students: Octavian Ganea, Cristian Tălău, Laurențiu Dascălu

Problem statement

Our idea is to build a system to run SETI@HOME like projects which execute 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 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.

In the final version, we will expect that the programmer uploads three Java classes (*map worker*, *reduce worker*, *input generator*) and an input file. The classes are dynamically plugged into our system, and it starts executing his/her code as any MapReduce framework (e.g. Hadoop).

We have designed the interfaces that are to be used by the user.

Server-side interface

The server uses one *IInputGenerator* per project that run on that server.

```
interface IInputGenerator <K extends IsSerializable, V extends IsSerializable>
  boolean hasNextTask();
  Task<K, V> nextTask();
```

Client-side interface

On the client-side, we provide a collection classes that the programmer will customize. We provide a *JobConf* implementation that is responsible for the *map* and *reduce* workers instantiation, for the communication with the server (e.g. download tasks, upload results). From the client-side perspective, the programmer has just to define his/her map workers.

```
interface IOutputCollector<T>
  void emit(T data);

interface IMapWorker<K1, V1, K2, V2>
  void map(K1 key, V1 value, IOutputCollector<Pair<K2, V2>> output);

interface IReduceWorker<K, V>
  void reduce(K key, SVector<V> value, IOutputCollector<V> output);
```

Shared structures

The clients and the server share the *map* and *reduce* tasks structures, and other common types (e.g. serializable String, Integer, Boolean, Void).

```
class MapTask<K, V> extends Vector<Pair<K, V>> implements Task<K, V>
class ReduceTask<K, V> extends Pair<K, Vector<V>> implements Task<K, V>
interface Task<K, V>
```

Mandelbrot example

Currently, we created the system structure that statically uses some implementation a map worker and a reduce worker that compute the Mandelbrot fractal of given size. The *MandelbropMapWorker* computes a part of the Mandelbrot fractal image and the *MandelBrotReduceWorker* combines all these lines in a final image.

Interestingly, we found that computing a 256x256 Mandelbrot fractal using Java takes about 17 seconds, while the generated JavaScript code from that Java code takes less than 7 seconds. We believe that this is due to the poor performance of our JVM.

The *custom* classes are separated from the generic part of the system, so it should be easily generalized to support dynamically provided *mappers* and *reducers*. After this we will look at some features like: fault-tolerance, load balancing, security, performance issues.

We found out that combining images on the client is unfortunate, because the data transfer is more expensive than *reducing* it (store multiple matrices in a single matrix). Thus, we figured out that the programmer has to specify if a reduce task can be executed on the server directly.