A Peer-to-Peer Communication Function among Web Browsers for Web-based Volunteer Computing

MAKOTO KUHARA
Graduate School of
Natural Science and Technology
Okayama University
Okayama, Japan

KAN WATANABE
Graduate School of
Natural Science and Technology
Okayama University
Okayama, Japan
Email:kan.watanabe@okayama-u.ac.jp

MASARU FUKUSHI
Graduate School of
Science and Engineering
Yamaguchi University
Yamaguchi, Japan

NORIKI AMANO
Center for
Research in General Education
Saitama University
Saitama, Japan

YASUYUKI NOGAMI
Graduate School of
Natural Science and Technology
Okayama University
Okayama, Japan

***Abstract*—As an accessible and high-performance computing environment, we have studied Web-based Volunteer Computing (VC), in which users can join a VC with just accessing a specified URL on Web browsers. A major problem of Web-based VC has been the difficulty of implementing some advanced functions such as direct communications among users and file sharing, because all processes must be handled on Web browsers. For this problem, we propose a solution to realize a P2P communication function among Web browsers by using recent Web techniques such as Web Real-Time Communication (WebRTC) API and its library, PeerJS. As an useful application of our method, we demonstrate file sharing on Web browsers. Experimental result indicates that our method achieves almost 50% reduction in the total time of file distribution.**

I. Introduction

Volunteer Computing (VC) is a computing paradigm to build large-scale parallel systems using computing resources such as personal computers (PCs) on the Internet. As evidenced by the success of SETI@home [1] project, VC can realize a high performance system such as a supercomputer at a very low-cost. However, in the existing VC systems, participants must put an extra effort, e.g. installing a client software to their computers and registering their personal E-mail addresses, which will be barriers to join the VC.

As more accessible and high-performance computing environment, we have proposed Web-based VC, in which users can join a VC with just accessing a specified URL on Web browsers [11]. Compared with the conventional VC with middleware BOINC [2], the Web-based VC can omit complicated procedures to join a VC, for example, downloading and installing the client software, and user registration. Due to the capability of providing an easy way to join a VC, the Web-based VC is expected to gather more users and make VC more popular.

A major problem of the Web-based VC has been the difficulty of implementing some advanced functions such as direct communications among users and file sharing. This is because all processes for VC must be handled on Web browsers.

For this problem, this paper proposes a solution to realize a P2P communication function among Web browsers by using recent Web techniques such as Web Real-Time Communication (WebRTC) API [3] and its library, PeerJS [5]. WebRTC is one of the new functions on HTML5, which realizes an efficient direct communication among Web browsers. Because the details of WebRTC API are different in each Web browser (Firefox/Chrome), PeerJS written in JavaScript have been provided as a library of WebRTC. In our previous work [11], it is shown that some scientific computations written in C/C++ are convertible to JavaScript with a little performance degradation. Thus, in this paper, a management node and worker nodes in Web-based VC are also implemented in JavaScript. As an useful application of our implementation, we demonstrate file sharing on Web browsers.as an useful

The rest of this paper is organized as follows. Section II describes Web-based VC. Section III explains related works and peer-to-peer communication techniques for Web browsers. Section IV describes an implementation of file sharing with P2P communication among Web browsers. Section V evaluates a performance of file sharing, and Section VI concludes the paper and presents some insights about future work.

II. Web-based Volunteer Computing

A. The Way to Participate VC

In the existing VC systems based on BOINC, to join a VC, participants must put an extra effort as follows.

1 Downloading the BOINC client software
2 Installing it and rebooting PC
3 Starting the client software
4 Choosing VC projects to participate

5 Registering user information by e-mail address

Compared with the BOINC-based VC, Web-based VC can omit complicated procedures to join a VC [11]. The procedures in the Web-based VC are as follows.

1 Starting Web browsers (Executable simultaneously with the following procedure 2)
2 Opening a Web page of VC projects to participate

Due to the capability of providing an easy way to join a VC, it is expected to gather more users and make VC more popular.

B. Computation Model

The computation model of Web-based VC is the master-worker model which is the same as the existing BOINC-based VC. The model consists of a management node (master) which manages the whole VC system and participants' PCs (workers). The summary of this model is shown in Fig. 1 and explained below.

- A master divides a large scale computing project into N pieces of independent computing problems (jobs), and distributes the jobs to workers as a response to their requests.
- When a worker receives a job, the worker computes it and returns the result to the master.
- The master completes the computing projects when all N jobs are finished.

C. Reliability of Computation

Due to the nature of allowing anyone to join the computation, VC might collect erroneous results from malicious workers (saboteurs) [9]. Erroneous results are generated by several reasons in volunteer side (such as overclocking, software error and sabotage calculation). Today, VC systems employ sabotage-tolerant mechanisms based on the principle of majority decision for these problems.

In Web-based VC, the presence of saboteurs will represent a credible threat to reliable computation because saboteurs can easily change their user information (written in cookies of Web browsers) to prevent blacklisting by the master.

For this problem, credibility-based voting [12] is proposed which performs a weighted voting based on the credibility of each worker. In this method, even if saboteurs change their user information and rejoin to the VC, the results of saboteurs may not be accepted because its credibility is lower than others who are joining the VC longer term. The theoretical performance of the credibility-based voting has already studied and is shown to be better than the popular voting methods.

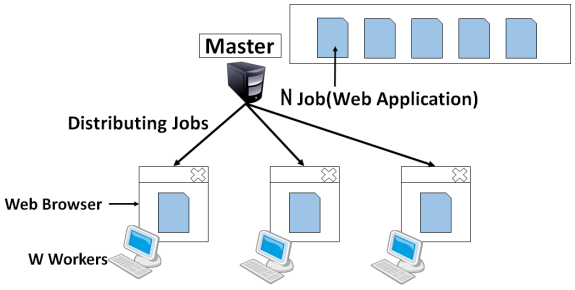


Fig. 1. Computation model in Web-based VC

III. Related Works

A. Dynamic Load Balancing by P2P Communication among Workers

In Ref.[10], dynamic load balancing with P2P communication improves the performance of the whole large-scale parallel computing system such as VC about three times at the maximum. With this load balancing, master regulates the amount of jobs among workers. Workers build an overlay network among them with P2P communication as shown in Fig.2, in order to quantify the impact of workers adjacent to a certain range. Certainly, the dynamic load balancing is also effective to the Web-based VC; however, its implementation is difficult because all processes must be run on Web browsers.

B. Communication among Web Browsers

Web is used in various purposes because of its facility and high versatility. Various services are supplied as Web applications (applications running on Web browsers with HTML and HTTP/HTTPS network communication). The communication among Web browsers is one of the current functions which has become possible with the recent advance in Web technologies. The communication among Web browsers makes it possible to implement the functions of file copying and dynamic load balancing with P2P communication. They are recently attracting attention in VC. The technologies and library used in this paper are as follows.

1 WebSocket [4] :
WebSocket is a protocol for bidirectional communication on the Internet. In WebSocket, efficient bidirectional communication is possible because the amount of communication resource consumption is improved. Once establishing a communication between a server and a client, we can transmit data without considering the transmission protocol unless the communication is completely disconnected.
2 WebRTC (Web Real-Time Communication) [3] :
WebRTC is an API to realize the bidirectional real-time communication with WebSocket without any plug-in for Web browsers. In 2011, communication among Web browsers with WebRTC was proposed by Google. Now it is standardized at W3C, and it's protocol is standardized at IETF.

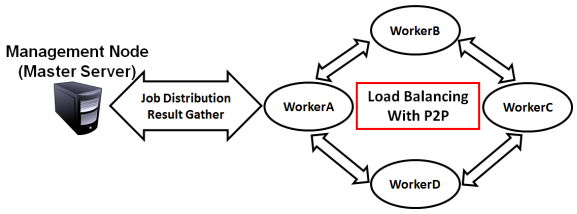


Fig. 2. Dynamic load balancing by P2P communications among workers

3 PeerJS [5] : PeerJS is a JavaScript library to implement P2P communication among Web browsers with WebRTC easily. In PeerJS, a client gets a PeerID (ID with 13 to 16 alphanumeric characters) to identify the client by the mediation server (called PeerServer). With the others' PeerIDs, P2P communication can be established on Web browsers.

IV. The Implementation of File Distributing Function with P2P communication among Web Browsers

A. Outline

As an application of P2P communication among Web browsers, we show that the function of file sharing among workers enables a faster file distribution than the original (master-worker based) file distribution method.

For the performance comparison, we implemented two types of file sharing functions. One is master-worker file sharing function that master directly distributes files to workers, and the other is P2P file sharing function that workers copy files among themselves by P2P communication among Web browsers. Section IV-B1 presents the common parts of the both functions. Section IV-B2 and IV-B3 describe the detailed implementation of each function. In Section IV-C, we discuss a method for an efficient file distribution.

B. Implementation of File Sharing Function

1) Common Part: The file to be distributed is stored in the master. The master copies the file and distributes it to workers.

We implemented the master with Node.js [6], and it is written in only JavaScript. The master communicates with workers by WebSocket.

We implemented the workers with HTML5 and JavaScript. One tab in a Web browser corresponds to a single worker. If the tab is updated (by user operation), it runs as a different new worker by getting a new worker ID. The processing at a worker starts when the worker opens the specific URL on Web browser. First, workers communicate with the master by WebSocket. Each worker receives the file to be distributed either from the master or one of the other workers then the worker archives the file into the data base of the Web browser. After archiving the file, the worker sends the message of "File Given" to the master.

2) Master-Worker File Sharing Function: The processing flow of the master-worker file sharing function is shown in Fig.3. When the worker starts, it sends a file demand and its unique ID to the master.

In this function, the master directly sends the file to the workers who send a file demand. The workers archive the file at the data base in their Web browsers, and send "File Given" to the master. On receiving it, the master increases $finFileSend$ counter and updates $workerList$.

3) P2P File Sharing Function: Unlike the master-worker file sharing function, file is sent to workers by those who have received it from other workers. This enables to control the disk I/O by file distribution, which has been a cause of high processing load at the master, and reduce the resource

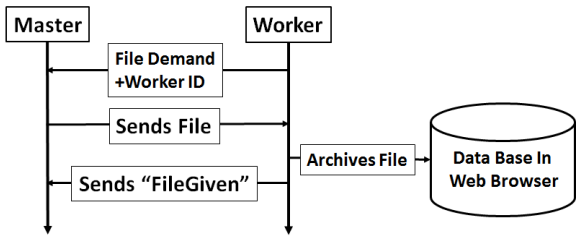


Fig. 3. The processing flow of master-worker file distributing function

consumption such as the master's communication bandwidth. The reduction of the master's processing load enables to build a large-scale system because the master can use its capacity for the management of job progress. It improves the performance of the whole VC system.

To realize the file copying among workers, in this paper, we implement the function for the master and the workers as follows.

- master :
  Master starts a PeerServer, a mediation server, inside itself to allow workers to P2P communication with PeerJS. In our method, the master classifies all workers into two categories, parent and child workers, in the order of their accesses; the first worker is decided as the parent and the next N-1 workers are decided as child. This dicision process is repeated for the subsequent workers. The master distributes a file to only the parent workers.
- worker :
  At the start of the worker process, each worker accesses the PeerServer and gets a PeerID. A parent worker receives a file from the master directly and copies it when receiving a demand from child workers by P2P communication. A child worker sends a file demand to parent worker with its PeerID, and receives a copy of the file.

Master judges whether a worker is a parent worker when a worker accesses the master in sequence. It is assumed that one parent worker distributes a file to $N-1$ workers. First, the worker accessing the master first is decided as a parent worker. After the master archives its PeerID as parent worker's ID, the master distributes the file to a parent worker. The workers that access from second to Nst are decided as child workers. The process of $N+1$ or subsequent is also the same process as heretofore.

The processing flow above the P2P file sharing function is shown in Fig.4. When a worker sends a file demand to the master, it is classified as either parent or child worker. If it is parent, it waits the request of "File Demand" from child workers. If it is child, it receives a parent worker's PeerID from the master. Each child worker sends a file demand to the parent worker with the PeerID, and sends "File Given" to the master after receiving and archiving the file.

C. The Optimum Number of parent workers

The communication bandwidth between the workers is limited. This is the reason why there is an optimum value

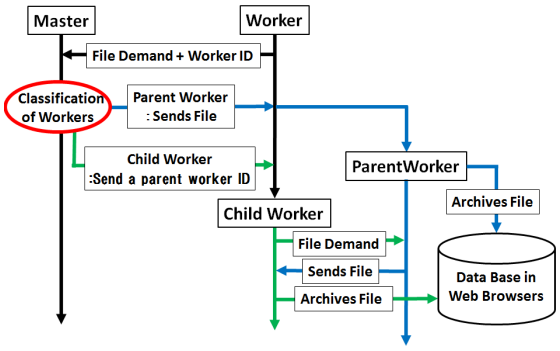


Fig. 4. The flow of the P2P file distributing function

for the parent workers. If the number of parent workers is too large or too small, the completion time of file distribution in the P2P file sharing function becomes longer.

We derive the optimum number of parent workers, w_{opt}, to minimize the completion time of file distribution, T_{trans}. The parameters used to the minimization of T_{trans} are as follows;

- The number of all workers: W
- The number of parent workers: $w(\leq$ W/2)
- The size of a file: F(MB)
- The bandwidth between the master and the workers: H_{MW}(MB/s)
- The bandwidth among the workers: H_{MW}(MB/s)
- The communication speed in P2P communication: S_{P2P}(MB/s)

In this paper, S_{P2P} is assumed about 0.1MB/s based on the default parameters in PeerJS. The time of distributing files from the master to parent workers is denoted as T_{MW}(s). T_{MW}(s) is equal to the time when the master distributes a file to w workers, and given by Eq.(1).

$$T_{MW}(w) = \frac{F}{H_{MW}} \times w. \tag{1}$$

A file is assumed to be sent from a parent worker to several child workers at the same time. Because of the limited H_{WW}, the number of child workers that a parent worker can distribute a file at the same time is also limited. At that time, the average of the communication speed is given by $(H_{WW}/(W-w))$. Therefore, the time of copying files from parent workers to child workers, T_{WW}, is given by Eq.(2).

$$T_{WW}(w) = \begin{cases} \frac{F \times (W-w)}{H_{WW}} & \text{if } (W-w) \times S_{P2P} \geq H_{WW}, \\ \frac{F}{S_{P2P}} & \text{otherwise.} \end{cases} \tag{2}$$

For simplicity, we assume file copying from the master to parent workers and from parent to child workers run in parallel (the file which doesn't finish transmitting can be sent). At that time, the file distribution time, T_{trans}, is equal to max(T_{MW}, T_{WW}).

Therefore, the optimum number of parent workers, w_{opt}, is given by Eq.(3). Eq.(3) shows that w_{opt} is independent of the size of a file, F.

$$w_{opt} = \begin{cases} \frac{H_{WW} \times W}{H_{MW} + H_{WW}} & \text{if } (W-w) \times S_{P2P} \geq H_{WW}, \\ \frac{H_{MW}}{S_{P2P}} & \text{otherwise.} \end{cases} \tag{3}$$

V. Performance Evaluation

A. Summary of Experiment

As the performance evaluation of two file sharing functions in Section IV-B, we measured the time to finish distributing a file to all workers. In this experiment, the master, workers and PeerServer run on one PC. The bandwidth between the master and the workers, and among the workers are limited by "shaperd[7]". The experiment condition is listed in Table. I.

TABLE I. EXPERIMENT CONDITION

OS	Ubuntu 12.04 LTS 32bit
CPU	Intel Core2 Quad Q6700
Memory	4GB (The real bandwidth: 4.685 [GB/s])
Node.js	ver. 0.10.20
PeerJS	ver. 0.3.8
Web browser	Chrome ver. 31

The workers participate by five per seconds. The number of all workers is 40 at a maximum to the extent that we can experiment, and F, the size of a file, is 8 MB. In the case of a typical VC such as Asteroids@home [8], F is about 50 to 200 kB. However, considering a small-scale experiment and an application such as image processing, we set the large size of a file.

In the case of a large-scale VC system, the real bandwidth of the master becomes narrow because many workers create connections to the master. Therefore, the bandwidth between the master and the workers is set to be smaller than that among the workers. Table II and III show the experimental parameter for the large-scale VC systems.

TABLE II. PARAMETER FOR EXPERIMENT1 (CHANGING THE NUMBER OF ALL WORKERS)

The number of workers W	5 - 40
The number of parent workers w	$\lceil \frac{W}{10} \rceil$
The size of a file F[MB]	8
The communication bandwidth [MB/s]	between the master and workers T_{MW}:2
	among workers T_{WW}:6

TABLE III. PARAMETER FOR EXPERIMENT2 (CHANGING THE NUMBER OF PARENT WORKERS)

the number of workers W	40
the number of parent workers w	4 - 20
the size of a file F[MB]	8
communication bandwidth [MB/s]	between master and workers T_{MW}: 1
	among workers T_{WW}: 3 or 6

B. Evaluation Results

1) Experiment 1 (in case of changing W): Fig.5 shows the file distribution time for the number of all workers. Fig.5 shows that the file distribution time with the P2P file sharing function (called T_{P2P}) becomes longer than that with the master-worker file sharing function (called $T_{MAS-WOR}$) when W is small. However, $T_{MAS-WOR}$ becomes longer than T_{P2P} as W increases. When W is above 20, T_{P2P} becomes shorter than $T_{MAS-WOR}$. T_{P2P} at W=40 is approximately half of

$T_{MAS-WOR}$. When W is small, the bandwidth T_{MW} is broad enough to distribute a file with the master-worker file sharing function. However, when W is large, $T_{MAS-WOR}$ increases in proportion to W because T_{MW} becomes a bottleneck. In the P2P file sharing function, this is unlikely to occur. This means that the master can manage more workers.

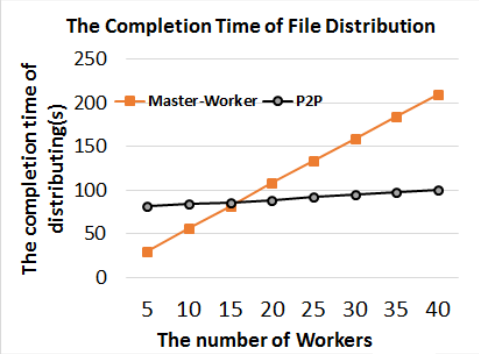


Fig. 5. File distribution time for the number of all workers

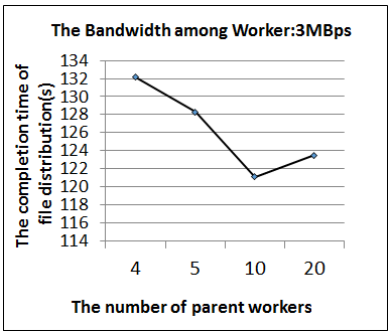
2) Experiment 2 (in case of changing w): Fig.6 shows the file distribution time for the number of parent workers. Fig.6 shows that the completion time of file distribution is significantly different depending on w and there is the optimum value for w. The optimum value of w, w_{opt}, is 10 when the bandwidth among the workers H_{WW} is 3(at Fig.6(a)), whereas w_{opt} is 4 when H_{WW} is 6(at Fig.6(b)). Therefore, w_{opt} changes for the different bandwidth.

In case of Fig.6(a), substituting the real P2P communication speed S_{P2P} = 1 into the calculation formula for the optimum value in Section IV-C(Eq.(3)), we obtain w_{opt} = 10 and it matches the optimum value in the experimental result. However, in the case of Fig.6(b), the value given by Eq.(3) is 1/0.1 = 10, and it is different from w_{opt} = 10 in the experimental result.

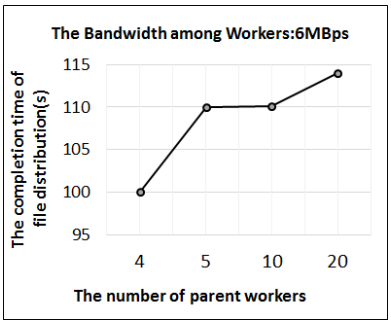
We consider that the reason why w_{opt} differs from the real optimum value is the waiting time of transmiting a file between the master and a parent worker. In the derivation of Eq.(3), we assume that the file which has not finished sending is also able to be sent. However, in contrast to the above assumption, the actual transmission between a parent worker and a child worker must be waited until the transmission between the master and the parent worker is finished. This waiting time for the file transmission is F/H_{MW} = 8 seconds per parent worker. Therefore, if H_{WW} is broad enough as in Fig.6(b), the file can be distributed to many child workers with the small number of parent workers at the same time. This is the reason why the file distribution is finished earlier if we cut the waiting time of file transmission by making the number of parent workers small. The derivation of w_{opt} considering the waiting time is one of challenges and remain as a future work.

VI. Conclusion and Future Works

In this paper, we propose a solution to realize a P2P communication function among Web browsers by using PeerJS. As an useful application of our implementation, we demonstrate file sharing on Web browsers. Experimental result indicates that our method achieves almost 50% reduction in the total time of file distribution when the number of workers is 40.



(a) T_{WW}=3MB/s



(b) T_{WW}=6MB/s

Fig. 6. File distribution time for the number of parent workers

Our future works include the derivation of the optimal number of middle workers for faster file distribution and the implementation of load balancing function on Web-based VC with P2P communication. A large-scale performance evaluation with multiple worker PCs and the performance evaluation of the whole VC system are also advisable.

References

[1] SETI@home: http:setiathome.berkeley.edu.

[2] BOINC: http://boinc.berkeley.edu.

[3] Johnston, Alan B., and Daniel C. Burnett, "WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web", Digital Codex LLC, 2012.

[4] Pimentel, Victoria, and Bradford G. Nickerson, "Communicating and displaying real-time data with WebSocket", Internet Computing, IEEE, Vol.16, Issue 4, pp.45-53, 2012.

[5] PeerJS: http://peerjs.com/

[6] Node.js: http://nodejs.org/

[7] shaperd: http://bto.la-terre.co.jp/support/shaperd.html

[8] Asteroids@home: http://asteroidsathome.net/boinc/

[9] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello, "Characterizing Error Rates in Internet Desktop Grids", 13th European Conf. Parallel and Distributed Comput., pp. 361–371, 2007.

[10] Y. Murata, T. Inaba, H. Takizawa and H. Kobayashi, "A Distributed and Cooperative Load Balancing Method for Large-scale Computing Environments", Information Processing Society of Japan, vol.49, no.3, p1214-1228, 2008.

[11] S. Takaki, K. Watanabe, M. Fukushi, N. Amano, N. Funabiki, T. Nakanishi, "A proposal of Web Browser-based Volunteer Computing Platform", IPSJ SIG Technical Report, 2014-HPC-143(29), pp. 1–8, 2014. (in Japanese).

[12] K. Watanabe, M. Fukushi and S. Horiguchi, "Expected-credibility-based Job Scheduling for Reliable Volunteer Computing", IEICE Trans. Inf.& Syst., Vol.E93-D, No.2, pp.306 – 314, 2010.