SERVER-SIDE FRAMEWORK COMPARISON

Rahul Parekh (rbparekh), Nikhil Aravind (naravind), Manasa Mannava (mmannava) and Aaditya Maheshwari (amahesh3)

1. Abstract

Server-side frameworks have provided an efficient way of developing web applications. With the increase in popularity of client-side frameworks, the traditional way of developing web applications purely on the server with presentation layer on the client side has been affected. This paper traces the rise of server side frameworks, their evolution, what they might be like in the future and provides a decision framework for choosing between server-side and client-side frameworks for web application development.

2. Introduction

Most web applications contain a certain number of repetitive tasks like data persistence, authentication, session management, caching, templating and code structuring. Over the years, developers started developing their own project specific structures which they followed internally. This lead to the rise of web application frameworks which helped developers save significant time building websites, web applications and web services [1]. Traditionally, these frameworks have been built on top of server side technologies and languages like Java, Python, Perl, Ruby, PHP etc. and are known as server side frameworks.

Some of the earliest server side frameworks were developed on Java platform. J2EE provides standardizations for interaction with many low level system services but does not provide a successful application programming model. APIs like Java Transaction API, Java Database Connectivity effectively shields the underlying wire protocols. However, they provide a cumbersome programming model with a rich potential for errors. Most large J2EE projects have traditionally used in-house frameworks that solve problems specific to that project. This generated a need of generic frameworks that address generic problems with the concept of division between application code and framework code. J2EE itself defined several frameworks, such as Enterprise Java Beans (EJB's) [2], MVC frameworks like Struts which was first released in 1999 [3] and the Java-based Rails-like development framework, Grails [4] which was released in 2005 [3].

Popular frameworks written in languages other than Java include Django for Python which began developing in Fall 2003 [5] and Ruby on Rails which was first released in 2004 [6].
All these frameworks rely on concepts such as, the application developer writes code which the framework calls at runtime. This is often called as the Hollywood principle – "Don't call us, we'll call you". The advantages that such frameworks provided are that the developer only needs to write code that he needs. They do not have to concern themselves with the complexities of the underlying API's. This is one of the key value propositions of server side framework development. Additionally, a well-defined framework provides a clear structure which makes it

easier for other developers to join the project. A good framework provides best practices and helps set industry standards. [2]

JavaScript is the most popular programming language for the browser today [7]. This has seen a rise in client side frameworks like Ember.js, AngularJS, Backbone.js, jQuery etc over the last few years.

3. Popularity

Since the late 90's, server-side frameworks have always been used by most websites and applications, be it enterprise level Java frameworks, the more recent open source frameworks like Django and Grails or even frameworks like Ruby on Rails which was extracted from Basecamp's internal codebase[8]. Since the web application development domain changes and evolves extremely quickly, keeping track of the popularity of frameworks is extremely important to understand, predict and stay on top of future trends.

Using extremely popular and widely used websites by web developers like Google.com, Github.com, Stackoveflow.com and Youtube.com, one can gauge the popularity of the open source frameworks and how the development community is adapting to and using these frameworks.

The following table shows total number of questions asked on Stackoverflow.com questions till date, Stars and Contributors each official repository has currently on Github.com and the approximate number of search results on YouTube.com.

Metric	Django	Spring	Ruby on Rails	Grails
Stackoverflow Questions	93,323 [10]	62, 704 [9]	200,607 [12]	21,269 [11]
Stars on Github	13,850 [14]	5,436 [13]	25,748 [16]	1,288 [15]
Github Contributors	838 [14]	116 [13]	2675 [16]	154 [15]
Youtube Results	~31,500 [18]	~120,000 [17]	~80,600 [20]	~3,770 [19]

Table 3.1: Popularity Metrics for server side frameworks

Here are Google.com's search statistics for these frameworks from 2004 to 2015[21]:

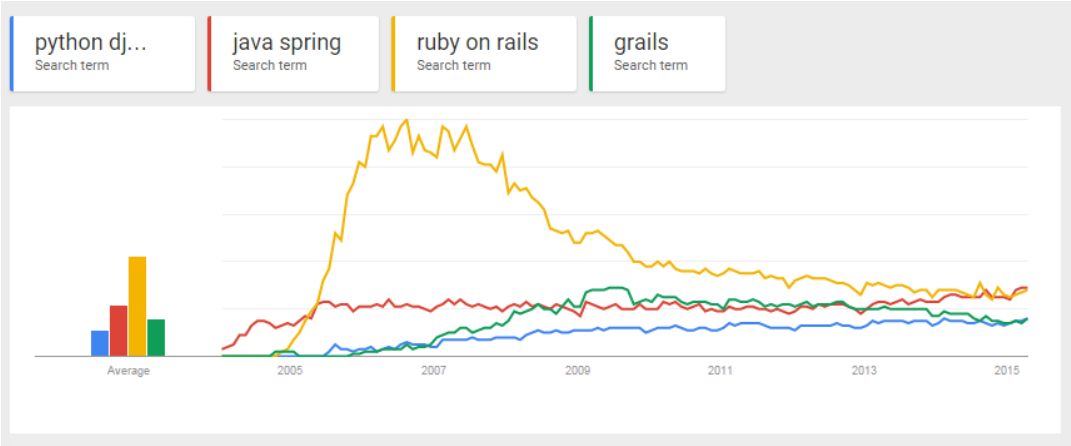

Figure 3.1: Google Statistics – Frameworks Search Term (Server side)

Based on the statistics above, we observe:

- **Django** – It kept gaining popularity at a steady rate from its release date and has more or less steadied over the last few years. It is also one of most actively contributed to and watched frameworks based on the Github.com statistics.
- **Spring –** It was released in the same year as Django [5][22] and has largely been similar in terms of popularity. It has less contributors and people following the official GitHub.com repository than Django but has seen increase in popularity which coincided with the release of major v4.0 in 2013 [23]
- **Ruby on Rails –** Rails showed the largest growth and the sharpest decline in popularity. It is still one of the most popular frameworks and has by far the highest number of contributors and followers on Github.com with the most questions asked on Stackoverflow.com.
- **Grails –** It never achieved the same popularity as the other Java based Spring or Rails – the framework which it was modeled around. It has seen a decrease in number of Google searches. It has lowest number of search results on YouTube as well as questions on Stackoverflow.com and stars on Github.com which suggests lack of adaptability in the development community.

3.1. Client Side Frameworks

With the increase in computing power and introduction of smartphones, web browsers can do much more with the web. In 2009, the W3C community started working on HTML5 specification [24] which introduced a lot of APIs like WebSockets, Media, Web storage, Time API etc. The same year, NodeJS was released – a JavaScript environment for server side applications [25]. This meant that developers could now code in JavaScript on the client-side as well as server-side leading to a more uniform code base.

All these things allowed developers to move more code onto the client side and reduce reliance on traditional server side processing to do many tasks. This lead to the launch of many client-side JavaScript web frameworks like AngularJS, Backbone.js, Ember.js

Let's look at the same metrics which we saw for the server side frameworks.

Metric	AngularJS	Backbone.js	Ember.js
Stackoverflow Questions	89,900 [26]	17,806 [27]	14,823 [28]
Stars on Github	37,695 [29]	21,483 [30]	13,464 [31]
Github Contributors	1,219 [29]	255 [30]	476 [31]
Youtube Results	~120,000 [32]	~15,900 [33]	~10,200 [34]

Table 3.1.1 Popularity Metrics for Client Side Frameworks

As we can see, AngularJS is by far the most popular framework and the other two frameworks also have a lot of contributors and stars on Github.com despite being released relatively recently compared to the server side frameworks we saw previously. AngularJS's first stable release in 2010 [35], Backbone.js was also released in 2010[36] and Ember.js was released in 2011 [37].

Here are Google.com's search statistics for these frameworks from 2004 to 2015[38]:

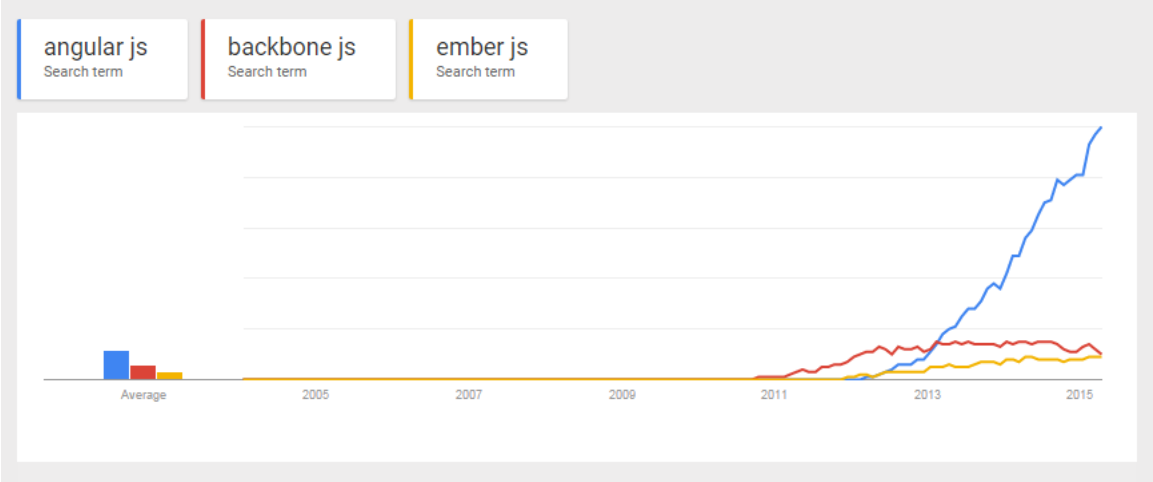

Figure 3.1.1: Google Statistics – Frameworks Search Term (Client side)

These statistics show extremely high popularity for Angular JS and relatively good popularity for the other two.

Let's see how the most popular JS framework Angular JS fares against the most popular server side framework Ruby on Rails in terms of searches [39]:

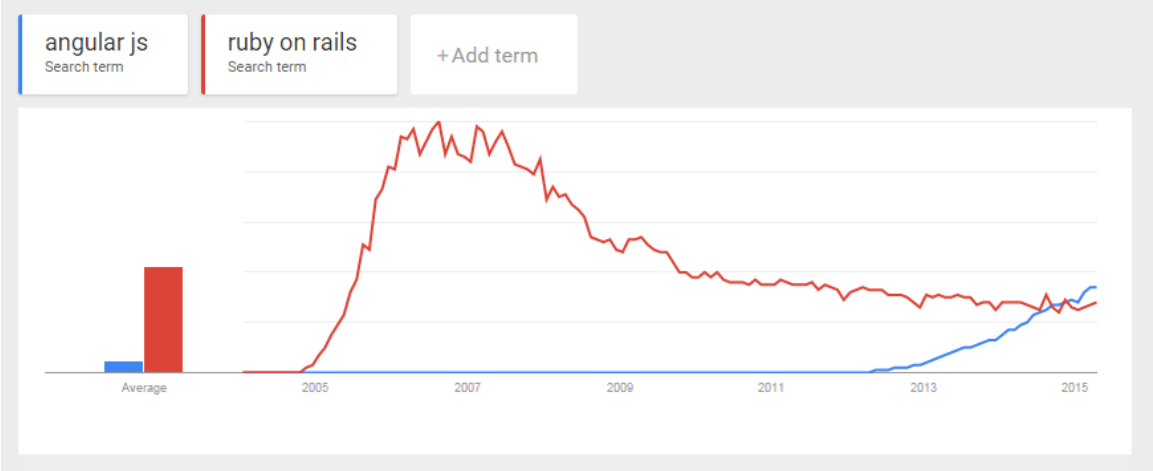

Figure 3.1.2: Google Statistics – Frameworks Search Term (Angular vs Ruby on Rails)

This shows that even though the client-side frameworks haven't reached the peak which server-side frameworks like Ruby on Rails did, they are on the rise and are becoming at least as popular as server-side frameworks if not more.

4. Comparison

4.1. Performance

Performance can conveniently be equated to the efficiency of the framework. In a simple way, it is the quality of system outputs in response to user inputs. The server side frameworks we've seen have been built upon for years, follow best practices and are constantly optimized and

improved. This definitely helps developers to quickly get started with the application logic instead of worrying about performance issues.

Lot of performance related issues depend upon the underlying environment the framework is built upon as well as poor coding practices by developers. That said, using a standard metrics and similarly built programs can go a long way into giving us an insight about how these frameworks fare against each other.

On the internet, response time is often most critical to retaining the target audience. Therefore, number of responses per second shows how quickly a system can response and is a good metric to evaluate and compare framework performances.

TechEmpower.com has been running performance tests on popular frameworks since 2013 [40]. All these tests were performed on a Dell R720xd dual-Xeon E5 v2 + 10 GbE server.

Here are some of the tests the results [41]:

1. **JSON serialization**:
   In this test, each response is a JSON serialization of a freshly-instantiated object that maps the key message to the value Hello, World!

Framework	**Best performance (higher is better)**
Spring	97,354
Grails	81,388
Django	11,064
Trinidad Rails	2,377

Table 4.1.1 Test results with JSON serialization

2. **Single Query**:
   In this test, each request is processed by fetching a single row from a simple database table. That row is then serialized as a JSON response.

Framework	**Best performance (higher is better)**
Grails	60,521
Spring	55,291
Django	9,245
Trinidad Rails	2,573

Table 4.1.2 Test results with Single Query

3. **Multiple Queries**:
   In this test, each request is processed by fetching multiple rows from a simple database table and serializing these rows as a JSON response. The test is run at 20 queries per request.

Framework	Best performance (higher is better)
Spring	3,916
Grails	3,393
Django	1,698
Trinidad Rails	1,498

Table 4.1.3 Test results with multiple queries

4. **Data Updates**:
   In this test, each request is processed by fetching multiple rows from a simple database table, converting the rows to in-memory objects, modifying one attribute of each object in memory, updating each associated row in the database individually, and then serializing the list of objects as a JSON response. The test is run at 20 updates per request.

Framework	Best performance (higher is better)
Spring	1,526
Trinidad Rails	1,128
Grails	902
Django	790

Table 4.1.4 Test results with Data Updates

5. **Plaintext**:
   In this test, the framework responds with the simplest of responses: a "Hello, World" message rendered as plain text. The size of the response is kept small so that gigabit Ethernet is not the limiting factor for all implementations.

Framework	Best performance (higher is better)
Spring	123,174
Grails	77,072
Trinidad Rails	2,007
Django	NA

Table 4.1.5 Test results with Plaintext

From this data, we can see that Spring has been by far the best and Grails has been a close second. Django and Rails have performed at a similar level to each other but much worse than the former two.

4.2. Scalability

Scalability is defined as the ability of the system to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate said growth.

Holistic scalability of a system is often a combination of its architecture design. Several server-side frameworks achieve varying degree of scalability through integration of architectures like REST – for API modules that deals with high traffic flow. Data modules can also be scaled by employing techniques such as batch processing and multithreading. Use of effective caching mechanisms help in reducing the server load as page requests can be served of the cache and not be sent to the server every time. This also means that the selection of the parts impacts the whole. An unsuited component can cause severe bottlenecks even when all the other components are working with just a part of their maximum load.

Frameworks do help to achieve scalability as they try to enforce certain best practices and set guidelines. That said, different use cases require different implementations to achieve scalability and thought needs to be put into choosing the right framework for a certain use case. For example, an application that receives high traffic needs to concentrate on web server optimization while another application that is highly data centric should primarily focus on their database and choose a framework which is known to perform data centric tasks much better than others.

Scalability is achieved in Spring with the help of several supporting frameworks. Spring MVC can employ RESTful principles to build a scalable application [52]. Spring Batch is a lightweight, comprehensive batch framework designed to enable the development of robust batch applications vital for the daily operations of enterprise systems. This enables extremely high-volume and high performance batch jobs through optimization and partitioning techniques. Simple as well as complex, high-volume batch jobs can leverage the framework in a highly scalable manner to process significant volumes of information. [51]

Scalability is achieved to a certain extent in Django with the use of caching frameworks that avoids server hits for requests that have previously been served. Varnish is an extensible caching framework that is popular with high traffic Django sites. [53]

Let's take a look at highly scaled websites which use these frameworks:

1. **Django:**
    a. Used by Disqus.com to power 8 billion page views [42]
    b. Used by Instagram.com to scale the site to 14 million+ users in little over a year [43].
    c. Used by BitBucket.com – 200TB of Code and 2.5 million+ users [44][45]
2. **Ruby on Rails:**
    a. Basecamp.com users manage 8 million+ projects using Ruby on Rails [46].
3. **Spring:**
    a. Spring has passed the five million downloads mark and now has a bigger market share than any Java Application Server. It is the predominant approach for Java Enterprise Applications. A survey by BEA in 2006 showed that 64% of all

enterprises were already using Spring. So, Spring has become the de facto standard for Enterprise Java. It is also interesting to note that in the job market Spring has passed other technologies like EJB. The job market is interesting because it shows what companies are actually willing to pay money for and what skills they want in their staff. These numbers are a clear indication that organizations are using Spring extensively [54].

b. Bank of America serves more than 59 million consumer and small business relationships with more than 6,100 retail banking offices, nearly 19,000 ATMs and online banking with nearly 24 million active users. One Spring application at Bank of America is a middle office data intensive Trading Analysis Application where the development team replaced the old Excel/Reporting Databases system with a Spring-based Java application. Spring allows Bank of America to focus on their application as opposed to the framework while providing patterns for simplifying data access.[54].
c. HSBC is one of the largest banking and financial services organizations in the world. HSBC's international network comprises over 10,000 offices in 83 countries and territories in Europe, the Asia-Pacific region, the Americas, the Middle East and Africa. HSBC based its next-generation architecture on Spring. This led to 40% reduction in framework code and 25% reduction in reference application code. HSBC has a long relationship with SpringSource and purchased Enterprise-wide Support from SpringSource [54].

4. **Grails:** LinkedIn.com, which serves, more than 300 million+ users uses Grails. [47]

4.3 Security

The practice of securing private data stored online, from unauthorized access and modification, is what is called security. In order to understand security of web applications, it is important to identify how each of these server side frameworks incorporates security features and the level of data access provided to its users.

4.3.1 Rails Security

While the rails framework is easy to use, it is also not hard to misuse [55]. The possible set of security issues include user account hijacking, bypass of access control, reading or modifying sensitive data, or presenting fraudulent content [56]. As such, developers on this framework should perform code review, penetration testing, static analysis and integrate security as a part of Software Development Life Cycle. Securing Ruby apps means to amalgamate best practices in coding plus an apt usage of helper methods that to fend off XSS attacks.

Ruby agrees to filters out necessary request parameters from log files by adding them to config.filter_parameters in the application configuration [57]. As a result, some pressing ruby logging vulnerabilities are prevented on the rails framework.

4.3.2 Spring Security

The Spring Security is also known as Acegi Security. It accommodates an all-inclusive security solution for managing authentication and authorization. The Spring Security namespace provides directives for the most operations, consenting complete security configuration in just a few lines of XML.

Spring Security has inbuilt authentication providers for many occasions. The most vital ones include the DaoAuthenticationProvider for retrieving user information from database; LdapAuthenticationProvider for authentication against a Lightweight Directory Access Protocol (LDAP) server and JaasAuthenticationProvider for retrieving user information from a JAAS login configuration.

4.3.3 Grails Security

By Grails security, we mean that the application should enable authenticating to users and authorizing them to do the activities they desire in the system. The conventional approach of ensuring security through login forms is no longer the best. This is where Spring Security collection comes into picture and facilitates connecting to an extensive range of data sources to obtain access gen from them instead of storing them manually.

Securing the grails app usually works this way. Upon creation of the app, a Spring Security Core plugin has to be installed. The Spring Security Core creates the necessary models and controllers. It provides the nitty-gritties necessary for access control in an easy-to-use package centered on users and roles. Thus ensuring security of the app.

4.3.4 Django Security

Django's security features [50] include the following:

Cross site scripting (XSS) protection: Django templates are extensively used to protect against the majority of XSS attacks.

Cross site request forgery (CSRF) protection: Usage of the CSRF middleware and template tags provides easy-to-use protection against Cross Site Request Forgeries.

SQL injection protection: Django's query sets could be exploited, so that the resulting SQL will be properly escaped by the database driver.

Clickjacking protection: Django supports clickjacking protection using the X-Frame Options middleware, which in a supporting browser thwarts the site from being rendered inside a frame.

4.4 Comparison Table

Comparison between frameworks [48]:

Best Practice	Django	Spring	Ruby on Rails	Grails
AJAX Support	Yes	Yes	Yes	Yes
Cloud Computing	Yes (dotCloud, Google App Engine and Amazon EC2)	Yes	Yes (Amazon EC2, Linode, Rackspace and Heroku)	Yes (CloudFoundry, Google App Engine, Amazon EC2 and Heroku)
Custom Error Messages	Yes (modelvalidationError)	Yes	Yes (File yml)	Yes (File properties)
Customization and Extensibility	Yes (Django-Utils e.g. safestring, translation, among others)	Yes	Yes (Plugins e.g. LessCSS, Authlogic, among others)	Yes (Utility e.g. iCalendar, Smartionary, among others)
Debugging	Yes (Django Debug Toolbar)	Yes (Log4j plugin commonly used)	Yes (debug, to_yaml and inspect)	Yes (X-Grails-Resources-Original-SrcHeader)
Documentation	Yes (Sphinx)	Yes (Spring Docs)	Yes (Rdoc)	Yes (grails doc)
Form validation	Yes	Yes	Yes	Yes
HTML5 support	Yes (HTML5 Boilerplate, H5BP)	Yes	No	Yes (Modernizr)
JavaScript frameworks support	Yes (jQuery, ExtJS, Dojo, among others)		Yes (jQuery, script.aculo.us, Dojo, among others)	Yes (jQuery, Dojo, script.aculo.us, among others)
ORM(Object Relational Mapping)	Yes (Django ORM)	Yes (Hibernate)	Yes (ActiveRecord)	Yes (GORM)
REST Support	Yes (Django REST framework)	Yes	Yes	Yes
Security	Yes	Yes	Yes	Yes
Template Framework	Yes (Django template language)	Yes (Freemarker)	Yes	Yes (Layout and SiteMesh)
Testing	Yes (Unit test, doctest, nose)	Yes	Yes (Rspec)	Yes (GMock and EasyMock)

Table 4.4 Comparison between frameworks

5. Server-side or Client-side?

Client-side frameworks offer a lot of advantages but they come with their fair share of negatives too.

5.1 Advantages of Client-Side Frameworks vs Server Side Frameworks:

1. **Thin Server -** Most of the application logic is ported over to the client leading to a thick client – thin server architecture.
2. **Bandwidth -** Bandwidth usage is reduced as most of the code on the client side is loaded once and page refreshes are avoided by the use of AJAX.
3. **Server load –** It is reduced as most of the processing is performed by the client (usually a web browser).
4. **Caching -** It is easier to cache client-side HTML templates [49].
5. **User Experience** - Usually leads to an improved user experience as the page doesn't refresh from page to page due to AJAX leading to a more native application feel.
6. **Flexibility -** It is easier to build the application for multiple platforms as the server can act like an API and different type of clients can consume it. For example, an application with a thin server can be used by clients build upon the web platform as well as native smartphone applications.
6. **Reduced Latency –** Once the page is loaded, all the client side code is in the browser and when subsequent requests are made, the client can re-render selected parts of the page as supposed to a complete re-render for servers. This leads to better responsiveness.
7. **Uniformity** – For large scale applications, multiple technologies/frameworks may be used on the server with different engines to render views (eg. JSP, GSP etc). This makes it difficult to share User Interface code between frameworks. Client side frameworks solve this problem by having one uniform codebase.

5.2 Disadvantages of Client-Side Frameworks vs Server Side Frameworks:

1. **Security -** Client side frameworks are not secure as all the source code is visible. Sensitive information needs to be carefully kept on the server.
2. **Browser compatibility** – Not all browsers support each and every feature, HTML5 API etc. Therefore, depending completely on a client side framework can lead to losing control over your application. For example, older browsers like Internet Explorer 7 and 8 do not fully support HTML5 and therefore may require additional work on the client.
3. **Inconsistency** – Since, we lose control over the performance of our application when client side frameworks do most of the work, it can lead to inconsistent results depending on the client specifications. For e.g. low powered mobile devices may not render a web page correctly as initial rendering on client side takes more time due to JavaScript needing to be completed loaded and then rendering the page.

4. **Testing** – Since there are too many web browser vendors and various types of devices ranging from desktops to phones, testing the application becomes difficult as performance depends upon clients instead of a single server.
5. **Initial Rendering –** Server side rendering of a web page is much quicker for the first page load as the server outputs the HTML straight to the web page and it's rendered without any JavaScript framework files are loaded or further requests are made to the server. This can be crucial if your application requires a really low page load time. For example, for an e-commerce website, showing the pricing and product information as quickly as possible is important to convert views to sales and in such a scenario a client-side framework may not be suitable.
6. **Search Engine Optimization –** Since data is loaded on subsequent AJAX requests once the framework is loaded, the data is hidden from search engines. Server side rendering of web pages ensures that all the content is visible to the search engine. Therefore, a mixture of initial server side rendering and then subsequent requests being rendered by the client may be ideal.
7. **Resource intensive computation –** If the application is supposed to do intensive computational work, a server side framework is better than a client-side framework as servers are faster than most client machines and can therefore offer a consistent and fast response.

Based on these advantages and disadvantages one can decide whether to go with a client-side framework or a server side framework. If the application is meant to be doing largely CRUD operations and is being built on multiple platforms, then a client-side framework might be useful with a thin server. If the application logic contains a lot of sensitive information, a server side framework should be used as that doesn't expose the information and would help prevent attacks on the application.

Ideally, a combination of both, client-side frameworks and server-side frameworks should be used. Up to various degrees as the client-side framework can reduce the burden on the server and increase flexibility while the server can do complex computational work, protect sensitive data etc.

6. Conclusion

In conclusion, the rise of client-side frameworks has changed the way web applications are developed and have had an impact on server-side frameworks. Websites can now be less reliant on the server and push a lot of processing to the client which helps in better server utilization. This has meant that server-side frameworks do not have as much of an impact as they once did but are and will be heavily utilized in the near future.

7. References

[1] http://docforge.com/wiki/Web_application_framework. Retrieved 2015-03-02

[2] Johnson, R. J2EE Development Frameworks. *Computer (Volume:38 , Issue: 1) IEEE Computer Society.* Jan. 2005
[3] Ian Darwin's Java Frameworks webpage: http://www.darwinsys.com/jwf/jwf-frameworks.html. Retrieved 2015-03-02
[4] José Ignacio Fernández-Villamor, Laura Díaz-Casillas, and Carlos Á. Iglesias. A Comparison Model for Agile Web Frameworks. *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems,* (Sep. 2008).
[5] Django's official Project FAQ webpage: https://docs.djangoproject.com/en/1.7/faq/general/#why-does-this-project-exist. Retrieved 2015-03-02
[6] RubyGems webpage for Ruby on Rails release versions: https://rubygems.org/gems/rails/versions. Retrieved 2015-03-02
[7] Andreas B. Gizas, Sotiris P. Christodoulou and Theodore S. Papatheodorou. Comparative Evaluation of JavaScript Frameworks. *Proceedings of the 21st international conference companion on World Wide Web,* (April 2012) 513-514.
[8] David Heinemeier Interview: https://web.archive.org/web/20130225091835/http://dev.mysql.com/tech-resources/interviews/david-heinemeier-hansson-rails.html. Retrieved 2008-06-08.
[9] Java Spring Stackoverflow.com Popularity: https://api.stackexchange.com/docs/tags#order=desc&sort=popular&inname=spring&filter=default&site=stackoverflow. Retrieved 2015-04-20
[10] Python Django Stackoverflow.com Popularity: https://api.stackexchange.com/docs/tags#order=desc&sort=popular&inname=django&filter=default&site=stackoverflow. Retrieved 2015-04-20
[11] Grails Stackoverflow.com Popularity: https://api.stackexchange.com/docs/tags#order=desc&sort=popular&inname=grails&filter=default&site=stackoverflow. Retrieved 2015-04-20
[12] Ruby on Rails Stackoverflow.com Popularity: https://api.stackexchange.com/docs/tags#order=desc&sort=popular&inname=ruby-on-rails&filter=default&site=stackoverflow. Retrieved 2015-04-20
[13] Github repository of Spring: https://github.com/spring-projects/spring-framework. Retrieved 2015-04-20
[14] Github repository of Django: https://github.com/django/django. Retrieved 2015-04-20
[15] Github repository of Grails: https://github.com/grails/grails-core. Retrieved 2015-04-20
[16] Github repository of Ruby on Rails: https://github.com/rails/rails. Retrieved 2015-04-20
[17] Youtube search results for Java Spring: https://www.youtube.com/results?search_query=java+spring. Retrieved 2015-04-20
[18] Youtube search results for Python Django: https://www.youtube.com/results?search_query=python+django. Retrieved 2015-04-20
[19] Youtube search results for Groovy & Grails: https://www.youtube.com/results?search_query=groovy+grails. Retrieved 2015-04-20
[20] Youtube search results for Ruby on Rails: https://www.youtube.com/results?search_query=ruby+on+rails. Retrieved 2015-04-20
[21] Google Trends for Server Side Frameworks – 2004 to 2015:

https://www.google.com/trends/explore#geo&q=python+django,+java+spring,+ruby+on+rails,+grails&cmpt=q. Retrieved 2015-04-22
[22] Spring Framework 1.0 Release: https://spring.io/blog/2004/03/02/spring-framework-1-0-release-candidate-2-released. Retrieved 2015-04-22
[23] Spring Framework 4.0 GA Release: https://spring.io/blog/2013/12/12/announcing-spring-framework-4-0-ga-release/. Retrieved 2015-04-22
[24] Frequently Asked Questions (FAQ) about the future of XHTML: http://www.w3.org/2009/06/xhtml-faq.html. Retrieved 2015-04-22
[25] Node JS initial release: https://github.com/joyent/node/tags?after=v0.0.4. Retrieved 2015-04-22
[26] AngularJS Stackoverflow.com Popularity: https://api.stackexchange.com/docs/tags#order=desc&sort=popular&inname=angularjs&filter=default&site=stackoverflow&run=true. Retrieved 2015-04-22
[27] Backbone.js Stackoverflow.com Popularity: https://api.stackexchange.com/docs/tags#order=desc&sort=popular&inname=backbone.js&filter=default&site=stackoverflow&run=true. Retrieved 2015-04-22
[28] Ember.js Stackoverflow.com Popularity: https://api.stackexchange.com/docs/tags#order=desc&sort=popular&inname=ember&filter=default&site=stackoverflow&run=true. Retrieved 2015-04-22
[29] Github repository of AngularJs: https://github.com/angular/angular.js.
[30] Github repository of Backbone.js: https://github.com/jashkenas/backbone. Retrieved 2015-04-22
[31] Github repository of Ember.JS: https://github.com/emberjs/ember.js. Retrieved 2015-04-22
[32] Youtube search results for AngularJS: https://www.youtube.com/results?search_query=angular+js. Retrieved 2015-04-22
[33] Youtube search results for Backbone.js: https://www.youtube.com/results?search_query=backbone+js. Retrieved 2015-04-22
[34] Youtube search results for Ember.js: https://www.youtube.com/results?search_query=ember+js. Retrieved 2015-04-22
[35] AngularJS Releases: https://github.com/angular/angular.js/releases?after=v0.9.1. Retrieved 2015-04-22
[36] Backbone.js Releases: https://github.com/jashkenas/backbone/releases?after=0.1.1. Retrieved 2015-04-22
[37] Ember.js Releases: https://github.com/emberjs/ember.js/releases?after=v0.9.1. Retrieved 2015-04-22
[38] Google Trends for Server Side Frameworks – 2004 to 2015: https://www.google.com/trends/explore#q=angular%20js%2C%20backbone%20js%2C%20ember%20js&cmpt=q&tz=. Retrieved 2015-04-22
[39] Google Trends for Server Side Frameworks – 2004 to 2015: https://www.google.com/trends/explore#q=angular%20js%2C%20ruby%20on%20rails&cmpt=q&tz=. Retrieved 2015-04-22
[40] Previous Rounds - TechEmpower Framework Benchmarks: https://www.techempower.com/benchmarks/#section=previous-rounds&hw=peak&test=json&f=754-0-0-0. Retrieved 2015-04-22

[41] Round 10 results – TechEmpower Framework Benchmarks: https://www.techempower.com/benchmarks/#section=data-r10&hw=peak&test=json&f=754-0-0-0. Retrieved 2015-04-22
[42] What Powers Instagram: http://instagram-engineering.tumblr.com/post/13649370142/what-powers-instagram-hundreds-of-instances. Retrieved 2015-04-22
[43] Scaling Django to 8 Billion Page Views: http://blog.disqus.com/post/62187806135/scaling-django-to-8-billion-page-views. Retrieved 2015-04-22
[44] Bitbucket year in review: https://blog.bitbucket.org/2015/02/05/bitbucket-2014-in-review/. Retrieved 2015-04-
[45] Django Success Stories = Bitbucket: https://code.djangoproject.com/wiki/DjangoSuccessStoryBitbucket. Retrieved 2015-04-22
[46] Basecamp Customers: http://web.archive.org/web/20140202213318/https://basecamp.com/customers. Retrieved 2015-04-22
[47] Grails at LinkedIn: http://blog.linkedin.com/2008/06/11/grails-at-linkedin/. Retrieved 2015-04-22
[48] María del Pilar Salas-Zárate, Giner Alor-Hernández, Rafael Valencia-García, Lisbeth Rodríguez-Mazahua, Alejandro Rodríguez-González, José Luis López Cuadrado. Analyzing best practices on Web development frameworks: The lift approach. (January. 2015).
[49] Template Cache – AngularJS https://docs.angularjs.org/api/ng/service/$templateCache. Retrieved 2015-04-22
[50] Django Security: https://docs.djangoproject.com/en/1.8/topics/security/. Retrieved 2015-04-22
[51] Spring Batch Framework http://projects.spring.io/spring-batch/. Retrieved 2015-04-22
[52] RESTful Spring Applications https://spring.io/understanding/REST. Retrieved 2015-04-22
[53] Varnish Cache https://www.varnish-cache.org/. Retrieved 2015-04-22
[54] Spring Managers Overview http://assets.spring.io/drupal/files/Spring_A_Managers_OverviewVer05.pdf . Retrieved on 2015-04-22
[55] 10 Most Common Rails Mistakes: http://www.toptal.com/ruby-on-rails/top-10-mistakes-that-rails-programmers-make. Retrieved on 2015-04-22
[56] Ruby on Rails Security Guide: http://guides.rubyonrails.org/security.html. Retrieved on 2015-04-22
[57] Configuring Rails Applications: http://guides.rubyonrails.org/configuring.html. Retrieved on 2015-04-22