CS 188/219

Scalable Internet Services

Andrew Mutz

May 27, 2015

Final Steps

Final presentations are 9 days from now!

You should be focusing on load testing, performance optimizations, write-up and presentation.

- Remember to use significant size data sets!

Now is the time that I expect the most AWS usage

- I will be aggressive about terminating stacks
- **You should be aggressive about scping data off.**

Final Steps

Each presentation should be roughly 15 minutes and include all group members.

- Project Overview
  - Demo of your working application, Application architecture
- Experiments and Results
  - Critical user paths used for load testing
  - Scaling performance results
  - Optimizations you performed and performance results
- Conclusions and lessons learned
  - Team organization, pair programming, Test Driven Development
  - Building a scalable web service, Any other of interest

Final Steps

Each presentation should be roughly 15 minutes and include all group members.

- We will want a live demo of your application running on EC2, but be prepared to fall back to slides if there is a network problem.
- Attend all student presentations during your lab group (if you are the 2pm lab, stay for the whole 2pm group, if you are in the 4pm lab, stay for the whole 4pm group), learn from others, **and ask questions**.

Final Steps

The final course write-up is where you record the various scaling improvements and optimizations that you have made and the performance improvements that have resulted.

- A brief description of your project
- A brief description of the critical user paths and data set size used to evaluate performance
- For each of your performance & scalability improvements
  - A description of what was changed and improved
  - A description of why this improves your application
  - A quantitative demonstration of the effects: graphs & numbers

Introduction

The last lecture focused on the increasing dominance of javascript.

This lecture looks past that a bit.

For Today

A post-javascript client

- Asm.js
- Emscripten
- MRuby, Pypyjs

Introduction

http://coolwanglu.github.io/vim.js/experimental/vim.html

http://danielsadventure.info/html5fractal

http://crypt-webgl.unigine.com/game.html

Introduction

Web-based applications have taken over many domains

- Email, calendaring, word processing, social, etc.
- HTML5 has Geo-location, WebRTC, etc.

What can’t web browsers do?

Introduction

Web browsers struggle with compute heavy tasks...

- Games
- Speech recognition
- Image recognition
- Image processing, effects

Lots of work has been put into making JS fast, but its still a scripting language.

Introduction

Competitive pressure on browser vendors.
More and more is desired from browsers.

Javascript engine performance has been improving

- Dynamic compilation
- Advanced garbage collection
- Other VM optimizations

Introduction

There is no other language on the client.

- `<script type=”text/c”>` doesn’t exist
  - Why doesn’t this exist?
- This means we are limited in the compute performance on the client.
- This also means we don’t have language heterogeneity on the client
  - Why is this bad?

Introduction

Key observation:

- Javascript engines are limited in the performance they can deliver because of the rich feature set of Javascript.
  - Garbage collection, Dynamic dispatch, etc.
- If Javascript didn't have some of these features, we could build really high performance VMs
- **More usefully, if restrict ourselves to the fast subset of features, VM designers can make that go very fast.**

Asm.js

Example:

```
f = function(a, b){
  return a + b;
}
f(5, 2)
=> 7

f(“foo”, “bar”)
=> “foobar”
```

Asm.js

Example:

```
f = function(a, b){
  return a + b;
}
f(5, 2)
=> 7

f(“foo”, “bar”)
=> “foobar”
```

How is `f()` compiled to machine code?

Which “+” should be used?

Depends on the args passed to `f()` at runtime

Asm.js

Example:

```
f = function(a, b){
  a = a|0;
  b = b|0;
  return a + b;
}
f(5, 2)
=> 7
```

The bitwise `|0` doesn't affect the value, but guarantees the result is an integer

We've removed the need to check types to implement "+"

Asm.js

Another example:

```
result = ""
for(i=10; i>0; i--){
  result += "a"
}
```

In this code, the variable “result” is getting repeatedly reassigned.

The garbage collector here is cleaning up after each.

Asm.js

Another example:

```
result = ""
for(i=10; i>0; i--){
  result += "a"
}
```

If strings are immutable in Javascript, how do we avoid this?

Asm.js

Another example:

```
buff = Int32Array(10);
for(i=10; i>0; i--){
  buff[i] = "a".charCodeAt(0)
}
result = String.fromCharCode.
apply(
  String, buff);
```

Avoid the garbage collector completely.

Allocate a reusable array of integers and manage your memory by hand.

Asm.js

A subset of Javascript that VMs can execute very fast.

All valid JS. Any interpreter can execute it, but some will optimize.

Developed at Mozilla in 2013

Asm.js

Only number types are used.

- No strings, booleans, or objects
- Higher level concepts can be built on numbers (like traditional assembly language)

Asm.js

No VM-provided GC

- All non-stack data is stored in one large array, called the heap.
- It is up to the application code to manage this heap. (like traditional assembly language)

Asm.js

No dynamic dispatch

- By only using numbers, many dispatch issues are avoided completely
- Coercion is applied to ensure compiler can be sure about types:

```
var x = a | 0
var y = + b
```

Asm.js

Asm.js is organized into modules. Modules are a series of function definitions with references to the JS stdlib, a foreign function interface, and a heap array for storage

```
function MyAsmModule(stdlib, foreign, heap) {
    "use asm";  // marks this function as an asm.js module
    // module body:
    function f1(...) { ... }
    function f2(...) { ... }
    ...
    return {
        export1: f1,
        export2: f2,
        ...
    };
}
```

Asm.js

The string “use asm”; is a hint to the VM that the code should be asm-compliant.

```
function DiagModule(stdlib) {
    "use asm";
    var sqrt = stdlib.Math.sqrt;
    function square(x) {
        x = +x;
        return +(x*x);
    }
    function diag(x, y) {
        x = +x;  y = +y;
        return +sqrt(square(x) + square(y));
    }
    return { diag: diag };}
```

Asm.js

The string “use asm”; is a hint to the VM that the code should be asm-compliant.

If FFI or heap is not passed in, defaults are provided.

```
// Browsers: this === window
var fast = DiagModule(this);

console.log(fast.diag(3, 4));  // 5
```

Asm.js

So, by restricting our use of Javascript to static typing and manual memory management, we can generate JS that can be executed very quickly.

Who would ever want to write code like this?

LLVM

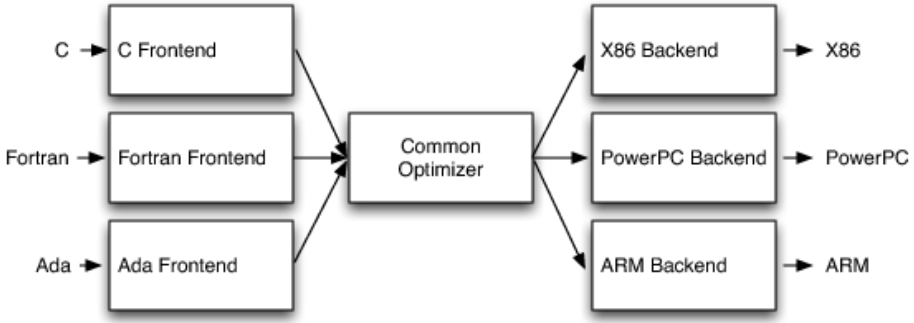
LLVM: Low Level Virtual Machine project

- Language-agnostic low level bytecode that can be translated to machine code.
- Many languages support generating LLVM: C, C++, Python, Ruby, Rust, Scala, etc.

LLVM

Basic idea:

- Have many front-end libraries able to generate LLVM bytecode
- LLVM optimizations are language/architecture agnostic
- Many back-ends to generate bytecode for each architecture.



LLVM

Sample:

```
#include<stdio.h>

int main() {
  printf("hello, world!\n");
  return 0;
}
```

LLVM

Sample:

```
@.str = private unnamed_addr constant [15 x i8] c"hello, world!\0A\00", align 1
; Function Attrs: nounwind ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  store i32 0, i32* %1
  %2 = call i32 (i8*, ...)* @printf(
    i8* getelementptr inbounds ([15 x i8]* @.str, i32 0, i32 0))
  ret i32 0
}
```

Asm.js

So we have many languages compiling to LLVM, and many backends that translate that to machine instructions.

What do we get if we write a backend that translates to Javascript?

Emscripten

Emscripten translates LLVM bytecode into Asm.js

Emscripten

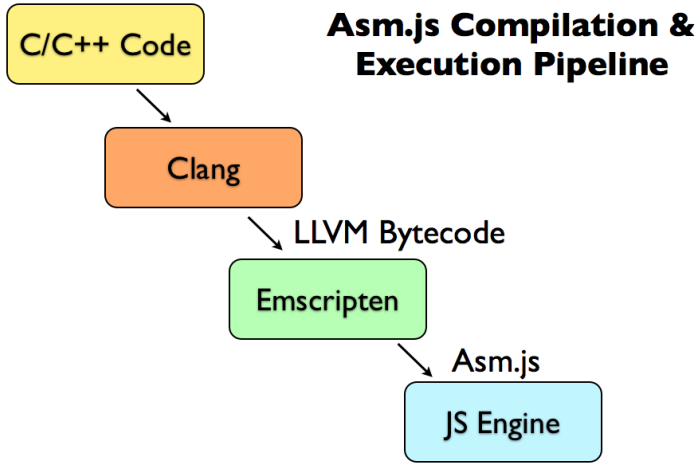
Example:

```
//LLVMIR
define i32 @func(i32* %p){
  %r= load i32* %p
  %s= shl i32 %r, 16
  %t= call i32 @calc(i32 %r, i32 %s)
  ret i32 %t
}
```

```
// JS
function func(p){
  var r = HEAP[p];
  return calc(r, r << 16);
}
```

Emscripten

Because we have many languages that compile to LLVM, we can now use Emscripten to execute these languages in browsers



Emscripten

What happens to application performance?

- What % performance decrease do you think?

Emscripten

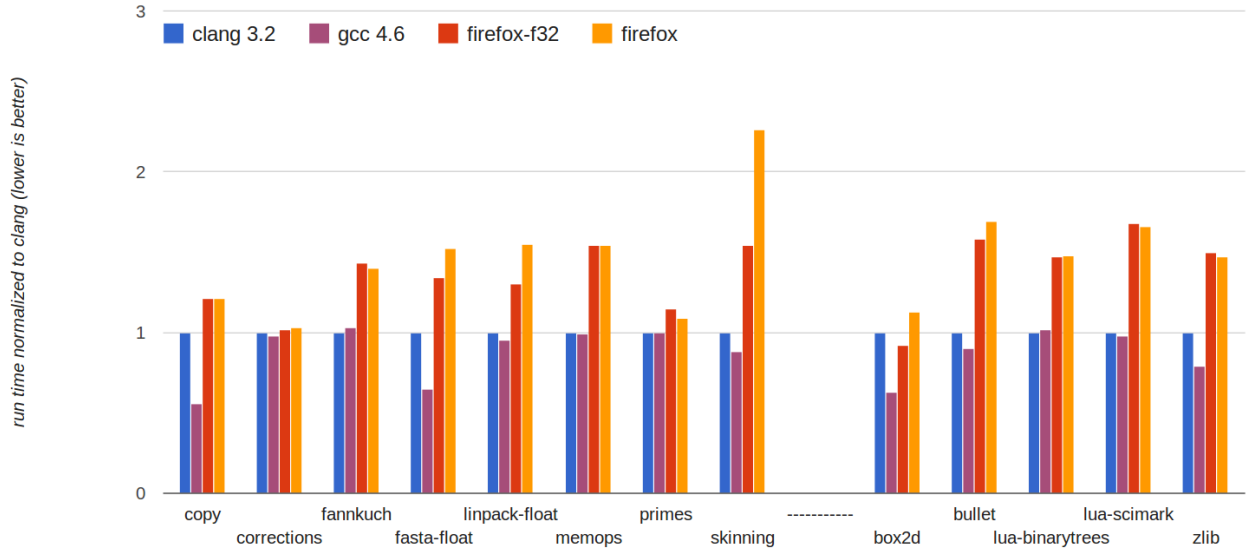
What happens to application performance?

- What % performance decrease do you think?

Project authors suggest 2X slowdown over native code!

Emscripten

What happens to application performance?



MRuby

Demo!

http://joshnuss.github.io/mruby-web-irb/

http://pypyjs.org/

MRuby

Why don’t we see client-side frameworks being built in these languages?

MRuby

Why don’t we see client-side frameworks being built in these languages?

- jquery: 96KB
- angular: 441KB
- MRuby: 1.7MB
- PyPyJS: 2.7MB

Conclusion

What does this mean in the long term?

There is a path towards language heterogeneity on the client.

Javascript won't necessarily take over all of web application development.

For Next Time...

Monday we have a guest speaker, Jon Walker, Founder and CTO of Appfolio.

Friday is the last lab session before our final presentations.

- In addition, I will be available via email and piazza for questions (piazza preferred)