**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

- Linux Kernel in JS
  - http://bellard.org/jslinux/
- HTML 5
  - http://slides.html5rocks.com/#web-workers
- Getting the V8 JS engine
  - https://code.google.com/p/v8/wiki/UsingGit
- Building V8
  - https://code.google.com/p/v8/wiki/BuildingWithGYP
- Running console JS
  - ./shell hello.js
- Node.js
  - http://nodejs.org/
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
Communication

- **WebSockets**
  - `websocket = new WebSocket("ws://echo.websocket.org/");
  - `websocket.onopen = function(evt)`

- **WebRTC**
  - P2P browser communication intended for video streaming.
  - Using ICE framework (STUN TURN)
WebRTC
WebRTC
• 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.
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));
    }
});
As an example that proves Internet browsers are quite capable of distributed computing, we present a typical scientific study from bioinformatics 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
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.
<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="job2" dest="job3" paramdest="b"/>
    <edge id="fourth" source="job1" dest="job3" paramdest="b"/>
  </edges>
  <finalnode id="job3"/>
</workflow>
Scheduling Step - 1

Job Queue
Scheduling Step - 2

Job Queue

val5 -> val1 -> val2
  
job2 -> job1
  
job3
Scheduling Step - 3

Job Queue

val5 -> val1 -> val2

job3 -> job1 -> job2
Scheduling Step - 4
Scheduling Step - 5
Scheduling Step - 7
Scheduling Step - 8
Scheduling Step - 9

Job Queue
Scheduling Step - 10
Scheduling Step - 12

Job Queue

Diagram showing dependencies between jobs and values.

- val5
- val1
- val2
- job2
- job1
- job3
Url

goo.gl/RD1LtT
WeevilScout UI - 2

```java
// Based On OPAL [http://opal.przyjaznycms.pl/en/]
// Version: 1.0
// Author: Adam Skowron [http://www.przyjaznycms.pl]
// fasta sequence format supported
// function weevil main(sequence1, sequence2, type, score, mismatch_mward, mismatch_penalty, matrix, penalty, linear_gap, open_gap, prolongation_gap, output_format) {
var retStr = new String();
var options = {};
var options.seq1_header = sequence1.match(/^[a-zA-Z]+$/); 
var options.seq2_header = sequence2.match(/^[a-zA-Z]+$/); 
var options.sequence1 = options.sequence1.replace(/[^A-Z][^a-z]/g, ''); 
var options.sequence2 = options.sequence2.replace(/[^A-Z][^a-z]/g, '');
var options.type = (type ? type : 1);
var options.score = (score) ? score : 1;
var options.mns = (mismatch_mward) ? mismatch_mward : 1;
var options.mnp = (mismatch_penalty) ? mismatch_penalty : 0;
var options.matrix = (matrix) ? matrix : 'blosum';
var options.penalty = (penalty) ? penalty : 'linear';
var options.gp = (linear_gap) ? linear_gap : 0;
var options.opengap = (open_gap) ? open_gap : 0;
var options.prolongationgap = (prolongation_gap) ? prolongation_gap : 0;
var options.outputformat = (output_format) ? output_format : 'html';
var nucleotides = 'ACTG';
var aminoacids = 'ACEDQGRKHPSTFYW';
var cols = options.sequence1.length + 1;
var rows = options.sequence2.length + 1;
var final_score = 0;
var final_seq = '', final_seq = '';
var seq_line = ''; 
var $ = new Array(!$); // macierz podstawien
```
### WeevilScout

#### WeevilNet

<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>d2390b3-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d27b82-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d292f0-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d2509f5-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d277a1-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d2b8e0-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d238c1-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d2b1b1-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d2335b-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d234d2-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d235e9-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
<tr>
<td>d272e2-4af9-11e2-ac9c-8778775648ff</td>
<td>matrix-mult</td>
<td>complete</td>
<td>2013-06-13 16:59:31</td>
<td>0:00:00</td>
<td>results</td>
</tr>
</tbody>
</table>
What Next?

- Combining different computing platforms as one Browser, Cloud, Desktop
- Platform agnostic data processing protocols
- Browser overlay networks (Think bittorrent and CDNs)