Expressing Object-Oriented Thoughts Functionally

Clarence J M Tauro
Center for Research
Christ University
Bangalore

Infant Arockiaraj A
Dept. of Computer Science
Christ University
Bangalore

Dhanashree K
Dept. of Computer Science
Christ University
Bangalore

ABSTRACT
Scaling web applications by adding processors is important as the number of users increase by the day, database size is on the rise which results in huge volume of work. The complication that exists in scaling does not just depend on increasing the count of new processors as the paradox also exists in extensive scaling of Java applications. This limitation is bounded specifically to Java programming language and not to Java platform as a whole.

We explore the Object oriented functional languages such as Scala, Clojure and Groovy in Java platform. The peripheral languages take advantage on the infinite scalability of Java platform. The interoperability of Scala, Clojure and Groovy is an asset in Java platform as they run parallel with other Java applications. Additionally, we discuss on how the elasticity and adaptability of the Object Oriented functional languages allows simple and efficient execution in Java platform.

1. INTRODUCTION
Web Applications are gaining unprecedented popularity due to their integration in business models. Leading platforms for Web application developments are Java and .NET [1]. The major goals of developing a web application are to increase productivity, ensure quality and take advantage of more sophisticated performance optimizations available in modern hardware [2].

Designed with many advanced features, such as automatic memory management, cross-platform portability and enforced security check, Java is an excellent programming language used on various platforms. The Multi-threading feature of Java is also employed on platforms, largely on the server systems to gain higher throughput and faster response time. Java is undisputedly leading in development of Web applications in current market scenario. However, the performance and scalability of Java applications running on multicore systems remain as a major concern and were rarely reported [3].

These were rarely reported because, scaling Java applications until the Number's of processors are in single digit it works well but the real issue arises when the processors are added beyond [2].

The limitation is subjected to Java applications alone but not to Java platform as a whole. This opens new possibilities for Java applications to scale linearly with dependencies.

Scala and Clojure are two of the most successful Object Oriented functional languages used in order to rectify the shortcomings of Java applications. These Object Oriented languages seamlessly integrate features of Object Oriented and functional languages, allowing Java and other language programmers to be more fruitful [4].

"Scala is a hybrid Object-Oriented (OO) and functional language developed by Martin Odersky - one of the original authors of the Java compiler. Scala runs on the Java virtual machine (JVM), compiles to .class files and is completely interoperable with the Java programming environment" [5].

"Clojure is a dynamic, Lisp like programming language that originally targeted the Java Virtual Machine (JVM). It has been successful on the JVM platform because of its combination of expressiveness, performance and host interoperability" [6].

"Groovy is a powerful Object oriented agile programming language which works on the Java Virtual Machine (JVM) and utilizes all the benefits of Java. Groovy can also be used as a scripting language for the Java platform [4]." In the following segments this paper highlights the shortcomings of Java applications and how Java platform overcomes scalability issues.

2. PARADOX IN JAVA
The paradox of scalability in Java isn't something unforeseen. There has been many works that has been put in place to address such issues. We put forth three of the most efficient projects in overcoming the drawbacks in the form of Object Oriented programming languages named Scala, Clojure and Groovy. The paradox in Java provides differential view points. The contradictor statement lies in the fact that the java programming language possesses characteristics of high scalability. The constructs in Java are said to help while there is increase in scalability in all the systems it is used in. However, we do understand that Java encourages scalability mechanisms through the uses of features such as multi-threading, but they do not provide provision for scalability on an infinite scale.

The time taken to run a program is halved when the numbers of processors are doubled is called linear scaling,.e. a single processor takes twice the time when compared to two processors. Practically there is no ideal linear scaling but the emphasis has always been on linear scalability. The apprehension on why the Java application fails to scale boils down to locking.

The drawback Java faces in linear scaling: The question which is common in our minds is that the code is same whether there is one processor or ten processors. In Java programming the composition of threads and locks implement concurrency. On adding the number of processors, Java programs invest more time on worrying about the locks. This occurs primarily when there aren't any fundamental data connection issues, as a result you initiate scaling programs into many processors due to the deadlock that is created.

The performance issues in Java are apparently hard to deal with due to its complexity. Achieving maximal performance is the bigger challenge faced as Multi-threaded Java applications scale considerably. The simultaneous Java programs use threads that lock to other pieces of data when access is required. Consider two threads X and Y of a particular program. If X needs access to a particular set of data then the data is locked i.e. unavailable for thread Y to

access. The data is available for thread Y only when method X unlocks the data.

We all do understand that Java is a core programming language, being used extensively for many applications and systems. Java possesses object-Oriented characteristics that are widely used in the field of development of applications on multiple platforms such as mobile applications, web applications, servlets, etc. Though Java is a powerful language used for development it does have few shortcomings, one of them being the inconsistency to cope with the working of Domain Specific Languages (DSL's).

These shortcomings are limited to the programming language alone, Java platform as a whole is vast field and the drawbacks of Java programming are overcome with the introduction of code of other programming languages into Java. Inspite of the drawbacks, Java language can however be brought back to life by use of Object Oriented programming languages such as Groovy, Scala and Clojure. These languages have additional features and functionalities that further aid the shortcomings of the scalabilities of Java and hence, end to enhance an overall scalability and functionality of the system in which they are used.

To further answer questions on Java programming we compare to Scala, Clojure or Groovy on the basis of which being the more compact code, which of these needs more effort and how the functional imperatives are being used. We analyze moving ahead.

2.1 Scala

The Java based language Scala, adds to the scalability of Java. Scala is a multi-paradigm programming language, helping programmers to build more concise and less complex code, and at the same time being more functional, which java lacks on its own. Scala offers better functional features than Java [8].

The functional features and Object Oriented features are uniformly integrated that complying Java to be more fruitful. This productivity is evident as many companies are currently migrating to Scala to boost their productivity. Recently, due to their fast growing tweet rates, Twitter has moved their core message queue from Ruby to Scala.

Since Scala runs in JVM, the existing Java code is functional. They both are highly interoperable and we can invoke Scala methods from Java, and vice-versa. The Scala language provides many features such as, type inferencing, function passing, etc. This clearly provides options to the programmer to reduce the amount of code used in the program by use of efficient structures and syntax found in Scala. Scala reduces complex tasks and hence increases scalability of Java [4].

According to the official release by Scala the below are the key aspects of Scala [8].

- Scala has a uniform object model, in the sense that every value is an object and every operation is a method call.
- It has uniform and powerful abstraction concepts for both types and values.
- Scala is extensible; it supports composition of data structures through mixins and self type annotations.
- By pattern matching, Scala allows decomposition of objects.
- It also allows external extensions of components using views.

Scala language also offers capabilities where it can even simplify an inheritance mechanism in Java, and therefore reduce the amount of code used in the program.

Scala being highly functional in nature provides very simple syntax. It can be used to represent very complex code, in very simple form. Scala can be used to define unified type systems, where all objects can be a subclass of a single class. Scala also gives the provision of pattern matching models, which are used extensively. This feature presents Scala more suitable for building web applications.

Scala also increases scalability in Java by helping the programmer add constructs of different programming languages by use of a combination of unique mechanisms. This feature helps to invoke methods of different code, which can perform multiple operations to produce the desired results. We also understand that Scala shares similar compilation models as Java. This will increase the interoperability between Java and Scala thereby inducing seamless working between the two languages.

Figure 1: Sample program in Scala and Java

```
// Java
class PrintOptions {
  public static void main(String[] args) {
    System.out.println("Options selected:")
    for (int i = 0; i < args.length; i++)
        if (args[i].startsWith("-"))
           System.out.println(
                 args[i].substring(1));
  }
}

//Scala
object PrintOptions {
   def main(args: Array[String]): unit = {
     System.out.println
           ("Options selected:")
     for (val arg <- args)
         if (arg.startsWith("-"))
            System.out.println(
                arg.substring(1));
   }
}
```

Scala is statically checked duck typing via structural typing like dynamic languages but in a type-safe way. It has higher-order function, the first-class functions that other functions can pass as arguments and accept as return types. Scala has lexical closures, the bedrock of functional programming support. Scala's closure support leads to easy development of control abstractions and domain-specific languages.

Scala also has immutable data structures which are included as a part of the standard library, which encourage developers to design referentially transparent abstractions. Advanced generator constructs: for example, for-comprehension that makes code more expressive and succinct. Patterns in Scala are represented internally as partial functions, which developers can compose using various combinators to construct extensible abstractions. Lastly Scala is event-driven programming via the actor model. These features make Scala practically feasible in JVM.

Scala boasts that it requires smaller number in terms of the lines of code when compared to Java. As per the research conducted in "ClojureScript: Functional Programming for JavaScript Platforms [6]" the result upholds the claims. Scala language also provides additional facilities to define new control structures without the use of macros.The applicability in production environment in last few years has affirmed its high dependability. Scala has the advantage of being functional and simultaneously Object Oriented. The extensibility provides a unique combination of language Mechanisms that makes it effortless to add new language constructs in the form of libraries.

Thus we can safely say that, the Scala programming language helps in overcoming the shortcomings of scaling applications developed in Java. The features of Scala can be effectively constructed into Java programs to increase scalability with use of less complex, concise, expressive and high functionality code.

2.2 Clojure

Clojure a recent dialect from the Lisp language family compiles directly to the Java Virtual Machine, CLR and Java Script. "Clojure is designed to be a general purpose language, combining the approachability and interactive development of a scripting language with an efficient and robust infrastructure for Multi-threaded programming [10]".

The discrete characteristic of lisp programming language is that 'code is data'. The syntax of lisp is simple yet extensible data structure. The adaptation of nested lists is subsequent and has definitive conventions as executable code. The benefit in formulating the code in data structure is that lisp can easily generate and execute other lisp code. From the beginning the programmers have materialized this through lisp macros [11].

In comparison with Java, the application of multicore architecture through Multi-threading can be benefited to the programmers by concurrency API that clojure possesses. As in Java the low level thread co-ordination tasks doesn't stress programmers.

The functional programmability of Clojure has its advantages. It is a combination of immutable data and first class functions. Clojure inherits the various feature of lisp programming language such as simplicity, compliancy and high expressiveness. The data structures are consistent and immutable that supports recursion.

Designed for pure functional programming, Clojure has data structures highly optimized such as vectors, lists, maps and sets. It offers extensive support for concurrency. Clojure was outlined and designed for Java Virtual Machine with easier interoperability with other languages including Java being its objective [11].

Multi-threading has always been Java's way of supporting concurrency. The usage of multicore machines has been fully optimized by concurrency, i.e. concurrency is execution of multiple threads simultaneously. Genuine thread based concurrency is not supported by every language with Multi-threading API, Ruby and Python being standard examples. Nevertheless, the programmer is challenged in Java multithreading on whether requiring Multi-threads are safe or not. Let us consider a program writing to the same memory location simultaneously. The same address space are being shared by threads in a single process, this results in the programmer being responsible for synchronizing threads rather than the operating system.

This becomes a challenging task for programmers as a Multi-threaded program runs too long in a single thread node due to excessive synchronization, which decreases the performance and may even lead to a deadlock. The reliability and correctness of a program are undermined due to insufficient synchronization as it promotes race conditions. Hence the daunting task is to have synchronization just enough [16].

The interfaces and Java classes can be completely utilized by Clojure. Even without importing a package java.lang classes can be used in Java. By either individualizing their package or adopting the import function other preferred packages can be used in Java classes. The concurrency issues in Java being disaster for the programmers, Clojure overcomes its impacts by modeling data structures as immutable objects represented by interfaces. In other ways an own class system is not presented to clojure itself [12].

The support for thread-safe concurrency has been significantly improved by Java in the past few years. The basic Java construct continues to remain the synchronized block.

Figure 2: Basic Java construct

```
synchronized (lock) {
     // code that must be
     //executed one thread at a time
}
```

`java.util.concurrent` one of the latest Java libraries provides support for thread coordination at a higher level. The foundation of Java multithreading API is a combination of synchronization constructs and explicit locking constructs formed as a synchronized block [16].

The below are the features of Clojure that makes Clojure advantageous when compared to Java [10].

- Dynamic development: The interactivity of Clojure sets it apart. It is not just a program you compile and run but there is a dynamism associated with this programming language. Clojure is an environment, not just a language abstraction where virtually all the language constructs are ratified.
- Functional Programming: This feature of Clojure provides tools and helps avert inconsistent state of Java. Clojure articulates recursive iteration in place of side-effect based loop. To make the program robust, the ideology in Clojure is to write most parts of programs functionally.
- Runtime Polymorphism: Employing runtime polymorphism makes it easier for a system to be extensible and change.

Three levels of polymorphism that clojure supports,

- Java Interface defines proximately all core infrastructure data structures at runtime.
- Using proxy, Clojure supports implementation of Java Interfaces.
- Multimethod in clojure is the first and foremost language construct in polymorphism.

In Object Oriented languages such as Java, a particular object is the programming construct that encapsulates state. Let us consider an object of point type. This object will have `x` and `y` coordinates and their values would compose its state. In order to access this state of object, which is combination of get

functions or set functions such as `p.getx()`. benefitted for `x`'s coordinate the current value is returned. `p.set(val)` updates `x` coordinate to val. By design since objects are mutable, through any thread of execution they are subjected to change. By estimation the, flow of control must be anticipated by the programmer where the properties of objects can be modified by multiple threads. Without appropriate synchronization if two threads modifies point `x`'s coordinate then the state of an object becomes indeterminate and hence the accuracy of the program will be threatened.

The advantage of Clojure and other Functional advantages is that they treat mutability as an exception and not a rule. A program is collection of immutable structures, pure functions and not collection of objects in various states. A pure function is defined as the mapping of same arguments to same values and other than the value returned by the invoked function it has no external effects. To be precise, a pure function has neither states nor side effects.

In the example,

```
(def teams {:gracie "george" :lou "bud"})
```

An immutable map is defined here, a collection of value pairs called teams.

The below statement,

```
(assoc teams :dickie "tommy")
```

It-creates a new expanded map with an additional value pair from the original map. By definition the function 'assoc' clones the existing map and then appends a new member to the clone rather than original. In combining the immutable structure and pure functions they are secure from unintentional modifications during the program execution. Clojure advocates the use of both pure functions and immutable structures and refrains from the ideals to facilitate efficiency and convenience of programs.

Clojure's multithreading API's advantages when compared to Java are highlighted in the above examples. Also the interoperability with JVM, the rapidly expanding infrastructure of Clojure has made it preferable for numeric processing.

The stakeholders and customers have major investments in platforms like JVM. This is result of being convenient and comfortable with stability, security and stability of JVM. Though the productivity and flexibility of dynamic languages are coveted by Java developers, the performance running on customer authorized infrastructure and access to libraries, existing code base have been cause of concerns [14].

The existing issue of concurrency in Java using native threads and locking has resulted in efforts to overcome them. Clojure is an efficient dynamic design language in this respect. It aims to be general purpose language applicable in areas where Java fails. It emulates the fact that for the future of concurrent programming omnipresent and liberate mutation simply has to go [11].

By embracing the industry standard platform such as JVM clojure adheres its objective. Renovating an age old language Lisp, adopting functional programming and advance it with immutable persistent data structures. Through asynchronous agents and software transactional memory Clojure provides built in concurrency support. Thus overcoming the practical implications of Java and resulting in fast, robust and practical language [13].

2.3 Groovy

Groovy is a swift and dynamic language for JVM. Though groovy has features influenced from the likes of Python, Smalltalk and Ruby are mostly built upon the strength of Java. Java developers get access to the latest programming features with negligible leaning curve. It is probably the first language written at a targeted community of programmers i.e. Java programmers [16].

The upcoming performance issues in Java in recent years are simply because of the decisions made during its early development and now it only complicates it further than simplifying it. Here we brief points which are identified as the core issues in Java for scalability and performance issues, how Groovy capabilities enable Java to be convenient and useful [18].

2.3.1 Static typing in Java

Though static typing was seen as an advantage in Java, the composition of dynamic binding and static typing had enough structure in Java to capture the issues immediately and yet had adequate space to implement and apply polymorphism. Java has dynamic binding constraints overriding is not possible unless the two classes are connected by inheritance.

Operator overloading and optional typing are features in Groovy that enable the developers to have superior flexibility with comparatively less code.

2.3.2 In Java methods must be contained in a class

In Java you cannot have a method by itself. It is a must that, a method has to reside within classes. This may not seem to be an issue always but when it comes to sorting strings it is definitely not recommendable. Though collections have been a part of Java since beginning, they do not have native support for collections like Clojure. All this may not seem to be a greater problem in smaller applications but when the application is scaled to higher number of processors they just are composed to a priority issue at hand.

The complexity of the code is reduced in Groovy; also the automatic imports and the native syntax for collections minimize the amount of required code. The ability of Groovy being autonomous to treat method as objects called as Closures is an advantage during extending.

2.3.3 Java being over extensive

Figure 4, is a POJO (Plain Old Java Object). It consists of a class (Project) probably a part of project management. The name, priority, start and end dates of the task represent the attributes.

Although the below Java code can be auto generated through IDE still, the result is being extensive.

Figure 3: POJO (Plain Old Java Object)

```
import java.util.Date;
public class Project {
    private String title;
    private int preference;
    private Date beginDate;
    private Date finalDate;

    public Project () {}
        public Project
       (String title, int preference,
        Date startDate, Date endDate){
         this.name = name;
         this.preference = preference;
          this.beginDate = beginDate;
          this.finalDate = finalDate;
        }
        public String getTitle(){
         return title;
        }
        public void setTitle(String title)
        { this.title = title; }

        public int getPreference(){
               return preference;
        }
        public void setPreference
                    (int preference){
            this.preference = preference;
        }
        public Date getBeginDate(){
        return beginDate; }
        public void setBeginDate
                    (Date beginDate){
            this.beginDate = beginDate;
        }
        public Date getFinalDate(){
              return finalDate;
        }
        public void setFinalDate
                        (Date finalDate){
        this.finalDate = finalDate;
        }
        @Override
        public String toString() {
             return "Task [title="
                    + title
                    + ", preference="
                    + preference
                    + ",beginDate="
                    + beginDate
                    + ", finalDate="
                    + finalDate + "]";
        }
}
```

Now we take a look at POGO (Plain old Groovy object),

Figure 4: Plain Old Groovy Object

```
class Project {
    String title
    int preference
    Date beginDate
    Date finalDate
    String toString() {
          "($title,$preference,
          $beginDate,$finalDate)"
}
```

The capabilities of Groovy such as dynamic generation exceptionally reduce the burden of code in a class and hence the focus is largely on essence and not the ceremony.

We have seen earlier achieving maximum performance in scaling Multi-threaded Java applications being a major preference. The above example as an indication of Groovy overcoming the extensibility issues faced in Java.

The Groovy language is highly dynamic in nature. Hence it offers multiple functionalities where the programmer can dynamically add or call the functions or methods during run-time of a program. Such useful meta-programming characteristics, help us to dynamically work with domain specific languages in a more efficient and seamless manner. One example of such a case could be the use of SQL programming statements in a Java program, where the SQL statements that perform some database operations will be extended into the Java program using Groovy to work dynamically [17].

Groovy is a Java based language that helps the domain specific languages to work seamlessly with Java. Java has internal structural differences that do not help it to work with domain specific languages, where its structure and syntax do not offer much support. Also java being a static language, furthermore opposes the seamless working of domain specific languages. However, the use of the Groovy programming language, offers more flexibility in terms of structure and syntax to handle domain specific languages more efficiently [16].

One of the biggest advantages of Groovy is that it is, able to integrate the libraries and Java classes seamlessly. Since the compilation is directly on Java byte code, groovy can be used anywhere Java is used. The above mentioned features of Groovy establish it as a very good platform for prototyping, developing utilities, and also they are adequately constructed into Java programs to increase scalability with use of less complex, precise, meaningful and high functionality code. The Groovy provides multiple syntax and structures to efficiently incorporate domain specific languages into Java, turning into a real boon for the Java language [17].

3. Conclusion

Java without, any doubt is a very powerful language and large. Acquiring certain problems and inconsistencies over a period of time due to some decisions made during the initial stage of its development. Java is still dominant in the market and is everywhere. Its infrastructure, tools, libraries and infrastructure are still favorable and useful.

In this paper we have presented the peripheral languages as an extension to Java. We have listed the capabilities of Scala, Clojure and Groovy that will be a rich source of ideas in order to abridge the short comings of the programming language Java. As powerful as the peripheral languages and also fun to use, this paper does not recommend replacing the existing Java with Scala, Clojure or Groovy instead we advocate a blended approach. Our ideology is to use the peripheral languages wherever it helps the most.

The paper still leaves scope for the performances of the peripheral languages to be evaluated on various parameters such as concurrency, memory utilization, response time, code efficiency and memory utilization which could be taken up for further studies.

4. REFERENCES

[1] Stella, L.F.F.; Jarzabek, S.; Wadhwa, B.; , "A comparative study of maintainability of web applications on J2EE, .NET and Ruby on Rails," Web Site Evolution, 2008. WSE 2008. 10th International Symposium on , vol., no., pp.93-99, 3-4 Oct. 2008 doi: 10.1109/WSE.2008.4655401

[2] Pankratius, V.; Schmidt, F.; Garreton, G.; , "Combining functional and imperative programming for multicore software: An empirical study evaluating Scala and Java," Software Engineering (ICSE), 2012 34th International Conference on , vol., no., pp.123-133, 2-9 June 2012 doi: 10.1109/ICSE.2012.6227200

[3] Kuo-Yi Chen; Chang, J.M.; Ting-Wei Hou; , "Multithreading in Java: Performance and Scalability on Multicore Systems," Computers, IEEE Transactions on , vol.60, no.11, pp.1521-1534, Nov. 2011 doi: 10.1109/TC.2010.232

[4] Bhat, M.S.; Nair, D.G.; Bansal, D.; Vaishnavi, J.; , "Data structure based performance evaluation of emerging technologies — A comparison of Scala, Ruby, Groovy, and Python," Software Engineering (CONSEG), 2012 CSI Sixth International Conference on , vol., no., pp.1-5, 5-7 Sept. 2012 doi: 10.1109/CONSEG.2012.6349515

[5] Ghosh, D.; Vinoski, S.; , "Scala and Lift Functional Recipes for the Web," Internet Computing, IEEE , vol.13, no.3, pp.88-92, May-June 2009 doi: 10.1109/MIC.2009.68

[6] McGranaghan, M.; , "ClojureScript: Functional Programming for JavaScript Platforms," Internet Computing, IEEE , vol.15, no.6, pp.97-102, Nov.-Dec. 2011 doi: 10.1109/MIC.2011.148

[7] Yue Luo; John, L.K.; , "Workload characterization of multithreaded java servers," Performance Analysis of Systems and Software, 2001. ISPASS. 2001 IEEE International Symposium on , vol., no., pp.128-136, 2001 doi: 10.1109/ISPASS.2001.990688

[8] Scala Official website: http://www.scala-lang.org/node/25, Dec 18,2012.

[9] Scala Official release: http://www.scalalang.org/docu/files/ScalaOverview.pdf, Dec 18,2012

[10] Clojure official website: http://clojure.org, Dec 23,2012.

[11] Hinsen, K.; , "The Promises of Functional Programming," Computing in Science & Engineering , vol.11, no.4, pp.86-90, July-Aug. 2009 doi: 10.1109/MCSE.2009.129

[12] Djuric, D.; Devedzic, V.; , "Incorporating the Ontology Paradigm Into Software Engineering: Enhancing Domain-Driven Programming in Clojure/Java," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on , vol.42, no.1, pp.3-14, Jan. 2012 doi: 10.1109/TSMCC.2011.2140316

[13] Hinsen, K.; , "The Promises of Functional Programming," Computing in Science & Engineering , vol.11, no.4, pp.86-90, July-Aug. 2009 doi: 10.1109/MCSE.2009.129

[14] Vanderburg, G.; , "Clojure Templating Libraries: Fleet and Enlive," Internet Computing, IEEE , vol.14, no.5, pp.87-90, Sept.-Oct. 2010 doi: 10.1109/MIC.2010.111

[15] Di Pierro, Massimo; Skinner, David; , "Concurrency in Modern Programming Languages [Guest editors' introduction]," Computing in Science & Engineering , vol.14, no.6, pp.8-10, Nov.-Dec. 2012 doi: 10.1109/MCSE.2012.111

[16] Martin Kalin, David Miller, "Clojure for Number Crunching on Multicore Machines," Computing in Science and Engineering, vol. 14, no. 6, pp. 12-23, Nov.-Dec., 2012

[17] Sateanpattanakul, S.; Walairacht, A.; , "JGroovy - an extensible Java Programming Language with Groovy," Advanced Communication Technology (ICACT), 2010 The 12th International Conference on , vol.2, no., pp.1139-1144, 7-10 Feb. 2010

[18] Bhat, M.S.; Nair, D.G.; Bansal, D.; Vaishnavi, J.; , "Data structure based performance evaluation of emerging technologies — A comparison of Scala, Ruby, Groovy, and Python," Software Engineering (CONSEG), 2012 CSI Sixth International Conference on , vol., no., pp.1-5, 5-7 Sept. 2012