Web Browser-based Social Distributed Computing Platform applied to Image Analysis

Mikel Zorrilla, Angel Martin, Iñigo Tamayo, Naiara Aginako, Igor G. Olaizola
Vicomtech-IK4
GraphicsMedia.net
Donostia - San Sebastian (Spain)
Email: {mzorrilla, amartin, itamayo, naginako, iolaizola}@vicomtech.org

***Abstract*—In this paper we introduce a new platform to perform image processing algorithms over big data. The main stakeholders of media analysis are the social services which manage huge volumes of multimedia data. While social service providers have already a big resources pool of connected assets through the devices of the community, they are not exploiting them for their processing needs and they usually deploy high performance systems that run batch works. Image processing requires parallelizable atomic and lightweight tasks that can benefit from a big community of thin devices executing seamless background processes while the user enjoys other social media contents. To provide such infrastructure a client-side browser solution based on JavaScript libraries has been developed. We also describe a performance model that establishes the contexts where the solution gets ahead in terms of available resources and the processing problem nature.**

I. Introduction

With rapid advances in multimedia production and the explosion of social media, the volume of multimedia that must be processed, analysed and tagged brings big data challenges. At the same time, dramatic decrease in the cost of commodity computing components brings large distributed computing platforms with tens or hundreds of thousands of unreliable and heterogeneous hosts. Grid systems cope with intensive processing tasks but barely face uncertainty of availability performing tasks with best-effort policy capturing all the required resources. Nowadays, Cloud Computing solutions rise to overcome elasticity in order to track processing request volume. All these solutions boost processing by deploying a monolithic, for grid, and elastic, for cloud, purpose-specific infrastructure. Future solutions must be driven by social and connectivity paradigms.

Social computing is an emerging field, which encompasses a diverse range of topics. It is mainly exploited as a vehicle for establishing and maintaining mainstream communication relationships between enterprises and customers. Major motivation of Social Business trends pursue the burgeon opportunities to monetise social knowledge, emphasising social intelligence.

However, a computing-oriented dimension for collaborative computing has not been explored yet. Social computing systems provide extra value than what is offered by computer systems alone. However, the next generation of the Internet envisages the Web as a computing rather than just publishing platform. This umbrella term has also been associated with Cloud Computing.

The rapidly increasing use of the Web as a software platform with truly interactive applications is boosted by emerging standards such as HTML5[1] is removing limitations, and transforming the Web into a real application platform middleware tackling hardware resources of the devices through JavaScript [13][12][1][15].

Common multimedia processing tasks such as segmentation, clustering and classification of multimedia streams from contents stored in a repository can be easily divided and distributed in atomic tasks. We can break up data into frames that can be processed independently as frames from different scenes can be considered unrelated. This lets us divide a large stream into small chunks that a thin device can analyse comparatively quickly. In this way, servers can dispatch the work to users willing to donate their spare CPU cycles.

Our approach deploys a promising solution to cope with the big data problem behind the batch analysis of the social media managed in a social service. This paper introduces a distributed user device platform on top of a social media community. The service provider takes benefit of the huge processing capacity of its big social community to seamless perform atomic image processing tasks. To achieve it, these lightweight works are embedded by the server to the different social content accessed.

This paper provides a state-of-the-art of big data infrastructures from on-going volunteer computing projects to web based distributed processing frameworks. We also summarise current image processing libraries exploiting browser capabilities of the user devices through JavaScript. The social distributed computing paradigm is also explained in the article giving rise to a system proposal to run multimedia analysis, making user device farm suitable for big data. A system architecture is detailed and we present performance models concluded from computational patterns studied in representative research. We introduce some image processing use cases where this infrastructure fits. Last but not least, a technical validation of our proposal is done emphasising on the performance.

II. Related work

Current image processing research mainly focuses on algorithm accuracy and infrastructure performance. Putting aside the first area, the analysis of huge datasets of multimedia content is a typical 'big data' problem that requires massive computational resources. On the one hand, buying that amount

[1]HTML5 standard specification (May 2011) http://www.w3.org/TR/html5/

of computational power would be incredibly expensive. On the other hand, main stakeholders of image processing solutions are the social media companies such as YouTube, Facebook or Twitter. Better and deeper tagging means increased relevance and more linked contents to engage user audience. These companies could avoid to invest in computing infrastructures, because they would have already available a huge network of user devices connected to their servers, removing the energy impact of high-performance computing systems.

Social computing research has been mainly focused on enabling technologies and specific applications guided by key directions including the emerging fields such as web science and dynamic network analysis, and the movement from social informatics to social intelligence, with an emphasis on managing and retrieving social knowledge. However, the future research lacks a core set of general scientific principles and a framework to guide social distributed processing. Putting aside the supporting technologies behind, Social Cloud research oversees a new type of computing paradigm, that inherits all the benefits provided by the conventional cloud but deployed over social community users who collectively construct a pool of resources to perform computational tasks.

SETI@home[2] is probably the main example of scientific experiment that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). Users participate by running a free program that downloads and analyses radio telescope data. But from the seed of this collaborative network model, numerous initiatives[3] related to unselfish research such as astronomy, climate, astrophysics, mathematics, genetics, molecular biology and cryptography have been raised where volunteers and donors share the computing time from personal devices.

Contrary to the previous initiatives our approach does not require downloading or installation of client software, atomic lightweight tasks are instead seamless embedded in a website and performed through JavaScript engine. So users do not need to be explicitly engaged to donate spare CPU cycles.

The most similar solution is brought by Plura Processing[4]. It provides a distributed computing solution using the web to power complex processing. The Plura affiliates earn revenue or drive donations by connecting computers to Plura network while their customers take benefit of one of the largest sources of computing power at affordable on-demand pricing. However, each approach focuses on different kind of problems and exploits different technologies. Moreover, our solution envisions a platform deployment on top of an already established network endorsed by a social media community.

This work [10] builds a distributed computing network based on a social graph of users you already trust assigning tasks to different nodes based on social acquaintance. It analyses several design options and trade-offs, such as scheduling algorithms, centralisation, and straggler handling. Our solution goes beyond providing the whole community resources to the service servers for specific multimedia processing tasks.

The great exponent of generic purpose massively collaborative computation with web technologies is MapReduce [2]. This framework for processing parallelisable problems was used by Google to completely generate Google's index of the World Wide Web. It defines a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. Available solutions forks [9] from open source Apache Hadoop[5] frameworks. This technology overcomes server-side tasks dispatching over a set of nodes. The highlights of this work [4] are ubiquitous nature of the image matching problems analysing some image processing algorithms specifically implemented for MapReduce technology. Another image processing projects hold by this technology are: HIPI[6] [11] that provides an API for performing image processing tasks in a distributed computing environment; and many more[7][8][9]. Current research goes further aggregating client-side nodes to work together with the server ones. In this direction, JSMapReduce[10] [8] is an implementation of MapReduce which exploits the computing power available in the computers of the users of a web platform by giving tasks to the JavaScript engines of any web browser. The atomised nature of multimedia processing tasks tackled in our work does not fit in with the MapReduce policy oriented to server side dispatching protocols.

Regarding JavaScript multimedia processing libraries there are an increasing number of solutions: IM.js[11] faces pixel image comparison; Resemble.js[12] deals with image analysis and comparison; Processing.js[13] addresses video and audio manipulation; Pixastic[14] manage pixel data access and manipulation of the image bringing color adjust, histogram, desaturate, edge detection, noise removal; jsfeat[15] provides image resample, equalise histogram, canny edges, fast corners feature detector, Lucas-Kanade optical flow, HAAR object detector, BBF object detector, linear Algebra module (Gaussian elimination solver, Cholesky solver, SVD decomposition, solver and pseudo-inverse, Eigen Vectors and Values) and Multiview module (Affine2D motion kernel, Homography2D motion kernel, RANSAC motion estimator, LMEDS motion estimator); ccv handles face detection and object detection[16]; nude.js[17] detects skin and body. All these features joined with JSMapReduce provide simpler and unique frontend for web developers that only must focus in JavaScript code writing and the results interpretation of multimedia analysis algorithms.

In terms of algorithm performance, historically JavaScript

[2]SETI@home Project (May 1999) http://setiathome.berkeley.edu/

[3]List of distributed computing projects (May 2013) http://en.wikipedia.org/wiki/List_of_distributed_computing_projects

[4]Plura Processing Service (2008) http://www.pluraprocessing.com/

[5]Apache Hadoop framework (2005) http://hadoop.apache.org/

[6]Hadoop Image Processing Interface (2012) http://hipi.cs.virginia.edu/

[7]EyeALike similarity across large datasets (2012) http://www.eyealike.com/

[8]Gum Gum in-image advertising platform (2012) http://gumgum.com/

[9]Kalooga discovery service for image galleries (2012) http://www.kalooga.com/

[10]JSMapReduce JS library (2013) http://jsmapreduce.com/

[11]IM.js Image comparison pixel by pixel (2013) http://tcorral.github.io/IM.js/

[12]Resemble.js Image analysis and comparison (2013) http://huddle.github.io/Resemble.js/

[13]Processing.js multimedia processing framework (2013) http://processingjs.org/

[14]Pixastic JavaScript Image Processing Library (2009) http://www.pixastic.com/lib/

[15]jsfeat Computer Vision library (2013) https://github.com/inspirit/jsfeat

[16]ccv Computer Vision library (2013) https://github.com/liuliu/ccv

[17]nude.js Nudity detection with JavaScript and HTMLCanvas (2010) http://www.patrick-wied.at/static/nudejs/

has been inefficient when compared to languages like C and C++, but Mozilla is recently bringing near-native application performance to the web Emscripten[18], an LLVM-to-JavaScript Compiler to translate C and C++ code into asm.js, a JavaScript subset for Firefox browser. This potential combined with the Mozilla's Web Workers solution to run scripts in background threads for Firefox browsers could provide the most powerful pure web platform for big data processing. However, it requires not only to translate the algorithms code through Emscripten but also it can only be performed on Firefox browsers. However, the browser ecosystem is heterogeneous (Chrome, Opera, Safari, IE) and the solution must fit in all of them.

Our goal is to advance current distributed computing approaches adding the social dimension and going into detail for multimedia processing purposes. To this end, our contribution in this paper is mainly twofold. First, we investigate the potential of the social distributed computing paradigm by introducing a design that construct distributing computing services. Second, we profile the potential of our approach to define the most suitable computationally-intensive problems regarding management overhead for task split and results splice.

III. System Architecture

The designed system architecture is based on a client-server architecture (Figure 1). It is completely oriented to have Web-based clients over HTML5, including specific JavaScript libraries. These specific purpose libraries are added inside a social media Web-based service transforming the client browser into a seamless image processing resource for the server. All the clients communicate with a main server interface through the Social Distributed Computing Manager (SDCM), which is in charge of attending them, balancing the load and distributing the requests through the different web servers available for the service.

Obviously, the different hardware and software stacks of the heterogeneous devices, span a wide spectrum of capabilities and performance. Emerging JavaScript libraries play a crucial role profiling the device capabilities to avoid impact on the user Quality of Service. Thus our approach uses minimal client resources to avoid affecting the user's experience. Thereby, on a first step, an initial process is launched in the client in order to evaluate the client device benchmark. Hence, the server decides to assign or not a processing task. In case of suitable devices, the rated analysis algorithm complexity and the image dimensions to be processed are matched according to this score. From that moment on, the data transfer begins between the social service client and the server delivering an image or a video frame and the order to process the data. This request includes the image processing algorithm or function to run for that image, the results format, etc.

On a second step, if the server has dispatched and set up a task, the client JavaScript engine will start the background work thread seamlessly while the user continues enjoying the social media service. Once the processing task is finished, the web browser will send the obtained result to the server

[18]Emscripten LLVM-to-JavaScript Compiler (2013) https://github.com/kripken/emscripten

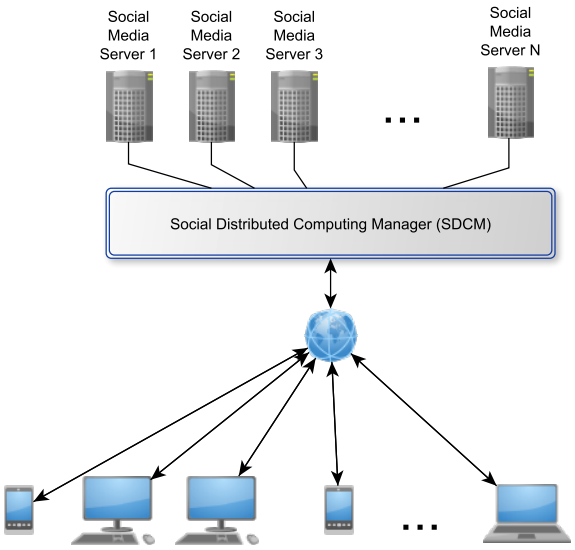

Fig. 1. System Architecture

triggering a new processing task request to the server in order to provide idle awareness.

These fragmented work tasks will be processed by the client along the user accesses a content from the social media service. So the server is responsible for managing the task distribution and monitor if the results are correctly received. In other case the server deals with uncompleted tasks re-sending them to another clients.

The following sections present a more detailed description of the different modules of the system architecture.

A. Client Architecture

The client is composed by several modules that are showed in Figure 2:

Communication Layer: This module is responsible for bridging the client-server communications, implementing the data and message protocols widely supported by HTML5 and JavaScript such as Websockets and AJAX. The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is the origin-based security model commonly used by web browsers. The protocol consists of an opening handshake followed by basic message framing, layered over TCP. The goal of this technology is to provide a mechanism for browser-based services that need two-way communication with servers that does not rely on opening multiple HTTP connections [5]. However, even if the Websocket implementation is in the roadmap of every Web Browser, nowadays there are restrictions to use it specially in mobile devices. A polling approach using Ajax technology, widely supported by any browser, is defined as an alternative to Websockets. Ajax (Asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With Ajax, web applications can send data to, and retrieve data from,

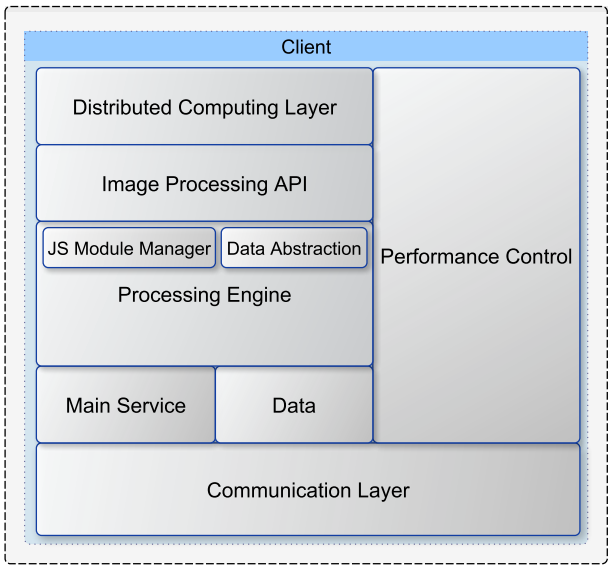

Fig. 2. Client Architecture

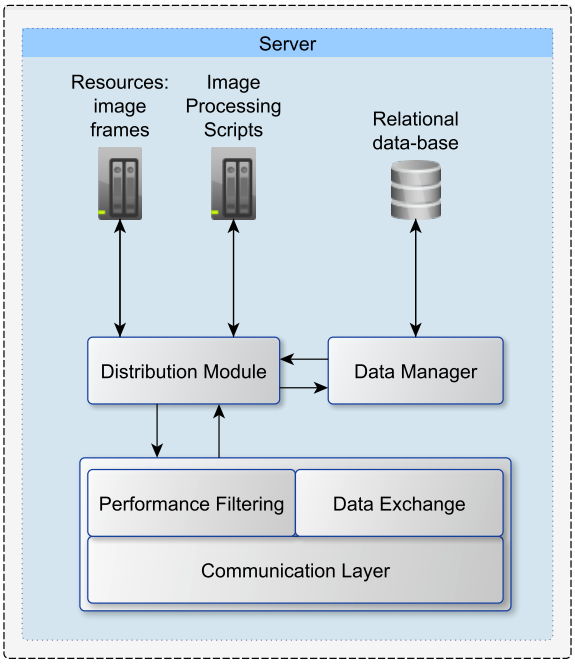

Fig. 3. Server Architecture

a server asynchronously keeping visual fluidity and behavior of the foreground Web application. Data can be retrieved using the XMLHttpRequest object [7].

Main Service: This module is the main Web application. It is a social media Web service where the user enjoys watching media content, interacting with friends, etc. The service keeps the highest priority to provide a good Quality of Experience (QoE) to the user interacting with social media contents. However, these applications do not usually require high computing performance in the client. The following modules add background processing tasks to the browser taking benefit of the spare CPU cycles of the connected users without interfering with the experience of the user with the foreground application. This way the service can enhance the tagging of the managed contents in order to empower content relevance and discover content relations boosted by the social community engine.

Performance Control: This module is a transversal task which is in charge of assessing client browser performance to distinguish client-side capabilities for extra job. If the client device is busy and cannot carry out additional work the other modules cease claiming incapable. In contrast, if the device can deal with concurrent tasks, all the modules in the client system start running and processing information.

Data: This module cope the transference of data between the client and the server. On the one hand, it will be very close to the Communication Layer to exchange data and messages with the server and on the other hand it will create and maintain the HTML5 Web Storage facilities for the received data. Once the local processing of the video frame is done, it will be also the responsible to adequate the results and send it to the Communication Layer to be transferred to the server. It is important to highlight that our solution is entirely run in memory by the web browser and do not access or record any client information.

Processing Engine: This is the core module of the client since it is the responsible to inject the downloaded image processing script, and run it with the downloaded frame. It also formats the obtained results to be sent to the server. It is based on two sub-modules:

- **JS Module Manager**: This module is in charge of the JavaScript image processing scripts injection and management. While the user is interacting with the social service, this module is the responsible of preparing and loading the scripts to be executed seamlessly without any impact in the user experience.
- **Data Abstraction**: This module adapts and formats the results of the execution of the script to be ready to send them to the server. The server will send information about how to prepare the output data so this module will format it to the adequate structure.

Image Processing API: This module benefits from the approach proposed in this paper by accessing to a simple Image Processing API in the server. Based on JavaScript code, it hides algorithms complexity and connects different methods to obtain specific features and they will be executed on a batch way through the different client browsers.

Distributed Computing Layer: This module is the orchestrator delegate running background tasks at end devices such as computers, tablets, smartphones, TVs, etc. by coordinating all the client-side modules to execute both the main social application and the other background processes concurrently and in a seamless way for the final user.

B. Server Architecture

Figure 3 depicts the different modules of the server:

Communication Layer: This module manages the communications between the server and the clients. As it is

mentioned in the client side, this layer is deployed on top of Websockets and AJAX communication protocols to exchange data and messages.

Performance Filtering: This block profiles the capabilities of each client in order to assess a specific task suitability. Each client will report its processing capability through a test done by the Performance Control to the server. According to the achieved score the server matches the different tasks depending their processing needs.

Data Exchange: It deals with data exchange (e.g. a image frame, scripts, etc.) through the communication layer between the client and the server.

Distribution Module: This is the core tasks dispatcher through the clients allocating the task, the image frame, the script and the needed information (how to format the answer, etc.). This module coordinates the communication layer, data exchange and performance filtering, but also communicates with the Data Manager and with the Resources (Image frames and image processing scripts).

Data Manager: This module interfaces the relational data-base which contains the information of the global tasks that should be done, in terms of what image processing scripts need to be applied to each image. This way, the Distribution Module will be able to assign a specific task relating a frame and a script following the priorities specified in the Data Manager. The relational data-base contains also information about the processing weight of the different scripts to fix to the performance capability of the client. This module is also responsible of gathering all the processing results and store them in the data-base.

IV. Use cases

The architecture described in the previous section is quite generic purpose oriented. In order to land the concept behind to specific social computing needs, we describe here some concrete image processing topics that fit on our approach while gather the main interests for social services are:

- Face detection and recognition for face similarity and recognition across a dataset.
- Logo recognition for image content based advertising.
- Character recognition for image content based auto-tagging for social media.
- Near similarity detection for image based video copy-right protection.

The aim for the different images is always to retrieve results independent from other contents, so the temporal dimension and correlation is completely removed from the distributed processing platform. Our perspective restrains problem complexity by removing the need of a subsequent multiple results consolidation. This way the social server would not need high demanding postprocessing to join client-side results that would mean processing overhead at the server-side.

Our target scenario is YouTube like social media services. This context provides a perfect environment for our system thanks to:

- Longer and continuous user sessions for videos. In contrast to text based social communities, it provides high availability of the resources with a more stable infrastructure in terms of elasticity.
- Not multi-task activities while watching videos. Users underuse their devices where background tasks can execute without interfering with the user experience.
- Residual overhead. The overhead of the data to be analysed is residual compared with the mainstream content.
- Continuous communication channel. The high transfer rate of a streaming server bridges a stable communication with lower latency.

It is very important to highlight that our solution manager anonymises the content to be analysed and the assigned tasks are entirely run in memory by the web browser and do not access or record any client information.

V. Performance Modeling

The performance of the proposed approach can be theoretically analysed by following the PRAM model [6]. According to this model, each processor has access to the shared memory and can also do computation on a local memory. However, this model has some drawbacks as it assumes that all processors work synchronously and that interprocessor communication is free. To overcome this limitations Culler et al. propose the LogP model [3]. The model is based on the following parameters:

- L: an upper bound on the latency or delay in communicating messages from the source to the target module.
- o: the overhead defined as the time that a processor is engaged in the transmission/reception of each message and cannot perform other operations.
- g: the gap or the minimum time interval between consecutive message transmissions/receptions at a processor.
- P: the number of processor/memory modules.

However, this model does not fit distinctly with the approach proposed in this paper as is it intended to deal with complex network topologies and simple processing units. Moreover, the LogP model is accurate only at the very low level hardware stack and not for practical communication systems with layers of protocols (e.g. TCP/IP).

In our case, the network topology is a star where processing nodes can dynamically be added or removed. It implies that:

- One processor acts as the central processor.
- Every other processor has a communication link with this central processor.
- Congestion may happen at the central processor (dispatcher).
- Remote processors can deal concurrently with messaging and processing tasks.

As a generalisation of PRAM and removing the overhead o of LogP, Valiant [14] proposes the Bulk Synchronous Parallel (BSP) model. In this model, each processor can follow different threads of computation and tasks are organised as *supersteps*. A supersteps includes three stages.

1) Concurrent Computation (asynchronous)
2) Communication
3) Barrier Synchronisation

The cost of a BSP algorithm is the total cost of the sum of all supersteps, and the cost of each superstep is given by Equation 3:

$$C_{superstep} = \max_{i=1}^{p}(w_i) + \max_{i=1}^{p}(h_i g) + l \quad (1)$$

$$C_T = \sum_{s=1}^{S} W_s + g \sum_{s=1}^{S} H_s + Sl \quad (2)$$

$$W = \max_{i=1}^{p}(w_i), H = \max_{i=1}^{p}(h_i) \quad (3)$$

where:

- p = number of processors
- S = number of supersteps
- l = synchronisation periodicity
- g = communication cost
- h = maximum number of incoming or outgoing messages per processor
- w = computation time

In our case, the communication channel is granted through the already established video streaming flows. It ensures good communication performance for relatively small data communications. When the required bandwidth is comparable to the video transmission bitrate, it may alter the QoS and g would increase.

The synchronisation periodicity could be removed for many use cases where the Social Distributed Computing Manager (SDCM) creates independent tasks. For example, to process all the instances of a database can be considered as a big single superstep that synchronises when all the databased is processed. In this case, we could model the computational cost as:

$$C_T = W_s + gH_s + l = \max_{i=1}^{p}(w_i) + \max_{i=1}^{p}(h_i g) + l \quad (4)$$

However this assumption does not allow the fact that fastest processors can start a new task once they finish the previous one. To do that, we define the model from the process perspective. According to this view, we can consider the total cost as the time needed by the network of resources to process each independent task by means of average communication and processing costs (Equation 5).

$$C_T = \frac{\hat{W}_s}{p_u} + \frac{\hat{g}\hat{H}_s}{p_t} + Sl \quad (5)$$

TABLE I. ESTIMATED PROCESSING AND COMMUNICATION PROPERTIES FOR DIFFERENT TYPES OF DEVICES.

ID	Device	Connectivity	Average Bandwidth	Average GFLOPS
(m)	Mobile phone	UMTS	3Mbit/s	0.05
(t)	Tablet	Wifi	8Mbit/s	0.08 [19]
(p)	PC	DSL	20Mbit/s	2.5
(s)	Server	SATA	6Gbit/s	p_sx82.8 [20]

In order to estimate the benefit of the presented approach compared with local processing, we can apply the same cost model (Equation 5) to a local multicore processor. In this case, the data access will be much faster but the number of processors might be much lower.

In order to give a comparative estimation, we will consider 4 types of devices with different connectivity and processing power (Table I).

H_s depends on the algorithm implementation, not on the device or communication infrastructures. So it is closely related to the defined use cases. Tasks are considered as batch processes, thus, it will not be contemplated in the cost comparison. The Sl factor will be reflected as a internal management factor that will require some computational power at the server side to be consolidated. However, it will not incur in extra time cost per operation as local parallel processing at the client side. Keeping in mind that we are in a streaming session context, initial delays can be disregarded as the data needed to communicate the server and the remote processors are directly added to the ongoing stream. The following equations depict a breakdown of the cost estimation under the conditions specified by Table I. f_{xy} represent the utilisation factor of the resources that will be available to process the tasks. In order to avoid any annoyance in the user experience, f will be set to 0.15 both for bandwidth and for processing power. The cost to establish a new thread and its management its denoted by $\hat{m}$. It is a fixed estimated value and mainly considers the cost on the server side.

Equation 6 represents the sum of the partial costs of the different type of end devices (mobile, tablet and PC) while equation 7 models the computational cost of each individual group of devices with similar characteristics.

$$C_T = \left(\sum_{i}^{n} \frac{1}{C_i} \right)^{-1} \quad (6)$$

$$C_i = \frac{\sum W_i}{f_{pi} \cdot F_i \cdot p_i} + \frac{\sum g_i}{f_{bi} \cdot \hat{b_i} \cdot p_i} + \hat{m} \cdot p_i \quad (7)$$

Following the same model, the cost of a multicore server with a single internal shared memory is:

$$C_s = \frac{W}{f_{ps} \cdot F_s \cdot p_s} + \frac{g}{f_{bs} \cdot \frac{\hat{b}_s}{p_s}} + \hat{m} \cdot p_s \quad (8)$$

In order to compare a distributed computing approach with a dedicated local server, different C_T and C_s have been

[19] http://www.tomshardware.com/reviews/fx-8150-zambezi-bulldozer-990fx,3043-14.html

[20] http://www.legitreviews.com/article/1988/2/

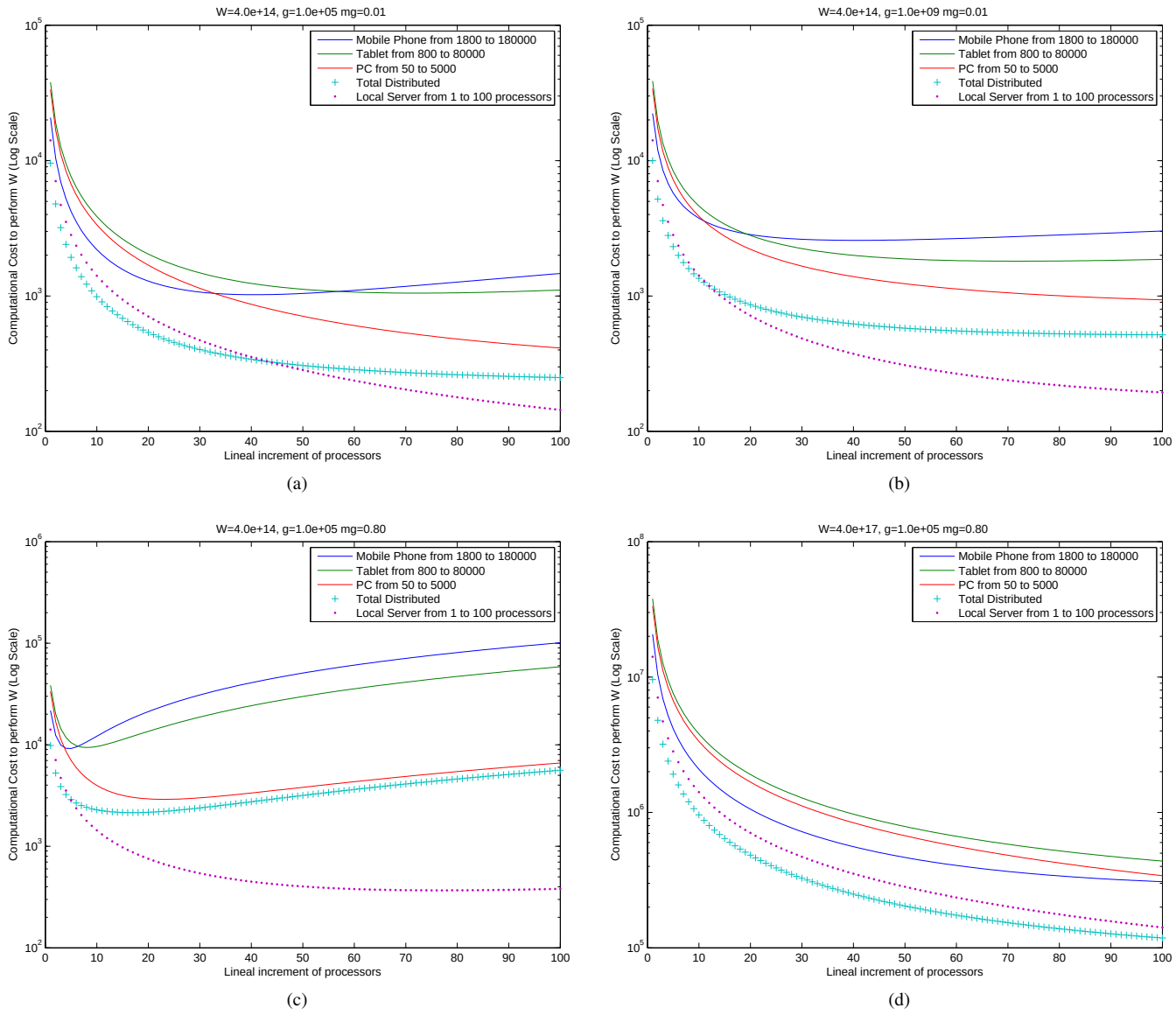

Fig. 4. Computational cost estimation under different sizes of work load (W) communication cost (g) and thread management cost ($\hat{m}$) .

calculated. Figure 4 shows different cases of performance behaviour for several values of model parameters. A lineal increment of processors is compared for different sizes of W g and $\hat{m}$. As it can be observed, the maximum benefit of our proposed social computing network is obtained for those use cases with higher computational load. The communication costs can be the main bottlenecks unless bandwidth conditions or utilisation factor are increased. Figure 4c shows that the efficient management of all the created threads becomes a critical factor as well.

VI. Validation

In order to provide some reference experimental values for the parameters involved in the theoretical performance modeling in a specific context, we have compared the performance of a distributed computing architecture with a dedicated local server approach. We have implemented the same testbed for both scenarios: processing of an image using JSFeat libraries in distributed devices and the OpenCV libraries for the standalone local server execution. The testing image resolution was 1200x1600 and compressed in JPEG. The processing method consisted on applying a grayscale converter and a Canny detector due to the necessity of finding reciprocal functions in both libraries to provide a homogeneous framework.

As validation parameters two elapsed times have been measured, the global processing time, which includes streaming of the image to be analysed and the actual algorithm processing, and the computation time that just includes the algorithm runtime. We have calculated the mean and the variance of 100 task samples. For distributed approach, client-side devices we used a computer, a tablet and a smartphone. As can be seen in Table (II), most of the global processing time is used for algorithm computing, justifying the capacity of this kind of architectures to minimise data streaming time consumption in relation to global processing time.

In conclusion, obtained results are coherent with the assumptions taken in the theoretical model and the relations of the different parameters involved. Even more, comparison of these results with the outcome of a standalone server execution (see Table III) underline the opportunity of using web based distributed platforms as a solution for large scale processing.

TABLE II. MEAN AND VARIANCE TIME CONSUMPTION FOR CLIENT-SIDE DEVICES IN A DISTRIBUTED ARCHITECTURE.

Architecture Client-side (JavaScript)	**Global Process** μ (ms.)	**Computation** μ (ms.)	**Global Process** σ^2	**Computation** σ^2
CPU 3 dual core	258	240	42	18
Asus Eepad transformer	2710	1744	97	73
Nexus 4	1694	1537	156	130

TABLE III. MEAN AND VARIANCE TIME CONSUMPTION FOR LOCAL STAND-ALONE PROCESSING.

Architecture Server-side (OpenCV)	**Global Process** μ (ms.)	**Computation** μ (ms.)	**Global Process** σ^2	**Computation** σ^2
CPU 2.3 dual core	277	101	24	37

VII. CONCLUSIONS

The social distributed computing platform leverages the computing power of potentially thousands of devices to come up with a cheap, flexible solution for social media analysis. Users contribute a fraction of their computing time. Despite the lower efficiency, compared to stand-alone and Grid solutions, the barrier to entry is low thanks to the potential larger cluster to reach an astounding number of machines for social services. This context enables us to solve big data problems previously unachievable. To be more specific the main stakeholders of image processing, the social services, can apply background work for batch analysis over the whole social media dataset.

However, the infrastructure deployment must keep in mind requirements and constraints of social services. On the one hand, there is a major requirement attached to keep the Quality of Service means to avoid draining the users' bandwidth and processing power. On the other hand, the solution must deal with resources availability due to the high elasticity related to the spontaneous presence nature of users. These conditions settle the framework of the image processing work which implies the necessity of atomic and lightweight image processing tasks.

The proposed system spreads over a client-server architecture working totally over Web-based clients. Emerging JavaScript technology enables to manage and run seamless background tasks while the user interacts and enjoys a social media service without affecting to the Quality of Experience. The approach provides a community of devices as a resource for computing tasks and there is no need to install or develop client applications but adding a distributed computing layer to the HTML-based main service.

Regarding a performance modeling over an adapted BSP model, the maximum benefit of the proposed social distributed computing platform is obtained for use cases with high computational load. This fits with the social media service providers needs, that often require complex image analysis processes for a huge volume of data. The target scenario is a social media content service (like YouTube) providing a favorable scenario. It brings beneficial fetures such as continuous communication channel, residual overhead compared with the video itself, not multi-task user activity, device underuse and longer sessions that foster high availability of resources for the distributed computing tasks.

In addition, validation experiments back up the theoretical performance modeling and remark the opportunity of using the Web-based social distributed computing solution for large scale processing in comparison with the outcome of a server grid approach.

To conclude, the proposed system deploys a promising solution to cope with the big data problem that the social media service providers deal with, through a social distributed computing platform. Service providers benefit from the huge processing capacity of the social community adding to the main service, via web browser, seamless background processing tasks for image analysis.

REFERENCES

[1] M. Anttonen, A. Salminen, T. Mikkonen, and A. Taivalsaari. Transforming the web into a real application platform: new technologies, emerging trends and missing pieces. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 800–807. ACM, 2011.

[2] F. Catak and M. Balaban. Cloudsvm: Training an svm classifier in cloud computing systems. 7719:57–68, 2013.

[3] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: towards a realistic model of parallel computation. *SIGPLAN Not.*, 28(7):1–12, July 1993.

[4] G. De Francisci Morales, A. Gionis, and M. Sozio. Social content matching in mapreduce. *Proc. VLDB Endow.*, 4(7):460–469, Apr. 2011.

[5] M. A. Fette, I. The websocket protocol. 2011.

[6] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 114–118. ACM, 1978.

[7] J. J. Garrett. Ajax: A new approach to web applications. 2005.

[8] P. Langhans, C. Wieser, and F. Bry. Crowdsourcing mapreduce: Jsmapreduce. pages 253–256, 2013.

[9] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang. Moon: Mapreduce on opportunistic environments. pages 95–106, 2010.

[10] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim. Socialcloud: Using social networks for building distributed computing services. *CoRR*, abs/1112.2254, 2011.

[11] C. Sweeney. Hipi: A hadoop image processing interface for image-based mapreduce tasks. *B.S. Thesis. University of Virginia, Department of Computer Science 2011.*, 2011.

[12] A. Taivalsaari and T. Mikkonen. The web as an application platform: The saga continues. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 170–174. IEEE, 2011.

[13] A. Taivalsaari, T. Mikkonen, M. Anttonen, and A. Salminen. The death of binary software: End user software moves to the web. In *Creating, Connecting and Collaborating through Computing (C5), 2011 Ninth International Conference on*, pages 17–23. IEEE, 2011.

[14] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[15] M. Zorrilla, A. Martin, J. Sanchez, I. Tamayo, and I. Olaizola. Html5-based system for interoperable 3d digital home applications. *Multimedia Tools and Applications*, pages 1–21, 2013.