CS429 Project Handout: Scaling Lucene with Solr

Michael Olmos and Frank Lockom

April 20, 2012

1 Intro

The Apache Solr project is a full fledged enterprise search application built on top of Lucene. Solr supports some basic functionality for scaling Lucene to multiple machines using both replication and distribution.

Amazon web services (AWS) and cloud computing in general offers a utility based model of computing where resources can be allocated dynamically and you only pay for what you use. For large scale applications this offers the ability to save resources by scaling the application up and down according to the demand at any given time. It also means a lower barrier to entry for new businesses which require a large amount of computational resources.

Critical to both these scenarios is to be able to scale an application up and down automatically (on-line) according to demand. For our project we investigate how to scale Solr, how AWS can help and how it can be done on-line with zero down-time.

2 Solr

Built on top of Lucene Solr can be used to integrate search into other applications or as fully functioning search engine out of the box. It supports a RESTful API using HTTP for performing queries and updating an index. It essentially makes using Lucene a system administrator task rather than a programming task at the expense of some power, of course.

2.1 Replication

Replication refers to running multiple instances of Solr which are servicing queries on the same index. The aim of replication is to increase throughput of queries.

Replication is accomplished using a master and multiple slaves. The master handles all updates to the index and the slaves poll the master for updates. The master is unaware of the slaves it merely services requests for updates and the slaves are unaware of eachother.

2.2 Distribution (Sharding)

Distribution refers to distribution of an index across multiple Solr instances. The aim of distribution is to decrease the latency of a single query (divide and conquer) and to handle a very large index using multiple smaller machines. Solr refers to each instance of a distrbuted index as a shard.

Distribution is accomplished by one instance relaying a query to all other instances, collecting and merging the results then sending them back to the client. Each instance in a distributed setup must be aware of all other instances so it can relay queries to them.

2.3 Distributed Indexing

When an index is broken across many shards Solr does not provide any built-in method for deciding which document should be indexed on which shard. The way in which the decision is made can effect how scaling the distribution works and vice-versa. It must also be taken into consideration that IDF values are not computed globally across shards, thus different rankings are possible with different distribution of documents.

3 Load Balancing

Load balancing refers to distributing a workload across multiple resources to achieve good utilization(latency/throughput) and avoid overload.

Solr provides no means for load balancing and so an external tool must be used. Since all interaction with a Solr instance is performed using HTTP the load balancer does not need to be coupled with Solr in any way[1].

Both distribution and replication must be load balanced. Incoming queries must be load balanced across all shards so that one shard is not responsible for all the dispatching and merging of queries to other shards. Within a shard, queries must be load balanced across all replicas so that high throughput is achieved.

3.1 HAProxy

HAProxy is a well known Load balancer which can operate at both Layer 4 (TCP) and Layer 7 (HTTP). HAproxy works as a simple forwarding service and given a list of servers will distribute incoming requests according to some rule (typically round-robin) to those servers. It will keep track of the health of its servers and divert requests away from servers which are offline or under high load. This is a very simplistic overview but is sufficient for our purposes.

[1]There could be some coupling if local state is developed, but this is not the case for a simple query

4 Amazon Web Services (AWS) and Cloud Computing

Amazon is the biggest provider of cloud Infrastructure as a Service (IaaS) services. IaaS is the lowest layer of cloud computing essentially providing bare metal or raw resources to developers. The other two layers, increasing in abstraction, are Platform as a Service (PaaS) such as Google AppEngine and Software as a Service (SaaS) such as GMail.

AWS is composed of many services addressing computation, storage, scalability and monitoring. We are looking at two in particular to scale Solr: Elastic Compute Cloud (EC2) and Elastic Load Balancing (ELB).

4.1 Elastic Compute Cloud (EC2)

EC2 is the primary service of AWS and essentially provides virtual machines on demand to customers. EC2 currently offers 13 instance types with different resources(CPU,RAM,Disk,Network,GPU) to suit applications. EC2 instances are billed by the hour at a rate dependent on the instance type. The second smallest and default instance type is "t1.small" offering 1.7GB ram 1 compute unit[2], and 160GB disk costing $0.08 per hour. The largest is "cc2.8xlarge" offering 60.5GB of RAM 88 compute units across 16 cores, 3370GB disk and 10Gb ethernet costing $2.40 per hour.

Amazon provides an api and a set of command line tools so that instances can be launched, destroyed, managed and monitored automatically. A web console is also available.

4.2 Elastic Load Balancing (ELB)

ELB is a Load balancer similar to HAProxy except that it is not restricted by a single network interface like an HAProxy server. ELB is not a piece of software installed on an instance but a service provided by AWS that can magically scale to "any" load. It is priced by $0.025 per Elastic Load Balancer-hour and $0.008 per GB of data processed by an Elastic Load Balancer.

In our case we will use ELB to balance all incoming queries across shards and HAProxy to balance requests within shards. This means as we scale up our first bottle-neck is likely the bandwidth through the network interface of the HAProxy server, limiting the number of machines in each shard. This limit is quite high though, Consider Google takes on the order of 1billion queries per day[2], so 10k per second. Considering the response size from a google search of "Google" is about 50KB and looking at the first figure in [1] we can see that HAProxy can handle about 20k connections per second with a response size of 50KB on a 10Gbit ethernet connection. Considering all this we can roughly say that we will not reach this bottle-neck. We can also conclude that we could use

[2] 1 compute unit is roughly defined by Amazon to be a 1.0-1.2GHz Opteron or Xeon from 2007

HAProxy instead of ELB. ELB will have a higher availability however and is likely less expensive.

5 Online Scaling, up and down

Here we consider a simple solution for scaling our described Solr cluster up and down. There could be many such solutions but this is the one we attempt to implement for simplicity.

5.1 replication

Changing the level of replication is fairly simple. To scale up:

1. allocate a machine using the EC2 API, use an image with solr pre-installed.
2. set its Solr configuration to point to the master server in its shard, wait until it downloads the index.
3. append the server to the shard's HAProxy config file and reload HAProxy (can be done with zero down-time)

A similar method can be used to scale down.

5.2 distribution

Changing the level of distribution is not so simple. First recall there are two goals of increasing distribution: reducing latency and accommodating a larger index.

If we have a constant size index and we want to reduce the latency we must move some of the documents from all the shards to the new one.If our index is growing then we do not necessarily need to do this. However we must then make our Indexer aware that all new documents should be directed to the new shard.

So we have two ways to increase distribution

1. move some documents from the old shards to the new one
2. index future documents to the new shard

The first method is most difficult to implement and has a high initial cost. The second method requires the Indexer to be more aware of the size of each shard.

There is an additional problem with the second method. IDF values are not calculated globally across shards. Since it is likely that documents indexed temporally close are similar, IDF values could become skewed on a new shard resulting in poor result ranking.

Additionally when downsizing we must move all documents on the shards to be removed to the other shards, thus we cannot avoid implementing the difficult part of the first method.

We will not give the steps here since we have not finalized a method but note that the difficulty lies in moving so many documents around while keeping the system on-line (and keeping results consistent). Changing the replication with method 1 will cause documents to be added or removed from every machine in the system. We are considering using a two step process where half the machines are converted to the new configuration while half continue servicing requests. Once finished the remaining half are updated.

References

[1] Willy Tarreau, 10GbE load-balancing with HAProxy, http://haproxy.1wt.eu/10g.html , 08/23/2009

[2] Matt McGee, By The Numbers: Twitter Vs. Facebook Vs. Google Buzz, http://searchengineland.com/by-the-numbers-twitter-vs-facebook-vs-google-buzz-36709, 02/23/2010