

Branislava Šandrih

Faculty of Philology, University of Belgrade

Vladimir Filipović, Saša Malkov, Aleksandar Kartelj

Faculty of Mathematics, University of Belgrade

DISTRIBUTED COMPUTING AMONG INDEPENDENT WEB BROWSERS APPLIED TO TEXT AND IMAGE PROCESSING

Abstract. Distributed computing implies presence of unused software resources available on multiple computers that work as a single system. This kind of computing uses a system with parallel architecture and varying node reliability. As a consequence, an adequate programming paradigm has to be used. Web application, described in this paper, is designed with such paradigm in mind. It is developed using popular technologies. Proposed approach can attract two types of users: ones that need additional computing resources (in further text *seekers*) and ones that are willing to contribute by putting their computing resources on disposal (in further text *helpers*). Seeker is obligated to share their data which is then divided into equal segments. Number of these equal segments is defined by seeker in advance. Secondly, seeker has to define processing procedure, i.e. code for processing these segments separately. Eventually, they should define the way how processed segments are reduced into final result. Described programming paradigm is known as MapReduce. Data can be in arbitrary format (at the moment, the system is evaluated for text and images) as long as the map-function handles it in the appropriate way. Helper is assigned a segment of the input data. Map-function, defined by the seeker, is then executed within helper's Web browser and its result is being returned to the system when processing procedure finished. The Web application's efficiency depends on the number and configuration of computing nodes. Four different use-cases are demonstrated in this paper: 1) word counting in file containing text, 2) finding the largest number in the text file that contains numbers, 3) sharpening of the corrupted image and 4) applying blur effect on the image file. Since its simplicity and universality, the system has potential for other more complex computations and could, in the future, be applied in the domain of distributed content digitalization, analysis of the data obtained from telescopes etc.

Keywords. Distributed computing, grid computing, meta computing, global computing, peer-to-peer computing, MapReduce paradigm, image processing, digitalization.

1. Introduction

The number of connected devices on the Internet is constantly rising. Similarly, computing power of these nodes is simultaneously increasing. Users on the Web create heaps of data every second. User needs are also getting extremely higher. Although today's computing resources are incomparable to those twenty years ago, apparently they will never be powerful enough to fulfill all incoming user needs. The goal is to harness these unused computing resources on the Internet and build a large parallel computer.

The idea is not fresh. One of the first projects of this sort was so called SETI@home project [1], [2] with a purpose of finding intelligent forms of life outside Earth analyzing incoming radio-signals from space in real time. In this paper we describe a Web application [3] designed for parallel computing distributed among the computer resources of the application users which is executed in their Web browsers.

It has simple architecture and also gives users possibility to specify their own tasks and let the application distribute it, which assures needed flexibility.

This paper is organized as follows. In section 2 some of the existing systems with similar purpose are listed and briefly described. In Section 3 we demonstrate our application and shortly describe its technical background. The prerequisites for defining user-owned tasks are explained in section 4, and section 5 shows results of the test examples described earlier. Finally, in section 6 we conclude and state our plans for future work.

2. Previous Work

In this section we name and shortly describe some of the so far developed Web applications for distributed computing. This list is obtained from search results of next terms: global computing, grid computing, peer-to-peer computing, distributed computing, meta computing, seamless scalable computing, scheduling.

Pioneer in the field of distributing computations through the net of Web browsers is Javelin [4] project from 1997. Authors recognized the potential of – at that point popular – Java applets. One could define their computation need, code it in Java and upload the applet. Users willing to help just had to visit the Web page and let the applet execute. Another similar initiatives, from the same year, are described in The Legion [5] and NetSolve [6]. Two years later, the Ninf library [7], global network-wide computing infrastructure project which allows users to access computational resources including hardware, software and scientific data distributed across a wide area network, was launched. Code was written on the client-side and executed on the server side. In the year 2001 is the already mentioned SETI@home [1], [2] launched, where abbreviation originates from *Search for Extraterrestrial Intelligence*, approaches so called ‘public-resourcing’ model in the means of distributing portions of computations to the unused computer resources of the volunteer Internet users. Different types of signals are captured by radio-telescopes and should be processed in real time in order to separate noisy input from signals emitted by other potentially existing living entities from space. These processings are computationally complex. On the other side, more radio-telescopes would increase chances of encountering other forms of lives. More projects that allow users to define their own tasks is XtremWeb [8] from 2001 and BOINC [9] project from 2004.

3. Technical specification

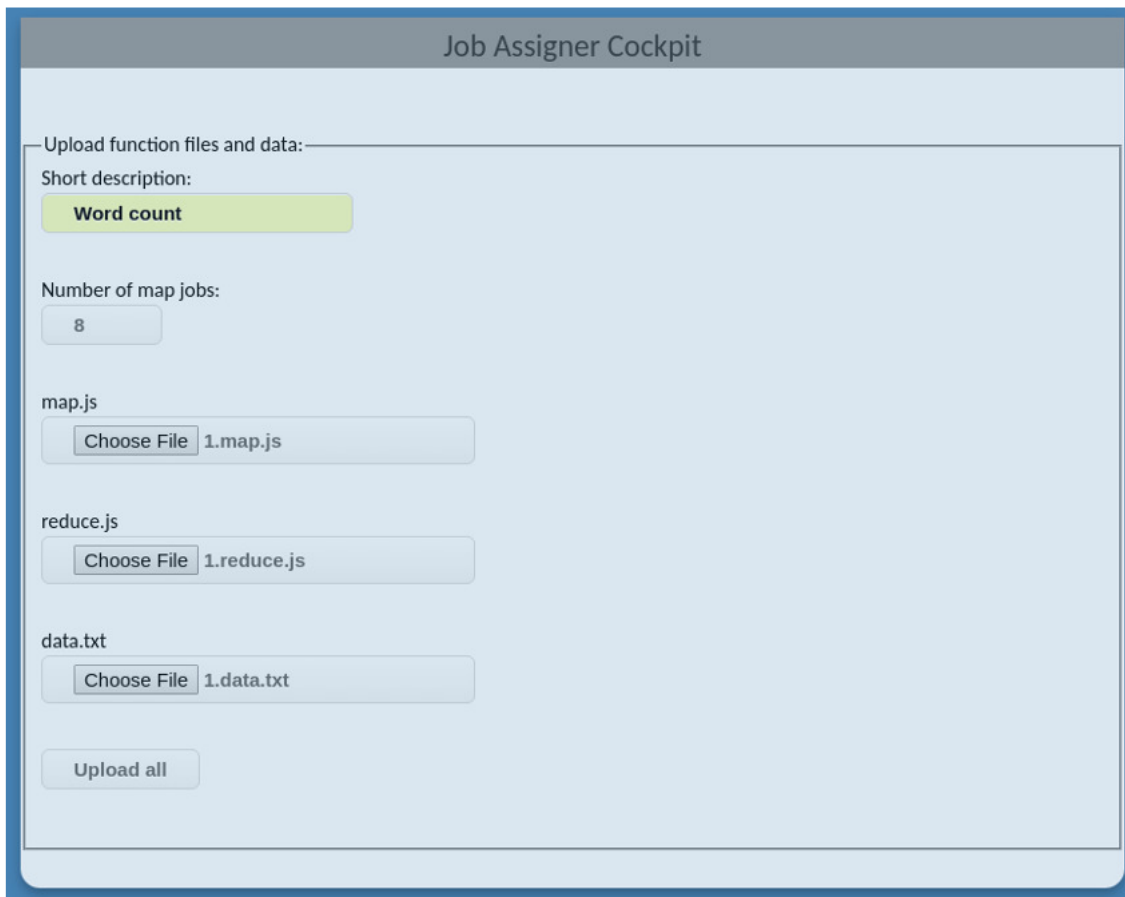
The Web application described in this paper can be used by two types of users: it can be used by ones who need additional computer resources (seekers) and by ones who are willing to put their computer resources at disposal (helpers). It is free and it only requires user registration. Although we name two different types of users, there is no hierarchy between users and one user can be both helper and a seeker at the same. The names exist only due to distinction of different aspects of use.

Application’s server code is written in Node.js technology [10], JavaScript asynchronous framework for development of scalable JavaScript server applications. MongoDB [11] document-oriented database is used for storing data, while data are exchanged between client and server in JSON format [12] using socket.io [13] JavaScript library for server-client communication in real time.

Web pages are rendered using EJS [14] templating technology, i.e. they are HTML pages with ability to directly include values received from server. Web Workers technology

[15] enables JavaScript code execution independent of graphical user interface. Finally, interactive diagrams for task visualization are displayed using GoJS [16] JavaScript library.

Upon login, users can choose whether they want to specify new task or help with an existing. In case of the first choice, user is redirected to Web page shown in Figure 1. User is then obligated to define task name and number of map-units (8 by default), and also upload two separate JavaScript files containing map-function and reduce-function, and finally file containing input data. Number of map-units specifies number of smaller task units, which results will be gathered after completion and reduced into final results representing processed input. Input data can be in any format, as long as map-function handles it appropriately. Once the task is defined, seeker is redirected to control page (known as cockpit, see Figure 2) where he/she can follow the status: how many tasks are assigned and being processed, how many tasks are completed and are there any neglected tasks. Task is neglected if it was assigned to helper, but the helper did not send any results back to server after certain time (network timeout, closing browser before job completion etc.). Cockpit is also reachable from the user profile page visible from the navigation menu.



The screenshot shows the 'Job Assigner Cockpit' interface. At the top, there is a header 'Job Assigner Cockpit'. Below it, a section titled 'Upload function files and data:' contains the following elements:

- Short description:** A text input field containing 'Word count'.
- Number of map jobs:** A numeric input field containing '8'.
- map.js:** A file upload button labeled 'Choose File' with the filename '1.map.js'.
- reduce.js:** A file upload button labeled 'Choose File' with the filename '1.reduce.js'.
- data.txt:** A file upload button labeled 'Choose File' with the filename '1.data.txt'.
- Upload all:** A button at the bottom of the section.

Figure 1: Assigning New Task

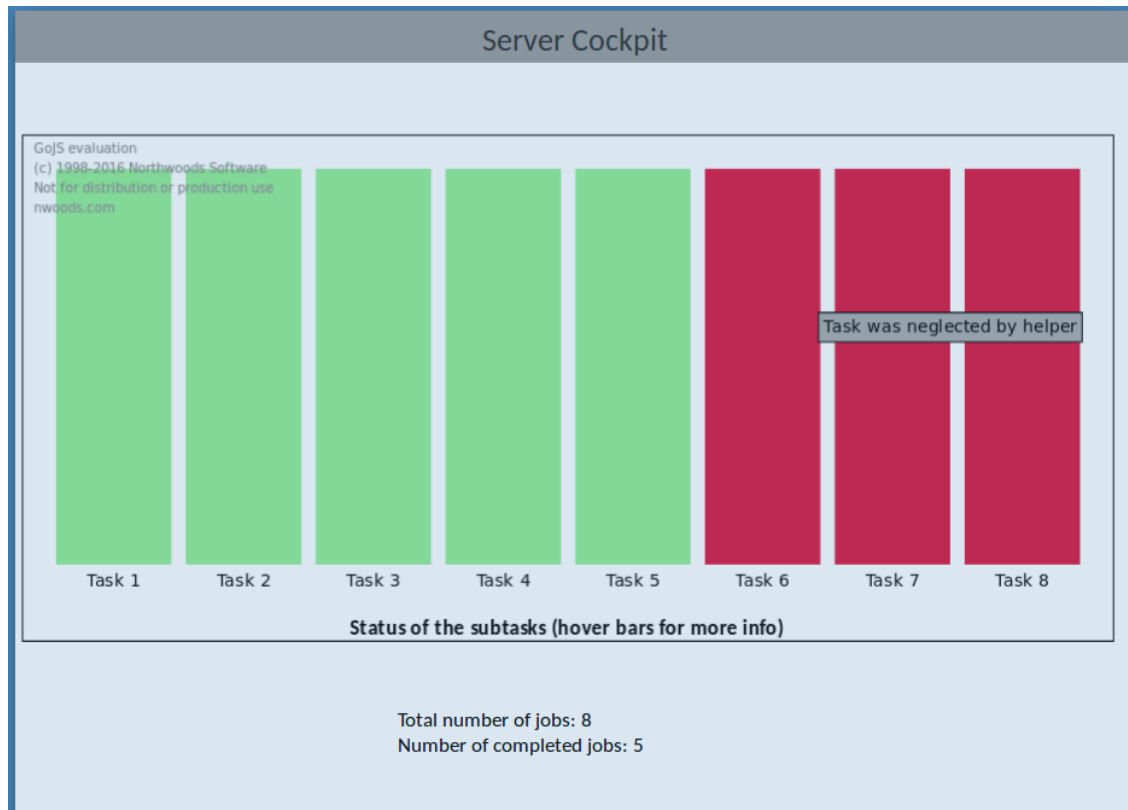


Figure 2: Cockpit for the certain task

4. Writing JavaScript functions for task definition

Task solving procedure should contain data splitting, parallel independent processing and reduction afterwards. This programming model is known as MapReduce paradigm [17]. There are free frameworks for MapReduce paradigm support in different programming languages with different optimization levels. One of the most famous open-source implementations is Apache Hadoop [18]. Another interesting implementation for MapReduce applications coded in Java is Twister¹ [19], that iteratively repeats procedure until the results are obtained.

Seeker is expected to have some experience with JavaScript. Following are the guidelines for tasks definition explained on concrete usage examples.

4.1. Obtaining word frequencies (WordCount). This is usually the first example used while demonstrating any MapReduce framework.^{2 3} Input file contains some text, e.g. text from a book. Output should contain list of all tokens present in text, along with their frequencies. For example, in case of the next input file:

This is some random text. We want to count words that appear in this text. This text is rather short.

1 Twister: Iterative MapReduce <http://www.iterativemapreduce.org/>

2 Apache Hadoop Java code for word counting: https://hadoop.apache.org/docs/r2.8.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example: WordCount_v1.0

3 Twister Java code for word counting: http://www.iterativemapreduce.org/samples.html#Word_Count

output could be in JSON format:

```
{ "this": 3, "text": 3, "is": 2, "some": 1, "random": 1, "we": 1, "want": 1, "to": 1, "count": 1, "words": 1, "that": 1, "appear": 1, "in": 1, "rather": 1, "short": 1 }
```

Formats of the map and reduce functions are given in the Table 1.

<pre>onmessage = function(e) { var words = e.data.split(' '); var wordsMap = new Object(); for(var i = 0; i < words.length; i++) { var word = words[i] .toLowerCase() .replace(/[\^\w\s]_ _/g, "") .replace(/s+/g, " "); if(!wordsMap[word]) wordsMap[word] = 0; wordsMap[word]++; } // sending results of the map job postMessage(wordsMap); };</pre>	<pre>function reduce(arrWordsMap) { finalResults = new Object(); for(var i = 0; i < arrWordsMap.length; i++) { result = arrWordsMap[i]; for(var word in result) { if(!finalResults[word]) finalResults[word] = 0; finalResults[word] += result[word]; } } // sending reduced results to server return finalResults; }</pre>
--	--

Table 1: Map (left) and reduce (right) JavaScript functions for WordCount

Client side uses Web Workers technology [15], i.e. it creates child process that communicates with its parent exchanging messages. Whenever child process is created, it should firstly await until its parent hands data for processing. This is visible from the lines:

```
onmessage = function( event ) {
  text = event.data;
}
```

Field 'data', from the event object, contains data in text format sent from parent. An 'on message' event triggers for the whole procedure of invoking map-function and data processing. Once the data is processed, Web Worker should notify its parent that the job is completed and return function results:

```
postMessage( wordsMap );
```

These steps are the only technical constraints during map-function definition. Once all map jobs are completed, results should be reduced. One should have in mind that the input for this function is a list of objects of the type returned in the mapping step. Output can be of any type or format. This is in all means typical JavaScript function.

Once the task is completed, seeker sees all bars of as green in their cockpit (Figure 2) and below a diagram, a hyper link for results download is displayed.

4.2. Finding the largest number (MaxNumber). For the problem of finding largest number in the file, containing numbers, map-function should contain code for handling segment of the input file in text form. We will not go into details, but rather just briefly describe the idea. After reading numbers into some appropriate structure (e.g. list), map-function should return the largest number for that segment of input file. Reduce-function receives list of numbers as its argument. It should find the largest number in this list and return it as a final result.

4.3. Improving image quality using sharpening technique (ImageSharp). Input could be provided in a form of a bitmap matrix with dimensions height×width×3. In case of the image with dimension 5×3, input file could have form given in Table 2: each pixel is represented with red, green and blue value from the [0, 255] interval and pixel values are separated with tab. Seeker is not constrained with input format, but should have in mind that file segmentation is later done based on number of lines (every map-job should receive same number of lines from the input file), and not based on amount of bytes.

0 0 0	255 255 255	0 0 0	
255 255 255	0 0 0	255 255 255	-1 -1 -1
0 0 0	255 255 255	0 0 0	-1 k -1
0 0 0	255 255 255	0 0 0	-1 -1 -1
255 255 255	0 0 0	255 255 255	

Table 2: Left: input file structure, right: image filter

Part of the code from the map and reduce-functions is given in. First the text data contained in e object should be read into structure that saves matrix input. Another matrix with same dimensions is then created and it will represent processed image. For this specific task, one needs an image filter. That is usually quadratic matrix with all the same values (0 or -1), except the one on the intersection of matrix diagonals (see Table 2, right). The value k is arbitrary and represents intensity of the sharpening technique (in our code it is set to 10). This filter behaves as a mask: groups of nine neighbor pixels are covered with mask, their red, green and blue components are respectively multiplied by the appropriate filter value, then summed up and resulting values are stored as red, green and blue component of the new image pixel.

4.4. Applying blur effect on image (ImageBlur). We will not describe this problem in detail, since the procedure is quite similar with ImageSharp procedure. Results of these processing are hand out in the next section.

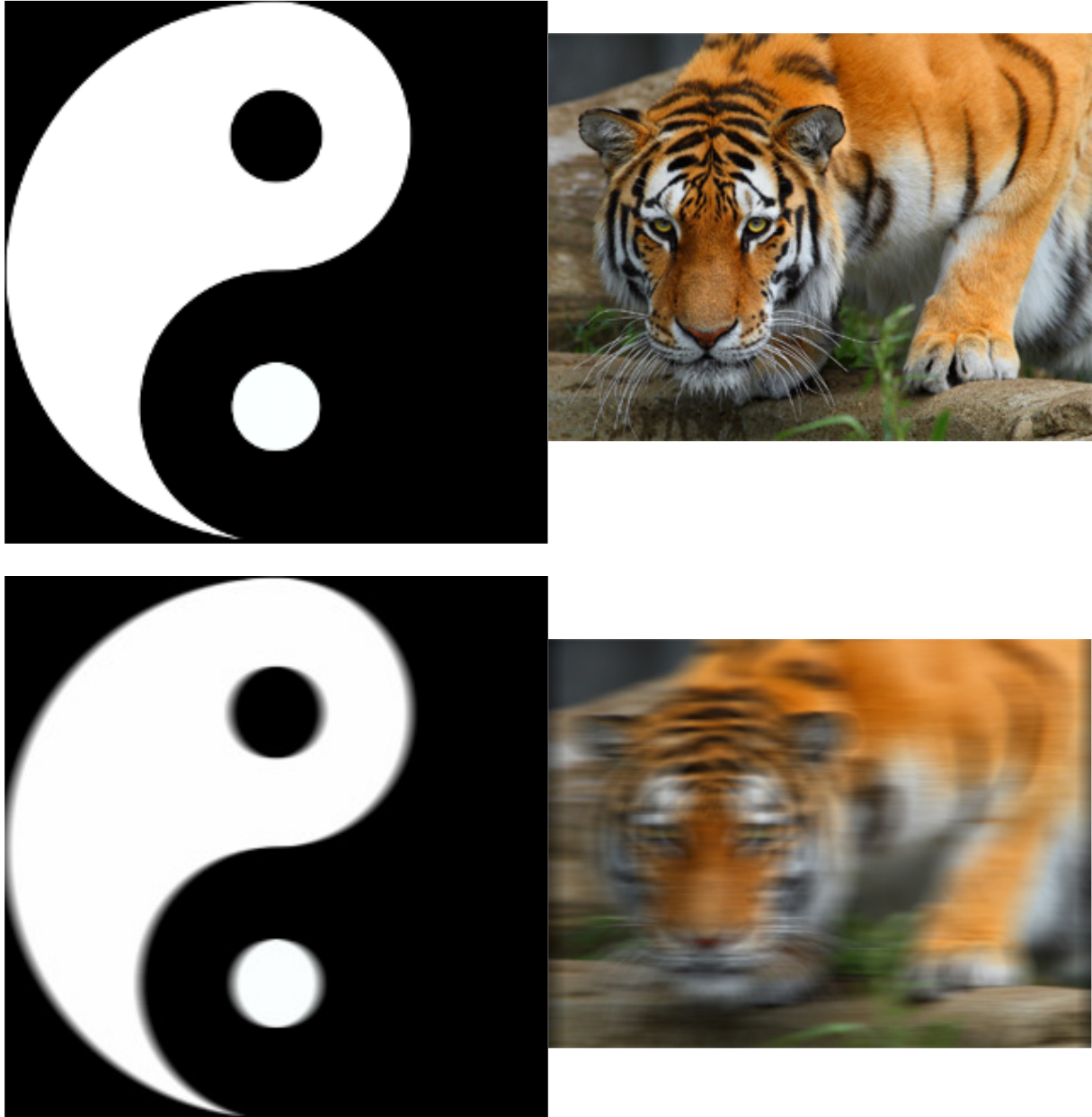
<pre> onmessage = function(e) { var imgArr = []; // ... reading bitmap matrix // columns (x) image width in pixels const M = imgArr[0].length; // rows (y) image height in pixels; const N = e.data.split('\n').length; // create array for processed image // based on input image dimensions imgB = createEmptyArray(N,M,3); // image filter (3x3, intensity = 10) filter = createFilter(3, 3, 10); // applying filter to each pixel for(var i = 0; i < N; i++) { for(j = 0; j < M; j++) { var sum_r = 0, sum_g = 0, sum_b = 0; for(var ro = -1; ro <= 1; ro++) { for(var co = -1; co <= 1; co++) { // sharpening algorithm sum_r+=imgArr[i+ro][j+co][0] *filter[ro][co]; sum_g+=imgArr[i+ro][j+co][1] *filter[ro][co]; sum_b+=imgArr[i+ro][j+co][2] *filter[ro][co]; } } // storing modified pixel // into new image bitmap imgB[i][j][0] = sum_r; imgB[i][j][1] = sum_g; imgB[i][j][2] = sum_b; } } // sending results of the map job postMessage(imgB); } </pre>	<pre> function reduce(listMat) { var finalResults = ""; // listMat contains list of matrices for(var ind = 0; ind < listMat.length; ind++) { var matrix = listfMat[ind]; var n = matrix.length; var m = matrix[0].length; // creating string representation // for each matrix // and concatenating these strings // ... } // sending reduced results to server return finalResults; } </pre>
---	---

Table 3: Map (left) and reduce (right) JavaScript functions for ImageSharp

5. Results

The two bitmap images, processed with application described in the paper, are displayed in table 4. Image on the left has only black, white and gray pixels. Since its pointed, clear edges, the outcome of the blur technique is more obvious. Right image is more colorful and the blurring technique returned nicely processed image. When this blurred image is used as an input for the sharpening technique, the resulting image looks like original, so we did not find it interesting for this case. Yet, what we would like to point out are the obvious eight

horizontal lines⁴ on the sharpened version of original image. This is due to segmentation step before map-function invocation. The sharpening algorithm is smooth and each pixel value depends on its neighbor pixels. When segments are divided and processed independently, edge pixels lose information about their position on the image and the neighbor pixels values. This can be overcome by sending larger image segments to Web Workers and by more thoroughly reduction afterwards.



4 Number 8 is due to the default number of map-jobs during task definition step shown in figure 1



Table 4: First row: original images, second row: blurred images, third row: sharpened images with parameter $k = 10$

6. Conclusions and future work

Application described in this paper has simple design and architecture, and its programmability allows higher level of flexibility. Javelin [4] used technology of Java applets that is today deprecated, so the project intercepted. Our application is developed in JavaScript as its programming language which is nowadays considered among the top 5 programming languages.^{5 6 7} The SETI@home [1], [2] project is live, but does not allow sharing own tasks, just contribution to existing ones. At the moment system is aimed at academic population but with simpler user interface even broader population could find it usable in the future. System currently supports only one reduction step. It could be redesigned in the means of adding multiple reduction-steps and an additional combine-step, inspired by Twister architecture [19]. These circumstances qualify our application as potentially usable in practice. It could find its applying text recognition, image and text filtering and many more.

References

- [1] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, *SETI@ home – Massively Distributed Computing for SETI*, *Comput. Sci. Eng.*, 3:1(2001),78–83.
- [2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, *SETI@ home: an Experiment in Public-Resource Computing*, *Commun. ACM*, 45:11(2002),56–6.
- [3] *MapReduce Distributed Web Browser Computing*, [Online]. Available: <https://liss2.matf.bg.ac.rs:3000>.

5 PYPL (PopularitY of Programming Language) <http://pypl.github.io/PYPL.html>

6 Stack Overflow <https://insights.stackoverflow.com/survey/2017#technology>

7 RedMonk <http://redmonk.com/sogrady/2017/03/17/language-rankings-1-17/>

- [4] P. Cappello, B. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauer, D. Wu, *Javelin: Internet-based Parallel Computing using Java*, ACM Workshop on Java for Science and Engineering Computation, 1997.
- [5] A. S. Grimshaw, W. A. Wulf, and others, *The Legion Vision of a Worldwide Virtual Computer*, Commun. ACM, 40:1(1997), 39–45.
- [6] H. Casanova, J. Dongarra, *NetSolve: A network-enabled Server for Solving Computational Science Problems*, Int. J. Supercomput. Appl. High Perform. Comput., 11: 3(1997), 212–223.
- [7] H. Nakada, M. Sato, S. Sekiguchi, *Design and Implementations of Ninf: Towards a Global Computing Infrastructure*, Future Gener. Comput. Syst.,15:5(1999), 649–658.
- [8] G. Fedak, C. Germain, V. Neri, F. Cappello, *XtremWeb: A Generic Global Computing System*, Cluster Computing and the Grid, Proceedings. First IEEE/ACM International Symposium on, 2001, 582–587.
- [9] D. P. Anderson, *BOINC: A System for Public–resource Computing and Storage*, Grid Computing, Proceedings. Fifth IEEE/ACM International Workshop, 2004, 4–10.
- [10] *Node.js* [Online]. Available: <https://nodejs.org/en/>.
- [11] *MongoDB* [Online]. Available: <https://www.mongodb.com/>
- [12] *JavaScript Object Notation (JSON)* [Online]. Available: <http://www.json.org/>
- [13] *socket.io*. [Online]. Available: <https://socket.io/>.
- [14] *Embedded JavaScript (EJS)* [Online]. Available: <http://www.embeddedjs.com/>
- [15] *Web Workers* [Online]. Available: <https://www.w3.org/TR/workers/>
- [16] *GoJS* [Online]. Available: <https://gojs.net/>
- [17] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, Commun. ACM, 51:1(2008), 107–113.
- [18] V. K. Vavilapalli et al., *Apache Hadoop Yarn: Yet Another Resource Negotiator*, Proceedings of the 4th annual Symposium on Cloud Computing, 2013, 5.
- [19] J. Ekanayake et al., *Twister: a Runtime for Iterative MapReduce*, Proceedings of the 19th ACM international symposium on high performance distributed computing, 2010, 810–818

branislava.sandrih@fil.bg.ac.rs

vladaf@matf.bg.ac.rs

smalkov@matf.bg.ac.rs

kartelj@matf.bg.ac.rs