Computing in Browsers?

Reggie Cushing
University of Amsterdam

14th June 2013

Objectives - distributed computing using web browsers.

Motivation - The proliferation of web browsers and the performance gain being achieved by current JavaScript virtual machines raises the question whether Internet browsers can become yet another middleware for distributed computing.

Web browser is ubiquitous - Every smart device, being a computer or a mobile device, is, nowadays, equipped with an Internet browser. At the heart of every Internet browser is a JavaScript engine.

Web Browser shapes our daily lives - by keeping us in contact with friends and collaborate with colleagues through social media such as Twitter and Facebook which have become a main stay in the way humans interact.
The social aspect of the current Web is enough to achieve **volunteer computing almost instantly**.

**How does it work:** Social media mediates the trust between the user and the volunteers asked to join the network.

- A user with a distributed application uses social media to get colleagues and friends to donate CPU.
- Colleagues and friends join the network by simply opening the shared URL.
- Computing can start instantly.
JavaScript is not the bottleneck anymore (quite optimistic statement) -- The interactive nature of websites put a demand for faster JavaScript engines which lead to a JavaScript engine arms race between the main contenders namely; Google, Mozilla, Apple, and Microsoft.

Web Technologies achievements make JavaScript engines more powerful:
Web workers: threads of JavaScript communicating over message passing
web sockets: bi-directional communication channels
WebGL: JavaScript API for rendering 3D graphics.
WebCL: standard for Javascript binding to OpenCL
JavaScript

- http://bellard.org/jslinux/
- http://slides.html5rocks.com/#web-workers
Ratios of the execution times of known algorithms compiled and run with 4 versions of Chrome’s V8 JavaScript engine to the respective GNU C execution times.
Notable performance gains in different versions of Google's V8 JavaScript Engine:

- Highly optimized Regex engine (Irregexp)
  http://blog.chromium.org/2009/02/irregexp-google-chromes-new-regexp.html

- Crankshaft optimization: ArrayBuffer and Float64Array contribute to SpectralNorm perf gain
WebCL

- WebCL is an API for OpenCL to compute directly on the GPU from within the browser.
- Follows WebGL for rendering on the GPU.
- Currently not part of browsers but work is being done: https://www.khronos.org/bugzilla/show_bug.cgi?id=792
Applicability to Science

- Many scientific applications are highly parallelizable.
- Data can be decomposed into *atomic* records.
- The data partitioning defines the concurrency in the application.
- Grids/Clusters/Clouds high performance but very tedious.
- Nothing stops Browsers adding resources to traditional resources.
- Applications that fit a browser scenario are called malleable applications:
  
  *Given a dataset a malleable task can compute on any subset of the dataset*

- Malleable applications:
  - Monte Carlo Simulations
  - Parametric studies
  - Indexing
The server side is composed of a REST service which creates the list of jobs to be executed while a website handles user interactivity.

Jobs are stored in a *runqueue* on a database. Server generates multiple jobs for multiple input parameters (*cross-product*).

Web browsers that load the website pull packaged jobs and send back job outputs to the REST service which in turn stores the results.
Weevil Jobs - 1

• Jobs are written in JavaScript (WebCL for future browsers)

• A job is encapsulated in a JavaScript function called `weevil_main()` (like C `main()`). The browser will invoke `weevil_main()` with the parameters.

• Invocation in a browser is done through message passing since WebWorkers are sandboxed (for security).
Listing 1: A simple JavaScript matrix multiplication function.

```javascript
function weevil_main(A, B) {
  var mA = JSON.parse(A);
  var mB = JSON.parse(B);
  var result = [];
  for (i in mA) {
    result[i] = [];
    for (var j in mB[0]) {
      var sum = 0;
      for (var k in mA[0]) {
        sum += mA[i][k] * mB[k][j];
      }
      result[i][j] = sum;
    }
  }
  return (JSON.stringify(result));
}
```
self.addEventListener('message', function(e) {
    var data = e.data;
    switch (data.cmd) {
        case 'start':
            weevil_main();
            break;
        case 'stop':
            self.close();
            break;
    }

    function weevil_main() {
        var A = e.data.A;
        var B = e.data.B;
        var mA = JSON.parse(A);
        var mb = JSON.parse(B);
        var result = [];
        for (i in mA) {
            result[i] = [];
            for (var j in mb[0]) {
                var sum = 0;
                for (var k in mA[0]) {
                    sum += mA[i][k] * mb[k][j];
                }
                result[i][j] = sum;
            }
        }
        self.postMessage({JSON.stringify(result)});
    }
},
false);
As an example that proves Internet browsers are quite capable of distributed computing, we present a typical scientific study from bio-informatics domain.

This study performs protein sequence alignments using the Needleman-Wunsch algorithm implemented in JavaScript (http://opal.przyjaznycms.pl).

Sequence alignment is a common method employed in bioinformatics as a way to order sequences of proteins and DNA to identify areas of similarity that could be attributed to some relationship between the sequences.

The data for the alignments was obtained from the UniProtKB http://www.uniprot.org/

The experiment was set to perform 33,000 alignments
Some Results - 2
Some Results - 3

![Graph showing estimated GFLOPS and completed tasks over time. The graph has two lines: one representing tasks and the other representing aggregated power (GFLOPS). The x-axis represents time in HH:MM format, and the y-axis represents the estimated GFLOPS and completed tasks on a logarithmic scale.]
Work Done But Not Online

Workflows in Browsers
Workflows in Browsers

- Executing a large number of independent jobs is not meaningful.
- It is better if jobs can communicate their result in a structured way (coordination).
- A workflow structure is a graph-based structure composed of a set of ‘value’ and ‘job’ nodes.
Parsing

```xml
<workflow>
  <nodes>
    <value id="val1" value="44" returntype="Number" />
    <value id="val2" value="13" returntype="Number" />
    <job id="job1" src="Add" returntype="Number">
      <parameters>
        <parameter paramtype="Number" id="a" />
        <parameter paramtype="Number" id="b" />
      </parameters>
    </job>
    <job id="job2" src="Add" returntype="Number">
      <parameters>
        <parameter paramtype="Number" id="a" />
        <parameter paramtype="Number" id="b" />
      </parameters>
    </job>
    <value id="val5" value="100" returntype="Number" />
    <job id="job3" src="Add" returntype="Number">
      <parameters>
        <parameter paramtype="Number" id="a" />
        <parameter paramtype="Number" id="b" />
      </parameters>
    </job>
  </nodes>
  <edges>
    <edge id="first" source="val1" dest="job1" paramdest="a" />
    <edge id="second" source="val2" dest="job1" paramdest="b" />
    <edge id="third" source="val5" dest="job2" paramdest="a" />
    <edge id="fourth" source="job2" dest="job3" paramdest="a" />
    <edge id="fourth" source="job1" dest="job3" paramdest="b" />
  </edges>
  <finalnode id="job3" />
</workflow>
```
Value node

- Holds a value
  - (e.g. “a string”, true, [1, 3, 5, 7], 1.71, …)
- Usually acts as an input to a job / pool collection node
Job node

- Represents an activity that must be executed on a remote worker
- 2 types:
  - WebWorker jobs
  - WebCL jobs
- Returns a result/value
WebWorker job node
WebCL job node

- main
  - kernel #1
  - kernel #2
  - kernel #3

- kernel #4
- kernel #5
- kernel #6

- CPU
- CD 1
- CD 2
- CD n
Pool collection node

- Array Input (n elements)
- N Value node
- N processor node
- A reducer node
- SIMD model
Executor vs Workflow Structure

- Workflow structure
  - Which output goes to which input port
  - Which node should be started first
  - Which node stores the final result of the workflow

- Dataflow Scheduler & Executor
  - When to queue the next available job / value / pool
  - How to pass result from one to another
Scheduling Step - 1

Job Queue

Diagram:
- val5 -> job2
- val1 -> job1, job3
- val2 -> job1
- job1 -> job3
- job2 -> job3
Scheduling Step - 2

Job Queue

Diagram showing dependencies among jobs and values.
Scheduling Step - 3
Scheduling Step - 4
Scheduling Step - 5

Diagram showing job dependencies and status: job1, job2, job3, val1, val2, val5.
Scheduling Step - 6
Scheduling Step - 7
Scheduling Step - 8

Job Queue

Diagram showing dependencies between jobs and values.
Scheduling Step - 9
Scheduling Step - 10

Job Queue

job3

val5

val1

val2

job2

job1

job3

job3

Validation and application of dependencies.
Scheduling Step - 11
Scheduling Step - 12
Workflow Execution

movie
www.elab.lab.uvalight.net/weevilscout/
### WeevilScout UI - 3

#### Weevil Network

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>status</th>
<th>queued time</th>
<th>duration</th>
<th>results</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>matrix-mul-1</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>67890</td>
<td>matrix-mul-1</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>23456</td>
<td>matrix-mul-1</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
</tbody>
</table>

#### Submit a Weevil

#### My Weevils

#### Quick How-To