Liquid IT : Toward a better compromise between development scalability and performance scalability not definitive

Etienne Brodu

June 9, 2015

Abstract

TODO translate from below when ready

Résumé

Internet étend l'économie à une échelle spatiale et temporelle sans précédent. Il permet à chacun de mettre un service à disposition de milliards d'utilisateurs, en seulement quelques heures. La plupart des grands services actuels ont commencés comme de simples applications créées dans un garage par une poignée de personnes. C'est cette facilité à l'entrée qui a permis jusqu'à maintenant une telle croissance sur le web. Google, Facebook ou Twitter en sont les exemples les plus flagrants. Pendant le développement d'une application, il est important de suivre cette croissance, au risque de se faire rattraper par la concurrence. Le développement est guidé par les besoins en terme de fonctionnalités, afin de vérifier rapidement si le service peut satisfaire l'audience. Des langages tel que Ruby ou Java se sont imposés comme les languages du web, justement parce qu'ils permettent d'intégrer facilement de nouvelles fonctionnalités.

Si une application répond correctement aux besoins, elle atteindra de manière virale un nombre important d'utilisateurs. Son audience peut prendre plusieurs ordres de grandeurs en quelques jours seulement, ou même en quelques heures suivant comment elle est relayée. Une application est dites scalable si elle peut absorber ces augmentations d'audience. Or il est difficile pour une application dont le développement est guidé par les fonctionnalités d'être scalable.

Au moment où l'audience commence à devenir trop importante, il est nécessaire de modifier l'approche de développement de l'application. Le plus souvent cela implique de la réécrire complètement en utilisant des infrastructures scalables qui imposent des modèles de programmation et des API spécifiques. Cela représente une charge de travail conséquente et incertaine. De plus, l'équipe de développement dois concilier cette nouvelle approche de développement scalable, avec la demande en fonctionnalités. Aucun langage n'a clairement réussi le compromis entre ces deux objectifs.

Pour ces raisons, ce changement est un risque pour la pérennité de l'application. D'autant plus que le cadre économique accorde peu de marges d'erreurs, comme c'est le cas dans la plupart des start-up, mais également dans de plus grandes structures.

Mon travail consiste à tenter d'écarter ce risque dans une certaine mesure. Ma thèse se base sur les deux observations suivantes. D'une part, Javascript

est un langage qui a énormément gagné en popularité ces dernières années. Il est omniprésent sur les clients, et commence à s'imposer également sur les serveurs avec Node.js. Il a accumulé une communauté de développeur importante, et est l'environnement d'exécution le plus largement déployé. De ce fait, il se place comme le langage principal du web, détrônant Ruby ou Java. D'autre part, l'execution de Javascript a la particularité de ressembler à un pipeline. La boucle événementielle de Javascript est un pipeline qui s'exécute sur un seul cœur pour profiter d'une mémoire globale. On observe le même flux de messages traversant cette boucle événementielle que dans un pipeline.

L'objectif de ma thèse est de permettre à des applications développées en Javascript d'être automatiquement transformées vers un pipeline d'exécutions repartis. Nous construisons un compilateur permettant d'identifier les fonctions de Javascript et de les isoler dans ce que nous appelons des Fluxions. Un conteneur qui peut exécuter une fonction à la réception d'un message, et envoyer des messages pour continuer le flux vers d'autres fluxions. Les fluxions étant indépendantes, elles peuvent être déplacées d'une machine à l'autre. En transformant automatiquement un programme Javascript en Fluxions, on le rend scalable, sans effort.

Contents

Chapter 1

Introduction

TODO 5p

Chapter 2

Context and objectives

Contents

2.1 The Web as a platform

2.1.1 From OS to Web platform

The focus of the software industry switched from native desktop applications to mobile and web applications. In this TODO continue here

2.1.2 The languages of the web

TODO This paragraph needs a lot of rewritting Python(1993), Javascript(1995), Ruby(1993), PHP(1995) and Java(1994) were all created around the same time, in the early 90's. Java imposes itself early as the language of the web, and never really decreased. It is a solid, professional language used by most of the industry. But it is too big, and too slow growing for the exact same reason. TODO precise this, too big and too slow are not precise enough. Maybe cite some comment about this

Python is older (1991), and was always seen as a general purpose language. It is only in 2003, with the Django framworks that Python start to be seen as a web language.

PHP short-lived as the easy-to-use scripting language, it is now backed by facebook, but is on the decline. TODO precise this, why is php on the decline ? numbers and arguments please

Ruby took-off in 2005 with Rails, and is still in active use. But it seems it is going to be eclipsed by Javascript. TODO precise this, again, with arguments and numbers

Since a few years, Javascript is in a constant rise as the main language of the web, because of Ajax first, and then Node.js What is so different with Javascript ? It is omnipresent, from every browser, to the server. And it is a target for LLVM. Because of this position, it became fast (V8, ASM.js ...) and usable (ES6, ES7).

2.1.3 Explosion of Javascript popularity

2.1.3.1 In the beginning

Javascript was created by Brendan Eich at Netscape around May 1995, and released to the public in September. The initial name of the project was Mocha, then LiveScript, the name Javascript was finally adopted to leverage the trend around Java. The latter was considered the hot new web programming language at this time. It was quickly adopted as the main language for web servers, and everybody was betting on pushing Java to the client as well. The history proved them wrong.

In 1995, when Javascript was released, the world wide web started its wide adoption.[1] Browsers were emerging, and started a battle to show off

[1] `http://www.internetlivestats.com/internet-users/`

the best features and user experience to attract the wider public.[2] Microsoft released their browser Internet Explorer 3 in June 1996 with a concurrent implementation of Javascript. They changed the name to JScript, to avoid trademark conflict with Oracle Corporation, who owns the name Javascript. The differences between the two implementations made difficult for a script to be compatible to both. At the time, signs started to appear on web pages to warn the user about the ideal web browser to use for the best experience on this page. This competition was fragmenting the web.

To stop this fragmentation, Netscape submitted Javascript to Ecma International for standardization in November 1996. In June 1997, ECMA International released ECMA-262, the first specification of ECMAScript, the standard for Javascript. A standard to which all browser should refer for their implementations.

The base for this specification was designed in a rush. The version released in 1995 was finished within 10 days. Because of this precipitation, the language has often been considered poorly designed and unattractive. Moreover, Javascript was intended to be simple enough to attract unexperienced developers, by opposition to Java or C++, which targeted professional developers. For these reasons, Javascript started with a poor reputation among the developer community.

But things evolved drastically since. When a language is released, available freely at a world wide scale, and simple enough to be handled by a generation of teenager inspired by the technology hype, it produce an effervescent community around what is now one of the most popular and widely used programming language.

2.1.3.2 Rising of the unpopular language

TODO why is Javascript unpopular ? cite some blog post like : https://wiki.theory.org/YourLangua and add many blog posts titles in a mozaïc

Javascript started as a programming language to implement short interactions on web pages. The best usage example was to validate some forms on the client before sending the request to the server. This situation hugely improved since the beginning of the language. So much that web-based, Javascript applications are currently now favored instead of rich, native desktop applications.

[2] to get an idea of the web in 1997 : `http://1x-upon.com/`

ECMA International released several version in the few years following the creation of Javascript. The first and second version, released in 1997 and 1998, brought minor revisions to the initial draft. However, the third version, released in the late 1999, contributed to give Javascript a more complete and solid foundation as a programming language. From this point on, the consideration for Javascript keep improving.

An important reason for this reconsideration started in 2005. James Jesse Garrett released *Ajax: A New Approach to Web Applications*, a white paper coining the term Ajax [**Garrett2005**]. This paper point the trend in using this technique, and explain the consequences on user experience. Ajax stands for Asynchronous Javascript And XML. It consists of using Javascript to dynamically request and refresh the content of a web page. The advantage is that it avoids to request a full page from the server. Javascript is not anymore confined to the realm of small user interactions on a terminal, it can be proactive and responsible for a bigger part in the system spanning from the server to the client. Indeed, this ability to react instantly to the user started to narrow the gap between web and native applications. At the time, the first web applications to use Ajax were Gmail, and Google maps[3].

Around this time, the Javascript community started to emerge. The third version of ECMAScript had been released, and the support for Javascript was somewhat homogeneous on the browsers but far from perfect. Moreover, Javascript is only a small piece in the architecture of a web-based client application. The DOM, and the `XMLHttpRequest` method, two components on which AJAX relies, still present heterogeneous interfaces among browsers. To leverage the latent capabilities of Ajax, and more generally of the web, Javascript framework were released with the goal to straighten the differences between browsers implementations. Prototype[4] and DOJO[5] are early famous examples, and later jQuery[6] and underscore[7]. These frameworks are responsible in great part to the wide success of Javascript and of the web technologies.

In the meantime, in 2004, the Web Hypertext Application Technology

[3] A more in-depth analysis of the history of Ajax, given by late Aaron Swartz `http://www.aaronsw.com/weblog/ajaxhistory`

[4] `http://prototypejs.org/`

[5] `https://dojotoolkit.org/`

[6] `https://jquery.com/`

[7] `http://underscorejs.org/`

Working Group[8] formed to work on the fifth version of the HTML standard. This new version provide new capabilities to web browsers, and a better integration with the native environment. It features geolocation, file API, web storage, canvas drawing element, audio and video capabilities, drag and drop, browser history manipulation, and many mores It gave Javascript the missing pieces to become a true language for developing rich application. The first public draft of HTML 5 was released in 2008, and the fifth version of ECMAScript was released in 2009. With these two releases, ECMAScript 5 and HTML5, it is a next step toward the consideration of Web-based technologies as equally capable, if not more, than native rich applications on the desktop. Javascript became the programming language of this rising application platform.

However, if web applications are overwhelmingly adopted for the desktop, HTML5 is not yet widely accepted as ready to build complete application on mobile, where performance and design are crucial. Indeed web-technologies are often not as capable, and well integrated as native technologies. But even for native development, Javascript seems to be a language of choice. An example is the React Native Framework[9] from Facebook, which allow to use Javascript to develop native mobile applications. They prone the philosophy *"learn once, write anywhere"*, in opposition to the usual slogan *"write once, run everywhere"*.[10]

2.1.3.3 Current situation

"When JavaScript was first introduced, I dismissed it as being not worth my attention. Much later, I took another look at it and discovered that hidden in the browser was an excellent programming language."

—Douglas Crockford

The success of Javascript is due to many factors ; I mentioned previously the standardization, Ajax libraries and HTML5. Another factor, maybe the most important, is the View Source menu that reveals the complete source code of any web application. *The view source menu is the ultimate form of open source*[11]. It is the vector of the quick dissemination

[8] `https://whatwg.org/`

[9] `https://facebook.github.io/react-native/`

[10] Used firstly by Sun for Java, but then stolen by many others

[11] `http://blog.codinghorror.com/the-power-of-view-source/`

of source code to the community, which picks, emphasizes and reproduces the best techniques. This brought open source and collaborative development before github. TODO neither open source nor collaborative development are the correct terms Moreover, all modern web browsers now include a Javascript interpreter, making Javascript the most ubiquitous runtime in history [**Flanagan2006**].

When a language like Javascript is distributed freely with the tools to reproduce and experiment on every piece of code. When this distribution is carried during the expansion of the largest communication network in history. Then an entire generation seizes this opportunity to incrementally build and share the best tools they can. This collaboration is the reason for the popularity of Javascript on the Web.

It seems to also infiltrate many other fields of IT, but it is hard to give an accurate picture of the situation. There is no right metrics to measure programming language popularity. In the following paragraphs, I report some popular metrics and indexes available on the net. More detailed informations are available section A.

Search engines The TIOBE Programming Community index is a monthly indicator of the popularity of programming languages. It uses the number of results on many search engines as a measure of the activity of a programming language. Javascript ranks 6th on this index, as of April 2015, and it was the most rising language in 2014. However, the measure used by the TIOBE is controversial. Some says that the measure is not representative. It is a lagging indicator, and the number of pages doesn't represent the number of readers.

On the other hand, the PYPL index is based on Google trends to measure the activity of a programming language. Javascript ranks 7th on this index, as of May 2015.

From these indexes, the major programming languages are Java, C/C++ and C#. The three languages are still the most widely taught, and used to write softwares. But Javascript is rising to become one of these important languages.

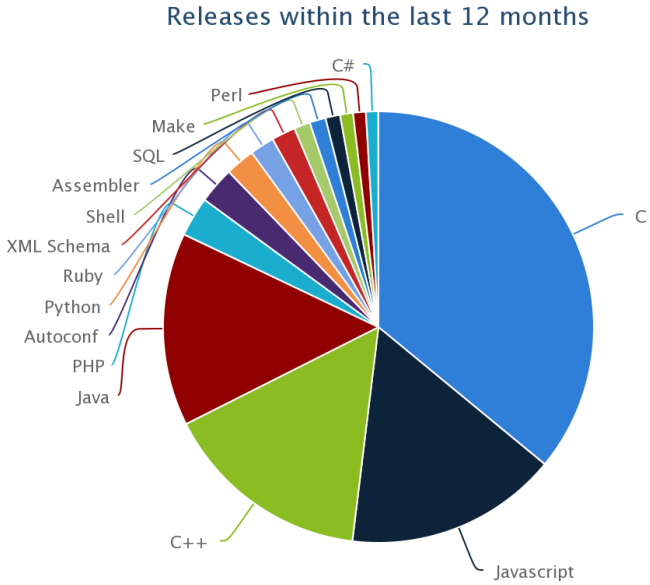
Developers collaboration platforms Github is the most important collaborative development platform, with around 9 millions users. Javascript is the most used language on github since mid-2011, with more than 320 000

repositories. The second language is Java with more than 220 000 repositories.

TODO : graph of Github repositories by languages

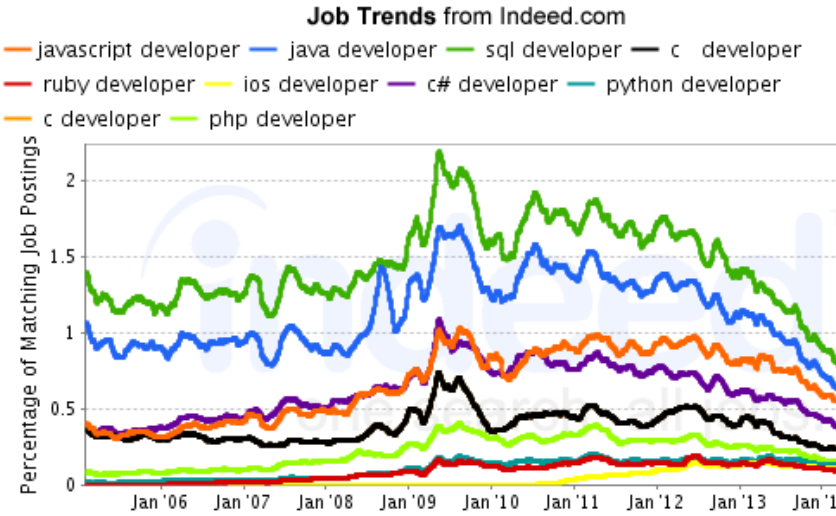
StackOverflow, is the most important Q&A platform for developers. It is a good representation of the activity around a language. Javascript is the second language showing the most activity on StackOverflow, with more than 840 000 questions. The first one is Java with more than 850 000 questions.

Black Duck knowledgebase analyzes 1 million repositories over various forges, and collaborative platforms to produce an index of the usage of programming language in open source communities. Javascript ranks second. C is first, and C++ third. Along with Java, the four first languages represent about 80% of all programming language usage.



TODO redo this graph, it is ugly.

Jobs All these metrics are representing the visible activity about programming language. But not the entire software industry is open source, and the activity is rather opaque. To get a hint on the popularity of programming languages used in the software industry, let's look at the job offerings. Indeed provide some insightful trends. Javascript developers ranked at the third position, right after SQL developers and Java developers. Then come C# and C developers.



TODO redo this graph, it is ugly.

All these metrics represent different faces of the current situation of Javascript adoption. We can safely say that Javascript is one of most important language of this decade, along with Java, C/C++. It is widely used in open source projects, and everywhere on the web. But it is also trending, and maybe slowly replacing languages like Java.

Future trends TODO

Code reuse. Why it never worked ?

em-scripten

https://github.com/kripken/emscripten Javascript is a target language for LLVM, therefor everything can compile to Javascript : JS is the assembler of the web.

Isomorphic Javascript

Server-side Javascript

https://www.meteor.com/ https://facebook.github.io/flux/ Javascript can be executed both on the client and the server. That allow use-cases never possible before (server pre-rendering, same team ...)

Reactive

http://facebook.github.io/react/ Javascript is used to model the flow of propagation of state in a web application

—

Some facts to include : https://www.destroyallsoftware.com/talks/the-birth-and-death-of-javascript The Atom editor is written in Javascript node.js. Now, major PaaS (which one) support node.js by default. Heroku support

Python, Java, Ruby, Node.js, PHP, Clojure and Scala Amazon Lambda Web service support node.js in priority. npm raises 8m. http://techcrunch.com/2015/04/14/popular-javascript-package-manager-npm-raises-8m-launches-private-modules/

With numbers from Worldline about Java and Javascript usages.

It ends with some success stories with Javascript (Paypal, Linkedin and so on ...).

-> There is a growing mass of developers for Javascript. And even for those who don't program in Javascript, there is transpilers. Javascript is here to stay : http://www.javaworld.com/article/2077224/learn-java/is-javascript-here-to-stay-.html (an article from 1996, where Java was still the hot new language)

2.2 The pivot Problem

2.2.1 Separation of concerns

The languages presented in the last section all have large developing community. Some are functional, others are object-oriented. But they all provide some ways to apply the separation of concerns principle : modules. Think of a module as an object in OOP, or a monad(?) in FP.

It assure that the behavior of a module is completly contained in this module, and does not depends on external parts. Splitting a large program into many modules allow developers to safely modify a module only by understanding itself, and its interface. It is the divide and conquer strategy, applied to computer science.

The separation of concerns principle incites to design simple, generic, reusable modules. Some modules can provide more complex behavior, by relying on, and composing with other modules. The separation of concern impose a module to be generic. It must be independent from where, and how it is used, to be able to be reused. De facto, a module is meant to be disseminated through the code base of the program. Implying that if the module holds state, this state is available in a global memory, and that there is no concurent access.

This principle allows to quickly develop, and maintain an application. For a new web application to meet the requirements of its user community, it is crucial to be able to quickly modify its codebase. And that is why the development of most, if not all, web applications starts using language

allowing with modularity.

2.2.2 Performance Scalability

Yet, because of the global memory used to centralize module state, while disseminating the module methods, it is difficult for this first approach to scale. Indeed, scalability is known to come from parallelism, which is incompatible global, shared memory space. TODO explain these shortcuts TODO I completly skip the point about invariance. The developer provides insight about parallelization by manipulating the points where invariance of the state is assured. Isolation, locks, or yields.

At some point, the development team discard this first approach, for a distributed execution model. Such model distribute the execution onto different computers communicating by messages. To simplify the argument, I called these isolated computers, actors. TODO change that name. The essence of an actor is to have only one execution thread, to be able to isolate its state. And if its state is isolated, there is no need to reduce the scope of this actor to reach other actors. TODO this argument is not strong enough

Actors are not composable the way modules are. The different functions of a module are used in the call / return fashionn while actors are used in the fire-and-forget fashion. An actor sends its result to another actor, than the one which sent the initial message. It is not possible to compose an actor, like it is possible for a module. That is, it is not possible to use an actor in different contexts.

2.2.3 The holy graal

Is it still possible to combine the advantages of modules, with the advantages of actors ? Is it possible for an actor to be the boundary for some modules ? Or is it possible for a module to be composed of actors ?

To answer the first question. If a module is limited inside an actor, then the reusability of the module is te be questioned. It loses its point as a generic component, reusable through the codebase. Or it is stateless. Or its state is boundable by the actor.

To answer the second question. For a module to be composed of actors, means that the messages the actors receive contains somehow information about the context they are executed. That is, at least, the actors to get the result back.

So, in some extents, it is possible to establish an equivalence between modules and actors. However, the transition is too tedious to be done manually. That is why most web applications neglect the module approach to adopt the actor approach, when true scalability is needed. At the lost of the development ease.

TODO why ?

The transposition from the module organization, to the actor organization is a tedious process, and should be handled by a compiler. Therefore, to have both performance through parallelism and still attract a large developer community, we need to find an equivalence between the module organization, and the performance point of view on parallelism.

TODO now talk about the limitations of this transition. you mention earlier that actors need to receive information about their context, and that modules needs to be stateless, or that their states needs to be bounded by the actor.

So the hypothesis is : by providing a tool to transform the invariance of cooperative scheduling into isolation and message-passing, it should be possible to provide a developer-friendly scalable paradigm. I don't know if hypothesis is the right term for this assumption

This transformation is done by isolating the memory use by each atomic execution of the event-loop (the callbacks). But this isolation is not possible for every kind of application. Applications could be classified in the level of coordination necessary in their design. (See below for a classification from Gunther's Universal Scalability Law) At a very end of the spectrum, there is static content servers, where there is no coordination, everything is parallel. At the very other end of this spectrum, there are consistent databases, where the coordination require the application to be highly sequential.

Web applications that are designed in a certain way (flow-based web applications -> no retropropagation) are in the middle of this spectrum. We advance as the thesis that it is possible for these types of web applications to isolate the memory for each atomic execution of the event-loop (the callbacks).

The different kind of applciations, from Gunther's Universal Scalability Law (USL) : Application classes for the USL model.

A	**Ideal concurrency** $(\alpha, \beta = 0)$ Single-threaded tasks Parallel text search Read-only queries
B	**Contention-limited** $(\alpha > 0, \beta = 0)$ Tasks requiring locking or sequencing Message-passing protocols Polling protocols (e.g., hypervisors)
C	**Coherency-limited** $(\alpha = 0, \beta > 0)$ SMP cache pinging Incoherent application state between cluster nodes
D	**Worst case** $(\alpha, \beta > 0)$ Tasks acting on shared-writable data Online reservation systems Updating database records

2.3 Proposal and Hypothesis

2.3.1 LiquidIT

A general definition of Liquid IT. And more specifically the focus on dev and perf scalability. Liquid IT should try to bring a solution to this compromise leveraging the particular position of Javascript and its event-loop.

2.3.2 Parallelization and distribution of web applications

2.3.3 Hypothesis and Thesis

Chapter 3

State of the art

3.1 Javascript

3.1.1 Overview of the language

3.1.1.1 Functions as First-Class citizens

3.1.1.2 Lexical Scoping

3.1.1.3 Closure

3.2 Concurrency

3.2.1 Two known concurrency model

3.2.1.1 Thread

3.2.1.2 Event

3.2.1.3 Orthogonal concepts

3.2.2 Differentiating characteristics

3.2.2.1 Scheduling

3.2.2.2 Coordination strategy

3.2.3 Turn-based programming

3.2.3.1 Event-loop

3.2.3.2 Promises

3.2.3.3 Generators

3.2.4 Message-passing / pipeline parallelism -> DataFlow programming ?

3.3 Scalability

3.3.2 Scalability outside computer science (only if I have time)

If I have time, I would like to try to explain why scalability is at the core of material engagement and information theory, and is at the core of our universe : the propagation of Gravity wave is an example : it is impossible to scale

3.4 Framworks for web application distribution

3.4.1 Micro-batch processing

3.4.2 Stream Processing

3.5 Flow programming

3.5.1 Functional reactive programming

3.5.2 Flow-Based programming

3.6 Parallelizing compilers

OpenMP and so on

3.7 Synthesis

There is no compiler focusing on event-loop based applications

Chapter 4

Fluxion

4.1 Fluxionnal Compiler

Some parts of this are already written in the first paper. It needs a lot additional explanations and rewritting

4.1.1 Identification

4.1.1.1 Continuation and listeners

4.1.1.2 Dues

4.1.2 Isolation

4.1.2.1 Scope identification

Scope leaking

4.1.2.2 Execution and variable propagation

4.1.3 distribution

4.2 Fluxionnal execution model

Everything here is already written in the first paper : flx-paper. It only needs to be rewritten

4.2.1 Fluxion encapsulation

4.2.1.1 Execution

4.2.1.2 Name

4.2.1.3 Memory

4.2.2 Messaging system

Chapter 5

Evaluation

5.1 Due compiler

5.2 Fluxionnal compiler

5.3 Fluxionnal execution model

Chapter 6

Conclusion

Appendix A

Language popularity

A.1 PopularitY of Programming Languages (PYPL)

[1] The PYPL index uses Google trends[2] as a leading indicator of the popularity of a programming language. It search for the trend for each programming language by counting the number of searches of this language and the word "tutorial".

PYPL for May 2015

[1] http://pypl.github.io/PYPL.html
[2] https://www.google.com/trends/

Rank	Change	Language	Share	Trend
1		Java	24.1%	-0.9%
2		PHP	11.4%	-1.6%
3		Python	10.9%	+1.3%
4		C#	8.9%	-0.7%
5		C++	8.0%	-0.2%
6		C	7.6%	+0.2%
7		Javascript	7.1%	-0.6%
8		Objective-C	5.7%	-0.2%
9		Matlab	3.1%	+0.1%
10	2× ↑	R	2.8%	+0.7%
11	5× ↑	Swift	2.6%	+2.9%
12	1× ↓	Ruby	2.5%	+0.0%
13	3× ↓	Visual Basic	2.2%	-0.6%
14	1× ↓	VBA	1.5%	-0.1%
15	1× ↓	Perl	1.2%	-0.3%
16	1× ↓	lua	0.5%	-0.1%

A.2 TIOBE

[3]

The TIOBE index uses many search engines as an indicator of the current popularity of programming languages. It count the number of pages each search engine find when queried with the language name and the word "programming". This indicator indicates the number of resources available, and the discussions about a given programming language.

Javascript was the most rising language of 2014 in the TIOBE index.

TIOBE for April 2015

[3] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Apr 2015	Apr 2014	Change	Programming Language	Ratings	Change
1	2	↑	Java	16.041%	-1.31%
2	1	↓	C	15.745%	-1.89%
3	4	↑	C++	6.962%	+0.83%
4	3	↓	Objective-C	5.890%	-6.99%
5	5		C#	4.947%	+0.13%
6	9	↑	JavaScript	3.297%	+1.55%
7	7		PHP	3.009%	+0.24%
8	8		Python	2.690%	+0.70%
9	-	2× ↑	Visual Basic	2.199%	+2.20%

A.3 Programming Language Popularity Chart

[4]

The programming language popularity chart indicates the activity of a given language in the online communities. It uses two indicators to rank languages : the number of line changed in github of, and the number of questions tagged with a certain language.

Javascript is ranked number one in this index. The Javascript community is particularly active online, and in the open source.

indeed.com

A.4 Black Duck Knowledge

[5]

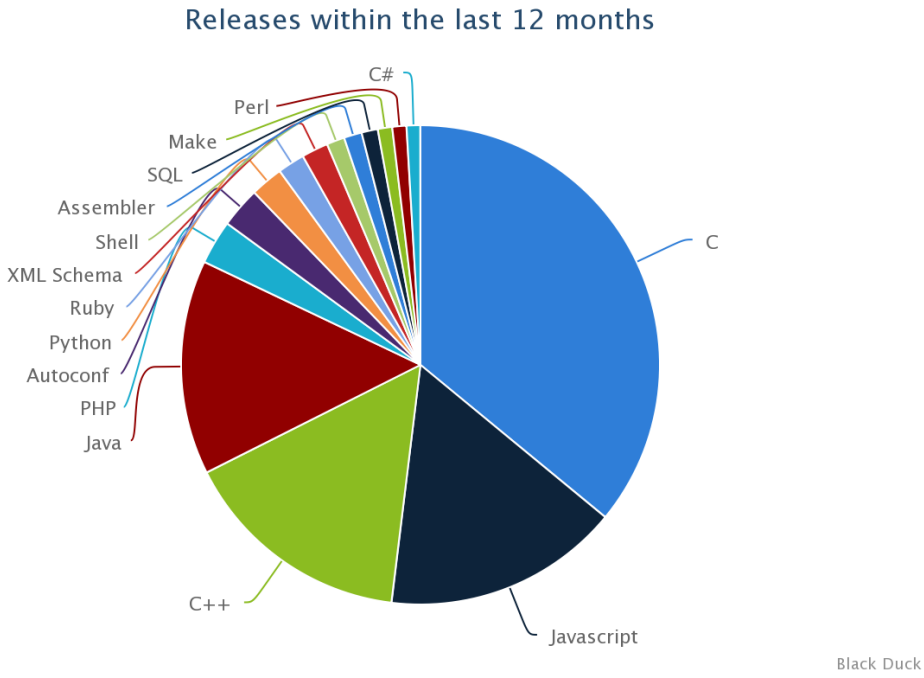
The black-duck, which analyze the usage of language on many forges, and collaborative hosts, rank Javascript number 2, after C, and with about the same usage as C++.

github.com sourceforge.net cpan.org rubyforge7.org planetsourcecode.com ddj.com

[4] http://langpop.corger.nl

[5] https://www.blackducksoftware.com/resources/data/this-years-language-use

Language	%
C	34.80
Javascript	15.45
C++	15.13
Java	14.02
PHP	2.87
Autoconf	2.65
Python	2.15
Ruby	1.77
XML Schema	1.73
Shell	1.18
Assembler	1.16
SQL	1.07
Make	0.94
Perl	0.92
C#	0.90



A.5 Github

http://githut.info/

A.6 HackerNews Poll

https://news.ycombinator.com/item?id=3746692

Language	Count
Python	3335
Ruby	1852
JavaScript	1530
C	1064
C#	907
PHP	719
Java	603
C++	587
Haskell	575
Clojure	480
CoffeeScript	381
Lisp	348
Objective C	341
Perl	341
Scala	255
Scheme	202
Other	195
Erlang	171
Lua	150
Smalltalk	130
Assembly	116
SQL	112
Actionscript	109
OCaml	88
Groovy	83
D	79
Shell	76
ColdFusion	51
Visual Basic	47
Delphi	45
Forth	41
Tcl	34
Ada	29
Pascal	28
Fortran	26
Rexx	13
Cobol	12