**How to build Java based Cloud Application**

Posted by [sgdev-blog](https://weblogs.java.net/blog/sgdev-blog) on May 20, 2014 at 8:23 AM PDT

[](http://2.bp.blogspot.com/-AsmQeE-ci2o/U3DshEOOAOI/AAAAAAAAAq0/1u-ItVOoq-c/s1600/cloud-computing-characters.jpg)Recently, we were tasked to develop a [SAAS](http://en.wikipedia.org/wiki/Software_as_a_service) application for big data analysis. To do data mining, the system need to store multi billion public posts in the database and run the classification process on them.

Classification in our context is a slow, resource intensive and painful process to assign a topic or sentiment to any record in the database. The process can last up to 24 hours with our testing data.

To cope with these requirements, our obvious choice is to build a cloud application on Amazon Web Services. After working on the project for a while, I want to share my own thought, understanding and approach to build Java based cloud application.

**What is Cloud Computing**

Let start with Wikipedia first:

"[*Cloud computing involves distributed computing over a network, where a program or application may run on many connected computers at the same time*.](http://en.wikipedia.org/wiki/Cloud_computing)"

The definition may be a bit ambiguous but it is understandable as *In The Cloud* itself is more of a marketing term rather than technical term. For a newbie, it is easier to understand if we define it with a more practical way:

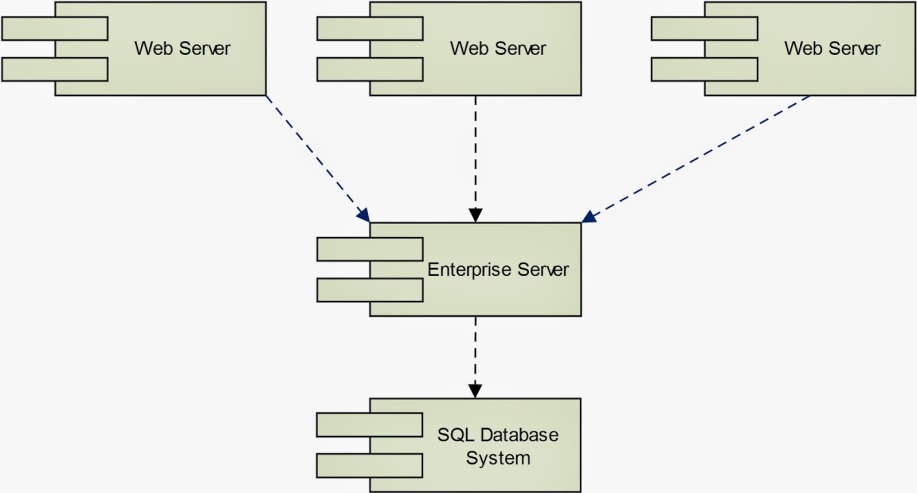
*The only difference between traditional web application with the cloud web application is the ability to scale perfectly. Cloud application should be able to cope with unlimited amount of works given unlimited hardware.*  
  
Cloud application is getting popular nowadays because of higher requirement for modern application. In the past, Google is famous for building high scale application that contains almost all available information in the internet. However, for now, many other corporates need to build applications that serve similar scale of data and computation (Facebook, Youtube, LinkedIn, Twitter,.. and also the people who crawl and process their data like us).

This amount of data and processing cannot be achieved with the traditional way of developing application. That lead us to an entirely different approach to build application that can scale very well. This is cloud application.

**Why traditional approach of developing web application does not scale well enough**

**Traditional Approach of developing web application**

Let take a look on why traditional application cannot serve that scale of data.

[](http://4.bp.blogspot.com/-9r6USt5q5LU/U2-iE-NXKBI/AAAAAAAAAqg/ieXFDs9zeYM/s1600/traditional_application.jpg)

If you have developed one traditional web application, it should be pretty much similar to the diagram above. There are some other minor variations as merging of application server and web server or multiple enterprise servers. However, most of the time, the database is relational. Web servers are normally stateful while enterprise servers can serve both stateless and stateful services.

There are some crucial weaknesses that cause this architect does not scale well enough. Let start our analysis with defining perfect scalability first.

*Perfect scalability can be achieved if a system can always provide identical response time for double amount of work given double amount of bandwidth and double amount of hardware.*

Perfect scalability cannot be achieved in real life. Rather, developers only aim to achieve near perfect scalability. For example, DNS servers are out of our control. Hence, theoretically, we cannot serve higher amount of requests than the DNS servers. This is the upper bound for any system, even Google.

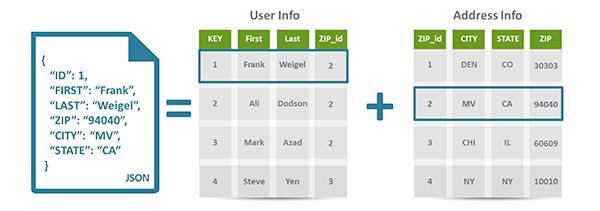
**SQL**

Come back to the diagram above, the biggest weakness is the database scalability. When the amount of requests and size of data are small enough, developers should not notice any performance impact when increasing load. Continue to increase the load higher, the impact can be very obvious, if the CPU is 100% utilized or memory fully occupied. At this point, the most realistic option is to pump more memory and CPU to the database system. After this, the system may perform well again.

Unfortunately, this approach cannot be repeated forever whenever problems arise. There will be a limit where no matter how much ram and CPU you have, performance will slowly getting worse. This is expectable because you will have some certain records that need to be*create, read, update, delete (CRUD)* by many requests. No matter whether you choose to cache them, store them on memory or do whatever trick, they are unique records, persisting in a single machine and there is a limit on amount of access requests that can be sent to a single memory address.

This is the unavoidable limit as SQL is built for integrity. To ensure integrity, it is necessary that any information in SQL server should be unique. This characteristic still applicable even after data segregation or replication are done (at least for the primary instance).

In contrast, NoSQL does not attempt to normalize data. Instead, it chooses to store the aggregate objects, which may contain duplicated information. Therefore, NoSQL is only applicable if data integrity is not compulsory.

[](http://www.couchbase.com/sites/default/files/uploads/all/images/WhyNoSQL/Figure5a.png)

Above example (from [couchbase.com](http://www.couchbase.com/why-nosql/nosql-database)) shows how data is stored in a document database versus relational database. If a family contains many members, relational database only store a single address for all of them while NoSQL database simply replicate the housing address. When a family relocate, the housing addresses of all members may not be updated in a single transaction, which cause data integrity violation.

However, for our application and many others, this temporary violation is acceptable. For example, you may not need the amount of page views on your social page or amount of public posts in a social website to be 100% accurate.

Data duplication effectively removes the concurrent access to a single memory address that we mentioned above and give developers the option to store data anywhere they want, as long as the changes in one node can be slowly synced up to other nodes. This architect is much more scalable.

**Stateful**

The next problem is stateful service. Stateful service requires the same set of hardware to serve requests from the same client. When the amount of clients increase, the best possible move is to deploy more application servers and web servers into the system. However, the resource allocation cannot be fully optimized with stateful services.

For traditional applications, load balancer does not have any information of system load and normally spread the requests to different servers using [Round Robin technique](http://en.wikipedia.org/wiki/Round-robin_DNS). The problem here is not all requests are equals and not all clients are sending identical amount of requests. That cause some servers are heavily overloaded while others are still idle.

**Mixing of data retrieval and processing**

For traditional applications, the server that retrieve data from database ends up processing it. There is no clear separation of processing data and retrieving data. Both of the two tasks can cause bottle neck to the system. If the bottle neck come from data retrieval, data processing is under-utilized and vice versa.

**Rethinking best approaches to build scalable application**

Look at what have been adopted in our IT fields recently, I hardly found them as new inventions. Rather, they are adoption of the practices that have been used succesfully in real life to solve scalability issue. To illustrate this, let imagine a real life situation of tackling scalability issue.

**Hospital**

Assume that we have a small hospital. For our hospital, we mostly serve loyal customers. Each loyal customer have a personal doctor, who keeps track of his/her medical record. Because of this, customers only need to show the ICs to be served by the preferred doctors.

To make things challenging, our hospital is functioning before the internet era.

**Stateless versus stateful**

Is the description above look similar enough to stateful service? Now, your hospital is getting famous and the amount of customers suddenly surges. Provide that you have enough infrastructure, the obvious option is to hire more doctors and nurses. However, customers are not willing to try out new doctors. That cause the new staffs are free while old staffs are busy.

To ensure optimization, you choose to change the hospital policy so that the customers must keep their medical records and the hospital will assign them to any available doctors. This new practice helps to resolve all of your headache and give you the option to deploy more seasonal staffs to cope with sudden surge of clients.

Well, this policy may not make the customers happy but for IT fields, stateless and stateful services provide identical results.

**Data Duplication**

Let say the amount of customers constantly surge and you start to consider opening more branches. At the same time, there is a new rising problem that customers constantly complain about the need of bringing medical records while visiting hospital.

To solve this problem, you come back to the original policy of storing the medical records at the hospital. However, as you are having more than one branch, each branch need to store a copy of user medical records. At the end of the day or the week, any record change need to be synced to every branch.

**Separation of Services**

After running the hospital for a few months, you recognize that the resources allocation are not very optimized. For example, you have blood test and X-ray faculty in both branch A and B. However, there are many customer doing blood test in branch A and many people taking X-ray in branch B.

It cause the customers keep waiting in one branch, while no one visit the other branch. To optimize resource, you shutdown the under-utilized faculties and setup unique blood test centre and X-ray centre. Customers will be sent from the branches to the specialized centres for special services.

**Adhoc Resource**

It is hard to do resource planning for hospital. There are seasonal diseases that only happens at a certain time of the year. Moreover, catastrophe may happen any time. They cause sudden surge of warded patients for a short period. To cope with this, you may want to sign agreement with the city council to temporarily rent facilities when needed and hire more part-time staffs.

**Apply these ideas to build cloud application**

Now, after looking at the example above, you may feel that most of the ideas make sense. It only take a short while before developers start to apply these ideas into building web application.

Then, we move to the cloud application era.

**How to build cloud application**

To build a cloud application, we need to find way to apply the mentioned ideas into our application. Here is my suggest approach

**Infrastructure**

If you start to think about building cloud application, infrastructure is the first concern. If your platform does not support adhoc resource (dynamically bursting of existing server spec or spawning new instance), it is very hard to build cloud application.

At the moment, we choose [AWS](https://aws.amazon.com/) because it is the most matured platform in the market. We have moved from internal hosting to AWS hosting one year ago due to some major benefits

* **Mutiple Locations**: Our customers are coming from all 5 continents, using Amazon Region, we can deploy the instance closer to customer location, through that, reduce the response time.
* **Monitoring & Auto Scaling**: Amazon offers quite a decent monitoring service for their platform. Due to server load, it is possible to do Auto Scaling.
* **Content Delivery Network**: Amazon CloudFront give us the options to offload static contents from our main deployment, [which will improve page load time](http://sgdev-blog.blogspot.sg/2014/01/maximum-concurrent-connection-to-same.html). Similar to normal instances, static contents can be served from the nearest instances to customer.
* **Synchronized & Distributed Caching**: [MemCache](http://memcached.org/) has been our preferred caching solution over the years. However, one major concern is the lack of support for synchronization among the nodes. Amazon Elastic Cache give us the option to use MemCache without worrying about node synchronization
* **Management API**: This is one major advantage. Recently, we start to make use of Management API to spawn up instance for a short while to run integration test.

**Database**

Provide that you have select the platform for developing cloud application, the next step should be selecting the right database for your system. The first decision you need to make is whether SQL or NoSQL is the right choice for your system. If the system is not data intensive, SQL should be fine, if the reverse is true, you should consider NoSQL.

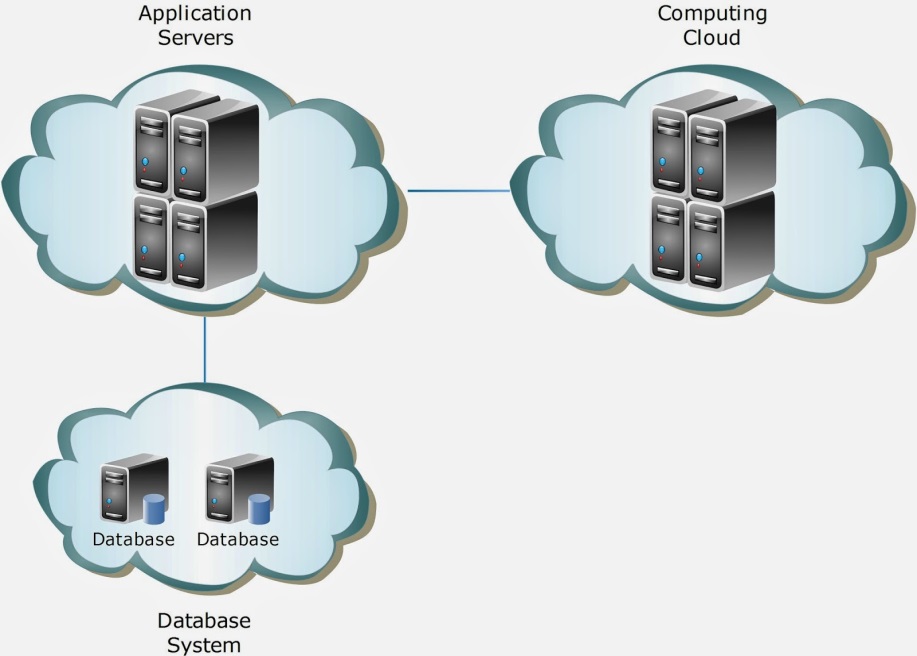
Sometimes, multiple databases can be used together. For example, if we want to implement a Social Network application like Facebook, it is possible to store system settings or even user profiles in SQL database. In contrast, user posts must be stored in the NoSQL database due to huge volume of data. Moreover, we can choose SOLR to store public posts due to strong searching capability and Mongo DB for storing of user activities.

If possible, please choose the database system that support clustering, data segregation and load balancing. If not, you may end up implement all of these features yourself. For example, SOLR should be the better choice compare to Lucene unless we want to do our own data segregation.

**Computing Intensive or Data Intensive**

It is better if we know that the system is data intensive or computing intensive. For example, Social Network like Facebook is pretty much data intensive while our big data analysis are both data intensive and computing intensive.

For data intensive system, we can let any node in the cloud retrieve data and do processing as well. For computing intensive node, it is better to split out data retrieval and data processing.

[](http://2.bp.blogspot.com/-SQyULcNVJtY/U3Dv96l0quI/AAAAAAAAArA/gQRu-XDUj2o/s1600/cloud_computing.jpg)

Data intensive system normally serve real-time data while computing intensive system run the background jobs to process data. Mixing these two heavy tasks in the same environment may end up reducing system effectiveness.

For computing cloud, it is better to have a framework to monitor load, distribute tasks and collect results at the end of computing process. If you do not need the processing to be real time, [Hadoop](http://hadoop.apache.org/) is the best choice in the market. If real time computation is required, please consider [Apache Storm](http://storm.incubator.apache.org/).

**Design Pattern for Cloud Application**

To build a successful Cloud Application, there are something that we should keep in mind.

1. **Stateless**

It is a must to make all your services and server stateless. If the service need user data, include them as parameter in the API.

It is worth noticed that to implement Stateless Session on Web Server, we have a few choices to consider:

* Cookie based session
* Distributed Cache session
* Database Session

The solutions above are sorted from up to down with lower scalability but easier management.

2. [**Idempotence**](http://en.wikipedia.org/wiki/Idempotence)

For Cloud Application, most of the API call will happen through the network rather than internal method calls. Therefore, it is better if we can make the method calls safe. If you stick to the Stateless principle above, it is  likely that the services you implement are already idempotent.

3. [**Remote Facade**](http://martinfowler.com/eaaCatalog/remoteFacade.html)

Remote Facade is different with [Facade](http://en.wikipedia.org/wiki/Facade_pattern) pattern. They may look similar in term of practice but aim to fix different problems. As most of your API calls happen over the network, the network latency contribute a great part to the response time. With Remote Facade pattern, developers should build a coarse-grained API so that the amount of calls can be reduced.

In layman's terms, it is better to go to supermarket and buy 10 things in one shot rather than visit 10 times, each time buy 1 thing.

4. **Data Access Object**

As you may transfer the data around, be careful with the amount of data you transfer. It is best to only give the minimum data as required.

5. **Play Safe**

This is not a design pattern but you will thanks yourself for playing safe in the future. Due to the nature of distributed computing, when something go wrong, it is very difficult to find out which part is wrong. If possible, implement health check, ping, thoroughly logging, debug mode to every component in the system.

**Conclusion**

I hope this approach to build Cloud Application can bring some benefit to everyone. If you have other opinions or experience, kindly feedback and share with us.

In the next article, I will share the design of our Social Monitoring Tool.