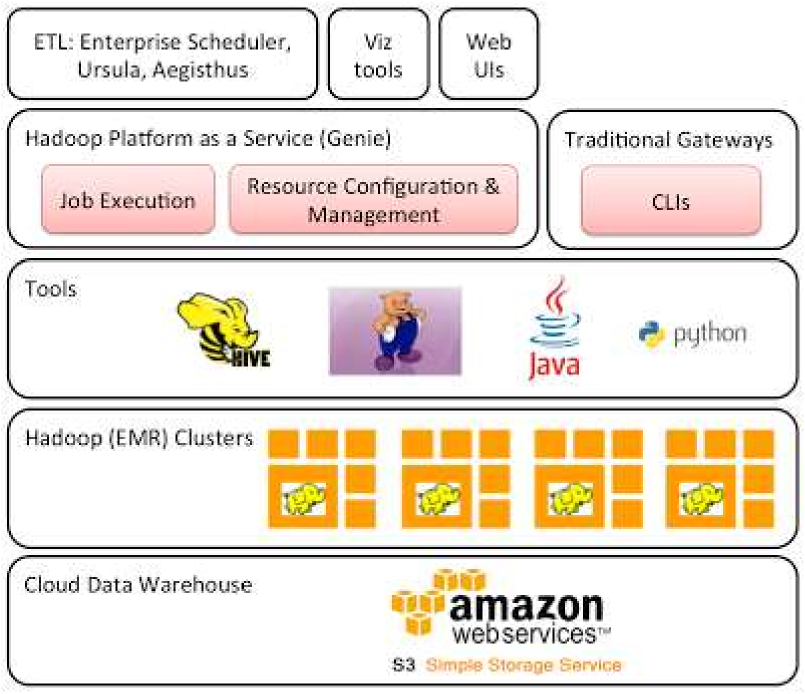
Hadoop Platform as a Service in the Cloud

by Sriram Krishnan and Eva Tse, Cloud Big Data Platform

Hadoop has become the de facto standard for managing and processing hundreds of terabytes to petabytes of data. At Netflix, our Hadoop-based data warehouse is petabyte-scale, and growing rapidly. However, with the big data explosion in recent times, even this is not very novel anymore. Our architecture, however, is unique as it enables us to build a data warehouse of practically infinite scale in the cloud (both in terms of data and computational power). In this article, we discuss our cloud-based data warehouse, how it is different from a traditional data center-based Hadoop infrastructure, and how we leverage the elasticity of the cloud to build a system that is dynamically scalable. We also introduce Genie, which is our in-house Hadoop Platform as a Service (PaaS) that provides REST-ful APIs for job execution and resource management.

Architectural Overview

In a traditional data center-based Hadoop data warehouse, the data is hosted on the Hadoop Distributed File System (HDFS). HDFS can be run on commodity hardware, and provides fault-tolerance and high throughput access to large datasets. The most typical way to build a Hadoop data warehouse in the cloud would be to follow this model, and store your data on HDFS on your cloud-based Hadoop clusters. However, as we describe in the next section, we have chosen to store all of our data on Amazon's Storage Service (S3), which is the core principle on which our architecture is based. A high-level overview of our architecture is shown below, followed by the details.


ETL: Enterprise Scheduler, Ursula, Aegisthus
Viz tools
Web UIs
Hadoop Platform as a Service (Genie)
Job Execution
Resource Configuration & Management
Traditional Gateways
CLIs
Tools
HIVE
Java
python
Hadoop (EMR) Clusters
Cloud Data Warehouse
amazon webservices™
S3 Simple Storage Service

S3 as the Cloud Data Warehouse

We use S3 as the "source of truth" for our cloud-based data warehouse. Any dataset that is worth retaining is stored on S3. This includes data from billions of streaming events from (Netflix-enabled) televisions, laptops, and mobile devices every hour captured by our log data pipeline (called Ursula), plus dimension data from Cassandra supplied by our Aegisthus pipeline. So why do we use S3, and not HDFS as the source of truth? Firstly, S3 is designed for 99.999999999% durability and 99.99% availability of objects over a given year, and can sustain concurrent loss of data in two facilities. Secondly, S3 provides bucket versioning, which we use to protect against inadvertent data loss (e.g. if a developer errantly deletes some data, we can easily recover it). Thirdly, S3 is elastic, and provides practically "unlimited" size. We grew our data warehouse organically from a few hundred terabytes to petabytes without having to provision any storage resources in advance. Finally, our use of S3 as the data warehouse enables us to run multiple, highly dynamic clusters that are adaptable to failures and load, as we will show in the following sections. On the flip side, reading and writing from S3 can be slower than writing to HDFS. However, most queries and processes tend to be multi-stage MapReduce jobs, where mappers in the first stage read input data in parallel from S3, and reducers in the last stage write output data back to S3. HDFS and local storage are used for all intermediate and transient data, which reduces the performance overhead.

Multiple Hadoop Clusters for Different Workloads

We currently use Amazon's Elastic MapReduce (EMR) distribution of Hadoop. Our use of S3 as the data warehouse enables us to spin up multiple Hadoop clusters for different workloads, all accessing the exact same data. A large (500+ node) "query" cluster is used by engineers, data scientists and analysts to perform ad hoc queries. Our "production" (or "SLA") cluster, which is around the same size as the query cluster, runs SLA-driven ETL (extract, transform, load) jobs. We also have several other "dev" clusters that are spun up as needed. If we had used HDFS as our source of truth, then we would need a process to replicate data across all the clusters. With our use of S3, this is non-issue because all clusters have instant access to the entire dataset. We dynamically resize both our query and production clusters daily. Our query cluster can be smaller at night when there are fewer developers logged in, whereas the production cluster must be larger at night, when most of our ETL is run. We do not have to worry about data redistribution or loss during expand/shrink because the data is on S3. And finally, although our production and query clusters are long-running clusters in the cloud, we can treat them as completely transient. If a cluster goes down, we can simply spin up another identically sized cluster (potentially in another Availability Zone, if needed) in tens of minutes with no concerns about data loss.

Tools and Gateways

Our developers use a variety of tools in the Hadoop ecosystem. In particular, they use Hive for ad hoc queries and analytics, and use Pig for ETL and algorithms. Vanilla java-based MapReduce is also occasionally used for some complex algorithms. Python is the common language of choice for scripting various ETL processes and Pig User Defined Functions (UDF). Our Hadoop clusters are accessible via a number of "gateways", which are just cloud instances that our developers log into and run jobs using the command-line interfaces (CLIs) of Hadoop, Hive and Pig. Often our gateways become single points of contention, when there are many developers logged in and running a large number of jobs. In this case, we encourage the heavy users to spin up new pre-baked instances of our "personal" gateway AMIs (Amazon Machine Images) in the cloud. Using a personal gateway also allows developers to install other client-side packages (such as R) as needed.

Introducing Genie—the Hadoop Platform as a Service

Amazon provides Hadoop Infrastructure as a Service, via their Elastic MapReduce (EMR) offering. EMR provides an API to provision and run Hadoop clusters (i.e. infrastructure), on which you can run one or more Hadoop jobs. We have implemented Hadoop Platform as a Service (called “Genie”), which provides a higher level of abstraction, where one can submit individual Hadoop, Hive and Pig jobs via a REST-ful API without having to provision new Hadoop clusters, or installing any Hadoop, Hive or Pig clients. Furthermore, it enables administrators to manage and abstract out configurations of various back-end Hadoop resources in the cloud.

Why did we build Genie?

Our ETL processes are loosely-coupled, using a combination of Hadoop and non-Hadoop tools, spanning the cloud and our data center. For instance, we run daily summaries using Pig and Hive on our cloud-based Hadoop data warehouse, and load the results into our (order of magnitude smaller) relational data warehouse in the data center. This is a fairly common big data architecture, where a much smaller relational data warehouse is used to augment a Hadoop-based system. The former provides more real-time interactive querying and reporting, plus better integration with traditional BI (business intelligence) tools. Currently, we are using Teradata as our relational data warehouse. However, we are also investigating Amazon’s new Redshift offering. We use an enterprise scheduler (UC4) in our data center to define dependencies between various jobs between our data center and the cloud, and run them as “process flows”. Hence, we need a mechanism to kick off Hadoop, Hive and Pig jobs from any client, without having to install the entire Hadoop software stack on them. Furthermore, since we now run hundreds of Hadoop jobs per hour, we need this system to be horizontally scalable, especially since our workload will increase as we migrate more of our ETL and processing to Hadoop in the cloud. Finally, since our clusters in the cloud are potentially transient, and there is more than one cluster that can run Hadoop jobs, we need to abstract away physical details of the backend clusters from our clients.

Why build something new?

Why did we build Genie, as opposed to using something else that is already available? The simple answer is that there was nothing that was already out there in the open source community that handled our requirements—an API to run jobs, abstraction of backend clusters, an ability to submit jobs to multiple clusters, and scalable enough (horizontally or otherwise) to support our usage. The closest alternative that we considered was Oozie, which is a workflow scheduler similar to UC4. It is not a job submission API like Genie (hence not an apples-to-apples comparison). We ruled out the use of Oozie as our scheduler, since it only supports jobs in the Hadoop ecosystem, whereas our process flows span Hadoop and non-Hadoop jobs. Also, when we started our work on Genie, Oozie didn't support Hive, which was a key requirement for us. A closer alternative to Genie is Templeton, which is now part of HCatalog. However, Templeton doesn't support concurrent job submissions to more than one cluster, is still evolving, and doesn't appear quite ready for production.

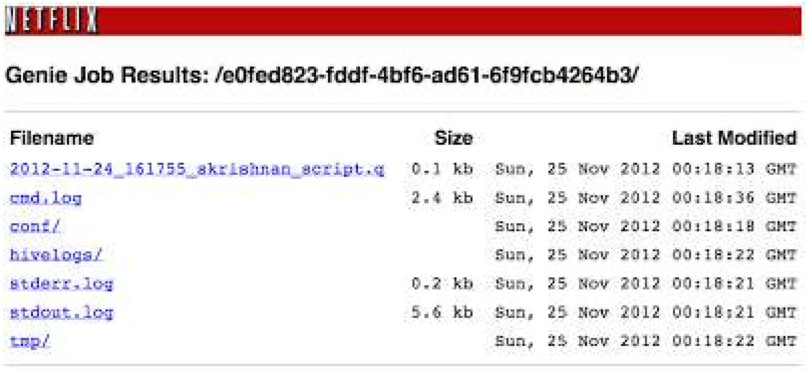
What is Genie?

Genie is a set of REST-ful services for job and resource management in the Hadoop ecosystem. Two key services are the Execution Service, which provides a REST-ful API to submit and manage Hadoop, Hive and Pig jobs, and the Configuration Service, which is a repository of available Hadoop resources, along with the metadata required to connect to and run jobs on these resources.

Execution Service

Clients interact with Genie via the Execution Service API. They launch a job by sending a JSON or XML message to this API to specify a set of parameters, which include:

- a job type, which can be Hadoop, Hive or Pig,
- command-line arguments for the job,
- file dependencies such as scripts and jar files (e.g. for UDFs) on S3,
- a schedule type, such as "ad hoc" or "SLA", which Genie uses to map the job to an appropriate cluster, and
- a name for the Hive metastore to connect to (e.g. prod, test, or one of the dev ones).


NETFLIX

Genie Job Results: /e0fed823-fddf-4bf6-ad61-6f9fcb4264b3/

Filename	Size	Last Modified
2012-11-24_161755_skrishnan_script.q	0.1 kb	Sun, 25 Nov 2012 00:18:13 GMT
cmd.log	2.4 kb	Sun, 25 Nov 2012 00:18:36 GMT
conf/		Sun, 25 Nov 2012 00:18:18 GMT
hivelogs/		Sun, 25 Nov 2012 00:18:22 GMT
stderr.log	0.2 kb	Sun, 25 Nov 2012 00:18:21 GMT
stdout.log	5.6 kb	Sun, 25 Nov 2012 00:18:21 GMT
tmp/		Sun, 25 Nov 2012 00:18:22 GMT

If a job submission is successful, Genie returns a job id for the job, which can be used to get the job status, and the output URL. The output URL is an HTTP URL pointing to the working directory of the job, which contains the standard output and error logs (see screenshot below). Each job id can translate to multiple MapReduce jobs depending on the number of intermediate stages in the Hive or Pig query being run.

Configuration Service

The Configuration Service is used to keep track of all the clusters that are currently running, and the schedule types that they support. For instance, our query cluster has been configured to support "ad hoc" jobs, whereas our production cluster has been configured to support "SLA" jobs. When a cluster comes up, we publish to the Configuration Service the type of jobs that it supports, and also the set of configuration files for that cluster (viz. the mapred-site.xml, core-site.xml, hdfs-site.xml for Hadoop, and hive-site.xml for Hive). We also mark its status as "Up". Similarly, when a cluster goes down we mark it as "Terminated". There is also an "Out of Service" state for clusters, which means it is alive, but not supporting any new job submissions. This is useful during upgrades and the termination process, when a cluster should no longer accept any new jobs, but should let all running jobs finish before it is terminated. This service is complementary to Eureka, which is designed to be a repository of metadata for ephemeral instances (and not clusters) in the cloud. When the Execution Service receives a job submission request, it maps the job to an appropriate cluster using the Configuration Service. If there are multiple clusters that could satisfy the job requirements, it currently picks one of the candidate clusters at random. However, this behavior could be modified by implementing a custom load balancer (e.g. based on available slots). It then fetches all the configuration files for that cluster, and forks off the individual Hadoop, Hive or Pig jobs into separate individual working directories for each job (see screenshot above), thereby providing isolation between Genie and the jobs themselves. A single Genie instance can thus support multiple concurrent job submissions to different clusters, all completely abstracted out from the clients.

How do we use Genie for dynamic resource management?

Various engineering teams in Netflix run their services on reserved instances on ASGs (auto-scaling groups) on AWS, which they expand and shrink based on load. Most of the ETL jobs run after midnight (PST), which conveniently happens to be the time when most of these ASGs have scaled down (due to the natural viewing pattern of Netflix in North America). Hence, we use these surplus reserved instances to spin up additional "bonus" clusters supplementing our other production clusters. We register them with the Configuration Service, and the Genie clients (i.e. ETL jobs) access these new clusters using the Execution Service API. When the engineering teams need their instances back, the bonus clusters are terminated and de-registered, and are no longer available via Genie clients. There is no need to perform any rolling upgrades, which is common in traditional Hadoop clusters. If we need to perform an upgrade of our production (SLA) cluster, one option is to spin up a new production cluster with the upgraded software stack, and stop routing traffic to the old cluster by simply setting its status to "Out of Service" (we call this a "red-black" push). Alternatively, we can perform an upgrade on a running cluster by setting its status to "Out of Service", and temporarily marking another running cluster as an SLA cluster, while the upgrade is being performed. If we do not want running jobs to fail, we wait for them to finish before terminating the old cluster or upgrading a running cluster. This is similar to the capability provided by Asgard for application deployments and cloud instance management.

What is the current deployment status of Genie?

Although it continues to be a work in progress, we use Genie heavily in our production environment. It is currently deployed in a 6–12 node Auto Scaling Group (ASG), spanning three Availability Zones for load-balancing and fault tolerance purposes. For horizontal scalability, we adjust the number of nodes based on load (i.e. the number of concurrent running jobs). This is configured via CloudWatch alarms, coupled with auto-scaling policies managed by Asgard. Genie instances are registered with Eureka, and clients use the Eureka API to choose an available instance to run their job. To avoid client skew, Genie also forwards job requests to other lightly loaded instances, when an instance gets more requests than it can handle. Genie currently supports hundreds of concurrent job submissions at any time. On a daily basis, it runs hundreds of Hive jobs from our visualization tools and our custom Hive/Pig Web UI, plus thousands of Hive and Pig-based ETL jobs. Scaling to thousands of concurrent jobs is theoretically possible, simply by increasing the number of instances in the ASG.

Summary

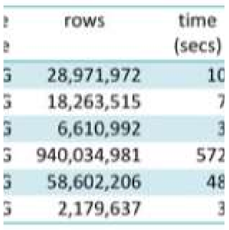
In this article, we described the cloud-based Hadoop architecture at Netflix, which is significantly different than data-center based ones. We are able to leverage the elasticity of the cloud to scale up and down, and we can spin up multiple clusters dynamically to address different kinds of workloads. We also described Genie, which is a horizontally scalable Hadoop Platform as a Service. Genie exposes APIs to submit and manage jobs, as well as manage and abstract out backend Hadoop resources. We are considering open sourcing Genie in the near future, and would love to hear your feedback on whether this might be useful in your big data environment. If building critical big data components like this sounds exciting to you (for a service that millions of people love worldwide!), take a look at http://jobs.netflix.com.

See Also:

Aegisthus—A Bulk Data Pipeline out of Cassandra

a constantly running Hadoop cluster processing sstables as they are created by any Cassandra data source

medium.com



rows	time (secs)
28,971,972	10
18,263,515	7
6,610,992	3
940,034,981	572
58,602,206	48
2,179,637	3

Asgard: Web-based Cloud Management and Deployment

Asgard, a web interface for application deployments and cloud management.

medium.com

. . .

Originally published at techblog.netflix.com on January 10, 2013.