DART Evolved for Web - A Comparative Study with JavaScript

Sabyasachi Mohanty
Department of Computer Sc. &Engg.
Centurion University of Technology & Management,
Bhubaneswar, India

Smriti Rekha Dey
Department of Computer Sc. &Engg.
TempleCity Institute of Technology & Engineering,
Odisha, India

ABSTRACT
We live in a data centric world with the availability of information on web to satisfy the two folded objectives of secured easy access and quick processing. The web solutions, which connect users to information, are well equipped to provide the best facility to its users and web browsers as a platform, are used to deliver services offered by the large and complex applications. JavaScript (JS) is the most popular and pervasive web scripting language because it is supported by all the web browsers. But, new alternatives are always evaluated to overcome its shortcomings. With the advancement in technology, programming languages have also added the new computing strategies. One such introduced by Google is Dart. With the latest set of programming features, it is such a programming language which is designed to utilize direct hardware capabilities just like C/C++ along with maximum utilization of CPU. The idea discussed in this paper is that the use of Dart as a Programming Language will definitely enhance the user experience. Apart from its use in business applications, it has the potential to revolutionize the application domains such as 3D Graphics, Cryptography, Vector Math Computation, Medical Image Processing, etc., where the volume of data to process is very high.

General Terms
Emerging Trends in Computing, Programming Language

Keywords
Dart, JavaScript, Dart VM, dart2js, SIMD, Web Browser

1. INTRODUCTION
Today, web is everywhere. It means technology enables web access for anyone, anytime, anywhere, using any device - from hand held smart mobile devices to interactive television sets to traditional desk-tops. A web browser on any device can make the web pages, full of data and information, accessible with internet connectivity. No installation or update procedures make the user experience very pleasant [4].

Several languages are used for server side programming but with browser at the client side, we are limited to JavaScript for the last two decades. Though Flash was one of its competitors, but due to its high memory usage, incompatibility with devices like iPads, iPhones etc., and unsuitability for Search Engine Optimization, JavaScript, despite its drawbacks, became the widespread web scripting language.

Because JavaScript is error prone and errors are difficult to spot, it causes problems to programmers. Even though many JavaScript based libraries and frameworks such as JQuery, Backbone.js, CanJS, Node.js, etc., are available, still development and maintenance of large scale, complex web applications are not easy. To address these issues, new programming languages are emerging to meet the demands of recent computing [5] and communication developments. Dart, as a structured language, is ideal for building rich featured, heavy browser based web applications.

The goal of this paper is to evaluate different problems of JavaScript which can be solved by Dart. To achieve this, a comparative study of language features and performance is made.

2. LIMITATIONS OF JAVASCRIPT
Alongside the beauty of JavaScript is its unpleasantness in debugging, performance across browsers, and security on client devices. With the advent of new programming languages like Dart, the drawbacks of JavaScript can be overcomed by the programmers. The major shortcomings that are associated with JavaScript are as follows.

- Lack of Modularity: [2] JavaScript does not follow the concepts of namespace and import statement. Here since the code is divided into libraries which again depend on other libraries, the developers need to know all the interdependencies beforehand. This makes the job of a programmer difficult.
- No Access Modifiers: JavaScript does not have any of the Access Modifiers which helps in setting the level of access to the members.
- No Type Systems: Explicitly declaring the types of variables are not mandatory. A variable can store with any data type say first stores a string and later an integer.
- No Compilation: As there is no compiler, programmers cannot take the advantage of checking errors or identify misuse of data types during compilation time.
- Lack of Generics: JavaScript can store different types of values in a single List. But with Generics a List can store data of only one data type.

It is really a challenging task to develop a large scale application in JavaScript because it lacks many favorable features. Absence of features like build-in module system will definitely make the task of a programmer difficult. There is no particular method to ensure that the third party libraries don't conflict with each other in one application. When there are no boundaries between modules and everything is dependent on everything then to avoid producing festering pile of messy code developers should adhere to a strict coding discipline. This will definitely increase the cost of software development.

3. DART AND ITS COMPARISON WITH JAVASCRIPT
Dart is a new platform for scalable web app engineering [1]. It is an open source, structured programming language for creating complex, browser-based web applications. The applications created in Dart can be executed either by using a browser that directly supports Dart code or by compiling your Dart code to JavaScript. Dart has a familiar syntax, and it is class-based, optionally typed, and single threaded. It has a

concurrency model called isolates that allows parallel execution. Dart code can be converted to JavaScript with dart2js compiler. This means Dart apps can run across all modern web browsers. It can be hosted in the Dart VM (Dart Virtual Machine), allowing both the client and the server parts of applications to be coded in the same language [3]. In addition to running Dart code in web, Dart code can also be run on the command line.

The useful features [9] of the Dart programming language that enables the programmer to build the next generation web apps are

- Easy to Learn: Developers from any domain can learn Dart quickly. It is an object oriented language with classes, single inheritance, top-level functions, lexical scope, and a familiar syntax.
- Optional Static Types: Dart supports types without requiring them. Darts optional types are static type annotations leading to´ better warning and error messages along with the requirement of fewer comments to document the code.
- Innocent until proven guilty: Dart minimizes the situations that result in compilation-time error. Warnings are the conditions that do not stop the program from running.
- Lexical Scope: Visibility of variables are defined by the program structure.
- Real Classes baked into the Language: Dart uses classes naturally.
- Top-level functions: Programmers can define functions at the top level, outside of any class. This makes composition of library to feel more natural.
- Classes have implicit interfaces: Elimination of explicit interfaces simplifies the language.
- Named Constructors: Developers are free to assign constructor names, which helps with readability.
- Factory Constructors: A factory constructor can return a singleton, an object from a cache, or an object of a sub-type.
- Isolates: Dart supports safe, simple concurrency execution of code with isolates. Communication happens by sending messages over ports.
- Dart compiles to JavaScript: Dart has been designed to compile to JavaScript using dart2js so that apps developed in Dart can run across all modern web.
- Dart runs in the client and on the server: Dart can be used for full end-to-end apps. The Dart virtual machine (Dart VM) can be integrated into a web browser.
- Strong tooling: The Dart ships with an editor, to write, launch, and debug apps. It helps with code completion, detection of potential bugs, code navigation, quick fixes and refactoring.
- Libraries for reusability: Wide array of libraries include built-in types and fundamental features such as collections, dates and regular expressions. Dart has built-in library support for files, directories, sockets and even web servers. Programmers code import a library, and libraries can be re-exported leading to code sharing.
- Scalable: Dart scales from small scripts to large and complex apps. Programs can start small and grow over time with the support for top-level functions, classes and libraries.
- String Interpolation: Building strings with variables makes the life of programmer easy.
- noSuchMethod: Dart is a dynamic language, and it has the facility to make arbitrary method calls with noSuchMethod().
- Generics: Dart's generics are more simple.
- Support of code sharing: Dart package manager (pub) and features like libraries, can locate, install and integrate code across the web.
- Open source: Dart was born for the web.

Table 1. Feature Comparison of Dart and JavaScript

Features	Java-Script	Dart	Summary
Static Type Checking	-	✓	Static typed languages are those in which type checking is done at compilation-time
Classes	-	✓	.
Interfaces	-	✓	.
Modules	-	✓	.
String In-terpolation	-	✓	String that is built by inserting a string or replacing a variable with its value
Intellisense	-	✓	Intelligent code sense or auto completion of code
Code Brevity	-	✓	If the number of lines of code is reduced
Better Speed	-	✓	Improvement in the performance of the code

```
1  double scalar_average(Float32List data) {
2  var sum = 0.0;
3  for (vari = 0; i<data.length; i++) {
4  sum += data[i];
5  }
6  return sum / data.length;
7  }
```

```
1  double simd_average(Float32x4List data) {
2  var sum = new Float32x4.zero();
3  for (vari = 0; i<data.length; i++) {
4  sum += data[i];
5     }
6  var total = sum.x + sum.y + sum.z + sum.w;
7  return total / (data.length * 4);
8  }
```

Figure 1. Scalar (top) and SIMD (bottom) implementations of an algorithm to find the average of an array of numbers

Table 1 shows the feature comparison of Dart and JavaScript.

4. SIMD PROGRAMMABILITY MATTER TO THE WEB

Many algorithms can be executed faster by taking advantage of SIMD co-processors. A simple example is the averaging of an array of numbers. The algorithm adds all the data items and computes the average by dividing by the number of data items. The top of Figure 1 shows a scalar implementation. The bottom of the figure shows a SIMD implementation. This algorithm can trivially take advantage of SIMD co-processors by adding 4 numbers at the same time.

The bulk of the work is done in parallel and only after exiting the loop does the program need to fall back to scalar computation when computing the final sum and average.

Table 2. Comparison of Double and Integer

Operation	Double	Integer	Double Slowdown
Multiply	6	2	3x
Addition	4	1	4x
Load	2	2	N/A
Store	2	2	N/A

If the Float32x4 type were available to web programmers and the optimizing compiler is successful in generating code that is free of memory allocation and allows for temporary values to stay in CPU registers, the algorithm can be sped up by 500% [6].

5. COMPUTING FEATURE OF DART

Dart is designed to run fast by being less permissive. The new Virtual Machine (VM) opens up new possibilities with SIMD (Single Instruction Multiple Data). This SIMD programming model is designed to give direct control to the programmer.

5.1 New Types

Dart introduces three new 128-bit wide value types: Float32x4, Int32x4, and Float64x2. Each value type stores scalar values in multiple "lanes". For example, Float32x4 has four single precision floating point numbers in lanes labelled: x, y, z, and w. Each instance is immutable and all operations result in a new instance.

5.2 Distinction between Integer and Double Numbers

JavaScript only has double. So all operations involve double and in computation point of view, Double arithmetic operation is slower than integer arithmetic operation. This difference is very high for mobile processors. Dart has both double and integer. So it gives choice to developers. Table 2 shows the difference.

5.3 Primitive Operations

Float32x4 supports standard arithmetic operations (+, -, *, /) as well as approximate square root (sqrt), reciprocal square root (rsqrt), and reciprocal. It also supports absolute value (abs), minimum (min), maximum (max), and clamp operations [6]. All of these operations are performed for each lane. For example, the minimum of two Float32x4 is the Float32x4 with the minimum of each individual lane.

5.4 Type Conversion

Value cast operations between Float32x4 and Int32x4 as well as Float32x4 and Float64x2 are available [6]. Also, Bit-wise cast operations between Float32x4, Int32x4, and Float64x2 are present.

5.5 Comparison and Branchless Selection

The result of comparison of SIMD values is not a single boolean value but a boolean value for each lane [6]. Consider the example of computing the minimum value of two values. Figure 2 shows the scalar and SIMD algorithms written in Dart.

The comparison results in an Int32x4 value with lanes containing 0xFFFFFFFF or 0x0 when the lane comparison is true or false respectively. The resulting mask is used to pick the desired value.

5.6 Lane Access

Direct access to each lane of a Float32x4 is done by accessing the x, y, z, or w instance properties. An example is shown in the average algorithm.

```
1  numscalar_min(num a, num b) {
2  if (a <= b) {
3  return a;
4    }
5  return b;
6  }
```

```
1  Float32x4 simd_min(Float32x4 a, Float32x4 b) {
2  Int32x4 mask = a.lessThanOrEqual(b);
3  return mask.select(a, b);
4  }
```

Figure 2. The scalar (top) and SIMD (bottom) minimum function

```
1  void copy(Float32x4List destination,
2  Float32x4List source,
3  int n) {
4  for (vari = 0; i< n; i++) {
5  var x = source[i]; // Load.
6  destination[i] = x; // Store.
7    }
8  }
```

Figure 3. The copy function copies an array of SIMD values

Because each instance is immutable it is not possible to change the value stored in the lanes. Methods, for example, withX allow for constructing new instances that are copies of an existing instance with an individual lane value changed [6]. For example, in Dart:

```
var x = new Float32x4(1.0, 2.0, 3.0, 4.0); var y =
x.withX(5.0);
```

5.7 Shuffling

Shuffling the order of lanes is also available [6]. Reversing the order of the lanes, in Dart:

```
Float32x4 reverse(Float32x4 v) {
  returnv.shuffle(Float32x4.WZYX);
}
```

The shuffle method uses an integer mask.

5.8 Memory I/O

Float32x4List, Int32x4List, and Float64x2List offer contiguous storage of Float32x4, Int32x4, and Float64x2 values. These lists do not store instances but their 128-bit payloads [6]. Figure 3 shows loading and storing SIMD values in Dart. Note that on the load on line 5 a new instance of Float32x4 is constructed.

6. DART VIRTUAL MACHINE

The Dart Virtual Machine (Dart VM) is the core of the Dart language [3]. One use is as an executable on the command-line VM, such as to start up an HTTP server or run a script, or any other console-based use of Dart. Another use is to embed it in another application, such as Dartium.

Dart VM stretches the performance envelop and makes new, magical experiences possible.

6.1 Unoptimized Code

When a function is first compiled by the Dart VM the generated code is completely generic. Every method call is looked up in the receiving object's class's function table. Every temporary value is allocated in the heap as a full object under control of the Garbage Collector (GC) [6].

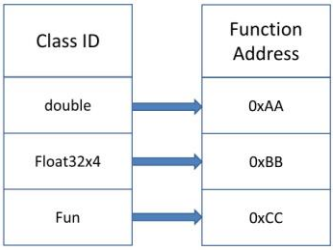

Figure 4.A call-site's type-cache.

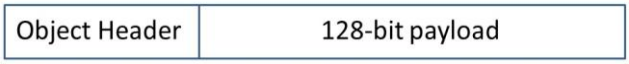

Figure 5.A boxed SIMD value.

6.2 Mapping from High Level to Low Level

The programming model is high level with each operation requiring a method call on a heap allocated object and results in a new heap allocated object holding the resulting value. Each value is immutable and storage of temporary values cannot be reused. When optimized code is generated, the overhead of the high level programming model can be removed. Almost all method calls will be mapped directly to a single CPU instruction. Instances will be stored directly inside CPU registers avoiding the cost of memory allocation and object creation.

6.3 Type Collection

The unoptimized code collects important type information that is used later by the optimizing compiler. At each method call the unoptimized code maintains a cache mapping from receiver class id to address of the class's corresponding function, as shown in Figure 4.

6.4 Boxed and Unboxed Values

The Dart compiler makes a distinction between boxed and unboxed values. Boxed values are pointers to objects which are allocated in the heap whose life cycle is managed by the Garbage Collector (GC). Unboxed values are stored in CPU registers. Operations on unboxed values are much more efficient because the values are already contained in CPU registers. Figure 5 shows the in-memory layout of an instance of Float32x4. The object header contains information used for type collection and GC [6].

6.5 Inlining

Both the Dart VM and the JavaScript VMs make heavy use of inlining to avoid method call invocation and unnecessary boxing of values. The first form of inlining is that the bodies of small functions are copied into the calling function, replacing the method call. The second form of inlining is replacement of runtime provided functions with compiler intermediate representation (IR) instructions.

6.6 Optimized Code

The optimized code first validates the class of each input value. After being validated the values are unboxed directly into CPU registers and the remaining operations are performed directly in CPU registers with no memory I/O. The last step is to box the result so that it can be returned.

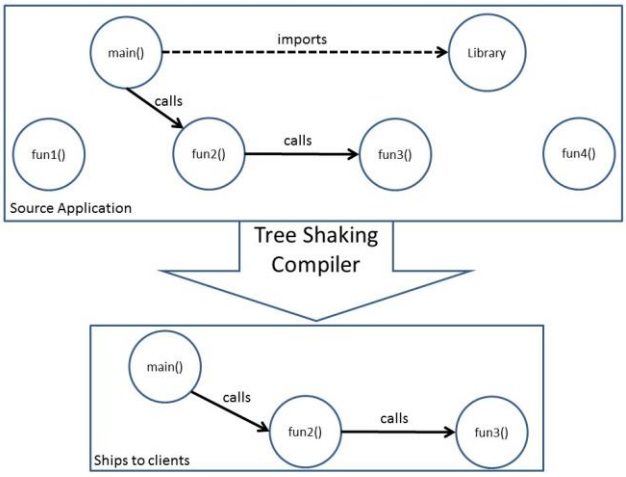

Figure 6.Tree Shaking.

7. DART2JS: COMPILING TO JAVASCRIPT

The dart2js tool is used to compile Dart to JavaScript. In other words, it produces a .js file that contains the JavaScript equivalent of Dart code at application level. For every feature of Dart, there is a corresponding chunk of JavaScript Code that gets included in the compiled output and the JavaScript generated includes shim code for the various Dart libraries [7]. The referenced Dart libraries are also added into the resulting JavaScript. If any error is detected during compilation, it helps the programmer by identifying where the errors occur.

In addition to all these, Dart has also implemented features enlisted below that reduces the execution time of the generated JavaScript Code and provides better performance than that of handwritten JavaScript.

- Minification of Code: When compiling to JavaScript, dart2js generates the smallest amount of bytes. Advantages of less bytes are smaller bandwidth bills, and faster load times, and longer battery life.
- Tree shaking: It is a technique to "shake" off unused code. Dart tools support tree shaking. As shown in the Figure 6, only the functions that are actually used are included in the generated output.
- Dead code elimination: During compile time, configuration values from the environment can be evaluated and potentially result in eliminating dead codes from the generated output.

8. PERFORMANCE

Performance can be judged with the standardized benchmarks. The point of the benchmark is to have an easy to run, reproducible stress test of the performance-sensitive algorithm. Google also used to promote new benchmark tests comparing Dart with JavaScript on four benchmarks (DeltaBlue, FluidMotion, Richards, and Tracer) [1]. Seth Ladd pointed out the Richards benchmark in particular, where he said Dart 1.1 ran 25 percent faster JavaScript [8].

In this section, we present a set of benchmark results for our benchmark programs. We ran our benchmarks on the Dart VM, V8, and JavaScript implementations. We used a Windows 7

(32bit) system with an Intel CPU (Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 2501 Mhz, 2 Core(s), 4 Logical Processor(s)).

Table 3. Benchmark Comparison results for Dart on Chromium

Benchmark	Dart (Scalar)	Dart (SIMD)
Average	17	4
Matrix Multiplication	111	18
Vector Transform	26	5
Mandelbrot	446800	163615

Table 4. Benchmark Comparison results for Dart and JavaScript V8 Engine

Benchmark	Java-Script	Dart
Average	20	21
Matrix Multiplication	113	112
Vector Transform	34	32
Mandelbrot	450600	450700

We ran each program with an interval of at least 5 seconds, then calculated the time consumed by a single iteration by dividing the result withthe number of iterations. We report absolute times in microseconds for both Dart and JavaScript benchmark implementations.
To cover different application domains such as 3D Graphics, Cryptography, 2D Image Processing, we have identified a small set of common operations as Benchmark Programs to showcase the impact of Dart on the Browsers. The benchmarks covered so far are listed as follows:

- Average : Compute the average
- Matrix Multiplication : Multiply 2 4X4 matrix
- Vector Transform : Vector Transformation 4 element vector
- Mandelbrot : Represent Mandelbrot set

Table 3 shows results of benchmark comparison for Dart executed with Chromium on our Intel machine and windows operating system. The time is reported in microseconds for both scalar and SIMD versions of the benchmark.

Table 4 shows results of benchmark comparison for Dart and JavaScript V8 Engine on the same Intel machine with windows operating system.

9. CONCLUSION

This paper explores the the design features of Dart. The results from its comparison with JavaScript, indicate that it has the potential to lead the world as a general-purpose language to build many different types of real-time applications. With concepts of classes, interfaces and modules, better IDE i.e., own editor and browser (Dartium) for software development, easy code debugging, a cleaner DOM API and libraries, developing rich featured client side web applications is easy. In addition to all these and improved computing capability with SIMD programmability, Dart really shines, when building complex web applications. Projects with the concept of data-driven decision making such as E-Governance, Enterprise Resource Planning, Gene Analysis, etc., can be highly benefited from Dart.

10. ACKNOWLEDGMENT

The results of this paper were obtained during our Ph.D. studies. We would like to express deep gratitude to Prof. (Dr.) Amit Kumar Mishra, for his valuable suggestions and technical directions.

11. REFERENCES

[1] Dart: Structured web apps, February 2014, https://www.dartlang. org/.

[2] Aansa Ali, Evaluation and comparison of alternate programming languages to javascript, Research Conference in Technical Disciplines (2013), 90–95, http://www.rcitd.com.

[3] Chris Buckett, Dart in action, Manning, 20 Baldwin Road, PO Box 261, Shelter Island, NY 11964, 2013, ISBN 9781617290862.

[4] Karan Dhiman and Benson Quach, Google's go and dart: parallelism and structured web development for better analytics and applications, Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '12) (2012), 253–254, http://dl.acm.org/citation.cfm?id=2399809.

[5] Sixto Ortiz Jr., Computing trends lead to new programming languages, the IEEE Computer Society 45 (2012), no. 7, 17–20, doi:10.1109/MC.2012.229.

[6] John McCutchan, HaitaoFeng, Nicholas D. Matsakis, Zachary Anderson, and Peter Jensen, A simd programming model for dart, javascript, and other dynamically typed scripting languages, Workshop on Pro-gramming Models for SIMD/Vector Processing (WPMVP '14) (2014), http://dx.doi.org/10.1145/2568058.2568066.

[7] Chris Strom, Dart for hipsters, The Pragmatic Programmers, 2012, ISBN-13: 978-1-937785-03-1.

[8] Rob Marvin (SD Times), Google releases dart 1.1 with new features and improved tools, January 2014, http://sdt.bz/content/ article.aspx?ArticleID=67599&page=1.

[9] Kathy Walrath and Seth Ladd, Dart up and running, O Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2012, ISBN 978-1-449-33089-7.