Building a Scalable News Feed Web Service in Clojure

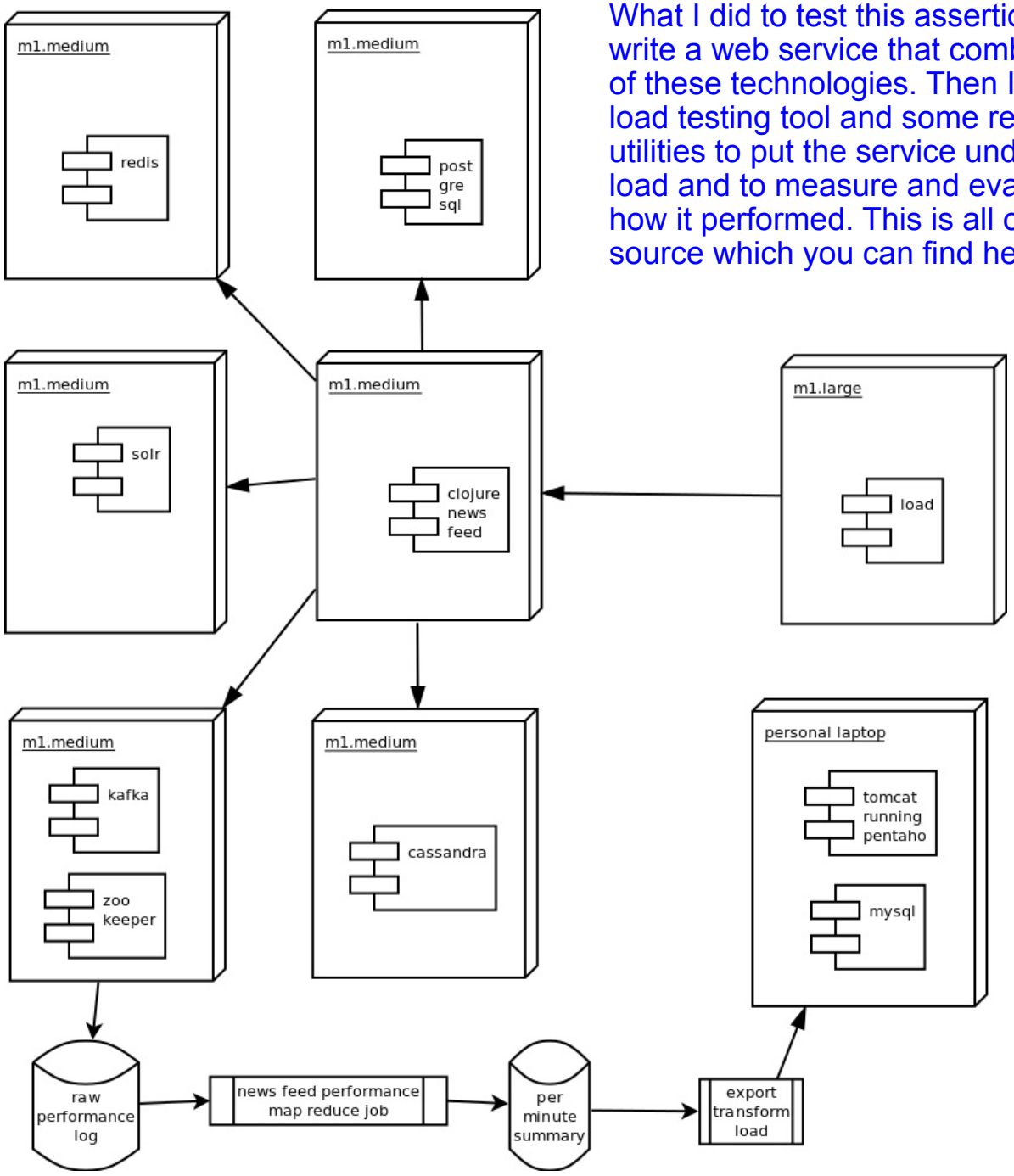
published 2014 by Glenn Engstrand

This is a good time to be in software. The Internet has made communications between computers and people extremely affordable, even at scale. Cloud computing has opened up technology possibilities and capabilities to people and organizations once thought to be too small to even dream of affording it. The open source software movement and component based architectures have made it incredibly easy for individual engineers/entrepreneurs or small technology startups to create great products with little or no funding.

Most experts agree that the commodifying influence of the Internet, the cloud, and open source has lowered cost based barriers to entry but what about quality? Do these technologies deliver reliable and accurate capability at scale? That was what I wanted to discover when I created this basic news feed service using recent and popular open source projects and tested its load capabilities when running on Amazon's public cloud.

Here are the open source projects that I put to the test. Clojure is a Functional Programming language built for the Java Virtual Machine. Cassandra is a column oriented big data NoSql data store well suited for storing large volumes of time series data. PostGreSql is a fully ACID compliant relational database management system. Solr is a web service with the popular search engine Lucene embedded within it. Kafka is an asynchronous messaging service well suited to aggregate event logs. Redis is an in-memory key-value data store most often used as a write behind cache.

https://github.com/gengstrand/clojure-news-feed



What I did to test this assertion was to write a web service that combined all of these technologies. Then I wrote a load testing tool and some reporting utilities to put the service under the load and to measure and evaluate how it performed. This is all open source which you can find here.

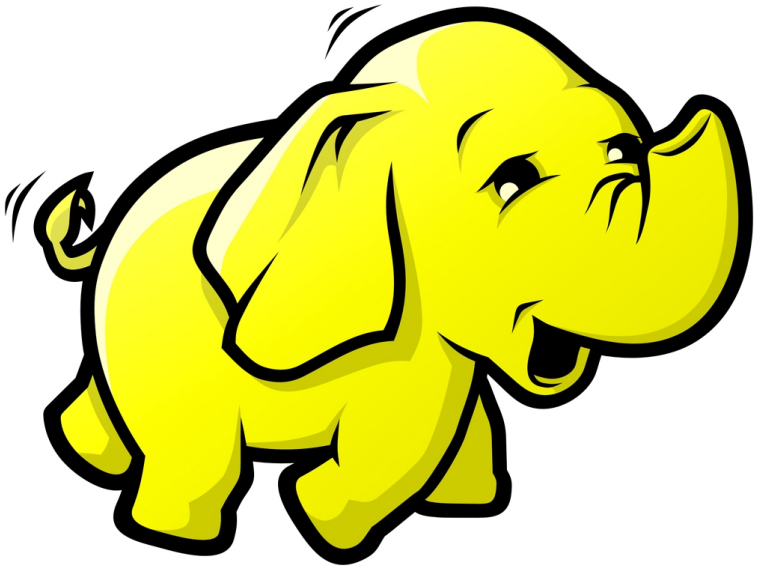
Clojure is a Functional Programming language which means that it is not easy to modify the state of objects once they have been created. One very important capability any web service needs is its ability to report its own performance metrics if queried to do so. On Java based services, that is done via a technology called JMX. This is not easy for Clojure to do so I wrote some Java code to do this for the proposed news feed web service. The support project is where that code is organized. Clojure and Java inter-operate very easily so it is no big deal to call Java code from Clojure. The support project also wraps access to the SolrJ API.

I wanted the outbound activity content to be keyword searchable which is where Solr comes in. The solr sub-project contains all the configuration files needed to index the outbound feed.

If you have ever used Facebook, then you should already be familiar with what a news feed is. This news feed is very basic. Participants are linked to each other through what is known as a social graph. A participant's activity is captured in his outbound feed. When a participant posts an activity, his friends (i.e. who he is directly connected to on his social graph) receive a copy of this activity in their inbound feeds.

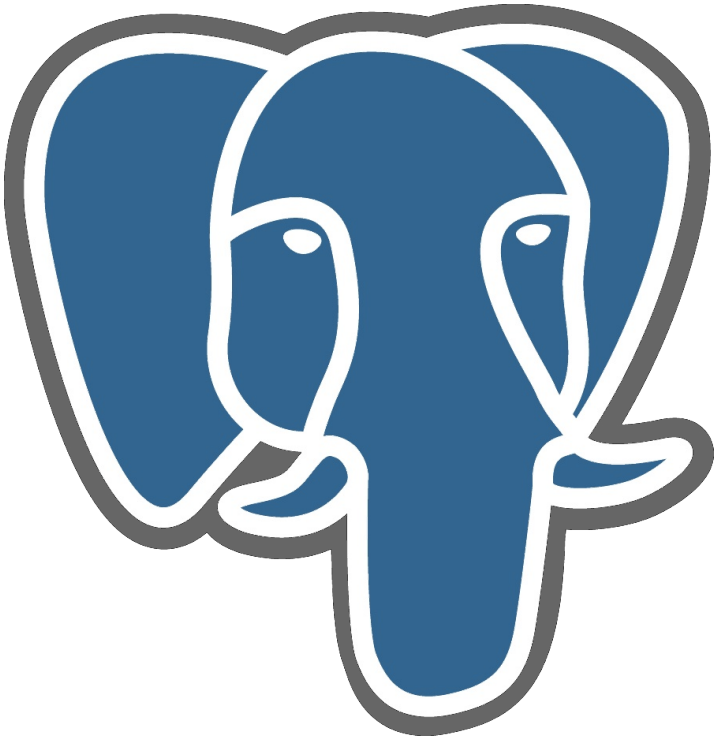
The affordability of open source is good news for small business. The bad news is that it is all about up sell in the cloud.

The feed project is where all the Clojure code, that implements the web service, is organized. Here are the main components to the feed service. The cache redis, cassandra, messaging kafka, mysql, postgres, and search name spaces all provide bindings to their respective technologies. The rdbms name space provides routing logic to either mysql or postgres depending on the configuration which is loaded with the settings name space code. This service does use connection pooling to the RDBMS which is currently set to 50 maximum pool size. The metrics name space provides bindings to the JMX part of the support project. The core component brings it all together with the ability to either load or store data from the underlying technology. It also logs performance data, both to JMX and to the Kafka feed topic. The handler name space maps HTTP requests to the proper functions in the core name space. For saves, I would use Clojure's support for polymorphism and the multi-method dispatch mechanism. For loads, I would use Clojure's support for higher order functions.



The news feed performance project is where all the Java code, that implements a Hadoop map reduce job for aggregating the performance data from the news feed web service, is organized.

The etl project contains a command line utility that takes the per minute aggregated output from the news feed performance map reduce job and populates a mysql database ready for access via the open source OLAP project Mondrian.



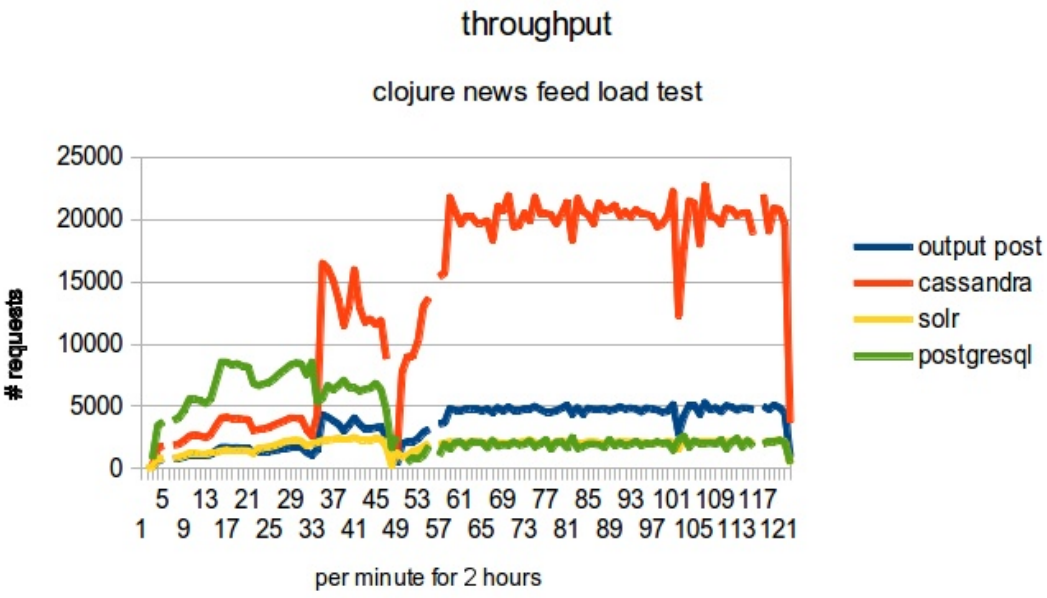
The load project contains the Clojure code that implements the load applications. This command line tool simulates a specified number of user sessions that create accounts, connect these accounts to each other socially, then post outbound activity. This is all done by calling the various end points to the web service.

Once the code was written, it was time to set the environment up in order to perform the load test. I used 5 m1.medium instances from EC2 and 1 db.m1.medium instance from RDS. What got installed on the EC2 instances was Cassandra, Solr, Kafka, Redis, and Jetty.

Each service got their own server but I installed ZooKeeper on the same instance as Kafka because Kafka depends on ZooKeeper. Solr can be set up as either single core or multi-core. I chose mult-core for its increased flexibility. The news feed service can run with either MySql or PostGreSql but I choose RDS running PostGreSql for the load test.

The news feed web service handles HTTP requests with Compojure which is a routing library for Ring which is a web applications library that can be run to integrate with Jetty. You can embed Jetty inside the JAR file which serves as the build artifact for the project. That is how I chose to build and release this experiment.

I ended up using the defaults for most of the configurations of these services. I already explained what you need to do to the Solr configuration. Cassandra is the other service that needs a tweak or two in its cassandra.yaml file. The rpc_address and listen_address will need to be set to the external IP of the host where Cassandra is running. Because the news feed service is using the CQL driver and not the Hector client, you will also need to set start_native_transport to true.



If you do decide to deploy this service to the cloud, then you will need to extensively change the feed/etc/config.clj file which contains all the connection settings to the various servers.

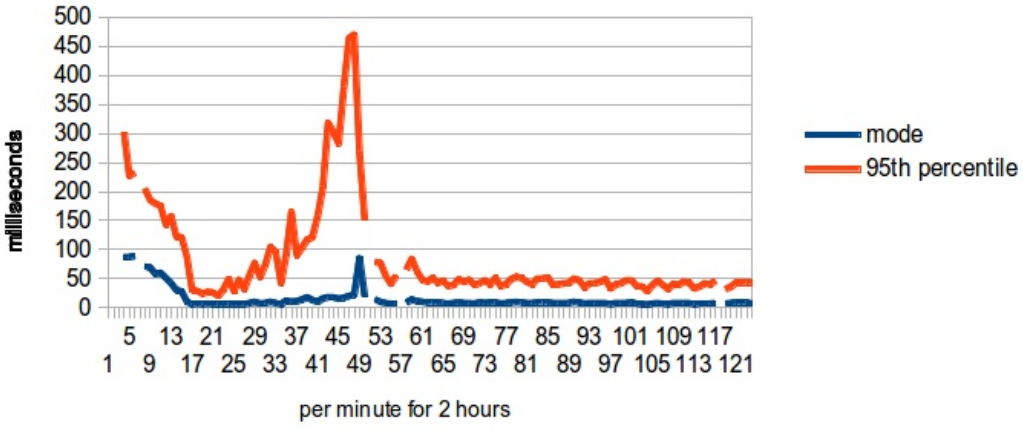
The defaults in that file are all localhost which is fine if you are just exploring the service for yourself. See the various README and docs/intro markdown files for much more detailed information.

Now that the service is all set up and running, how do we test its capacity at load? First thing to do is to run the load test application which is a command line tool, written in Clojure, that simulates a concurrent number of sessions. Some percentage of those sessions are just performing keyword search.

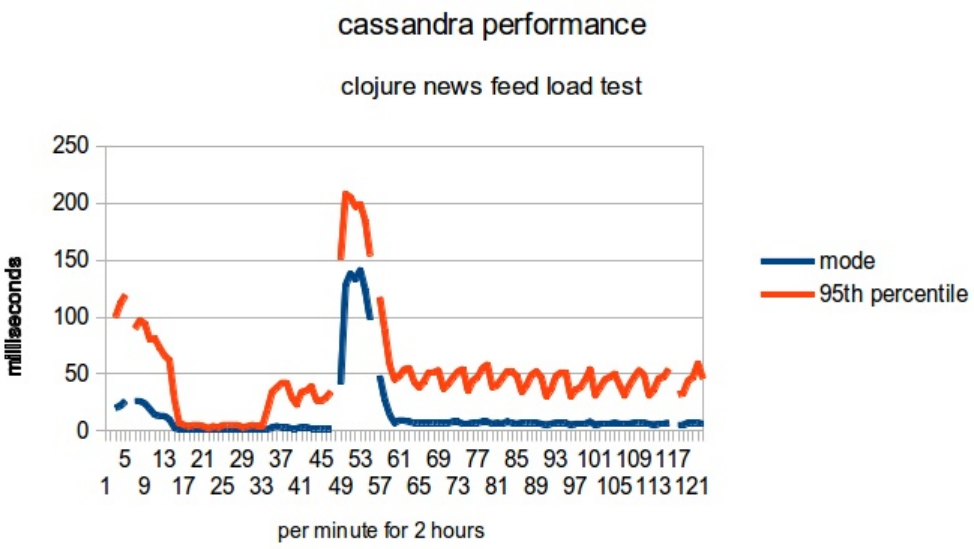
Configuration parameters for the load test application include host, port, participate batch size, minimum number of friends, maximum number of friends, how many words for the subject line, how many words in the activity content, how many activity posts per user, and how many searches per user. I ran this test with 100 concurrent users and 10% searches. The load application got to run on its own m1.large EC 2 instance.

Now that the load test is running, how do we collect the performance data? As mentioned previously, the service logs this data to Kafka which provides command line tools to read the data and redirect it to a text file. Once the test concluded, I transferred that file to Hadoop where I ran the map reduce job, from the news feed performance project, which input that raw performance information and output a text file which contains the per minute collection of performance metrics (throughput, mode, and 95th percentile) per entity (Participant, Friends, Inbound, Outbound) and activity (load, store, post, and search).

The load test has completed and we have a text file containing all the performance metrics. How do we access and analyze this data? I used the etl Clojure project (i.e. Extract, Transform, and Load) to input the summarized performance metrics file into a MySql database ready for access by a Mondrian OLAP cube.



You have to create the database and set up appropriate credentials. Run the etl/etc/starschema.sql script to create the tables and stored procedures that this etl program needs. Modify the etc/src/etc/core.clj program to connect to the database with the proper credentials. See the etl/doc/intro.md file for details on how to set up Mondrian with the news feed performance cube data.



Now that the data has been collected and analyzed, what were the findings? Performance at the beginning was not so good. That was because the Redis cache was cold. Once the cache warmed up, performance was great. Near the end of the first hour, performance really bottomed out briefly.

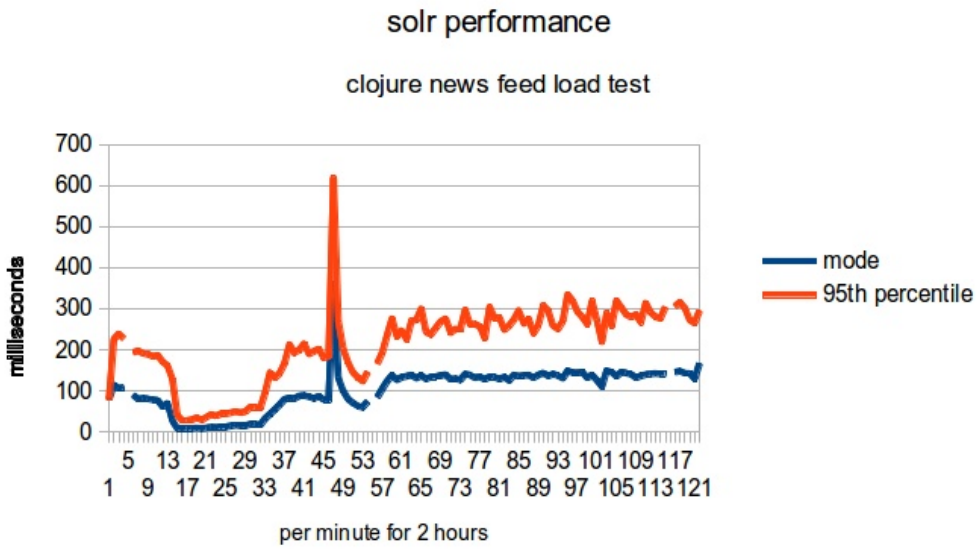
I suspect that the cause of the big spike can be attributed to the very nature of cloud computing itself.

I provisioned and ran the server side software on five EC 2 instances but is that really five different servers? No. Cloud computing uses virtualization. I sshed to each 64 bit Ubuntu LTS box but that is actually just the guest OS running along side other guest OS images on some host server that I never see. On the cloud, spikes in performance are more likely to be attributed to increases in activity in the images that reside along side yours on the real hardware. You can purchase instances with provisioned guarantees of input/output processing per second but that costs more. I did not do that for this test.

At first glance, it looks like Cassandra writes and Postgres writes have about the same latency. The steady-state numbers for mode are 10 ms for Postgres and 8 ms for Cassandra. The 95th percentiles are 50 ms for Postgres and 47 ms for Cassandra. What you have to take into account is that Cassandra was handling 14 times more writes than Postgres.

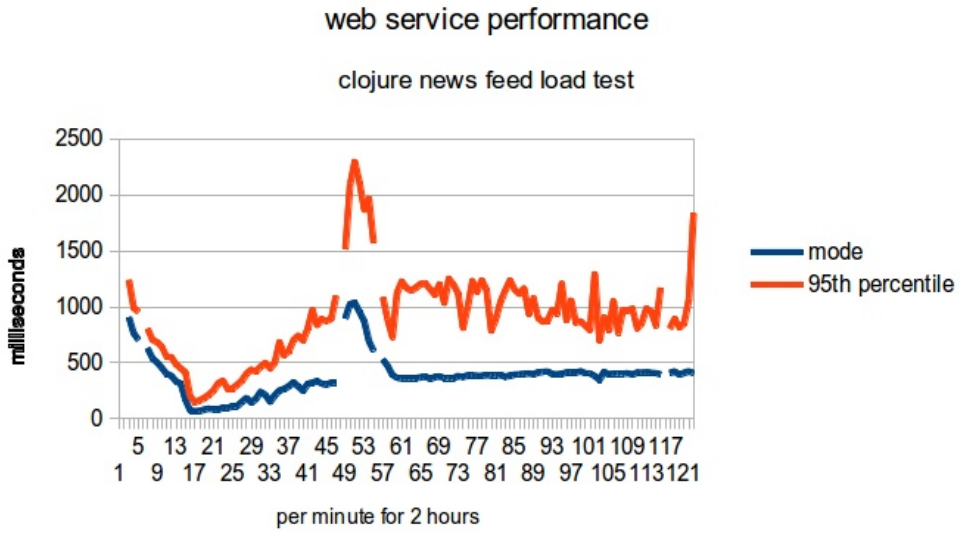
Solr performance is very consistent. It was handling 33 requests per second with a very steady 135 ms per request for the mode and 213 ms for the 95th percentile. Compare that with the 5X difference in the other data stores.

The second hour of the load test had consistent performance metrics so I will base most of my findings on that information alone.



Overall transaction throughput for posting outbound activity is a very respectable 58 to 83 transactions per second with a latency whose mode is 393 ms and whose 95th percentile is about 1 second. That is not bad at all considering how this is with one web server which you could easily scale out.

Does the Internet, cloud computing and open source really disrupt entrepreneurism by providing lean startups with the capability to serve large markets on a budget? Well, we have some mixed results here. The Internet is clearly disruptive in that consumers will gladly pay to benefit from its networked effects. Open source is a big win for small business in that this 1400 lines of code that one guy wrote over the holidays could easily deliver non-trivial news feed capability at a scale that the fortune 500 would gladly pay permium for merely a decade ago. Cloud computing; however, is a different story. The good news is that it cost me only $10 to run this test. That's about as commodity as it gets. The bad news is that it is all about up sell in the cloud. You would need three times as many servers running on at least two availability zones for a credible High Availability story.



Increase the RAM and CPUs and start provisioning IOPs and you will quickly find that the cloud is not about saving money, especially once you have achieved some success and have a customer base to protect.