Project proposal – SETI@EPFL

Talau Cristian, Laurentiu Dascalu, Ganea Octavian

**Main problem**

The problem we are trying to address with this project is how to make computational intensive applications (e.g. research projects) benefit from the idle machines within the Internet.

In accordance to recent statistics, just in August 2011 people spent over 41.1B minutes on Facebook[1] and almost half of it on Twitter[2]. This suggested us the fact that most of the time, people are keeping their personal computers in idle, thus consuming energy without any visible benefit. Why not using these resources for other more important applications?

**Our proposal**

Our idea is to build a SETI@HOME like system which executes multiple tasks on the machines of ordinary Internet users that are idling on websites. Our goal is to split high resource consuming applications into tasks and send them to run as JavaScript[3] code on people’s local machines. This code will be compiled from Java code allowing developers to use their favorite language to write distributed applications.

This approach has the additional advantage of offering high security guarantees for clients’ computers which are ensured by the browser; there is no need to run executable binaries on local machines. Moreover, Java code can be executed on different platforms without the need of having specific runtime environments installed (e.g. the Java Virtual Machine – JVM[4]).

**State of the art**

Nowadays, there are multiple projects which are using the BOINC[5] platform in order to benefit from volunteer computing. Some examples include SETI@HOME[6], LHC@HOME[7] and MILKYWAY@HOME[8].

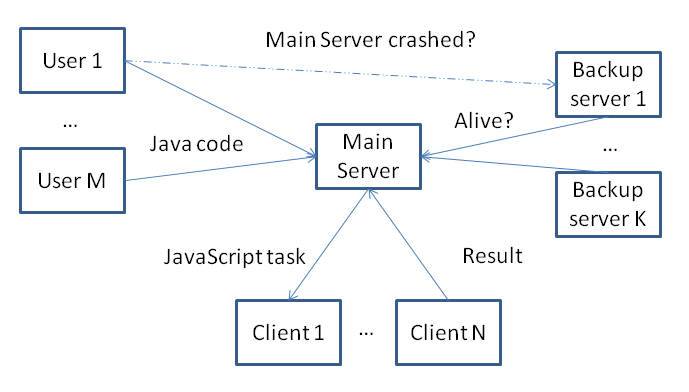
The main problem with this existent approach is that it requires users to download, install and run some executables on their local machines. This has the drawbacks of not being straightforward for the majority of the Internet users and also of offering vulnerability in front of viruses or other threats. Moreover, our implementation will allow new projects to be deployed much easier without having to worry about the platform differences.

**Proposed experiments**

We plan to compare the performance of running Java applications on a small computer cluster with that of running the same tasks using our system on multiple personal computers’ browsers. We will create a mini benchmark and will assess which kinds of applications are suitable for this approach.

**Functionality**

We will provide a website and a Facebook application for our system. Our model is client-server with multiple servers, in order to support fault tolerance, scalability, reliability and load balancing on both client and server sides.



Our approach assumes 3 kinds of roles: *servers*, *users* and *clients*. Users enter our website and send new Java code to be run on clients’ machines as JavaScript code. Clients represent idle machines that accept incoming JavaScript code and run it while keeping the browser open. There will be multiple servers that will receive new Java parallel applications from users, distribute them to the clients, receive and combine the answers from clients and, eventually, send the final results to the users.

**Main challenges**

* Security, clients may submit wrong results
* Fault tolerance on both server and clients sides
* Avoidance of clients' overloading in terms of memory, CPU and bandwidth utilization
* Possibility of users’ submissions of parallelized applications that would be run on multiple clients

**Addressing main issues**

1. Compile Java to JavaScript  
   We plan to integrate our application with Google Web Toolkit, thus generating Javascript source code from uploaded Java code.
2. Facebook application  
   In order to access a large pool of users, we plan to integrate our application with Facebook. The main motivation is that Facebook has many users, and they can help a research project if they stay online in the most popular social network.
3. Load balancing  
   We will test that the server is doing load balancing, by defining tasks of different complexity and monitoring how the tasks are mapped on clients.
4. Fault tolerance on client-server  
   We will make sure that our application is fault tolerant by running different tests for both client and server sides. A faulty client is not giving an answer in a limited amount of time, and the server should resend the unresolved tasks to other clients; a faulty client is excluded from future cooperation. A faulty server would not respond to clients' queries and the clients have to connect to a backup server; once the  main server synchronizes with the system, the clients starts cooperating with it.
5. Avoid clients overloading  
   Each client will provide before connection some maximum quotas for memory and CPU consumption and for bandwidth utilization depending on its idleness degree.
6. Security wrong results  
   We plan to test that the clients are not submitting wrong results, by sending the same task to 3 clients. If  all 3 results are different, then the server retries the task. If all 3 results are identical, then we consider the task successfully resolved. If only 2 results are identical, then we found a faulty client and we exclude it from future cooperation.

**Timeline for the project**

|  |  |
| --- | --- |
| 31.10 – 13.11 | Define the client-server and server-server communication protocols, and the tasks interface. |
| 14.11 – 27.11 | Build the web based user interface that would allow the user to upload Java source code. Integrate our server application with the Google Web Toolkit, in order to generate Javascript source code from Java code. The user can specify the amount of resources (e.g. CPU, memory, bandwidth) he/she wants to allocate. |
| 28.11 – 02.11 | Address the security issues, such as clients submitting wrong results. |
| 03.11 – 10.11 | Introduce fault tolerance support, on both server and client sides. |
| 11.11 – 13.12 | Avoid clients' overloading in terms of memory, CPU and bandwidth utilization. |
| 14.12 – 16.12 | Improve application performance using speculative execution. |
| 16.12 – 18.12 | Deploy SETI@EPFL on Facebook, ensure the proper deliverables are in place. |
| 19.12 – 23.12 | Demo; define several tasks to be run in our Internet browsers and make sure that everything works as expected. |

**Deliverables**  
  
We will deliver a document that describes and analyzes our solution, including information about the client-server and server-server communication protocols, the tasks interface, how the users can upload Java source code and other internal mechanisms. The proposed solution consists of source code deployed on servers that would user to upload and execute their Java source.

**References**

1. <http://facebook.com>
2. <http://twitter.com>
3. <http://en.wikipedia.org/wiki/JavaScript>
4. <http://en.wikipedia.org/wiki/Java_Virtual_Machine>
5. <http://boinc.berkeley.edu/>
6. <http://setiathome.berkeley.edu/>
7. <http://lhcathome.web.cern.ch/LHCathome/>
8. <http://milkyway.cs.rpi.edu/milkyway/>