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

Etienne Brodu

May 27, 2015

Abstract

TODO 1p, translate from below

Résumé

TODO reduce to 1p Le web permet à chacun de mettre un service à disposition d'une audience sans précédent. 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 une telle croissance sur le web. Google, Facebook ou Twitter en sont les exemples les plus flagrants. Au début du développement d'une telle application, il est important d'aller le plus vite possible afin de vérifier rapidement si le service peut rencontrer une audience. Le développement est guidé principalement par les fonctionnalités à fournir. C'est pourquoi ces projets commencent avec des langages faciles à développer, tel que Ruby ou Java qui ont pris une ampleur importante ces dernières années en se positionnant comme les langages du web.

Si cette application répond correctement à un besoin, 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. Dans la plupart des cas, cela implique de la réécrire complètement. Le plus souvent en passant par des infrastructures scalables qui imposent des modèles de programmation et des API spécifiques. Cela représente une charge de travail conséquente et incertaine. Une fois le changement fait, l'équipe de développement doit concilier deux objectifs : l'augmentation en fonctionnalité et l'optimisation de l'infrastructure scalable. Pour toutes ces raisons, ce changement est un risque pour l'évolution de l'application. Tout cela dans un cadre économique qui est souvent instable, comme c'est le cas dans la plupart des start-up. Mais ce risque est tout autant présent dans de plus grandes structures.

Ma thèse se base sur l'observation suivante. D'une part, Javascript est un langage qui a énormément gagné en popularité ces dernières années. Il était déjà omniprésent sur les clients, et avec Node.js il s'impose également de

plus en plus sur les serveurs. Il représente une communauté de developpeur importante. Il se place maintenant comme le langage principal du web, détrônant ruby ou Java. D'autre part, Javascript a la particularité de ressembler à un pipeline, car il est exécuté sur une boucle événementielle. L'execution d'un programme Javascript réagis à un évenement en invoquant une fonction. Cette fonction s'exécute, et envoie un évenement à une autre fonction pour qu'elle s'exécute à la suite etc .., On observe ici le même flux 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'execution reparti. Je construis un compilateur permettant d'identifier les fonctions de Javascript et de les isoler dans ce que j'appelle des Fluxions. Quelque chose qui serait à la fois une fonction et un flux. Un conteneur qui peut exécuter du code à la réception d'un message, et envoyer des messages pour continuer le flux vers d'autres fluxions. L'intérêt est que les fluxions sont complètement indépendantes, et qu'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 & Objectives

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

2.1.2 The languages of the web

Javascript(1995), Ruby(1993), PHP(1995) and Java(1994) were all created around the same time, in 1995. 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.

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.

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

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

(I already have a few pages written on this) The history of Javascript, from 1995 to now. With numbers about the popularity of Javascript on Github, StackOverflow and different indexes.

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

But many large company replaced the initial languages of the web with scalable solutions. Twitter : Ruby -> Storm Facebook : PHP -> HHVM / Flux ... Google : MapReduce, Kubernetes, Borg ...

2.2.1 Development & Performance Scalability

I explain exactly what I mean by Development scalability (I should change the term, it is misleading and not explicit enough), and Performance scalability.

2.2.2 A difficult compromise

I explain why the two are difficult to merge together. It is difficult to design a language for parallel programming which is simple enough, and scalable enough. Either it is scalable, or it has wide adoption (because it is simple enough)

2.3 Proposal and Hypothesis

2.3.1 Hypothesis

The hypothesis : It is hard for developers to assure the invariant on the memory. Most use synchronization, and fail, a very few use isolation (parallel languages), and between the two, there is the event-loop : turn-based programming which is an alternative to synchronization and isolation.

2.3.2 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.

2.3.3 Parallelization and distribution of web applications

The thesis : with the help from a compiler, it is possible to automatically transform this class of application : flow applications, into independent components (like difficult parallel languages)

The article your server as a function is similar to storm (except in C, apparently). It is a very good solution for a specific class of applications : stateless applications. For stateful applications, it relies on a database, like Storm. But a database is too much : if a stateless application is the best case for scalability, a database is the worst case. Some applications are in the middle, they don't need a database, but they are not stateless : flow application -> state without retro-propagation

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]

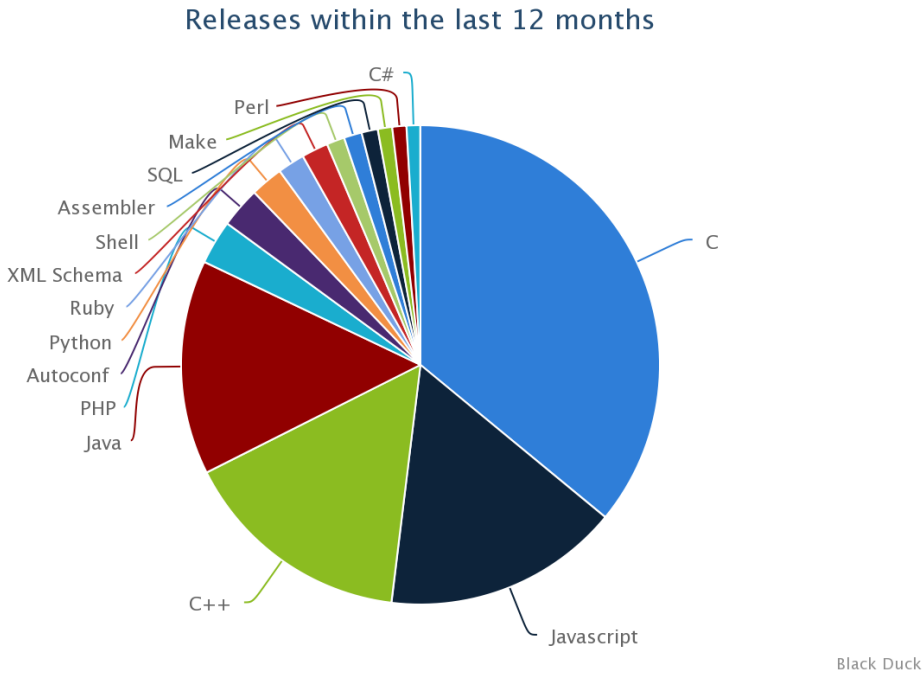
The black-duck, which analyze the usage of language on many forges, and collaborative hosts, rank Javascript number 2, after C, and with about the same usage as C++.

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