Some projects like set@home, do scientific computing with a super computer built out of home computers. Participating in such a computing requires too often custom client software to be installed. We have drafted, and started prototyping, a browser based computation platform that should allow people to participate in computations by simply pointing their web browser at a website. The platform is based on the MapReduce model of defining distributed computations.

BROWSER GRID COMPUTING

MAPREDUCE MODEL

The following picture shows the information flow of MapReduce, a possible model for handling distributed computations. Input data is read into partitions, each of which are handled by an independent maptask. The maptask results are divided into buckets, and a reduce task is launched for each bucket. Finally, results are stored.

In a browser centric implementation data can be read and stored using XMLHttpRequest. CORS and JSONP are required for cross-origin access to systems. In some edge cases generated input data, and no output may do. To communicate efficiently between workers, some peer-to-peer technologies are needed. We have considered Flash, and BrowserSocket. In future html5 may bring an alternative solution.

COROUTINE REQUIREMENT

Coroutines are required for implementing MapReduce platforms efficiently. The reducer functions contain state, and thus can not be stopped before the whole reduce task is completed. The problem that rises is that the reduce task is then required to download all contents of a bucket before it can start the second reducer for that reducetask. This eats up lots of memory.

The problem can be solved by using coroutines. Instead of doing a destructive return, a coroutine equipped programming language lets the programmer do yield, which allows one to continue execution at a later time. Thus one can process some chunk of work entirely before moving to the next one. See the picture above, for an example.

BEESWARM MAPREDUCE

ARCHITECTURE

BSMR architecture consists of a single master server that is responsible for queueing jobs, and commanding workers. The administrator may add jobs to the queue by using a console application. Multiple console applications may be used, but all of them are considered to be the same administrator. Visitors of the web site run one worker application each, and the workers register themselves with the master. The picture above shows the setup. All communications happen over Websocket connections towards the master. The master can share a TCP port with a web server.

ADDING JOBS

Adding a job is done through the console, like the one shown in the screen shot below. The administrator gets to decide the amount of maptasks, and reducetasks. i.e. the granularity of distribution. He has to further define the map and reduce operations as javascript code, and select the ways of input, output, and peer-to-peer communication from a list of plug-ins. He may also need to write custom plug-ins, if he is not doing something really trivial.

MONITORING JOBS

In the screen shot above, a mapreduce job is being executed. The master server sends status updates to all connected consoles periodically. The yellow boxes show maptasks that have started. Once finished, they turn green. The worker list shows all connected worker nodes. That would be the visitors of the site. Some technical details, like the peer-to-peer communication address is shown for each peer. For debugging reasons the console also shows the code that is being sent to workers.