Work Stealing Based Volunteer Computing Coordination in P2P Environments

Wei Li and William W. Guo
School of Engineering & Technology, Central Queensland University, Australia
Email: w.li@cqu.edu.au; w.guo@cqu.edu.au

Abstract —This paper aims at the evaluation of work stealing based Volunteer Computing (VC) coordination with the goal of confirming the scalability of VC for Peer-to-Peer (P2P) environments. Our previous work has successfully modelled work stealing for VC coordination and been evaluated by a few applications and a small number of machines. However, this paper argues that the evaluation of scalability of VC by the statistical data of real world applications or through mathematical modelling is either limited by the number of volunteer machines or difficult to achieve because of the peer churn in P2P opportunistic environments. This paper proposes a simulation model for the same VC functions but performing the virtual work so that the statistical data can be quickly obtained for the dynamic behaviors of a large number of volunteers. The initial evaluation results have demonstrated that the work stealing based VC coordination scales up to 10K volunteers against varying churn rate, communication cost and stealing granularity. The confirmation of scalability ensures that VC can be effectively applied to P2P opportunistic environments.

Index Terms—Work stealing, simulation, volunteer computing, peer-to-peer

I. Introduction

In parallel computing, work stealing was originally proposed by Blumofe & Leiserson [1] as a scheduling strategy of multithreading, where the underutilized processors steal threads from overloaded processors. Work stealing was demonstrated effective for dynamic work load balancing in multicore and shared or distributed memory systems [2]-[4], where parallel computations use processors in a time-varying manner. Work stealing makes every processor busy in such a dynamic environment to maximize the overall speedup.

Volunteer Computing (VC) [5] is to gain the potential computing capacity from millions of volunteer computers from the Internet for large-scale scientific computations. VC releases the reliance on expensive super-computers and therefore has attracted a large amount of research effort and has been applied to a large number of scientific projects [6] such as SETI@home [7]. VC is generally implemented through a centralized master/worker structure [8], which is criticized for poor scalability or reliability, susceptibility to churn (join, leave and crash of volunteers) and inefficient load balancing mechanisms [9]-[11].

Peer-to-Peer (P2P) is a distributed architecture in which all computers act as peers to connect over the Internet and share distributed resources with no distinction between the role of client and server. There is no centralized or hierarchical control in a P2P overlay, and each peer has identical functionality. When P2P environments are heterogeneous in computing power, storage and network bandwidth, and dynamics (peer churn), their merits such as no single point of failure, decentralization and self-organization, are the significant benefits for many distributed applications in terms of scalability and adaptability [12]-[14].

To our best knowledge, P2P based VC has not been well modelled to adapt to churn of peers or to ensure the scalability with the increase of peer numbers. For example, Dou *et al.* [13] attempted a P2P approach by forming volunteers into an unstructured cluster, but their model did not cope with work and data lost or the maintenance of an effective neighborhood when peers left or crashed. Ni & Harwood [15] used peers to form a tuple space as the work and result storage, where a peer was able to contribute to storage space, CPU cycles or both. However, peer communication in such an unstructured P2P overlay had to be message flooding, which incurred significant bandwidth usage. Zhao *et al.* [16] built VC coordination as a decentralized P2P system. Their unstructured P2P overlay was not one of the currently stability-or scalability-proved P2P overlays. If some peers crashed, the overlay would break into multiple parts and could not be reunited.

The above situation has motivated us to transform work stealing into P2P environments to model VC coordination. The work stealing based VC coordination model in our previous work [17] was based on Chord [18] P2P protocol and therefore has naturally inherited the proved performance, reliability and scalability of Chord; the model was evaluated as effective in a distributed environment. However, our previous evaluation with the limit number of peer machines could not give us enough confidence to answer whether work stealing scales for VC vs. a large number of peers. We argue that the evaluation of scalability through real world applications is difficult to achieve in two ways. P2P has no central server to perform data statistics or putting a peer as such a server is impractical in terms of peer churn or computing

Manuscript received July 26, 2017; revised October 12, 2017.
Corresponding author email: w.li@cqu.edu.au.
doi:10.12720/jcm.12.10.557-564

power or storage capacity. In addition, evaluation by mathematical modelling is impractical because of the uncertainty that is brought by peer churn. Those issues have motivated us to propose a simulation model in this paper to be fully compatible with the real work stealing based VC coordination model of our previous work. On this basis, this paper evaluates the scalability of work stealing for VC coordination in a virtual P2P environment for up to 10K peers against different churn rate, communication cost and stealing granularity of the entire work.

The rest of this paper has been structured as follows: related work is reviewed in Section 2. The work stealing based P2P VC coordination is briefly reviewed in Section 3. Section 4 describes the necessity of a simulation model for the evaluation of the scalability of work stealing in P2P environments. Section 5 details the simulation model for virtual work and virtual P2P environments. The initial evaluation of scalability of the work stealing VC coordination is detailed with results and analysis in Section 6. Section 7 concludes the initial evaluation that the work stealing based VC coordination scales for 10K peers in P2P opportunistic environments.

II. Related Work

The performance of work stealing has been studied in general or on particular concerns by the current literature. Tallent & Mellor-Crummey [19] proposed a profiling strategy to identify performance bottlenecks in work stealing based computations. They also implemented an HPCToolkit to quantify parallel idleness (waiting for work) and overhead (working on non-user code). They aimed to find the regions of a given computation that needed concurrency but failed with parallelization because of idleness and overhead. Their studies of the computations in Cilk language demonstrated that a decrease in stealing granularity could enhance parallel efficiency for the computation with high idleness and low overhead. On the contrary, an increase in stealing granularity could reduce overhead for the computation with high overhead and low idleness. However adjustment of stealing granularity would not help for high overhead and inefficient parallelism situations.

Perarnau & Sato [4] evaluated the different victim selection strategies for work stealing performance on K Computer, which is a supercomputer consisting of 80K nodes (each with 8 cores) and distributed memories. Among Deterministic Selection and Random Selection with Skewed Distribution and with or without Half-Stealing, Random Selection with Skewed Distribution and Half-Stealing behaved the best performance to scale up to 8,192 nodes.

Dinan *et al.* [2] designed and implemented a runtime system to support work stealing on distributed memory systems. Different work stealing strategies named ARMCI (Aggregate Remote Memory Copy Interface) Locks, Spin Locks and Spin Locks with Aborting Steals were evaluated by the Bouncing Producer-Consumer (BPC) benchmark, Unbalanced Tree Search (UTS) benchmark and Madness 3d Tree Creation Kernel (Madness) benchmark on a HP cluster with 2,310 nodes. When the other 2 strategies scaled up to 8,192 processors on UTS benchmark but only scaled to 6,144 processors on BPC and Madness benchmarks, the Spin Locks with Aborting Steals scaled up to 8,192 processors on all 3 benchmarks.

Kumar *et al.* [20] identified the key sources of overhead in work stealing, i.e. sequential overhead (from the special support from the runtime system for the initiation, state management and termination) and steal ratio (the fraction of tasks actually stolen). They optimized X10WS runtime system into X10WS (OffStack) by avoiding to maintain an explicit deque but allowing the runtime to extract the information from the worker's call stack; they also modified X10WS compiler to compile computations to X10WS (Try-Catch) to reduce the overhead of exception handling for the work stealing related operations. The evaluation results showed that those optimized runtime systems had better performance (about 15% overhead on an Intel Xeon E7530 machine with 12 cores for a number of embarrassingly parallel computations) than the traditional Fork-Join and original X10WS runtime systems.

Vu & Derbel [21] focused on designing an effective work stealing algorithm to deal with the heterogeneity of linked parallel computing resources, which were assumed to have heterogeneous computing and communication capacities. Their studies were closer to distributed environments such as computing grids but still away from P2P environments because they did not consider churn. Their proposed algorithms were the fine tune of the existing Probabilistic Work Stealing (PWS) and Adaptive Cluster-aware Random Stealing (ACRS) by introducing new adaptive control operations to increase work locality and decrease stealing cost. The evaluation results showed that the proposed algorithms could save about 30% computing time on the experimental environment of 16 clusters with 128 nodes.

Although the above studies have provided both a broad and a deep view of the performance of work stealing and there are more studies [3], [22], [23] in this domain, they are based on parallel or distributed environments with a certain number of stable computing nodes (most of the cases are homogeneous and in some cases are heterogeneous). The issue, whether work stealing is effective and scalable for VC in P2P opportunistic environments, remains open. This paper aims at filling up the gap by confirming the scalability to clarify the issue.

III. The Work Stealing Based P2P VC Coordination

Our previous work [17], [24] has modelled the VC coordination framework as a Chord [18] ring formed by available volunteers to take the advantages of theoretically proved performance, scalability and reliability of Chord protocol in P2P environments. Volunteers use the standard Chord operations to join and leave the P2P community or crash at any time. When a volunteer joins, it will collect and then compute a piece of unfinished work that is left by a peer who has already left or crashed. If there is no such a left-over work piece, a peer will steal a piece of work from another working peer, which splits its current work and yields half of the unfinished portion of the work to the thief peer. By the time a peer leaves or crashes, the finished portion of a left peer's work piece is effective and counted for the overall progress of the entire work. However, the whole work piece of a crashed peer needs to be recomputed. A peer returns the computing results back to the Chord ring and the termination condition is that the overall progress of the entire work is 100%. The work stealing model adapts to the heterogeneity of volunteers in terms of computing power, storage capacity and network bandwidth. No matter what the original distribution of work pieces is, a faster peer can dynamically obtain more work pieces to keep busy all the time. In addition, the model adapts to the churn of peers. When a new peer joins or an existing peer leaves or crashes, the existing work distribution is no longer valid. Work stealing is able to reflect churn and therefore re-balance workload among the dynamic peers. As a consequence, the overall speedup is maximized. The formal description of the model is as follows for a general VC scenario with peer churn considered.

- There are n number of peers $P = \{p_1, p_2, \ldots, p_n\}$, where p_1 is the work owner and the others are pure volunteers.
- The compute-capacity (in terms of computing time) for a peer $p \in P$ to independently solve the whole given VC problem (e.g. the N-Queen problem) is C_p.
- The work owner p_1 starts the work from time point 0. When another peer joins the community, it will get a piece of work to compute at time point jt_p. The join time points of all peers comprise the set $JT = \{jt_p\}$, where $p \in P$.
- Some peers, which comprise the set L and where $L \subset P$, will leave the community before the completion of the entire work. Leave means that the partial result is valid. That is, for $p \in L \wedge p \neq p_1$, when p leaves, the result of finished portion of the current piece of work of p is valid and the unfinished portion will be picked up by another peer.
- Some peers, which comprise the set CR and where $CR \subset P \wedge (CR \cap L = \phi)$, will crash. Crash means that the partial result is invalid. That is, for $p \in CR \wedge p \neq p_1$, it just crashes but is not able to upload any partial results. That is, if the last piece of work was accepted by p at time point $at_{last\text{-}p}$ and the time point when the peer p crashes is ct_p, the computing between $at_{last\text{-}p}$ and ct_p is totally wasted. The whole piece of work will be picked up by another peer to recompute.
- Except for the work owner, all the leave or crash time points of peers comprise the set $LCT = \{lt_p\}$, where $p \in L \cup CR$ and when a peer as the last peer to leave or crash, the entire work has not been completed.
- Except for the work owner, the time point of the last *join* or *leave* or crash of a peer is t_{last}, where $t_{last} \in JT \cup LCT$ and $\forall t(t \in JT \cup LCT \rightarrow t \leqslant t_{last})$ is true. After t_{last}, there will be no more join or leave or crash of peers and the entire work will be completed by the community $P\text{-}L\text{-}CR$.

Our previous work has successfully modelled such a work stealing based VC coordination for pure P2P environments by using the standard Chord protocol [18]. The model has been successfully implemented in Java by using the Open Chord APIs [25].

IV. The Necessity of VC Simulation

Our previous model has been evaluated for the effectiveness by a small number of peer machines in a distributed environment [17], [24]. Although the results showed linear speedup, it could not give us enough confidence on whether the model would scale for a large number of peers with churn, varying communication cost and stealing granularity. It could be argued that another way of evaluation is mathematical modelling. However, this section will describe why such a way is very difficult due to the uncertainty that is brought by peers' churn.

Based on the formal work stealing model as described in Section 3, we assume that the computing of a peer with compute-capacity of C_p is paused several times for stealing or supplying a piece of work or uploading results for t_s long in the time period t_1 to t_2. In that situation, the compute-capacity C_p of the peer needs to be adjusted by formula (1).

$$C_p^{t_1 \sim t_2} = \frac{t_2 - t_1 - t_s}{t_2 - t_1} C_p \tag{1}$$

That capacity is called adjusted capacity for the time period of t_1 to t_2 and denoted as $C_p^{t_1 \sim t_2}$. Under such an adjustment, the computing time for the entire work by the peers with churn as described in Section 3 will be determined by formula (2), where $t_{final\text{-}p}$ is the time point of peer p when it completes the last piece of the entire work and WL is the computing load of the entire work in terms of computing time.

$$\frac{WL - \left(\sum_{p \in P-L-CR} \frac{t_{last} - jt_p}{C_p^{jt_p \sim t_{last}}} + \sum_{p \in L} \frac{lt_p - jt_p}{C_p^{jt_p \sim lt_p}} + \sum_{p \in CR} \frac{at_{last-p} - jt_p}{C_p^{jt_p \sim at_{last-p}}} \right)}{\sum_{p \in P-L-CR} \frac{WL}{C_p^{t_{last} \sim t_{final-p}}}} + t_{last} \tag{2}$$

For a given VC scenario, although jt_p, lt_p, ct_p and t_{last} could be predetermined or calculated for the evaluation, t_s in formula (1) and $t_{final\text{-}p}$, $at_{last\text{-}p}$ in formula (2) cannot be predetermined/calculated. These dynamic factors, t_s, $t_{final\text{-}p}$ and $at_{last\text{-}p}$, are determined by:

- The randomness from whom a peer steals a piece of work.
- The stealing granularity (such as half-stealing or someway else).
- The stealable portion of a piece of work that is based on the computing progresses of each peer.

Consequently, the speedup of the given scenario cannot be obtained by the calculation of using formula (1) and (2). For example, the first peer p_1 is the work owner, when the second peer p_2 joins, it is certain that p_2 will steal a piece of work from p_1. However when the *i*th peer p_i joins, it could steal a piece of work from $p_1, p_2, \ldots, p_{i-1}$, depending on whom p_i is going to contact, the availability of p_1 to p_{i-1} for servicing a piece of work, and whether the current piece of work of p_1 to p_{i-1} is splittable. Therefore VC speedup in P2P environments can only be simulated rather than mathematically calculated.

V. The Simulation Model

The determination of the entire work (computing) load of a VC work is modelled relatively to peer compute-capacity in terms of computing time. Each peer owns a certain compute-capacity C_p. In real world applications, this C_p can be obtained by testing the computing time of a predefined benchmark on full concentration. If a standard compute-capacity C is chosen, the workload WL of the entire work will be certain. For example, the workload WL of 800M Time Units (TUs) means that a peer with the compute-capacity C_p that is equal to the standard compute-capacity C needs 800M TUs to complete the entire work on its own, where a TU could be a second, a hour or a day etc., and M stands for a million. Thus if a peer's compute-capacity C_p is half or doubled of the standard C, it can finish the same work in 1.6G or 400M TUs respectively, where G stands for a billion.

The computing progresses in different speeds at each peer in accordance to the peer's compute-capacity. When a peer is assigned a piece of work, its computing will progress step by step. A certain number of steps will complete a TU. For example, depending on the simulation requirement, a step or 10 steps could progress a TU. To describe in another way, if a peer with capacity C_p can progress a TU by a single step, another peer with compute-capacity of $C_p/10$ will progress a TU by 10 steps.

Peers commit churn in terms of join, leave or crash. A peer can join at any time. Once it joins, it starts to pick up a left piece of work or steal a piece of work from another peer to compute. A peer can leave at any time, e.g. in computing, uploading results or searching for another piece of work. If it leaves whilst computing, its current progress is treated as valid. The progress is check-pointed and the left work will be picked up by other peers in the future. If it leaves when uploading results, the model allows it to finish the uploading. A peer can crash at any time. Whilst a peer crashes, its current progress is treated as invalid. The whole piece of work of the peer will be re-computed by another peer who picks it up in the future. In simulation, a peer is assigned a join time, which is the time point in terms of TU since the start of the entire work from time point 0, or a leave or crash time if it leaves or crashes in the future. The churn of the peer will occur when the current simulation time matches those leave or crash time points.

The communication cost is counted for stealing a piece of work or uploading the result of a completed piece of work. When stealing a piece of work from another working peer or picking up a piece work from a left or crashed peer, a peer will pause for a certain time in terms of TU. When supplying a piece of work to another peer, a peer will pause for a certain time as well. The pause reflects the communication cost. During the paused time, a peer will not be able to do anything else except for the current demanding or supplying.

The stealing granularity of a piece of work is controllable. When the current piece of work is bigger than a predefined granularity in terms of TUs, the piece of work is splittable. Otherwise it is not splittable and the requesting peer must search another peer for available pieces of work. When a piece of work is split, the unfinished portion is divided into 2 halves in terms of TU and one half is sent to the requesting peer.

Every peer is modelled by a finite state machine; a peer exhibits 3 states: servicing, computing and terminating during its life cycle. A computing peer (i.e. a peer in computing state) is computing a piece of work to progress according to its compute-capacity. A computing peer can change to the servicing state if it completes current work to upload results or to steal a piece of work from another peer, or its work is being stolen and it is supplying a portion of it. A servicing peer will return to the computing state if it completes supplying work or receives a new piece of work. A computing or servicing peer will change to the terminating state if it is to leave or crash or there is no available work. If a terminating peer is to leave, it will upload the partial results; if a terminating peer is to crash, it will not do anything. A terminating peer will never go back to any other states. Such state changes of a peer are showed in Fig. 1.

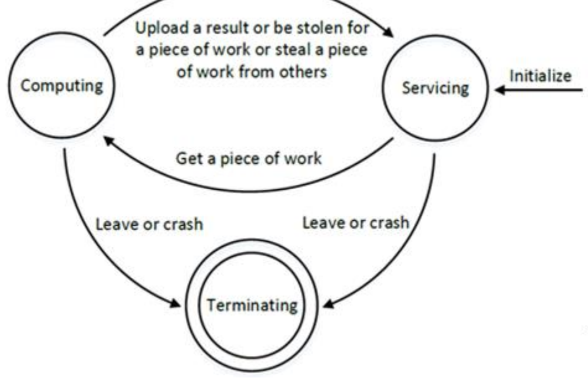

Fig. 1. The state change of a peer

The simulation procedure is to manage the set of all state machines for thousands of peers. The simulation monitor needs to initialize (according to the join time) every peer to the servicing state to get a piece of work, change the states between servicing and computing for many times, and change the computing or servicing peers to the terminating state (according to the leave or crash time). The termination condition of the entire work for all peers is the overall progress of 100%. The overall speedup is the division of *WL* by the termination time of the simulation. The simulation monitor is showed in pseudo code in Fig. 2.

```
/* Set simulation scenario such as the overallProgress=0,
   the currentTime=0 and the computing load of the entire work WL
   and peer profiles such as compute capacity etc.
*/
setScenario();
initialise(); //Initialise the work owner into computing state
while (overalProgress!=100) {
      /* Leave or crash a peer if its leave or crash time is due,
         i.e. change the peer’s state into terminating
      */
      leavePeers();
      crashPeers();
      /* Join a peer if its join time is due, i.e.
         initialise the peer into servicing state.
         A newly joining peer can make another computing peer into
         the servicing state if the latter is stolen for a piece
         of work.
      */
      joinPeers();
      /* Make progress for every peer for 1 step. A progress may
         result in a servicing peer into the computing state
         (if it gets a piece of work or if it finishes supplying
         a piece of work) or a computing peer into the servicing state
         (if it finishes its current work to upload the result or
          is stolen for supplying a piece of work).
      */
      setCurrentTime(getCurrentTime()+1);
      makeProgress();
      /* Collect the current available results and
         count for the overallProgress.
      */
      collectResult();

}
```

Fig. 2. The simulation monitor

VI. Evaluation of Scalability

The evaluation has been performed by three particular settings to assess the influence of churn, communication cost and stealing granularity on the scalability of the model with increasing number of volunteers. The three particular settings are based on our investigations into a real world application SETI@home [26]. Everyday SETI collects 35GB data to process. A work unit is 0.25MB, so the number of units is 35GB/0.25GB=140K. A work unit needs some additional information. Consequently, a SETI work unit is 0.34MB (340KB). The return result of a work unit is 64KB. Based on the available statistical data from SETI, each work unit takes about 18 to 25 hours to process. Thus each day needs 140Kx18 or 140Kx25=2,520,000 to 3,500,000 (on average 3M) hours of computing time with a 233MHz or 300MHz computer.

A test of the speed of internet connection by ADSL2+ (a very common internet plan for home use) was 438KB/s for downloading and 81KB/s for uploading. We can assume that downloading a work unit or uploading the result of a work unit is less than 1 second. Based on the above data, downloading a work unit or uploading the result of a work unit is on average 1/10G of the total computing load, where G stands for billion. Similarly, downloading 10K work units or uploading the results of 10K work units is on average 1/1M of the total computing load.

A. *The Scalability against Churn*

The setting of the overall workload *WL* is 800M TUs that are big enough to simulate a common VC work. The download of a piece of work is 80 TUs, which are 1/10M of *WL*. The upload of a result is 40TUs, which are 1/20M of *WL*. The setting of download and upload time is big enough to simulate the task exchange of an embarrassingly parallel computing. The stealing granularity of the entire work is 80 TUs, which are 1/10M of *WL* and small enough for a common VC work. The numbers of peer of this evaluation are set to 2K, 4K, 6K, 8K and 10K and peers join the community sequentially in every 20 TUs (randomly chosen). The standard compute-capacity of peer is 800M TUs and half peers have the standard capacity and the other half peers have the capacity of 400M TUs (half of the standard capacity). The numbers of churn peers are set to 10%, 30%, 50%, 70% and 90% of the total peers, of which half leave and the other half crash. The leave or crash peers are distributed from the middle backward and forward. For example, if the number of peers is 8K and the churn rate is 50%, there will be 4K peers to leave or crash. The middle position is P_{4000} and then the first leave or crash peer will be P_{2000} and the last leave or crash peer will be P_{5999}. Peers start to leave or crash when half (randomly chosen) of the total peers have joined. A peer will leave or crash in 20 TUs (randomly chosen), which is the same as the peer join interval.

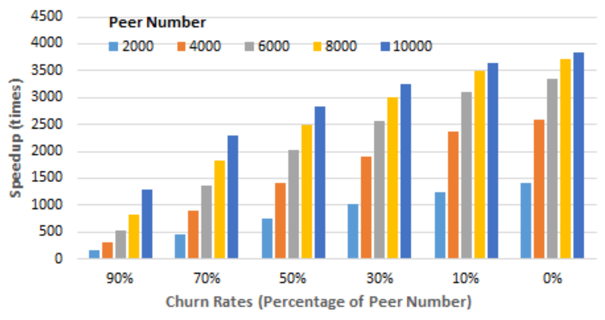

Fig. 3. The speedup vs. different churn rates.

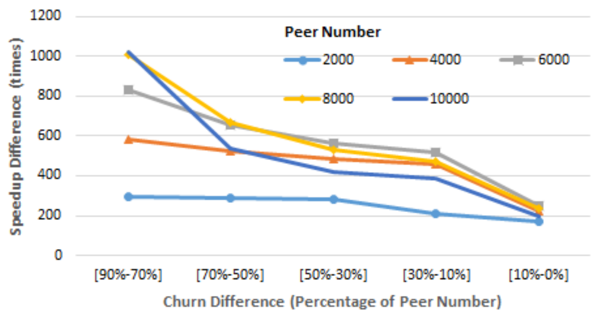

Fig. 4. The speedup differences between neighbor churn rates

The speedup evaluation is reported in Fig. 3. It shows that the speedup scales with the increase of peer numbers vs. different churn rates. The speedup differences between neighbor churn rates are reported in Fig. 4. What can be concluded from the speedup difference vs. churn difference diagram (Fig. 4) is that the speedup is affected much more significantly by a higher churn rate than a lower churn rate for a given number of peers. That is confirmed by the observation that for a given number of peers, the speedup difference always decreases with the same churn difference (20% or 10%), starting from the highest churn difference of [90% -70%] to the lowest churn difference [10% -0%].

Another observation from Fig. 4 is that 10,000 peers show mostly lower speedup differences for a churn difference range of [70% -10%] than that for 4,000, 6,000 and 8,000 peers. For example, for 20% churn difference between 70% and 50% churn rate, the speedup difference is 539 times for 10,000 peers but 666 times for 8,000 peers. However, for 20% churn difference between 90% and 70% churn rate, the speedup difference is 1,020 times for 10,000 peers but 1,009 times for 8,000 peers. From the above, we cannot draw the conclusion that given a churn difference, the speedup difference is directly proportional or inversely proportional to peer numbers. The reason for such a uncertainty comes from two aspects: first 20% churn rate incurs more peers (2,000) to leave or crash for 10,000 peer overlay than that (1,600) for 8,000 peer overlay, but it also keeps more peers (8,000) working for 10,000 peer overlay than that (6,400) for 8,000 peer overlay. Second, a peer contributes more to the speedup if it commits churn in the later stage of the computation. On the contrary, a peer contributes less to the speedup if it commits churn in the earlier stage of the computation. However, when peers leave or crash is random. Based on the above and in fact comparing speedup differences vs. peer numbers is meaningless in the scalability evaluation against churn rate in this paper. In short the useful conclusions are: Fig. 3 shows the scalability of VC in terms of peer numbers vs. churn rates and Fig. 4 shows a higher churn rate affects more on speedup than a lower churn rate does.

B. The Scalability against Communication Cost

The setting of this evaluation is the same as the setting of *A* except:

- The churn rate is fixed as 30%.
- The communication cost varies for stealing a piece of work and uploading the result of a piece of work as 8K TUs and 4K TUs (1/100K and 1/200K of the entire *WL* of 800M TUs), 4K TUs and 2K TUs (1/200K and 1/400K of the entire *WL*), 2K TUs and 1K TUs (1/400K and 1/800K of the entire *WL*), 800 TUs and 400 TUs (1/1M and 1/2M of the entire *WL*), 80 TUs and 40 TUs (1/10M and 1/20M of the entire *WL*) and 8 TUs and 4 TUs (1/100M and 1/200M of the entire *WL*).

The speedup evaluation is reported in Fig. 5, where 100K/200K represents 1/100K (work download) and 1/200K (result upload) of the entire *WL* of 800M TUs to shorten the labels. It shows that the speedup scales with the increase of peer numbers vs. different communication cost. The speedup differences between neighbor communication cost are reported in Fig. 6. It shows that the speedup is affected much significantly by higher communication cost than lower communication cost for any number of peers.

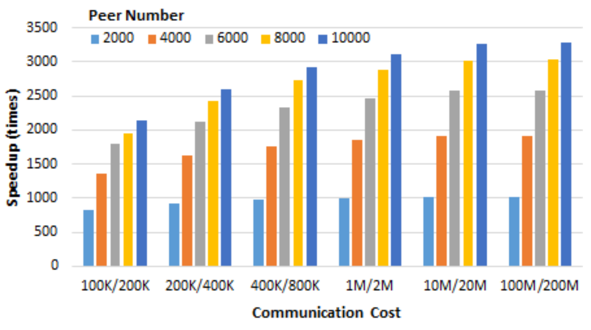

Fig. 5. The speedup vs. different communication cost.

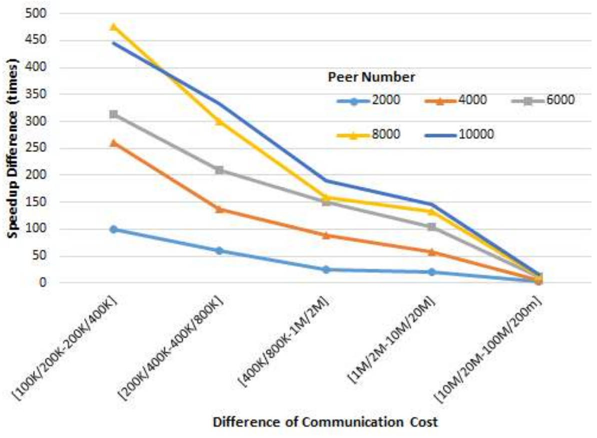

Fig. 6. The speedup differences between neighbor communication cost.

C. The Scalability against Stealing Granularity

The setting of this evaluation is the same as the setting of *A* except:

- The churn rate is fixed as 30%.
- The stealing granularity varies as 8K TUs (1/100K of the entire *WL* of 800M TUs), 4K TUs (1/200K of the entire *WL*), 2K TUs (1/400K of the entire *WL*), 1K TUs (1/800K of the entire *WL*), 800 TUs (1/1M of the entire *WL*) and 80 TUs (1/10M of the entire *WL*).

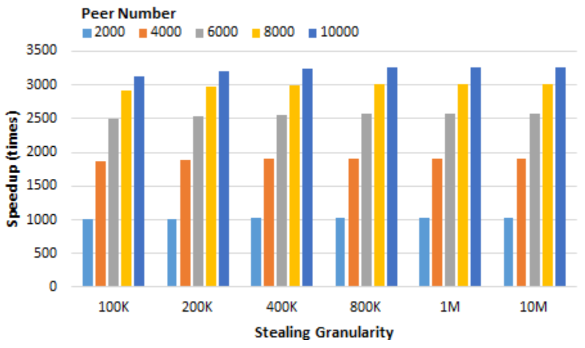

Fig. 7. The speedup against different stealing granularities.

The speedup evaluation is reported in Fig. 7, where 100K represents 1/100K of the entire *WL* of 800M TUs to shorten the labels. It shows that the speedup scales with the increase of peer numbers vs. different stealing granularities. The speedup differences between neighbor stealing granularities are reported in Fig. 8. It shows that the speedup is affected much significantly by coarse grained works than by fine grained works.

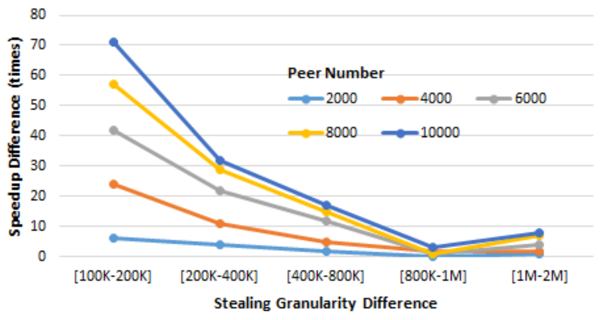

Fig. 8. The speedup differences between neighbor stealing granularities

VII. Conclusions

Work stealing based volunteer computing has been modelled for P2P environments and the effectiveness of the model has been evaluated for a small number of volunteer machines [17]. This paper transforms the model into a simulation version to evaluate the model's performance, not being influenced by the underlying hardware limits (such as the number of machines) and conditions (such as physical computing time). The results from three evaluations have confirmed that the work stealing based VC coordination scales for a larger number (up to 10,000) of volunteers in P2P opportunistic environments against different churn rates, communication cost and stealing granularities of the entire work. This implies that VC can be effectively applied to P2P opportunistic environments.

Future work goes into 2 directions. More intensive evaluations for scalability against a very large number of volunteers such millions will be conducted by using an optimized simulation algorithm for a higher time efficience in simulation. Remodeling work stealing to fit for non-embarrassingly parallel applications such as data-intensive applications and evaluating its scalability is also a necessity.

References

[1] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM*, vol. 46, no. 5, pp. 720-748, 1999.

[2] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha, "Scalable work stealing," in *Proc. International Conf. on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 53-63.

[3] J. Lifflander, S. Krishnamoorthy, and L. V. Kale, "Steal tree: low-overhead tracing of work stealing schedulers," *ACM SIGPLAN Notices*, vol. 48, no. 6, pp. 507-518, 2013.

[4] S. Perarnau and M. Sato, "Victim selection and distributed work stealing performance: A case study," in *Proc. 28th IEEE International Symp. on Parallel and Distributed Processing*, 2014, pp. 659-668.

[5] L. Sarmenta, "Volunteer computing," PhD thesis, Massachusetts Institute of Technology, 2001.

[6] BOINC. [Online]. Available: http://boinc.berkeley.edu/projects.php

[7] E. J. Korpela, "SETI@ home, BOINC, and volunteer distributed computing," *Annual Review of Earth and Planetary Sciences*, vol. 40, pp. 69-87, 2012.

[8] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proc. Fifth IEEE/ACM International Workshop on Grid Computing,* 2004, pp. 4-10.

[9] D. P. Anderson and J. McLeod, "Local scheduling for volunteer computing," in *Proc. IEEE International Symp. on Parallel and Distributed Processing*, 2007, pp. 1-8.

[10] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in *Proc. IEEE International Symp. on Parallel & Distributed Processing*, 2009, pp. 1-8.

[11] F. Costa, J. N. Silva, L. Veiga, and P. Ferreira, "Large-scale volunteer computing over the Internet," *Journal of Internet Services and Applications*, vol. 3, no. 3, pp. 329-346, 2012.

[12] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335-371, 2004.

[13] W. Dou, Y. Jia, H. M. Wang, W. Q. Song, and P. Zou, "A P2P approach for global computing," in *Proc. Parallel and Distributed Processing Symp.*, 2003, pp. 1-6.

[14] R. Rodrigues and P. Druschel, "Peer-to-peer systems," *Communications of the ACM*, vol. 53, no. 10, pp. 72-82, 2010.

[15] L. Ni and A. Harwood, "P2P-Tuple: Towards a robust volunteer computing platform," in *Proc. International Conf. on Parallel and Distributed Computing, Applications and Technologies*, 2009, pp. 217-223.

[16] Z., Zhao, F. Yang, and Y. Xu, "PPVC: a P2P volunteer computing system," in *Proc. 2nd IEEE International Conf. on Computer Science and Information Technology*, 2009, pp. 51-55.

[17] W. Li, W. Guo, and E. Franzinelli, "Achieving dynamic workload balancing for P2P volunteer computing," in *Proc. 44th International Conf. on Parallel Processing Workshops*, 2015, pp. 240-249.

[18] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17-32, 2003.

[19] N. R. Tallent and J. M. Mellor-Crummey, "Identifying performance bottlenecks in work-stealing computations," *Computer*, vol. 42, no. 12, pp. 44-50, 2009.

[20] V. Kumar, D. Frampton, S. M. Blackburn, D. Grove, and O. Tardieu, "Work-stealing without the baggage," *ACM SIGPLAN Notices*, vol. 47, no. 10, pp. 297-314, 2012.

[21] T. T. Vu and B. Derbel, "Link-heterogeneous work stealing," in *Proc. 4th IEEE/ACM International Symp. on Cluster, Cloud and Grid Computing*, 2014, pp. 354-363.

[22] U. A. Acar, A. Charguéraud, and M. Rainey, "Scheduling parallel programs by work stealing with private deques," *ACM SIGPLAN Notices*, vol. 48, no. 8, pp. 219-228, 2013.

[23] G. Varisteas and M. Brorsson, "DVS: Deterministic victim selection to improve performance in work-stealing schedulers," in *Proc. MULTIPROG 2014: Programmability Issues for Heterogeneous Multicores*, 2014.

[24] W. Li and E. Franzinelli, "Decentralizing volunteer computing coordination," in *Proc. International Conf. of Young Computer Scientists, Engineers and Educators*, 2016, pp. 299-313.
[25] S. Kaffille and K. Loesing, *Open Chord (1.0.4) User's Manual*, The University of Bamberg, Germany, 2007.
[26] SETI@home. [Online]. Available: http://setiathome.ssl.berkeley.edu/

Dr Wei Li holds a PhD degree in computer science from the Institute of Computing Technology of Chinese Academy of Sciences China. He currently works for the School of Engineering & Technology, Central Queensland University Australia. His research interests include dynamic software architecture, P2P volunteer computing and multi-agent systems. Dr Wei Li has been a peer reviewer of a number of international journals, including IEEE Transactions on Software Engineering, ELSEVIER Journal of Systems and Software and John Wiley & Sons Journal of Software Maintenance and Evolution: Research and Practice, and a program committee member of more than 30 international conferences.

Dr William Guo is currently a professor in applied mathematics and computation at Central Queensland University Australia. His research interests include applied mathematics and computational intelligence, simulation and modelling, data mining, and STEM education.