Electronic Preprint for Journal of Information Processing Vol.24 No.2

Regular Paper

A Reliable Volunteer Computing System with Credibility-based Voting

Takeshi Sakai[1,a)] Masaru Fukushi[1,b)]

Received: June 30, 2015, Accepted: October 2, 2015

Abstract: In order to realize highly-reliable volunteer computing (VC), this paper develops a VC system with credibility-based voting. Credibility-based voting is known as an efficient technique for eliminating incorrect calculation results returned by participant (worker) nodes. Although its theoretical performance has been studied in detail, its implementation and practical performance have not been fully studied yet. Our VC system consists of a management server and a number of worker nodes. In the management server, each process that includes credibility-based voting is multithreaded, and all information is managed in a database for high performance and stable operation. Through performance evaluations, we found that the most time-consuming processes are the credibility-based voting in a network environment with a small delay (e.g., Intranet), and the process for sending jobs in a network environment with a moderate delay (e.g., domestic Internet). Moreover, multithreading is shown to be effective for those performance bottlenecks, and feasible system configurations are also revealed for a variety of request rates.

Keywords: volunteer computing, desktop grid, credibility-based voting, multithreading

1. Introduction

Volunteer Computing (VC) is a type of distributed computing paradigm, which allows any participants on a network (Intranet or Internet) to contribute their idle computing resources (CPU cycles) toward solving large parallel problems. Some examples of VC include SETI@home [1], Folding@home [2], and distributed.net [3]. By making it easy for anyone on the network to join a computation, VC makes it possible to build very large and high performance computing environments at a low cost. Nowadays, VC is used for scientific computations in various areas such as biology [4], astronomy [5], and physics [6].

One of the critical problems which must be addressed in VC is sabotage-tolerance [7], [8], [9]. Unlike grid systems that are appropriately managed by administrators [10], [11], [12], [13], participant nodes of a VC system are owned and managed by general users, and thus may behave erratically and return incorrect results. The incorrect results can be generated by several reasons on the participants' side (e.g., hardware/software errors, computer virus infection, or malicious behavior for falsifying the computation). It has been reported that a large fraction of participant nodes (about 35%) actually returned at least one incorrect result in a real VC [7]. In today's VC systems, majority voting is widely employed for this problem. BOINC [14], a major VC middleware, uses m-first voting which simply collects m matching results for each computation.

To enhance the efficiency of the voting, credibility-based voting [15] has been proposed. This method conducts a weighted voting based on the credibility of each participant. The key advantage of this method is the capability of guaranteeing the computational correctness mathematically. The theoretical performance of this method has been studied in detail and is shown to be better than the popular m-first voting; however, its implementation issue and practical performance have not been fully studied yet.

In this paper, for the purpose of realizing highly-reliable VC, we developed a VC system with credibility-based voting by improving our previous prototype implementation [16]. In the proposed VC server, each process that includes credibility-based voting is multithreaded, and all information is managed in a database (DB) for high performance and steady operation. Then, we reveal a bottleneck process in the VC server, and show the effectiveness of multithreading for the bottleneck.

The main contributions are as follows:

(1) a VC system with credibility-based voting and a DB is developed, and its operation is confirmed to be correct,
(2) performance bottleneck is found in the proposed VC server and relieved by multithreading, and
(3) a feasible system configuration is revealed for a network environment with a small or a moderate delay.

The rest of this paper is organized as follows. Section 2 presents the computation model of VC and its sabotage-tolerance mechanism. Section 3 describes the proposed VC system inclusive of DB structure and the access method. Section 4 evaluates the performance of the developed VC system under various VC environments. Finally, Section 5 concludes the paper.

1 Graduate School of Science and Engineering, Yamaguchi University, Ube, Yamagichi 755–8611, Japan
a) u017vk@yamaguchi-u.ac.jp
b) mfukushi@yamaguchi-u.ac.jp

© 2016 Information Processing Society of Japan

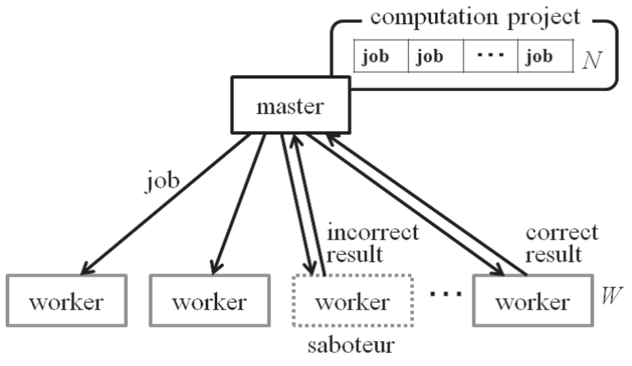

Fig. 1 Master-worker model.

2. Volunteer Computing

2.1 Computation Model

The popular master-worker model is assumed as the computation model of VC. This model is used in almost all VC systems practically. The details of this model are summarized in **Fig. 1** and below.

- A VC system consists of a management server (master) and W participant nodes (workers).
- A computation project to be processed in the system is composed of N independent jobs.
- The master manages the computation project, and assigns a job to a worker at the request of the worker.
- Each worker executes the assigned job and returns the calculation result to the master. Workers repeat this process as long as a job is assigned by the master.
- The computation project is completed when all N jobs are finished.

In this model, there exist $\lfloor f \times W \rfloor$ saboteurs, where f is the fraction of saboteurs. Saboteurs return incorrect results with a constant probability s, which is known as the sabotage rate. The values of f and s are unknown to the master.

2.2 Sabotage-tolerance Mechanism

To eliminate the effect of incorrect results, some sabotage-tolerance mechanism must be used in VC.

2.2.1 m-first Voting

The most commonly used method for the sabotage-tolerance of VC is voting [17], [18]. Basically, a master replicates a job and distributes them to several workers for a majority decision. In m-first voting, the result which collects m matching values first is accepted as the final result for the job. In practical VC systems like SETI@home [1], m is set to 3. Since it has a minimum redundancy of 3 regardless of f, the performance of VC systems may be significantly decreased, i.e., less than $1/m$.

2.2.2 Spot-checking

Some sampling techniques were developed to overcome the limitations in voting. Examples are naive, quizzes, ringers, and spot-checking [8]. In spot-checking, a master sometimes assigns decoy jobs called spotter jobs. The master knows the correct results of the spotter jobs, and thus can directly check whether workers behave correctly or not. If a worker returns an incorrect result, the master recognizes the worker as a saboteur. Then, the master can counter against the saboteur by the following two methods; invalidating all results returned from the saboteur (backtracking), and/or preventing the submission of any results in any subsequent computation (blacklisting).

2.2.3 Credibility-based Voting

In credibility-based voting [15], a master conducts a weighted voting which combines the functions of m-first voting and spot-checking. In this method, each system element such as worker, result, and job is assigned a credibility value which represents its correctness. Using these credibility values, the final result for each job is decided by weighted voting. This method guarantees the correctness of the final results mathematically, which is an important feature that no other voting method possessed. This method is also applied to the job scheduling problem of VC to reduce overall computation time [19], [20]. Therefore, in this paper, we focus on this method for realizing highly-reliable VC.

We briefly present the definition of credibility and the method of determining final results.

The credibility of worker w, denoted by $C_W(w)$, is determined by the number of times that w survives spot-checking, i.e., w returns correct results for spotter jobs. When w survives spot-checking k times, $C_W(w)$ is given by

$$C_W(w) = \begin{cases} 1 - f_{\max} & (k = 0) \\ 1 - \dfrac{f_{\max}}{k} & (k \neq 0), \end{cases} \tag{1}$$

where $f_{\max}$ is the maximum fraction of saboteurs assumed by the master.

The credibility of result r returned by worker w, $C_R(r)$, is equal to $C_W(w)$.

$$C_R(r) = C_W(w). \tag{2}$$

Suppose that the results returned for a job are divided into g result groups $G_1, G_2, ..., G_g$, each of which includes results having the same value. The credibility of result group G_a, denoted by $C_{RG}(G_a)$, is defined by

$$C_{RG}(G_a) = \frac{P_T(G_a)\prod_{i \neq a} P_F(G_i)}{\prod_{i=1}^{g} P_F(G_i) + \sum_{n=1}^{g} P_T(G_n)\prod_{i \neq n} P_F(G_i)}, \tag{3}$$

$$P_T(G_a) = \prod_{r \in G_a} C_R(r), \tag{4}$$

$$P_F(G_a) = \prod_{r \in G_a} (1 - C_R(r)). \tag{5}$$

$P_T(G_a)$ (or $P_F(G_a)$) represents the probability that all results in G_a is correct (incorrect). $C_{RG}(G_a)$ in Eq. (3) represents the conditional probability that the results in G_a are correct and those in all other groups are incorrect.

The credibility of job j, $C_J(j)$, is equal to the credibility of the result group that has the highest credibility in all result groups for job j.

$$C_J(j) = C_{RG}(G_x) = \max_{1 \leq a \leq g} C_{RG}(G_a). \tag{6}$$

When $C_J(j)$ is greater than or equal to a threshold $\theta (= 1 - \varepsilon_{acc})$, the result of G_x is accepted as the final result of job j, and then job j is finished. ε_{acc} is the acceptable error rate and the value can be set in accordance with the reliability requirement imposed for the computation project.

Credibility-based voting is an efficient voting method which balances the redundancy and the reliability of computations. However, to the best of our knowledge, no study has revealed its practical performance.

3. Developed VC System

For the purpose of realizing highly-reliable VC, we develop a VC system with credibility-based voting. In this section, details of the system structure are described inclusive of DB structure and the access method.

3.1 System Structure

The VC system we developed consists of a master and several workers as shown in the computation model in Section 2. **Figures 2** and **3** show the configurations of the master and the worker, respectively. In both figures, rectangles in solid lines represent independent processes. Each process is implemented as a thread and works in parallel. In this system, each thread is implemented with standard Linux Pthreads.

On the master side, a FIFO queue is employed between two adjacent threads to pass data between them. Information about workers, jobs, and result groups is managed in the DB. Before describing each thread, we define three types of requests: Participate Request, Job Request, and Send Result Request. Participate Request is sent once when workers join the VC, while Job Request and Send Result Request are sent every time when they need to obtain a job and return the result, respectively. Details of each thread are explained below.

- Accept Request
  This thread waits for requests from workers through accept and connect functions (accept() and connect()) of the BSD socket API. When the master is connected from a worker, connect() returns a socket number which is used for subsequent communication with the worker. The socket number is stored in the FIFO queue to be passed to the next thread.
- Receive Request
  This thread establishes communication with the worker to receive a request. Then, the socket number and the received request are stored in the queue.
- Identify Request
  This thread identifies the type of the received request and forwards it to the corresponding thread. If the request is a Participate Request, the socket number is passed to the next Accept New Worker thread. If it is a Job Request or a Send Result Request, the socket number and the worker ID are passed to the next corresponding thread.
- Accept New Worker
  This thread processes Participate Requests. This thread sends a new worker ID to the worker and then adds the worker ID to the DB.
- Send Job
  This thread processes Job Requests. This thread sends a job to the worker. When selecting a job to be sent, it checks whether the job is completed or not by inquiring the DB.
- Receive Result
  This thread processes Send Result Requests. This thread receives a result from the worker and then passes it to the next Credibility Calculation thread.
- Credibility Calculation
  This thread calculates the credibility associated with the received result and performs credibility-based voting as explained in the previous section.

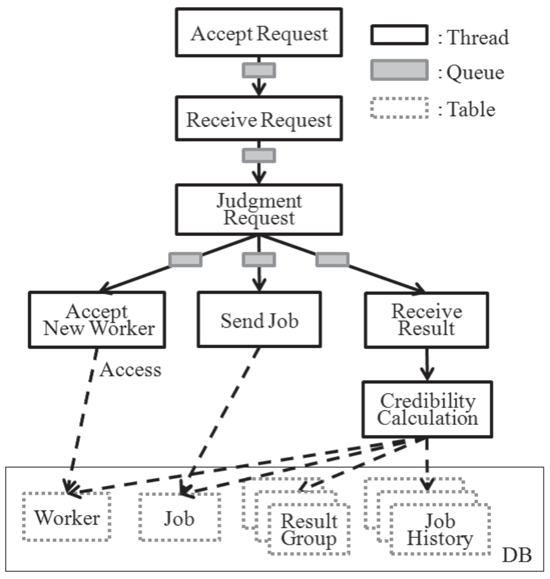

Fig. 2 Master structure.

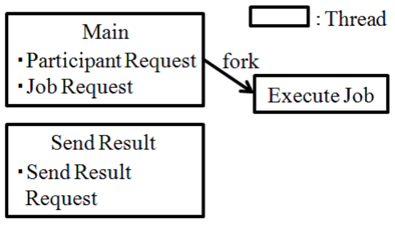

Fig. 3 Worker structure.

On the worker side, three threads run in parallel as shown in Fig. 3. Main and Send Result threads are memory-resident, while the Execute Job thread is forked by the Main thread as necessary. Details of each thread are explained below.

- Main
  This thread sends requests (either Participate Request or Job Request) to the master and manages the Execute Job thread. By a Perticipate Request, the worker receives a worker ID from the master. When sending a Job Request, the worker ID is also sent with the request.
- Execute Job
  This thread executes an assigned job. When the execution of the job is completed, this thread is killed by the Main thread.
- Send Result
  This thread sends Send Result Requests to the master to return the calculation result. Together with the request, the worker ID is also sent.

© 2016 Information Processing Society of Japan

Worker table

worker ID	number of times that the worker survives spot-checking (k)	credibility of the worker ($C_W(w)$)

Job table

job ID	credibility of the job ($C_J(j)$)	end flag

Result Group table

answer	worker ID

Job History table

job ID

Fig. 4 Data format of four tables.

3.2 Database Structure and Access Method

In the developed VC system, the DB is used to manage information about workers, jobs, and result groups. The DB is implemented using MySQL. In the previous system [16], DB access is a cause of performance bottleneck. Therefore, we propose a new DB structure and access method to improve the access times.

The proposed DB structure consists of one Worker table, one Job table, N Result Group tables, and W Job History tables, as shown in Fig. 2. The Worker and Job tables store the information of all W workers and J jobs as W and J entries, respectively. The Result Group table for job j stores the information of result groups for job j. The Job History table for worker w stores the information of jobs calculated by worker w. The data format of each table is shown in **Fig. 4**. In Worker and Job tables, the Worker ID and Job ID are unique.

In the previous system [16], the DB consists of only three tables, Worker, Job and Result Group tables. The Result Group table stores all information about result groups for all jobs, which makes the table size quite large. We found this is the cause of the long access times. For Worker and Job tables, the access time can be kept small by specifying the entry to be accessed. On the other hand, access to the Result Group table involves a full search because the result information is added in the order of returned results. Therefore, in the new DB structure, the Result Group table is divided by job to keep the table size small. This allows faster information search in the DB.

The master frequently accesses the DB when executing Credibility Calculation. DB access differs depending on the type of received result.

- Normal Process
  When the master receives a calculation result for a job j from worker w, then the master updates $C_{RG}(G_a)$ for $1 \le a \le g$ and $C_J(j)$ by the following procedures:
  (1) Add the received result and the worker ID into the Result Group table for job j.
  (2) Add the job ID into the Job History table for worker w.
  (3) Get all of the necessary data to calculate the job's credibility (i.e., result values and credibility of all workers who returned results to job j) from the Result Group and Job History tables.
  (4) Update the end flag in the Job table when $C_J(j) \ge \theta(= 1 - \varepsilon_{acc})$.
  (5) Check the end flag for all jobs in the Job table.

  Due to the modification of the DB structure, this access method requires two procedures to add data (i.e., procedures (1) and (2)). The main difference between the above proposed method and the previous method [16] is the number of SQL commands to obtain the necessary data. In the previous method, all data in the above procedure (3) are obtained sequentially by individual SQL commands, whereas, in the proposed method, it is done by only one SQL command.
- Success Process
  When the master receives the correct result for a spotter job from worker w, then the master updates $C_J(j)$ of all jobs executed by w. All $C_{RG}(G_a)$s are updated by the following procedures:
  (1) Update the credibility of worker w in the Worker table.
  (2) Get all j_n job IDs from the Result Group table, where j_n represents the number of jobs for which w returned the results.
  (3) Calculate the job's credibility by the same procedures of (3)–(5) in the Normal Process. This is repeated j_n times.
- False Process
  When the master receives an incorrect result for a spotter job from worker w, then the master deletes all results returned by w by the following procedures:
  (1) Get all j_n job IDs from the Result Group table.
  (2) Delete all j_n results from the Result Group table.
  (3) Calculate the job's credibility by the same procedures of (3)–(5) in the Normal Process. This is repeated j_n times.

In adding, updating and deleting data, the master locks and unlocks a entry of a table in the DB, which also needs two DB accesses.

4. Performance Evaluation

4.1 Experimental Overview

We have implemented the VC system described in Section 3 using the PCs listed in **Table 1**. The master runs on PC1 and is directly connected to PC2. At PC2, the round-trip communication delay is generated using the Linux tc command. Workers run on PC3 as multiprocess virtual workers. Although virtual jobs are used to simulate the VC, actual jobs can be used as well for performing VC. We have confirmed the correct operation of the implemented VC system.

We have conducted VC experiments to evaluate the practical performance of the credibility-based voting for various system configurations and VC environments. In the experiment, each worker sends a request (either a Job Request or a Send Result Request) to the master per second and the master responds to them. This process is continued for three minutes. The experimental parameters are shown in **Table 2**. Since 100 workers send a request per second for three minutes, the total number of requests the master receives is 18,000.

4.2 Evaluation of DB Access Methods

To compare the DB access methods in the previous and the proposed systems, we measured the total processing time and ac-

Table 3 Breakdown of Credibility Calculation.

process name	number of processes	total processing time (s) (previous system)	total processing time (s) (proposed system)
Normal Process	16,231 (90.2%)	30.28 (14.1%)	17.97 (26.7%)
Success Process	1,749 (9.7%)	184.00 (85.4%)	49.16 (73.1%)
False Process	20 (0.1%)	1.18 (0.5%)	0.11 (0.2%)
total	18,000 (100.0%)	215.46 (100.0%)	67.25 (100.0%)

Table 1 Specifications of PCs.

PC	OS	CPU	memory
PC1 (master)	Vine Linux 6.2	Intel Core i7 3.4 GHz (4 cores)	8 GB
PC2 (mediator)	Linux Mint 16	Intel Core 2 Duo 3.33 GHz (2 core)	3.9 GB
PC3 (worker)	Linux Mint 15	Intel Core i3 2.13 GHz (2 cores)	3.7 GB

Table 2 Experimental parameter.

number of jobs (N)	10,000
number of workers (W)	100
spot-check rate (q)	0.1
maximum fraction of saboteurs ($f_{\max}$)	0.35
fraction of saboteurs (f)	0.1
sabotage rate (s)	0.1
acceptable error rate (ε_{acc})	0.01

cess count for the three processes in the Credibility Calculation (i.e., Normal Process, Success Process, and False Process). The access count denotes the number of times the DB is accessed. In this evaluation, no communication delay is set by the tc command because the processes in the Credibility Calculation do not require any communication.

Table 3 shows the total processing time for the three processes. This table shows the breakdown of all 18,000 processes in the Credibility Calculation. As shown in this table, more than 90% are Normal Processes. Since the spot-check rate q is set to 0.1, the sum of Success Process and False Process is approximately equal to 10%. Although the proportion of Success Processes is less than 10%, the total processing time is significantly large compared to the other two processes in both systems. The proposed system achieves a total processing time that is about 60% smaller than the previous system.

Figures 5 – 8 show the access count of Normal Process and Success Process. In these figures, the x-axis is the order of the request received by the master. In Fig. 5, the access count in each request is almost constant. After the 10,001st request, the access count slightly increases because each job begins to receive a second result from a worker and information necessary for the credibility calculation is increased. In Fig. 6, the access count is always constant because the reference of all necessary data (i.e., procedure (3)) is done at a time by one SQL command.

The access count of the Success Process is quite large compared to that of the Normal Process there is a tendency to increase gradually as the computation of VC proceeds. When a result is returned for spot-checking by worker w, all jobs which have been processed by w need the credibility updated. The number of such jobs (i.e., j_n in Success Process) is increased as computation proceeds; hence the access count is increased accordingly. The proposed system substantially reduces the access count compared to the previous method. This is because of less DB accesses in procedures (3)–(5) for the Normal Process. Although both systems show almost the same access count as in Fig. 5 and Fig. 6, we found through investigating the DB log that the proposed system needs only two DB accesses for procedures (3)–(5), while the previous system needs four or five accesses. The other six accesses in the proposed system are for procedures (1) and (2) which involve a total of four lock and unlock operations. Since

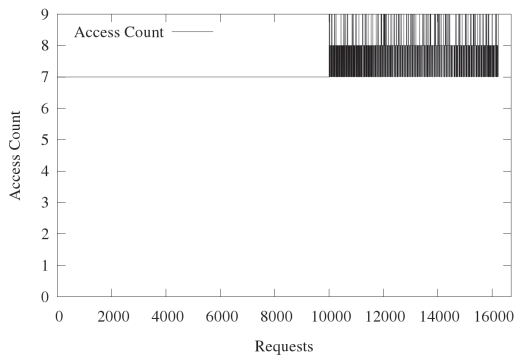

Fig. 5 Access count of Normal Process in the previous system.

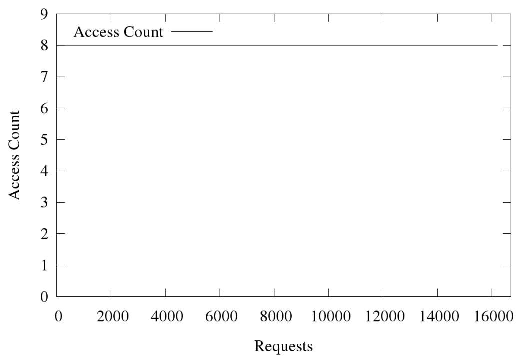

Fig. 6 Access count of Normal Process in the proposed system.

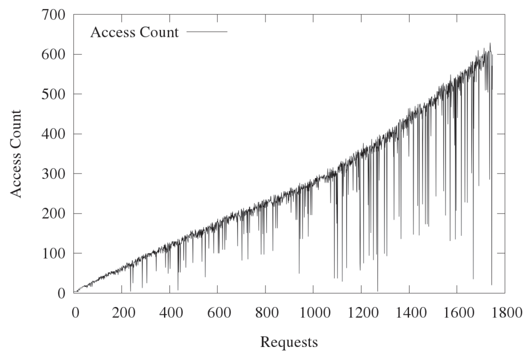

Fig. 7 Access count of Success Process in the previous system.

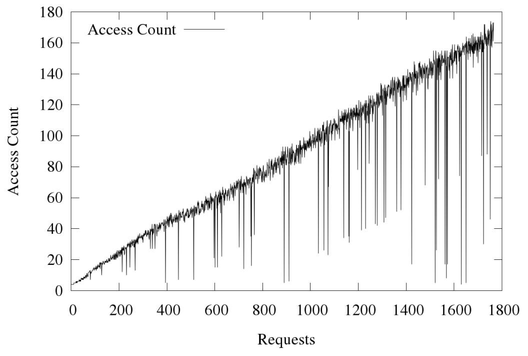

Fig. 8 Access count of Success Process in the proposed system.

procedures (3)–(5) are repeatedly executed in the Success Process, the difference becomes large, as shown in Fig. 7 and Fig. 8. This data indicates the reason for the reduction in total processing time in the proposed system.

Access count for the False Process is similar to that of the Success Process; therefore, the data is omitted here.

4.3 Evaluation of VC server

To evaluate the performance of the proposed system, we measured the average processing time, average waiting time, and average queue length of a request for four main processes in the master (i.e., Receive Request, Send Job, Receive Result, and Credibility Calculation).

4.3.1 Results under no additional delay

First, no communication delay is added by the tc command. In this case, the average communication delay between the master (PC1) and workers (PC3) was less than 1 ms. This corresponds to a delay time on the Intranet, e.g. in a university.

Figures 9, **10** show the experimental results. In the four main processes, Credibility Calculation has the longest processing and waiting time, and others are much shorter than the Credibility Calculation. Figure 9 also shows DB access time as a breakdown. The DB access time occupies almost 99% of the processing time of the Credibility Calculation, meaning that DB access is the cause of the bottleneck. These results indicate that credibility-based voting is a performance bottleneck and the main cause is spot-checking, which is the key to reducing redundancy in credibility-based voting. These results also provide important insight into the use of credibility-based voting; that is, it may need some restriction on updating credibility in Success Process in practical VC. Otherwise, a large number of DB accesses will be a critical bottleneck for the VC server. For example, updating the credibility of completed jobs can be stopped to avoid a continuous increase in the access count.

To further reduce the processing time, the number of threads is increased by multithreading. Let N_S, N_R, and N_C be the number of threads for Send Job, Receive Request, and Credibility Calculation, respectively. **Figures 11 – 13** show the average processing time, average waiting time, and average queue length as a function of the number of threads N_C. N_S and N_R are set to be one. Figure 11 shows that the average processing time increases with increasing N_C. This seems to be due to the overhead of thread switching. Figures 12 and 13 show that the waiting time and the

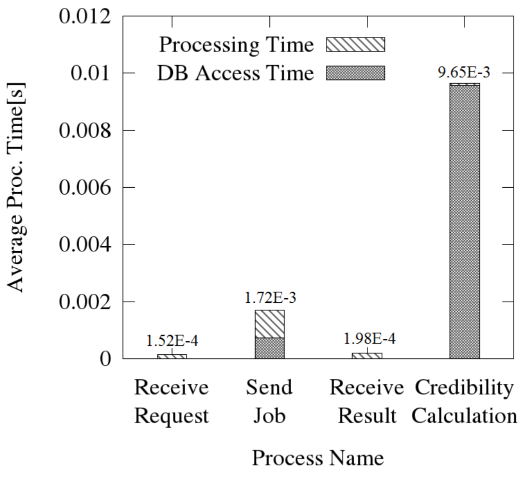

Fig. 9 Average processing time.

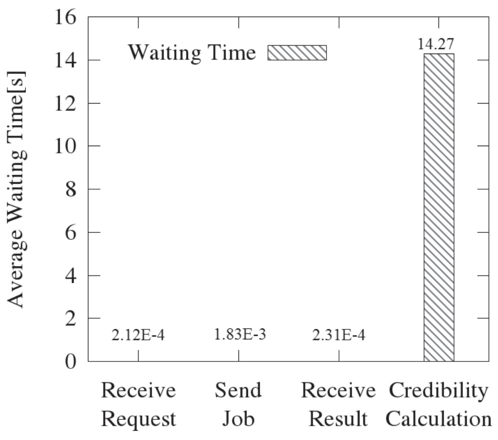

Fig. 10 Average waiting time.

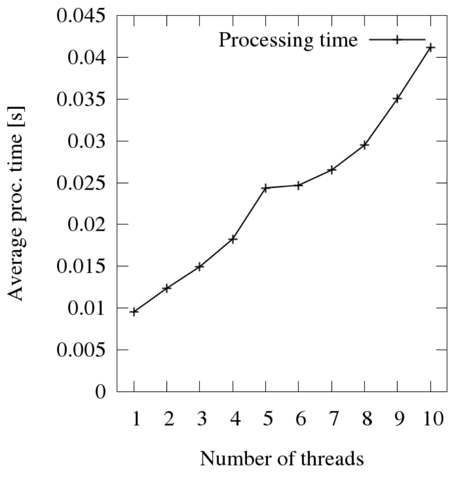

Fig. 11 Average processing time of Credibility Calculation.

queue length drop sharply at $N_C = 2$, then gradually decreases up to $N_C = 5$.

Figure 14 shows the average processing time of the Credibility Calculation for 100 requests. Since each of 100 workers send one request per second, the average processing time over one second indicates that the processing capacity of the master is not sufficient for the request rate. Figure 14 shows that the processing time for the Credibility Calculation can be significantly decreased by increasing N_C. For a VC system deployed in an Intranet, a VC server of $N_S = N_C = 1$ and $N_C = 5$ is sufficient to handle 100

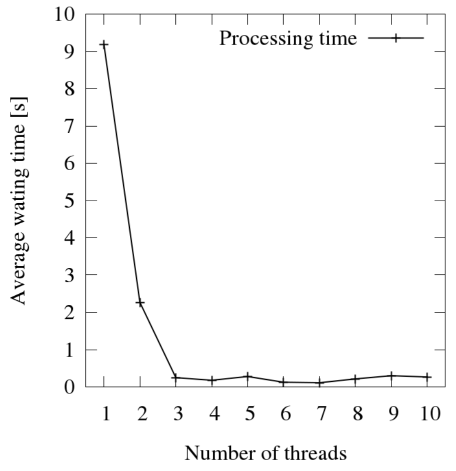

Fig. 12 Average waiting time of Credibility Calculation.

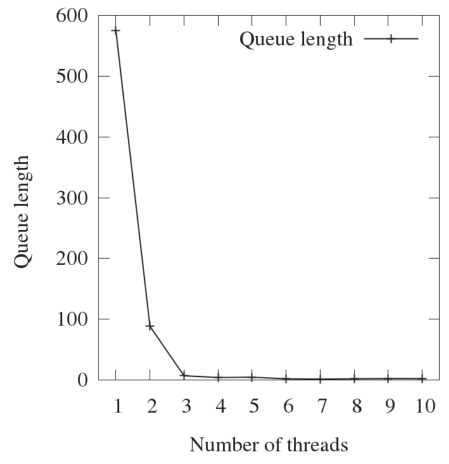

Fig. 13 Average queue length of Credibility Calculation.

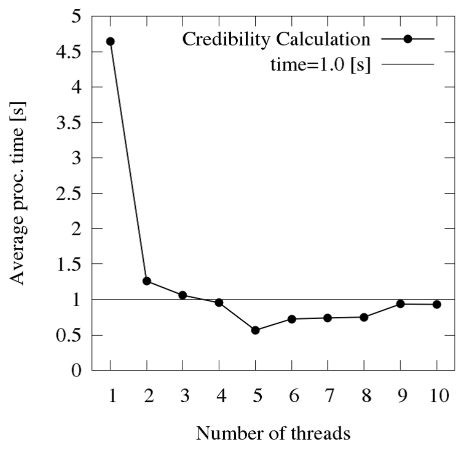

Fig. 14 Average processing time of Credibility Calculation (100 requests).

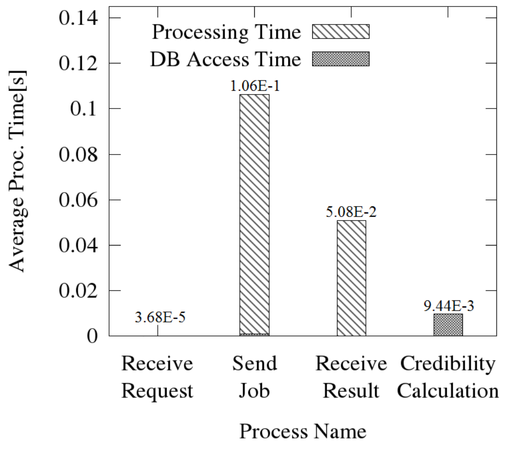

Fig. 15 Average processing time (communication delay: 50 ms).

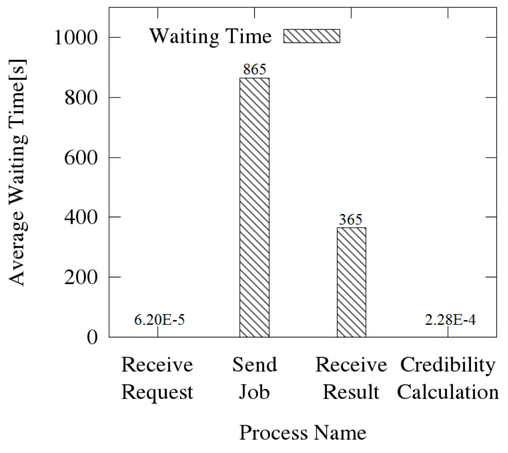

Fig. 16 Average waiting time (communication delay: 50 ms).

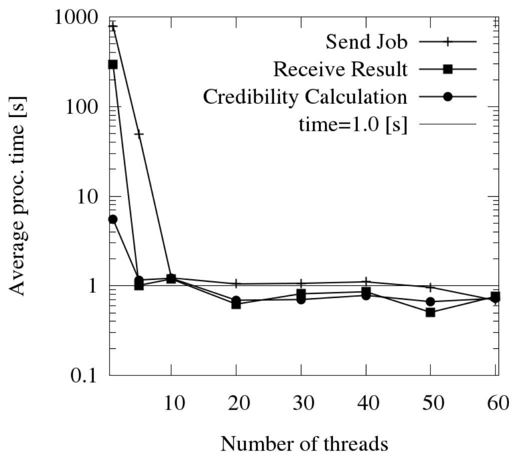

Fig. 17 Average processing time (communication delay: 50 ms).

requests per second.

4.3.2 Results Under Additional Delay of 50 ms

Next, a communication delay of 50 ms is added by the tc command. This corresponds to the delay time in a domestic Internet, e.g. within Japan.

Figures 15 and **16** show the experimental results for the VC server with $N_S = N_R = N_C = 1$. In contrast to the results in Fig. 9 and Fig. 10, the processing time and waiting time of Send Job and Receive Result threads are increased due to the communication delay. In a Send Job thread, two-stage communication occurs between the master and a worker for one request; that is, the master first sends a job ID and then the job file to the worker. Thus, the processing time and the waiting time are about twice as large as those of the Receive Request. These results indicate that the VC server of $N_S = N_R = N_C = 1$ will eventually hang up due to poor performance.

Figure 17 shows the average processing time of each thread for 100 requests. The results show that, in the case of a network with a delay of 50 ms, a VC server of $N_S = 60$, $N_R = 20$ and $N_C = 20$ is required to handle 100 requests in a second.

We investigated system configurations for a request rate R

Table 4 Feasible system configuration.

request rate (per second)	Send Job	Receive Result	Credibility Calculation
20	11	3	3
40	11	4	4
60	15	6	6
80	21	11	10
100	43	16	16

which can handle R requests in one second. **Table 4** shows the system configurations that consist of the minimum number of threads. For $R = 100$, a feasible system configuration is found to be $N_S = 43$, $N_R = 16$ and $N_C = 16$. In the case of SETI@Home, R is about 23 [1]; hence, the above server has adequate performance for practical use.

We discuss the relationship between the average request rate, average number of workers, and job's granularity. In the computation model presented in Section 2, one worker sends a Job Request to obtain a job, processes the received job, and sends a Send Result Request to return the calculation result. Suppose that all the three processes are completed in T seconds. Then, the average request rate R can be represented by $R = 2W/T$, provided that all W workers ideally return the results in T seconds. This assumes the worst case scenario. If some workers do not return the results, the master will receive requests less than $2W/T$ in T seconds. We can roughly estimate W using R and T. The VC environment of $T = 1{,}000$ and $R = 100$ roughly corresponds to that of $W = 50{,}000$ workers. Moreover, once the average W and T are estimated, we can determine a feasible system configuration for the server using our VC system. For a large scale VC over several hundred thousand workers, we need to allocate VC servers across several regions to distribute the processing load. It is expected that the proposed system and the above discussion will help us to decide the specification, the number, and the configuration of VC servers.

5. Conclusion

In this paper, we developed a VC system with credibility-based voting. In the proposed VC server, each process that includes credibility-based voting is multithreaded and all information is managed in a DB for high performance and stable operation. We found by performance evaluations that the credibility-based voting and the process for sending a job are the bottlenecks in an Intranet and the Internet, respectively. Moreover, multithreading is shown to be effective for these performance bottlenecks, and feasible system configurations are also revealed for a variety of request rates. These are new results that cannot be obtained theoretically.

As future work, we will deploy the developed VC system in a real network environment to show the feasibility of a highly-reliable VC.

Acknowledgments This work was supported by the Program to Disseminate Tenure Tracking System, MEXT, Japan and JSPS KAKENHI Grant Number 15K00171.

References

[1] SETI@home. (online), available from ⟨http://setiathome.berkeley.edu/⟩ (accessed 2015-10-15).

[2] Folding@home. (online), available from ⟨http://folding.stanford.edu/⟩ (accessed 2015-10-15).

[3] distributed.net. (online), available from ⟨http://www.distributed.net/⟩ (accessed 2015-10-15).

[4] Rosetta@home. (online), available from ⟨http://boinc.bakerlab.org/rosetta/⟩ (accessed 2015-10-15).

[5] Knispel, B. et al.: Pulsar Discovery by Global Volunteer Computing, *Science*, Vol.329, No.5994, p.1305 (2010).

[6] Virtual LHC@home. (online), available from ⟨http://lhcathome2.cern.ch/test4theory/⟩ (accessed 2014-02-06).

[7] Kondo, D., Araujo, F., Malecot, P., Domingues, P., Silva, L.M., Fedak, G. and Cappello, F.: Characterizing Error Rates in Internet Desktop Grids, *Proc. 13th European Conference on Parallel and Distributed Computing*, pp.361–371 (2007).

[8] Domingues, P., Sousa, B. and Silva, L.M.: Sabotage-tolerance and trust management in desktop grid computing, *Future Generation Computer Systems*, Vol.23, No.7, pp.904–912 (2007).

[9] Ling, X., Hong, W., Takizawa, H. and Kobayashi, H.: A Reliability Model for Result Checking in Volunteer Computing, *Proc. International Symposium on Applications and the Internet*, pp.201–204 (2008).

[10] Litzkow, M.J., Livny, M. and Mutka, M.W.: Condor - a hunter of idle workstations, *Proc. ICDCS*, pp.104–111 (1988).

[11] Sato, M., Boku, T. and Takahashi, D.: OmniRPC: A Grid RPC System for Parallel Programming in Grid Environment, *IPSJ Trans. ACS*, Vol.44, No.SIG11, pp.34–45 (2003).

[12] Foster, I. et al.: Globus: A Metacomputing Infrastructure Toolkit, *J. HPCA*, Vol.11, No.2, pp.115–128 (1997).

[13] Ninf-C. (online), available from ⟨http://ninf.apgrid.org/⟩ (accessed 2014-03-12).

[14] Virtual BOINC. (online), available from ⟨http://boinc.berkeley.edu/⟩ (accessed 2013-09-06).

[15] Sarmenta, L.F.G.: Sabotage-tolerance Mechanisms for Volunteer Computing Systems, *Future Generation Computer Systems*, Vol.18, No.4, pp.561–572 (2002).

[16] Sakai, T. and Fukushi, M.: Implementation of A Reliable Volunteer Computing System with Credibility-based Voting, *Proc. of Second International Symposium on Computing and Networking, CANDAR 2014*, pp.345–359 (2014).

[17] Zuev, Yu.A.: On the Estimation of Efficiency of Voting Procedures, *Theory of Probability and its Applications*, Vol.42, No.1, pp.73–81 (1998).

[18] Casanova, H.: Benefits and Drawbacks of Redundant Batch Requests, *Journal of Grid Computing*, Vol.5, No.2, pp.235–250 (2007).

[19] Watanabe, K., Fukushi, M. and Horiguchi, S.: Expected-credibility-based Job Scheduling for Reliable Volunteer Computing, *IEICE Trans. Inf. Sys.*, Vol.E93-D, No.2, pp.306–314 (2010).

[20] Watanabe, K., Fukushi, M. and Kameyama, M.: Adaptive Group-Based Job Scheduling for High Performance and Reliable Volunteer Computing, *Journal of Information Processing*, Vol.19, pp.39–51 (2011).

Takeshi Sakai received his B.E. degree from Yamaguchi University in 2014. He is currently a master grade student in the Graduate School of Science and Engineering at Yamaguchi University. His research interest is high-performance distributed systems.

Masaru Fukushi received his B.Sc. and M.Sc. degrees from Hirosaki University in 1995 and 1997, respectively, and his Ph.D. degree in information science from the Graduate School of Information Science at Japan Advanced Institute of Science and Technology (JAIST) in 2002. He is currently an associate professor in the Graduate School of Science and Engineering at Yamaguchi University. Prior to joining in the Yamaguchi University, he was an assistant professor in the School of Information Science at JAIST from 2002 to 2004, and in the Graduate School of Information Sciences at Tohoku University from 2004 to 2012. His research interests include dependable parallel VLSI architectures, dependable and high-performance distributed systems, and parallel and distributed computing. Dr. Fukushi is a member of IEEE, IPSJ, IEICE.