Social-Networking@Edge

Miguel Borges

Instituto Superior Técnico - University of Lisbon
miguel.a.borges@tecnico.ulisboa.pt

Abstract. Online Social Networks (OSNs) have become very popular nowadays. These organizations have millions of users and collect petabytes of data from their interactions. Having the capability of understanding that data is crucial for their success, as it enables them to infer trends, recommend friends and present content that is suited for users.
Some tools have been developed that enable the processing of such amount of data. One of the most used solutions is *MapReduce framework*, because it provides a simple programing model that eases the development of scalable distributed application and automatically handles the problems of replication, fault-tolerance and load balancing. However, using these tools requires considerable computing resources that are normally supported by massive centralized infrastructures. The creation and maintenance of such infrastructures leads to high ownership and environmental costs. At the same time, the machines used by the users of OSNs have an increasing computing capacity that is normally not being fully used.
This thesis proposes solution that makes use of users' computational resources, while they are using an OSN's website, for executing computation that would normally be done at data-centers. *Social-Networking@Edge* is a MapReduce framework that makes use of users' spare-cycles for executing tasks on their Web Browser while handling the problems of data distribution, fault tolerance and load balancing. By making use of such solution, OSNs would be able to support the same data processing requirements in a much more cost-effective manner and producing a smaller ecological footprint.

1 Introduction

Nowadays, OSNs are very popular and have millions of users, specially due to the success of Web Sites like Facebook, Twitter and Google+. Interactions provided by users on their systems produce enormous amounts of data, leading to data-warehouses with petabytes of data[11]. Having the capability to analyze that data is crucial for their success, because it enables them to infer trends, recommend friends and present insights to advertisers.

Some tools have been developed that enable the processing of such quantity of data, among which, the most popular is MapReduce[5] as it presents a simple programming model that eases the development of scalable distributed applications for data processing by automatically handling common problems for distributed application such as replication, fault-tolerance and scheduling. In order to use these tools it is required large amounts of processing power and storage capacity, leading to the creation and maintenance of large data-centers that have an high cost and ecological footprint. As the number of users increases, these data-centers need to grow as well.

The computing power made available at commodity machines owned by normal people has been increasing in terms of CPU and memory. However, most of the time, those resources are not being fully used. Systems that make use of processing capabilities donated by users such as BOINC[2] and SETI@home[3] have had great success, enabling the development of research programs in a very cost-effective manner but they are focused on specific projects and do not cope with OSN's requirements.

There have been some approaches that try to solve the scalability and data privacy problems of centralized OSNs by creating totally distributed OSNs, making use of their users' resources to allocate the data and computation needed to enable the OSN. However, building the same functionalities on a totally distributed environment while dealing with privacy issues and high rate of failure from users'

computers is a much difficult task, leading to OSNs with less functionalities and worse usability. At the same time, it is hard to convince users to migrate to a new OSN and donate their resources to the community. Other less radical approaches have also been tried but they are mainly focused on saving the OSN's data on their users and do not provide solutions for handling the processing of large amounts of data.

We propose a system that makes use of users' idle resources, while they are using an OSN's website, for executing computation that would normally be done at data-centers. With Social-Networking@Edge it is possible to execute MapReduce tasks on the users' browser, allowing OSNs to offload some of their processing requirements. By using this approach it is possible to increase the OSN's scalability but, at the same time, reducing their costs, ecological footprint and maintain the benefits of having a centralized architecture for other tasks.

In this paper we will start by describing the architecture of the proposed solution (Section 2), afterwards on Section 5 we show the results of evaluating our system and its discussion. Related work is presented on section 7 and at last a summary of the document is presented in Section 8.

2 Architecture

In this section we will present the overall architecture for Social-Networking@Edge, each of its main components and how they cooperate. We will start by presenting an overview of the system (section 2.1). Afterwards we detail the distributed architecture (section 2.2), and finally we present the approach used by the system to tolerate faults (section 2.3).

2.1 Architecture Overview

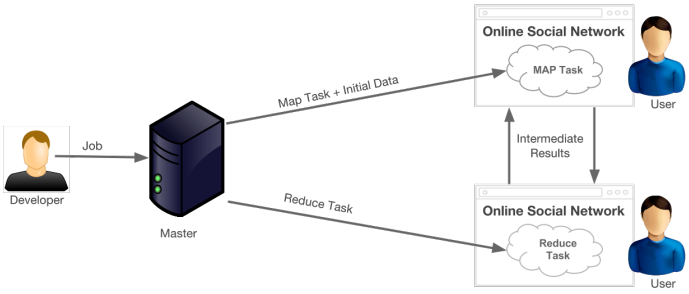


Fig. 1: Social-Networking@Edge Overview

Social-Networking@Edge is a MapReduce framework, that runs tasks at the browser of OSN's users. A normal use case of the system can be seen at figure 1. Users access the OSN's website as usual. While they do that, a client's software is executed in the background by their browser. Developers are able to submit MapReduce Jobs to the Social-Networking@Edge server that are then executed by the available users. A job's specification contains several parameter such as the map and reduce code to be executed and the number of mappers and reducers that should be used to execute the job.

The server will schedule Map or Reduce tasks to the available users' browsers, taking into consideration the resources on each machine. In order to improve the scalability of the system, user nodes can also exchange intermediate results of Map tasks with other nodes, enabling data processing to proceed without overloading the server. The system ensures the execution of the jobs submitted, taking

care of common problems of distributed applications such as parallelization, load-distribution and fault-tolerance. Building such a system brings several architectural challenges that will be addressed on the following sections.

2.2 Distributed Architecture

The solution presented here allows the distribution of MapReduce tasks among a large number of nodes. As we increase the number of nodes, or the capacity of each node, the system is able to improve its performance by raising the level of parallelization.

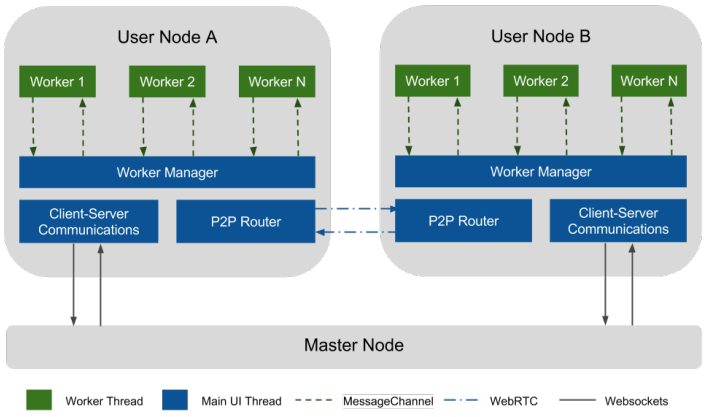


Fig. 2: Client's Components

The system has two types of nodes: one master node and multiple user nodes. The master node's main responsibilities are: a) to keep record of the jobs' specifications submitted by developers; b) know the availability of the user nodes at a given point in time and their computing resources; c) distributing MapReduce tasks among the available user nodes according to the job's specification; d) and saving the results of the MapReduce jobs, so that the developers can access them after their completion. User nodes are responsible for the execution of MapReduce tasks assigned by the master node.

Figure 2 presents a view of the main distributed components that execute on the system and the communication channels used by them. Each user node has a Worker Manager component and several Workers. The Worker Manager creates Workers, being responsible for the communications with the master node and with other user nodes. The number of Workers that are created on each user node depends on the available resources of the user's host machine and the level of participation with which the user is willing to contribute to the system. Workers are responsible for executing the tasks assigned by the master node and delivering the results when the execution ends. Multiple Workers can run in parallel and communicate with the Worker Manager by means of messages exchanged using a bidirectional message channel. The Worker Manager may send and receive information from the server or from other user nodes. Each message is identified with the sender and receiver so that the Worker Managers can dispatch the messages to the appropriate workers.

Communications between the master and user nodes are done by using a Socket based communication that is kept alive while the user is using the system. This way, the master node can take the initiative of sending messages to the available users. This is an important property as it allows the server to push tasks to the available users without needing to wait for the users nodes to request them. The connection with the server is managed at the level of the Worker Manager, meaning that when a user node has a large number of workers, there will be only one connection with the server. This will ensure that the

number of connections that must be managed by the master node is kept at low levels even when a large number of workers is using the system.

Workers are organized on a P2P network where each worker is a peer and the connections are maintained at the level of the Worker Manager. Similarly, there must be only one connection between user nodes, even when there are several workers on the same user node. By having a P2P router at this level, the system can also optimize the case where messages are exchanged between workers running on the same user node because they can communicate directly without using the network.

All the messages created by workers identify the sender and receiver and are then sent using the message channel. The Worker Manager intercepts the messages and delivers them using the appropriate channel. Each worker is identified by a unique pair (UserID, WorkerID), that is assigned by the master node when a user connects. The creation of P2P connections is lazy, meaning that they are only created when the first message between the two nodes is exchanged. This ensures that there are only connections between user nodes that need to communicate.

2.3 Fault Tolerance

Systems that deal with a large number of distributed nodes where each node has an high rate of failure must provide strategies to tolerate some of those failures. In our case, user nodes can fail frequently and our solution must provide ways for the system to detect and recover from failures. We propose a solution that detects worker's failures and reschedules their tasks to other workers, enabling partial re-execution of tasks when a failure occurs by using a checkpoints mechanism.

During the execution of jobs, workers save checkpoints of the data processed at that moment. Those checkpoints are replicated and saved by other user nodes available at the moment. When a failure occurs, the master node assigns the task to other of the available worker that can continue the task's execution from the last checkpoint. Checkpoints are made after the completion of each intermediate task and before the notification of the server. This ensures that enough replicas exist before the task being considered complete. The number of checkpoints saved has impact on the quantity of data transfered between the workers and on the time to complete the tasks. For that reason, it is important that developers can configure the level of replication wanted for each job. The decision of which workers should be used as replicas is done by the master node that ensures that a replica is always placed on a different user node.

3 Implementation

Previous section explained the main architectural decisions in such a way that the solution may be applied using different technologies. In this chapter we detail the main technologies and technical decisions that have been made for the implementation of a prototype that follows the architectural decisions explained previously.

3.1 Parallel execution of Workers

Nowadays, almost all Web Browsers have support for the execution of Javascript code and it is used for building most of the existing Web Sites. However, up until recently it was not possible to do intensive computing operations on a Web Browser without blocking the user interface nor it was possible to have P2P communications between Web Browsers. With the introduction of technologies such as WebRTC and WebWorkers, that are now supported by most of the Web Browsers, we have been able to provide parallel execution of tasks by each user node.

On figure 2 we can see that each computing unit runs on a WebWorker and the Worker Manager executes on the main thread. This enables our prototype to run computing intensive tasks on a Web Browser without blocking its main thread. Several Web Workers can be executed at the same time on different threads, allowing the execution of different MapReduce tasks in parallel on the same browser.

The number of Worker being executed is controlled by the Worker Manager and is given as parameter on its initialization. If no value is given, then the system creates as many workers as the number of cores on the user's host. There are some limitation on the API that can be accessed by Web Workers. For example, the context has no access to WebRTC API. Instead, all the P2P communication must then be handled by the main thread.

4 P2P Communications between Workers

Workers need to communicate with each other in order to exchange information and data during the shuffle phase. They also need to save checkpoints at certain points of the job's execution. A naive implementation would use the master node as a broker for delivering those messages. However, such implementation would create a bottleneck on the server that would limit the performance and scalability of the solution. A better alternative is to enable P2P communication where the workers communicate directly with each other. In order to implement this feature we used WebRTC [9]. This technology enables P2P communication between Web Browsers even when the peers are behind NAT routers by using Session Traversal Utilities for NAT (STUN) [7] to discover the real public address of each node.

When a worker sends a P2P message, it first creates the content of the message and specifies a pair (UserID, WorkerId) that identifies the worker to which the message should be delivered. This message is then passed to the main thread that may connect or reuse existing connection with the user identified by UserID. The message is sent using the WebRTC channel established between the two nodes. For the case where messages are to be sent to a worker of the same user sending the message, the Worker Manager simply delivers the message to the appropriate worker without needing to use the network.

By default, the communication done between workers and the main thread is done by value. For small messages this has no impact, but for larger messages the browser can block, resulting in the Web Page being terminated because the maximum buffer size of the channel is reached. Fortunately the protocol supports passing messages by reference when the type of the object is a native array. However, when we send messages by reference, the sender of that message can no longer use the original object. Our solution to overcome the current protocol's limitations was to use the default implementation (copying messages) for small control messages because it has a low overhead. For all the messages that contain data, such as shuffle messages or checkpoints' replicas, the worker copies the message to a native array and then sends it by value. With this solution we have a low overhead for small messages and the overhead of creating a copy of the data is on the worker side and can be done in parallel without disrupting the normal execution of main Javascript thread.

5 Evaluation

On our current solution, we have implemented user nodes and back-end, enabling us to execute map-reduce jobs on web-browsers. Intermediate data from map tasks is exchanged directly between user nodes using P2P communications. To evaluate our solution we experimented the platform with popular benchmark applications taking into account, the execution times of one job at a time with different cluster's configurations and different number of available workers. The tests make use of clients' machines to execute workloads expressed as map-reduce jobs, resorting to a typical example as benchmark: *wordcount*, with an input of 1 Gigabyte. The tests were run using virtual machines from DigitalOcean [1]. The hardware configuration used for running the user nodes is presented on table 1. For all the tests presented here, we used the configuration A to deploy the master node. The configuration used for the user nodes varied and is described bellow. The deployment process of a user node consists on simply opening a Web Browser on a URL being hosted by the master node. In order to deploy the master node it is needed to:

[1] https://www.digitalocean.com/

– start the execution of a signaling server for enabling WebRTC connections and;
– execute a *Java* server that manages the execution of jobs.

Configuration	Memory	CPU (Number of Cores)	Disk Size
A	2 Gigabytes	2	40 Gigabytes
B	4 Gigabytes	2	60 Gigabytes
C	8 Gigabytes	4	80 Gigabytes

Table 1: Virtual Machine's configurations

5.1 System performance for an increasing number of participants

One of the goals for the system was that it should be able to increase its overall performance simply by adding more participants. This means that when the number of available workers increases, the system should be able to use those resources and, by doing that, the time needed to complete jobs should decrease.

In order to assess the performance of the system for an increasing amount of resources, we tested the system with WordCount benchmark with a fixed input size of 1 gigabyte and an increasing number of participants. The virtual machines' configuration used for all the user nodes was the type A from table 1. This means that each of the user nodes has two cores and so they contributed with two workers.

We started by having 10 machines and 20 workers and then we increased that number until we reached a total of 50 machines and 100 workers. The jobs executed used 1 reducer and 1 replica. The number of mappers was increased at the same rate at which the number of available workers increased. Figure 3 presents the execution times of WordCount jobs while increasing the number of available workers and mappers and how it compares with the ideal speedup relative to the first execution.

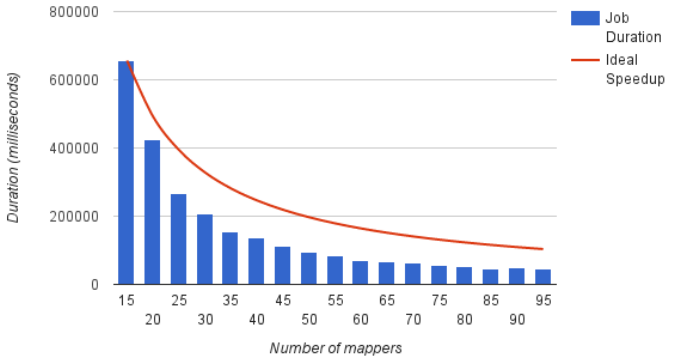


Fig. 3: M/R-WordCount times (ms) per phase with increasing number of workers.

As expected, the job's execution time decreases as the number of available workers increases. The rate at which the execution time decreases is lower than the ideal speedup. This means that system is able to scale vertically as the number of workers increases.

Notice that as the number of mappers gets closer to the maximum number of available workers, the relative time improvements start decreasing. There are two main reasons for this to happen. First, the overheads of communication between the workers become larger. Second, at the beginning the scheduler

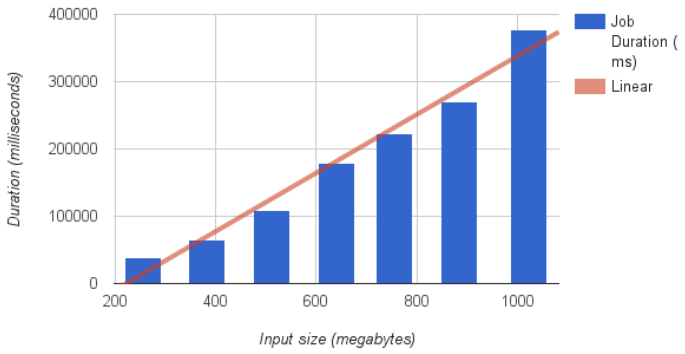


Fig. 4: M/R-WordCount times (ms) with increasing chunk size.

can distribute the system load across a large number of idle workers. However, when we get to a number of mappers that is equal to the number of idle workers, all the user nodes have to support more load, meaning that the resources' usage on those nodes is increased. In section 6 we analyze the resources usage in more detail.

5.2 System performance for an increasing input size

The main goal of this test is to evaluate the system's performance and its capabilities as we increase the size of the job's input that has to be processed. Increasing the input size of the job has the reverse effect of the previous test. As the initial input size increases, the computational requirements of each participant also increase. The input chunk that each worker has do download and process is larger and the amount of data exchanged between the participants increases. To perform these evaluations we used a fixed number of workers and increased the input size. The benchmark used was WordCount with 20 mappers, 5 reducers and replication factor of 1. In order to run the test we deployed 20 virtual machines of type B (Table 1), with a total of 40 available workers. We run several tests, starting by using an input of 256 megabytes and progressively increased it by 128 megabytes until we reached 1 gigabyte.

Figure 4 shows the jobs' duration as we increase the size of the input file. As can be seen, the jobs' execution times increase linearly with the input size.

Note that, for larger inputs the time to complete the jobs is above the linear level. This happens because we approached the maximum usage of the participants' resources. Currently the system has some limitations on the quantity of data that can be processed. These limitations are related with two design decisions:

1. All data is kept in memory while the workers are processing a given job. This means that the initial input chunk, replicas received from other workers, the results of processing and sorting data are saved on the worker's main memory. When we get to the point where the level of occupied memory exceeds the browser's capacity, the system starts degrading in terms of performance. Google Chrome has a memory limit of 1 Gigabyte. When that level is reached, the browser crashes and the user node fails.
2. For a given job, a worker can only process one map task. This means that the only way to process larger files is to increase the number of workers that participate on that job.

6 Resource Usage and Scheduling

User nodes donate a given part of their resources for the execution of MapReduce tasks on their Web Browser. The usage of resources on the participants is decided by the scheduler component. With this test we wanted to detect how well the available resources were used and the main overheads. For this test we executed WordCount benchmark with 20 participants of type B (table 1) that originate the participation of 40 workers. The job used 30 mappers, 5 reducers and replication factor of 1. During the job execution we used *dstat* [1] to scan the resources usage in terms of memory, CPU and network on the master and user nodes.

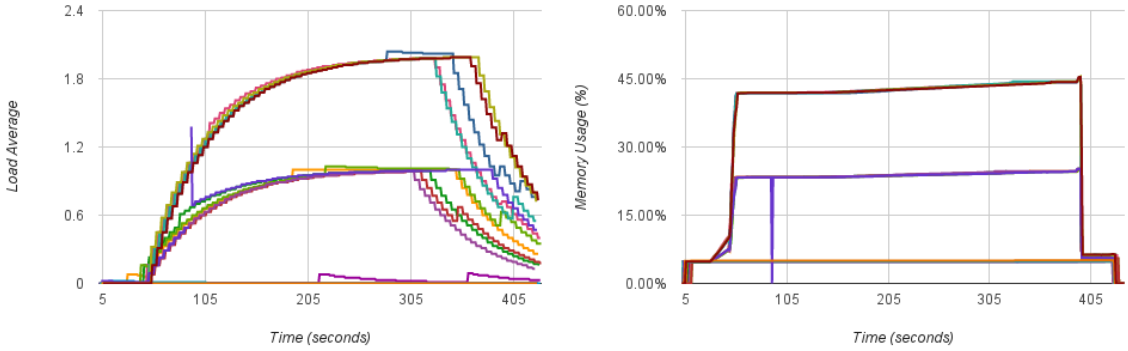
Figure 5a shows the resources' usage by the user nodes available during the job's execution. It is possible to see that the load increased exponentially after the start of the job and then stabilized at 90% for the majority of nodes and at 190% for a minority of nodes. Also, for a small number of nodes we can see that the average load was very close to 0%. The reason why this happens has to do with the level of resources donated by each participant and the scheduling decisions. On this test, each participant had two workers. This is why some of the nodes get an higher load than others. The scheduler distributes the job's working units by the available workers in a random manner. The users that donate more workers have an higher probability of getting more tasks assigned to them. On this case we can see that the scheduler does an acceptable distribution of load across the available resources and only a small number of nodes have no assigned tasks to perform.

On figure 5b we can see that the system uses 20% of memory when they have only one working unit assigned and approximately 45% when all the worker slots are assigned to the users. It is easy to assert that the memory usage is higher than expected if we consider that the total job input has one gigabyte and it was evenly distributed by 20 workers. The high usage of memory has to do with the design decision of keeping all the data in memory during the execution of a given job. After the execution of the job we can observe that the levels of memory drop to approximately 5% that is close to the normal usage while the machine is idle. When the job execution finishes, the user node kept executing the worker manager component. It is possible to notice that the resources' usage required by this component are very low.

The CPU usage of the user nodes is presented on figure 5c and they had a behavior similar to the memory usage. From this figure we can notice that some workers reduce their CPU usage before others. This happens because some of them finish their working unit before others. After the majority of nodes reduced their CPU usage to 0%, there is a small pause and then we can see that a small number of nodes had an high usage. Those nodes are the ones with reduce tasks assigned. The decrease of CPU between the map and reduce phase is due to the exchange of data with other nodes.

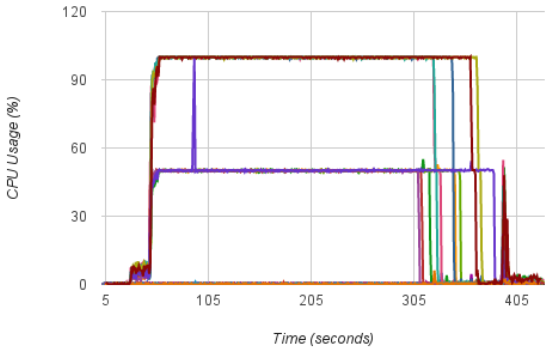
7 Related Work

Taking the problems of existing OSNs into account, some approaches of building a Distributed Social Network (DSN) have occurred such as PeerSoN [4] and SuperNova [10]. By distributing the data of the social network to the users' computers they tackle the problems of infrastructure costs and the control of data is done by each user instead of a unique entity. This does not mean that it stops having security problems but it is less likely the occurrence of a major data breach and the social network stops having as much power as it has on a centralized architecture. The implementation of such systems consists on building a P2P network where each user is a peer that stores part of the information of the OSN. Data security is enabled by using encryption or by saving information users that have social relations. As each user may disconnect frequently from the network, data must be replicated on several nodes so that its availability can be improved. The state-of-art solutions for DSNs are normally focused on distributing data and no solution has been provided for distributing the data processing operations that are crucial for the success of OSNs.



(a) Average load

(b) Memory usage



(c) CPU usage

Fig. 5: M/R-WordCount resources' usage on user nodes

In order to make use of computing resources from people waiting to contribute to a given project, there have also been introduced some solutions with great success. For example, BOINC [2] is an open-source project[2] that evolved from SETI@Home [3] and it provides the software infrastructure needed for the creation of cycle-sharing projects. It offers flexible and scalable mechanisms for distributing data, scheduling tasks and tolerate faults. Users wanting to contribute for projects with their spare-cycles need to install a client software. In order to provide incentives for users to donate resources, each user receives credits based on the resources (CPU, memory and bandwidth) provided to the project.

Using the same principal of using computing spare-cycles, JSMapReduce [6] and MRJScreates [8] introduced a solution capable of executing MapReduce tasks on the browser. Users wanting to donate resources just have to open their browser on a given page, having a much lower entry barrier. The Map and Reduce tasks are written in *Javascript* and executed on the clients' browser. The system makes use of some technologies provided by the browser such as WebWorkers[3] for processing several tasks and XHTMLRequest[4] for asynchronous communication with the server. The system maintains a centralized

[2] http://boinc.berkley.edu
[3] http://www.w3.org/TR/workers/
[4] http://www.w3.org/TR/XMLHttpRequest/

server that keeps track of MapReduce Jobs, assigns tasks to workers and supervises its execution, keeping track of each worker's availability. Failures from workers are detected using timeouts. The server is also responsible for saving the intermediate results from map tasks, sorting the intermediate results and saving final results from reduce tasks. However, as the server is used to save intermediate results, these solutions create a bottleneck on the server that prevents them from providing a scalable solution.

8 Conclusion

Online social networks have been growing to enormous numbers of users and they must find ways to analyze all the data produced in a scalable and cost-effective manner. We proposed a solution and described an architecture for a system that enables data analytic tasks to be performed on their users' browser by using the computing resources of OSN's users. Preliminary results from our implementation have proven that it is possible to implement such a system but further work must be placed in order to reduce the overheads of P2P communications between browsers and memory usage. The first problem can be solved by compressing data exchanged between workers. Reducing the usage of users' memory can be achieved by using technologies such as IndexdDB [5] or File API [6] that allow data to be saved locally to the browser.

References

1. Dstat. http://dag.wiee.rs/home-made/dstat/. Accessed: mar-03-15.
2. David P Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.
3. David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.
4. Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2p social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52. ACM, 2009.
5. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
6. Philipp Langhans, Christoph Wieser, and François Bry. Crowdsourcing mapreduce: Jsmapreduce. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 253–256. International World Wide Web Conferences Steering Committee, 2013.
7. J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session traversal utilities for nat (stun). RFC 5389, RFC Editor, October 2008. http://www.rfc-editor.org/rfc/rfc5389.txt.
8. Sandy Ryza and Tom Wall. Mrjs: A javascript mapreduce framework for web browsers. *URL http://www. cs. brown. edu/courses/csci2950-u/f11/papers/mrjs. pdf*, 2010.
9. Simon Pietro Romano Salvatore Loreto. *Real-Time Communication with WebRTC*, volume 1. O'Reilly Media, 2010.
10. Rajesh Sharma and Anwitaman Datta. Supernova: Super-peers based architecture for decentralized online social networks. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, pages 1–10. IEEE, 2012.
11. Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, and Hao Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1013–1020. ACM, 2010.

[5] http://www.w3.org/TR/IndexedDB/
[6] http://www.w3.org/TR/FileAPI/