Automatic pipeline distribution for monolithic web applications : Toward a better compromise between development scalability and performance scalability not definitive

Etienne Brodu

May 18, 2015

Abstract

TODO

Contents

Chapter 1

Introduction

Chapter 2

Context and Objectives

2.1 Javascript

2.1.1 Explosion of Javascript popularity

2.1.1.1 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 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.

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

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

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.

2.1.1.2 Rising of the unpopular language

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.

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 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.

[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/`

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

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]

2.1.1.3 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 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.

[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/`

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.

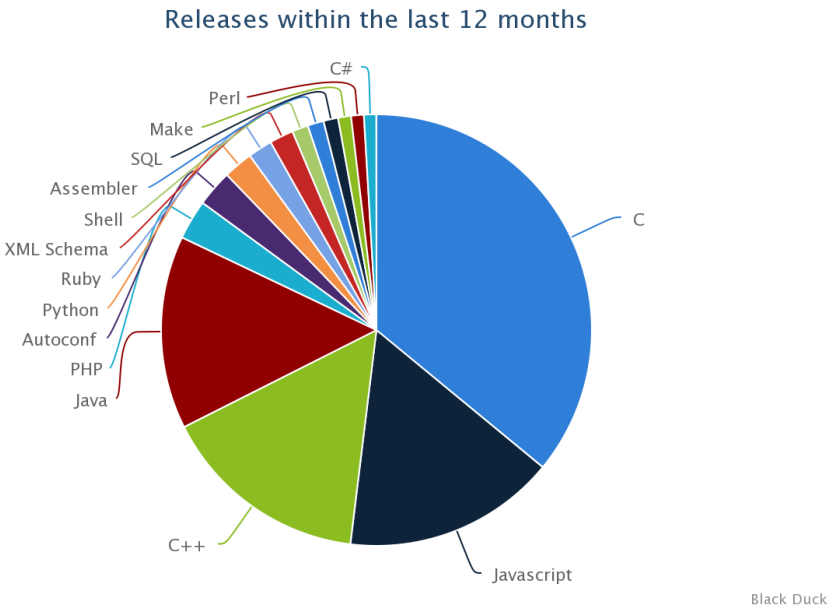
From these indexes, the major programming languages are Java, C/C++ and C#. The three languages are still the most widely taught, and used to write softwares. But Javascript is rising to become one of these important languages.

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.

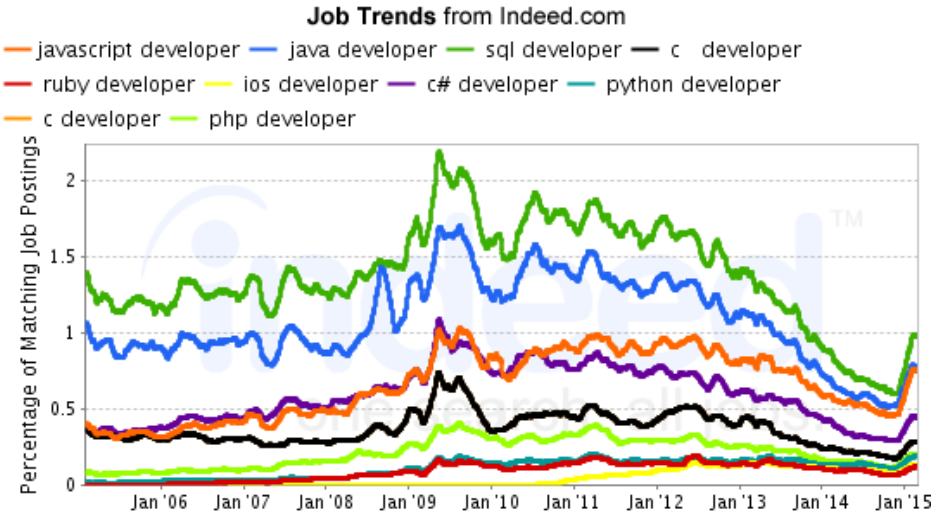
TODO : graph of Github repositories by languages

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.



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.



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/

2.1.2 Overview of the language

Javascript was released in a hurry, without a strong and directive philosophy. During its evolution, it snowballed with different features to accommodate the community, and the usage it was made on the web. As a result Javascript contains various, and sometimes conflicting, programming paradigms. It borrow its syntax from a procedural language, like C, and the object notation from an object-oriented language, like Java, but it provides a different inheritance mechanism, based on prototypes. Most of the implementation adopt an event-based paradigm, like the DOM[12] and node.js[13]. And finally, event though it is not purely functional like Haskel, Javascript borrows some concepts from functional programming.

In this section, we focus on the last two programming paradigm, functional programming and event-based programming. Javascript exposes two features from functional programming that are particularly adapted for event-based programming. Namely, it treats functions as first-class citizen, and allows them to close on their defining context, to become closures.

2.1.2.1 Functions as First-Class citizens

"All problems in computer science can be solved by another level of indirection"

—Butler Lampson

Javascript treats function as first-class citizens. One can manipulate functions like any other type (number, string ...). She can store functions in variables or object properties, pass functions as arguments to other functions, and write functions that return functions.

The most common usage examples of these features, are the methods `Map`, `Reduce` and `filter`. In the example below, the method `map` expect a function to apply on all the element of an array to modify its content, and

[12] `http://www.w3.org/DOM/`
[13] `https://nodejs.org/`

output a modified array. A function expecting a function as a parameter is considered to be a higher-order function. `Map`, `Reduce` and `Filter` are higher-order functions.

```
[4, 8, 15, 16, 23, 42].map(function firstClassFunction(element) {
  return element + 1;
});
// -> [5, 9, 16, 16, 24, 43]
```

Higher-order functions provide a new level of indirection, allowing abstractions over functions. To understand this new level of abstraction, let's briefly summarize the different abstractions on the execution flow offered by programming paradigms. In imperative programming, the control structures allow to modify the control flow. That is, for example, to execute different instructions depending on the state of the program. Procedural programming introduces procedures, or functions. That is the possibility to group instructions together to form functions. They can be applied in different contexts, thus allowing a new abstraction over the execution flow.

So, higher-order functions add another level of abstraction. It allows to dynamically modify the control of the execution flow. The ability to manipulate functions like any other value allows to abstract over functions, and behavior.

Higher-order functions replace the needs for some Object oriented programming design patterns.[14] Though object oriented programming doesn't exclude higher-order functions.

They are particularly interesting when the behavior of the program implies to react to inputs provided during the runtime, as we will see later. Web servers, or graphical user interfaces, for examples, interact with external events of various types.

2.1.2.2 Lexical Scoping

Closures are indissociable from the concept of lexical environment. To understand the former, it is important to understand the latter first.

Lexical environment A variable is the very first level of indirection provided by programming languages and mathematics. It is is a binding between a name and a value. Mutable like in imperative programming to represent the reality of memory cells, or immutable like in mathematics and functional

[14] `http://stackoverflow.com/a/5797892/933670`

programming. These bindings are created and modified during the execution. They form a context in which the execution takes place. To compartmentalize the execution, a context is also compartmentalized. A certain context can be accessed only by a precise portion of code. Most languages defines the scope of this context using code blocks as boundaries. That is known as lexical scoping, or static scoping. The variables declared inside a block represent the lexical environment of this block. These lexical environments are organized following the textual hierarchy of code blocks. The context available from a certain block of code, that is set of accessible variable, is formed as a cascade of the current lexical environment and all the parent lexical environment, up to the global lexical environment.

Javascript lexical environment [15]

Javascript implement lexical scoping with function definitions as boundaries, instead of code blocks. The code below show a simple example of lexical scoping in Javascript.

```
1  var a = 4;
2  var c = 6;
3  function f() {
4    var b = 5;
5    var c = 0;
6    // a and b are accessible here.
7    return a + b + c;
8  }
9
10 f(); // -> 9
11
12 // b is not accessible here :
13 a + b + c; // -> ReferenceError: b is not defined
```

Lexical scoping, or statical scoping, implies that the lexical environment are known statically, at compile time for example. But Javascript is a dynamic language, it doesn't truly provide lexical scoping. In Javascript, the lexical environments can be dynamically modified using two statements : `with` and `eval`. We explain in details the Javascript lexical scope in section **??**

2.1.2.3 Closure

"An object is data with functions. A closure is a function with data."

[15] `http://www.ecma-international.org/ecma-262/5.1/#sec-10.2`

—John D. Cook

A closure is the association of a first-class function with its context. When a function is passed as an argument to an higher-order function, she closes over its context to become a closure. When a closure is called, it still has access to the context in which it was defined. The code below show a simple example of a closure in Javascript. The function `g` is defined inside the scope of `f`, so it has access to the variable `b`. When `f` return `g` to be assigned in `h`, it becomes a closure. The variable `h` holds a closure referencing the function `g`, as well as its context, containing the variable `b`. The closure `h` has access to the variable `b` even outside the scope of the function `f`.

```
function f() {
  var b = 4;
  return function g(a) {
    return a + b;
  }
}

var h = f();
// b is not accessible here :
b; // -> ReferenceError: b is not defined

// h is the function g with a closure over b :
h(5) // -> 9
```

2.2 Concurrency

Javascript, like most programming languages, is synchronous and non-concurrent. The specification doesn't provide any mechanism to write asynchronous nor concurrent execution of Javascript. There is no reference to `setTimeout` nor `setInterval`, two well-known instructions to asynchronously post-pone execution. Indeed, like for many languages, concurrency is supported and provided by the host environment. For example, the DOM for Javascript, the JVM for Java, or the operating system for C/C++. These three examples provide different models for concurrency. We explore in this section, the two different models and their evolutions.

2.2.1 Two known concurrency model

We distinguish two types of concurrent models, tailored for two types of applications, CPU bound applications, and I/O bound applications. An ex-

ample of CPU bound application is a scientific application, like a physics simulation. Such applications rely heavily on the CPU to do demanding calculations. CPU bound applications need concurrency to increase the computing power and deliver a result faster. On the other hand, an example of I/O bound application is graphical user interface. It needs to display and react to multiple concurrent streams of interaction. Another example is a web server, it needs to react to multiple concurrent requests. I/O bound applications need concurrency to keep track of concurrent control flows in multiple contexts. The difference between these two needs is thin enough for one to be mistaken with the other. The two well-known models for concurrency are threads and events. It is now broadly admitted that threads are the better solution for CPU bound applications, while events are the better solution for I/O bound applications. However, this distinction is very blurry, because of a lack of precise definitions for both models. In the next section I explain what characterizes these two models and what are the differences between the two.

when there is the need for concurrency in a system (so the components are somewhat dependent on each others : causal relation, or state sharing), there is a need for coordination. If there is no need for coordination, then there is two independent systems. So concurrency is first a topology and coordination (communication?) problem. Synchronization is a type of coordination, asynchronous messaging is another type of coordination.

2.2.1.1 Thread

A thread is the lightest representation of a sequence of instructions executed on a computing core. Multiple threads can be executed concurrently. Threads are not to be mistaken with processes. A process always contain at least one thread, sometimes more. Multiple threads share the same memory, while multiple processes don't. The thread is a very broad concept, it generalize kernel threads, green threads or user threads and fibers[**Adya2002**].

A kernel thread is a set of registers on a computing core representing the state of a computation[16]. One of these registers being the instruction pointers, which is manipulated by the control flow, a thread indeed represent the execution of a sequence of instructions on this computing core. It is the

[16] `http://stackoverflow.com/a/5201879/933670`

smallest execution unit manipulable by the kernel scheduler. C/C++ uses kernel threads provided by the operating system.

A green thread, or user thread, is the implementation of a thread in user level, instead of kernel level. Java uses green threads provided by the JVM.

Threads, and particularly kernel threads are often preemptively scheduled. A fiber, on the other hand, is a thread non preemptively scheduled but cooperatively scheduled. I address in the next section exactly what is a preemptive or cooperative scheduler.

2.2.1.2 Event

An event is a significant change in the [application] state[**Chandy2006**]. In an event-based system, the concurrent executions are independent, they don't share any memory. To coordinate the execution, they exchange messages representing the occurrence of events impacting the execution at a larger scale than the local concurrent execution.

The actors model is an example of an event-based concurrent model for Artificial Intelligence purposes [**Hewitt1973a**].

2.2.1.3 Thread and Event are orthogonal concepts

Threads doesn't indicate anything about synchronization, but focus solely on execution. Events doesn't indicate anything about execution, but focus solely on synchronization. They are orthogonal concepts. It is completely pointless to compare them.

There is two differences between threads and events. The memory sharing, and the programming style. Threads share memory, while events don't, so threads needs synchronization to coordinate. Threads use synchronous programming, while events use asynchronous programming.

Events with shared memory is an event-loop : there is no parallelism.

Lauer and Needham [**Lauer1979**] presented an equivalence between Procedure-oriented Systems and Message-oriented Systems.

Adya *et. al.* analyzed this debate and presented fives categories through which to present the problem [**Adya2002**]. Their advantages and drawbacks were mistaken with those of thread and events. Adya *et. al.* explain in details two of these categories that are most representative, Task management and Stack management. These two categories were often associated with thread-based systems and event-based systems. We paraphrase these explanations.

Multi-process programming use Inter Process Communications (IPC). IPC regroups both shared memory(?) coordination, and message passing(?) coordination. Semaphore, shared memory, and files are shared-memory coordination mechanisms, while pipe, streams and message queues, are message passing coordination mechanisms.

Automatic stack management is as bad as preemptive scheduling : you don't know when the preemption happens. It is in reality impracticable to use cooperative task management with automatic stack management. Ayda said it : with closure, the problem of stack management disappear. It remains only the difference between sync / async programming : the control flow is broken in multiple handlers (async) or appear to be continuous (sync). It is linked with the task management. Cooperative task management provides a known invariant. While preemptive task management hides the invariant. Actually, it is neither task management, nor stack management, it is knowing yielding points, to know the invariant.

2.2.2 Differentiating characteristics

2.2.2.1 Scheduling

2.2.2.2 Coordination strategy

2.2.3 Turn-based programming

When an event-loop is used as the concurrency model for a partially functional language like Javascript, it creates what Douglas Crockford called turn-based programming[17].

2.2.3.1 Event-loop

2.2.3.2 Promises

2.2.3.3 Generators

Generators might be used to implement synchronous programming on top of an event-loop (asynchronous programming), a bit like fibers Generators use the yield keyword, which is good stuff. The long term goal would be to have asynchronous yield as rupture points.

[17] `https://youtu.be/dkZFtimgAcM?t=1852`

However, one very important thing, is that yield block the execution. And if it is a good thing for programmers to know when the execution is leaving the atomicity (yield vs fibers), I want to explore how it might reduce the concurrency expressiveness. yield impose a linear programming style, while promises keep the tree programming style.

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

"One early vision was that the right computer language would make parallel programming straightforward. There have been hundreds—if not thousands—of attempts at developing such languages ... Some made parallel programming easier, but none has made it as fast, efficient, and flexible as traditional sequential programming. Nor has any become as popular as the languages invented primarily for sequential programming."

—David Patterson

2.2.4.1 TODO

2.3 Scalability

2.3.1 Theories

2.3.1.1 Linear Scalability

2.3.1.2 Limited Scalability

2.3.1.3 Negative Scalability

Conclusion : scalability = concurrency + not sharing the resources that grows with the scale

2.3.2 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

2.4 Objectives

2.4.1 Problem : the pivot

2.4.1.1 Development & Performance Scalability

Where I explain the two (It should be clear from the two previous section), and why it is difficult to switch from one to the other.

2.4.1.2 A difficult compromise

It is impossible to truely merge development and true linear performance scalability : linear performance scalability means perfect parallelism, that is impossible for multiple reasons. It is impossible to reduce the time of an operation infinitely : an instant computation doesn't exist. It is impossible to decompose a set of operation past a certain point : the coupling (sharing / causality) is needed for computation : this composition is the fabric of computation.

2.4.2 Proposal and Hypothesis

2.4.2.1 LiquidIT

TODO liquid IT is roughly about improving the compromise between development and performance scalability (it is more than that, and I need to explain it briefly, but I need to narrow the field to focus on my thesis)

2.4.2.2 Pipeline parallelism for event-loop

The hypothesis is that it is possible for event-looped web applications to be pipeline parallelized, and so to be distributed

2.4.2.3 Parallelization and distribution of web applications

the proposal is to study and develop a solution to parallelize and distribute web applications.

Chapter 3

State of the art

3.1 Framworks for web application distribution

3.1.1 Micro-batch processing

3.1.2 Stream Processing

3.2 Flow programming

3.2.1 Functional reactive programming

3.2.2 Flow-Based programming

3.3 Parallelizing compilers

OpenMP and so on

3.4 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]

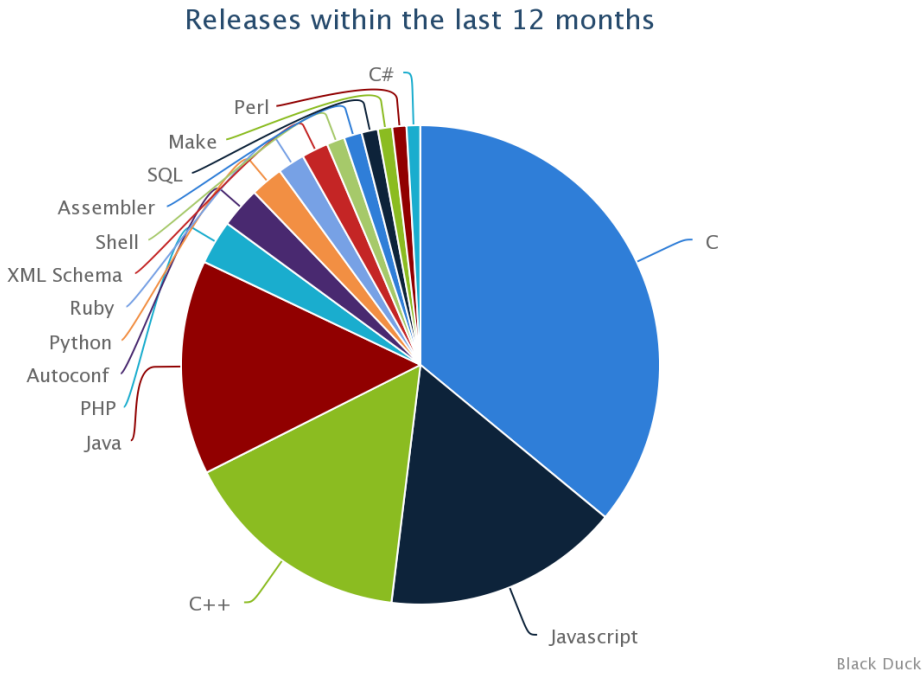
The black-duck, which analyze the usage of language on many forges, and collaborative hosts, rank Javascript number 2, after C, and with about the same usage as C++.

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
Smalltalk	130
Other	195
Erlang	171
Lua	150
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