big data and cognitive computing

MDPI

Article

The Optimization Strategies on Clarification of the Misconceptions of Big Data Processing in Dynamic and Opportunistic Environments

Wei Li [1,*] and Maolin Tang [2]

[1] School of Engineering & Technology, Central Queensland University, Rockhampton, QLD 4702, Australia
[2] School of Computer Science, Queensland University of Technology, Brisbane, QLD 4000, Australia; m.tang@qut.edu.au
* Correspondence: w.li@cqu.edu.au

Abstract: This paper identifies four common misconceptions about the scalability of volunteer computing on big data problems. The misconceptions are then clarified by analyzing the relationship between scalability and the impact factors including the problem size of big data, the heterogeneity and dynamics of volunteers, and the overlay structure. This paper proposes optimization strategies to find the optimal overlay for the given big data problem. This paper forms multiple overlays to optimize the performance of individual steps in terms of MapReduce paradigm. The optimization is to achieve the maximum overall performance by using a minimum number of volunteers, not overusing resources. This paper has demonstrated that the simulations on the concerned factors can fast find the optimization points. This paper concludes that always welcoming more volunteers is an overuse of available resources because they do not always bring benefit to the overall performance. Finding optimal use of volunteers are possible for the given big data problems even on the dynamics and opportunism of volunteers.

Keywords: bigdata; optimization; simulation; volunteer computing

Citation: Li, W.; Tang, M. The Optimization Strategies on Clarification of the Misconceptions of Big Data Processing in Dynamic and Opportunistic Environments. *Big Data Cogn. Comput.* **2021**, *5*, 38. https://doi.org/10.3390/bdcc5030038

Academic Editor: Min Chen

Received: 13 July 2021
Accepted: 11 August 2021
Published: 21 August 2021

1. Introduction

Data that are either of scientific projects to produce answers to scientific hypotheses or generated by business transactions or social events have become a huge size. Nowadays, the generation of big data is also extended to the areas where data are continuously collected from sensors such as those used in horticulture and farming fields or from cameras or telescopes such as those used in the search for extraterrestrial intelligence. No matter how they are generated, big data analysis has become pillars to support business operations or intelligence and scientific research [1]. In terms of computing a global result, big data are too big to be processed by a single commodity computer in a reasonable amount time. Ideally, big data are processed by a dedicated data center, where a high-performance cluster consisting of reliable computing nodes connected by fast networks, fuses, analyses and synthesizes the data for real-time response or warehousing. However, the reality is that small or medium business or scientific projects are unable to invest such a data center. What they can make use are existing commodity computers in the organization. To make use of these computing facilities, the corporate desktops or laptops are unreliable and are not dedicated for a single task. When this situation is extended to the Internet scale, the donated compute cycles are even dynamic and opportunistic. Both situations are the same in terms of a distributed, heterogeneous, dynamic and opportunistic environment. Harnessing millions of commodity computing resources together in such an environment to cope with large scale compute-intensive or data-intensive problems is termed *volunteer computing* (*VC*) [2]. Volunteer computing is a practical and cheaper way for big data processing, which has been already evidenced by scientific projects such as ATLAS@Home [3], Asteroids@Home [4] and Einstein@Home [5].

Our previous work [6] has classified the dynamics and opportunism of volunteer computing environments by eight impact factors: heterogeneity, download/upload speed, round trip time, churn rate, start position, occurrence interval, map/reduce ratio and redistribution factor. The previous work has also confirmed the scalability of volunteer computing for big data processing under the impact factors. The confirmation is promising as the idle compute and storage capacity of an organization or the donated capacities from the Internet can be utilized for big data processing, which traditionally can only be processed by using a dedicated data center. At the same time this previous work has also demonstrated that the dynamics and opportunism of volunteers impair the overall performance. That means the scalability is not a linear increase vs. the number of volunteers.

This paper takes our previous work further to investigate how the scalability goes with different numbers of volunteer and how the overall performance converges vs. the number of volunteers. We use the term *overuse* to refer to the situation of using more than necessary volunteers to bring very little benefit for the overall performance, while the resources can be used for other computing. The research goal of this paper is an optimal use of available volunteers, i.e., the least number of volunteers, for achieving the convergence performance. To achieve the goal, we need to construct a simulation/evaluation platform to:

1. Assess the behavior dynamics and opportunism of volunteers;
2. Process generic MapReduce big data problems;
3. Record system performance.

When the platform is available, we need an algorithm to operate big data. There are a number of big data algorithms [7], among which MapReduce [8] has been a successful algorithm widely used in a variety of field applications. As long as the application data can be organized into the form of <*key, value*> pairs, it can be processed in one or multiple rounds of three sequential steps: *map*, *shuffle* and *reduce* in a distributed environment. The local results are gained in parallel in the map step; the local results are sorted out in the shuffle step and finally the global results are synthesized in the reduce step. If multiple rounds are used, the system tries to speed up the production of local results by parallel computing in map steps and to decrease the data exchange between computing nodes in the shuffle steps. Reduce steps are necessary to synthesize the final global results. To integrate with the <*key, value*> pair representation of MapReduce and to accommodate the dynamics of volunteers, the selection of distributed protocols for the construction of the evaluation platform goes to *distributed hash table* (*DHT*). As a DHT protocol, the dynamics and opportunism of each volunteer are tolerated, and the overall overlay is guaranteed reliable at a certain cost such as $O(_{log}n)$ for lookup a data item and $O(_{log}{}^2n)$ for stabilization on dynamics as in Chord [9]. For a given big data problem, the integrated platform and algorithm support our research methodology, which consists of the following investigations on how scalability goes and converges vs.:

1. The number of volunteers;
2. The heterogeneity of volunteers;
3. A single overlay or multiple overlays;
4. The varying workload and/or varying volunteer numbers.

The platform has been designed on Chord DHT protocol and implemented on the Open Chord APIs [10]. The aforementioned investigations (simulations and evaluations) have been conducted by using a generic MapReduce big data application on the implemented platform. Based on the evaluation results and analysis, the contribution of this paper includes:

1. Compilation of a multiple factor profile to synthesize dynamics (free join, leave or crash) of large-scale volunteers and its impact on a large part of the overlay and on a large number of tasks.
2. Confirmation of scalability of volunteer computing for big data processing goes logarithm-like scale in terms of speedup and in reciprocal inverse-like scale in terms of speedup growth rate.

3. Identification of the convergence points of speedup growth rate.
4. Proposal of strategies to plan the optimal overlay size, the overlay numbers and the overlay structures for a given problem scale and given dynamics of volunteers.

The organization of this paper is as follows: related work is reviewed in Section 2. A preamble about the discussion of the concerned issues is given in Section 3. Section 4 proposes and clarifies the misconceptions on the scalability of VC on big data problems. In Section 5, optimization strategies are proposed, and a study case is detailed. Section 6 concludes that optimal use of volunteers can achieve the possible maximum performance without overusing resources.

2. Related Work

The existing work that is related to this paper has focused on two aspects: the impact of dynamics and opportunism on big data performance and various optimization approaches to coping with the uncertainties. In the area of data-intensive computing, the early study on the impact on the performance of computing include Fadika et al. [11] and Dede et al. [12]. Fadika et al. [11] studied the impact caused by certain data operations of MapReduce, the number of tasks, the replication of data and the network bandwidth. Dede et al. [12] studied the impact of heterogeneity, unreliability or unstable computing power of computing nodes. Both of them contrasted different implementations of MapReduce in a heterogeneous cluster, a homogeneous but load-imbalanced cluster and a cluster with unreliable nodes. Their evaluation results demonstrated the key impact factors on the algorithm implementations in processing data-intensive, CPU-intensive and memory-intensive applications.

Cheng et al. [13] proposed an *adaptive task tuning* to assign tasks to computing nodes based on their profiles. The profiling included monitoring task execution and nodes performance. The core of the model was a genetic algorithm to adjust workload in the course of computing. Based on the profiles, the algorithm could dynamically balance the workload in the whole cluster by fine-tuning *109* parameters of Hadoop [14] and MapReduce in cluster level, job level and task level. The model was evaluated on both a physical cluster and a virtual cluster. The results demonstrated a *11%* to *31%* improvement compared with default Hadoop and Starfish and Rules-of-Thumb optimizers. The model was limited to multiple round MapReduce applications as the genetic optimization algorithm must be multiple rounds. However, most of the real-world applications are single round in terms of minimizing data exchange in the course of computing.

Jothi and Indumathy [15] assumed that the compute capacities of computing nodes were different but certain. On the basis, they studied the impact of the heterogeneity of computing nodes on the overall performance of MapReduce in a distributed environment. They classified computing nodes based on compute capacity and split data into different sizes. To balance the workload, they proposed to use a centralized scheduling to assign larger data sets to faster computing nodes. They demonstrated the effectiveness of the approach, but the experimental environment or problem scales that the approach could be applied to, were not clearly given in the paper.

To reduce the impact of dynamics in distributed environments. Yildiz et al. [16] proposed a model to recover failure of computing nodes. The key idea was to prioritize tasks and to allow the prioritized tasks to pre-empt other tasks in failure recovery. The model tried to achieve recovering important tasks in certain time. In a dynamic environment, the model was able to alleviate the impact of node failures and demonstrated a reduction in the overall completion time. Their experimental environment including *17 GB* and *56 GB* data sets on a cluster of *19* nodes of *8*-core Intel Xeon CPUs connected by a *10 Gbps* Ethernet network was small to reflect a data-intensive application on the internet scale in terms of volunteer computing.

Singh et al. [17] stated that Hadoop [14] data locality could overwhelm load balancing and slow down the overall performance. They demonstrated that the heterogeneity of computing nodes was a key impact factor, and workload balancing was the critical

way to optimize system performance in such a heterogeneous environment. They stated that the impact on big data processing could be from two aspects: algorithm-specific or cluster-specific. They proposed a number of strategies, including using a combiner to maximize intra-node processing and minimizing inter-node data exchange; sacrificing redundancy or fault tolerance by disabling speculative execution to increase efficiency; removing slow nodes to decrease the overlay size, to improve the overall performance. Their optimization goal was to integrate MapReduce-based Apriori (a very efficient data analyzer) and Hadoop, but it was partly solved as reported in Singh et al. [17].

Ardagna et al. [18] stated two factors that need to be considered for the cost of a big data processing on a public cloud. First, an appropriate size of cluster was important to predict the budget to run Hadoop for applications in a public cloud. Second, the execution time of a MapReduce job was critical as well. In order to estimate MapReduce job execution time in Hadoop clusters that was managed by YARN, they used Petri-net-like models to provide performance analysis at design-time. They tested their analysis models on a public cloud Amazon EC2 (*120* CPUs supporting *240* containers) and an Italian supercomputing center CINECA (*120* cores with a container). The test results confirmed that the accuracy of the estimation was high in terms of that the relative error between the estimation and the actual measurement was *14%* to *32%*.

Work stealing, i.e., fast nodes steal work from slow nodes in order to balance workload, is a traditional optimization against the heterogeneity of distributed environments. Perarnau and Sato's [19] study was a distributed memory environment, consisting of *8192* nodes. They compared a number of algorithms, named Deterministic Selection and Random Selection with Skewed Distribution and with or without Half-Stealing, for victim selection strategies. A victim was the one that was identified to be stolen for work. Among the algorithms, Random Selection with Skewed Distribution and Half-Stealing was demonstrated overperforming others. Vu and Derbel [20] dealt with the heterogeneity of distributed nodes in terms of compute capacity and network speed. They introduced some adaptive control operations to the existing Probabilistic Work Stealing (PWS) and Adaptive Cluster-aware Random Stealing (ACRS) algorithms, aiming at increasing work locality and decreasing the cost of work stealing. They demonstrated a *30%* save in the computing time of the algorithms on a *128*-node cluster. In the distributed environment they studied, the dynamics of computing nodes was not considered. Zhang et al. [21] regarded the equity task allocation as the main reason of poor performance of MapReduce in heterogeneous environments. They stated that by holding tasks, the straggler nodes could slow down the completion of map step, preventing the reduce step from starting, or they could slow down the reduce step to affect the overall progress. They improved the situation by allowing faster nodes to steal some work from the stragglers. Their study was for heterogeneous environments but did not touch the dynamics or unreliability of the environments. Our previous work [22] has proposed two versions of work stealing to algorithmically optimize the processing for compute-intensive or data-intensive applications in consideration of the dynamics and opportunism of volunteer environments.

The motivation of generating datasets is similar between [23] and this paper in that both believe that the off-the-shelf real-world datasets have some limitations, and the generation of richer and more synthetic datasets is necessary for the study of the concerned issues. For this paper, the big size of datasets, the map/reduce ratio and redistribution factor of map results are the concerns of data generation against the dynamics and the large number of volunteers.

The aforementioned review concludes that the optimization comes from a variety of considerations but has not been well explored for the dynamics and opportunism. This paper takes the effort in this direction, aiming at an adaptive overlay structure to the properties of big data problem and the dynamics and opportunism of volunteer environments for optimizing data-intensive applications.

3. Preamble of Discussion

To ease the discussion of the complex issues, a number of conditions about the dynamics and opportunism of distributed environments, MapReduce workflow, the problem scale of big data and the measurement of performance is presented in this section.

3.1. MapReduce Workflow in Dynamic and Opportunistic Environments

When MapReduce is the most commonly used big data processing paradigm and DHT is a successful protocol to construct dynamic but still reliable overlay of internet volunteers, the integration of them has natural advantages to construct a big data processing platform. The nature is the use of <*key*, *values*> pairs by both MapReduce and DHT. Based on the CRUD operations on <*key*, *value*> pairs of DHT, the three steps of MapReduce are constructed as follows.

Assumption: a map task mt_i or a reduce task rt_j is identified by a key mk_i or rk_j, where $i \in \{1, 2, \ldots, m\}$, $j = \{1, 2, \ldots, r\}$, and m is the number of map tasks and r is the number of reduce tasks. A map task or a reduce task is a self-satisfied object, which includes the executable code and data that are encapsulated in a data structure that is appropriate to a particular MapReduce application. The data set of a map task is filled at the beginning of computing but the data set of a reduce task needs to be filled in the course of computing by using the results of map tasks.

Map Step: a volunteer looks up <mk_{i0}, mt_i> by using the key mk_{i0} for an available map task, where $i \in \{1, 2, \ldots, m\}$. If <$mk_{i0}$, mt_i> is available, the volunteer changes it to <mk_{i1}, mt_i, mts_i>, where mts_i is the timestamp of the task. Once a task is found, it is downloaded and put into execution. If the volunteer leaves before finishing the task, it will change it back to <mk_{i0}, mt_i>. When a map task is in execution, the volunteer will need to update the timestamp mts_i in a regular time interval *ui*. If failed with looking up <mk_{i0}, mt_i>, a volunteer looks up <mk_{i1}, mt_i, mts_i > by using the key mk_{i1} for an available map task that satisfies the condition: *(the current time—*mts_i*)* > *ui*. Such a map task was in execution by another volunteer that is treated as crashed already.

Shuffle Step: for the result set of a map task, the shuffle step is to redistribute it into a number of reduce tasks. The procedure is that all the <*key*, *value*> pairs with the same key in the result set will be merged together in the form of <*key*, *a list of values*>. The merged <*key*, *a list of values*> will be distributed into a reduce task by using a hash function so that the pairs emitted by different map tasks but with the same key must be distributed into the same reduce task. The redistribution is dependent on both the application features and the original data set in terms of the number of redistributions for each map result set.

Reduce Step: a volunteer looks up <rk_{j0}, rt_j> by using the key rk_{j0} for an available reduce task, where $j = \{1, 2, \ldots, r\}$. If <rk_{j0}, rt_j> is available, the volunteer changes it to <rk_{j1}, rt_j, rts_j>, where rts_j is the timestamp of the task. Once a task is found, it is downloaded and put into execution. If the peer leaves before finishing the task, it will change it back to <rk_{j0}, rt_j>. When a reduce task is in execution, the volunteer will need to update the timestamp rts_i in a regular time interval *ui*. If failed with looking up <rk_{j0}, rt_j>, a volunteer looks up <rk_{j1}, rt_j, rts_j> by using the key rk_{j1} for an available reduce task that satisfies the condition: *(the current time—*rts_j*)* > *ui*. Such a reduce task was in execution by another volunteer that is treated as crashed already. The result set of a reduce task is simply uploaded onto the volunteer overlay.

The dynamics and opportunism of the volunteers is that a volunteer can join freely at any time, and in each operation of looking up a task, downloading a task, computing a task or uploading a result set, the volunteer can leave or crash. The dynamics and its impact on MapReduce workflow are summarized in Table 1 and explained as follows.

Table 1. The dynamics and its impact on MapReduce workflow.

Dynamics	Workflow for Each Task			
	Lookup	**Download**	**Compute**	**Upload**
Join or re-join	No impact	-	-	-
Leave	No impact	No impact	Checkpointed	Must be done
Crash	No impact	No impact	Must be redone	Must be redone

A volunteer starts looking up a task when newly joining or re-joining (i.e., completing the current task and asking for another task) the overlay. A volunteer can leave from or crash on the overlay when looking up or downloading a task, but this has no impact on the task itself. When a volunteer leaves in the course of computing a task, the uncompleted task must be checkpointed so that it can be picked up by another volunteer in the future. When a volunteer leaves in the course of uploading results, it must finish the uploading before leaving. When a volunteer crashes in the course of computing or uploading the results, the computing on the task is fully wasted and the task must be redone by another volunteer in the future.

3.2. The Measurement of Performance

In addition to the completion time that can be directly used to measure the overall performance, we define two measurements to make a quantitative investigation. *Speedup* is used to measure the overall performance of a volunteer overlay; *Speedup Growth Rate* is to measure the change rate of speedup upon the change of an impact factor on the overlay, e.g., number of volunteers, the heterogeneity of volunteers or map/reduce ratio, etc.

$$Speedup = \frac{The\ total\ time\ to\ complete\ the\ entire\ problem\ by\ a\ volunteer\ overlay}{The\ total\ time\ to\ complete\ the\ entire\ problem\ by\ a\ single\ volunteer}$$

$$Speedup\ Growth\ Rate = \frac{The\ speedup\ of\ value\ 2 - The\ speedup\ of\ value\ 1}{The\ speedup\ of\ value\ 1} \times 100\%$$

Speedup growth rate is finer than speedup to reflect the overall performance because the former is able to check whether the scalability converges to a particular value.

3.3. The Setting of Dynamics and Workload

Our previous work [6] has confirmed that the following factors have significant impact on the overall performance of a volunteer overlay on processing big data problems. We borrow the factors for the discussion of this paper, and we need to base the evaluation settings of this paper on the factors.

1. *Heterogeneity (H)* reflects the difference of volunteers in compute capacity. If we assume that the base capacity is *tier 1*, then a two-fold slower volunteer is of *tier 2*.
2. *Download/Upload Speed (DUS)* reflects the internet speed of a volunteer. For example, if we assume a moderate internet speed of *25/10* Mbps for download/upload, the *DUS* is *20/51* s for a *64 MB* data.
3. *Round Trip Time (RTT)* reflects the time to establish an internet connection before or close the connection after a communication between volunteers. A reasonable *RTT* should be no more than *8* for a moderate speed internet.
4. *Map/Reduce Ratio (MRR)* reflects a big data application, being a data aggregation (*input* > *output*), data expansion (*input* < *output*), data transformation (*input* ≈ *output*) or data summary (*input* >> *output*) [23]. The most common big data applications are data aggregation. For example, a *20% MRR* means that the workload and data scale of reduce tasks will be *20%* of map tasks.
5. *Redistribution Factor (RF)* reflects the difference of the keys of an intermediate result set. For example, a *RF* of *200* means that an intermediate result set from the map step needs to be redistributed into *200* reduce tasks in the shuffle step.

6. *Churn Rate (CR)* reflects the percentage of total volunteers who behave dynamics or opportunism in terms of leave or crash in the course of computing.
7. *Start Position (SP)* reflects how long a volunteer stays on the overlay before committing dynamics.
8. *Occurrence Interval (OI)* reflects the time period within that a volunteer could commit churn.

The three factors *CR*, *SP* and *OI* work together to represent the churn of volunteers. For example, a churn of *CR* of *30%*, *SP* of *250 K* and *OI* of *30* reflects that there are *30%* volunteers committing dynamics. For each churn volunteer it will stay at least *250 K* long on the overlay after joining, and it could commit churn between *250,000* to *250,030* time period. The three churn factors together determine the churn occurrence pattern. The selections of the three churn factor values must make the churn pattern as random as possible. As shown in Figure 1, the churn happens once for the map and shuffle step and once for the reduce step. The ascending lines represent the joins of volunteers, and the descending lines represent the leaves or crashes of volunteers. The flat lines represent that the overlay is stable without dynamics.

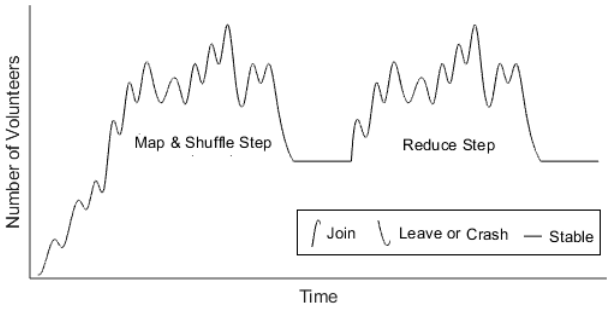

Figure 1. The random churn patterns.

Instead of using real world datasets such as Word Count or TeraSort, the datasets used by this paper is generated and independent of or not restricted by any real world applications. The generality/flexibility of dataset generation allows to apply any computing and communication intensities to the quantitative evaluations. A dataset is generated by applying:

1. The number of map tasks (*NMT*) and the number of reduce tasks (*NRT*);
2. The computing load of each task (*CLET*);
3. The size of a map or a reduce task or a map or a reduce result set;
4. The lookup time of a map task or a reduce task on the overlay;
5. The communication speed.

The aforementioned parameters can produce and vary:

1. The compute intensity, e.g., if the *NMT* is *1,400,000* (*1.4 M*) and *NRT* is *280,000* (*0.28 M*) and the *CLET* is *8000* (*8 K*) time units, the overall computing load is (*1.4 M* + *0.28 M*) $\times$ *8000* = *11.2 G* + *2.24 G* = *13.44 G* time units.
2. The problem scale, e.g., if the size of a map or a reduce task or a map or a reduce result set is *64 MB*, the overall data size to be processed is *1,400,000* $\times$ *64* + *280,000* $\times$ *64* = *107,520,000 MB* $\approx$ *108 TB*.
3. The communication intensity, e.g., if the communication speed is *5* download/upload speed (in *Mbps*) tiers: *12/1*, *25/5*, *25/10*, *50/20* and *100/40* as provided by Australia National Broadband Network (NBN), the download/upload speed (in seconds) of a *64 MB* dataset is *5* tiers: *43/512*, *20/102*, *20/51*, *10/26*, *5/13*.
4. The problem property. By varying *NMT* and *NRT*, the data aggregation (*input* > *output*), data expansion (*input* < *output*), data transformation (*input* $\approx$ *output*) and data summary (*input* >> *output*) [24] can be configured, e.g., if *NRT/NMT* = *20%*, a data aggregation application is set.

The proposed dataset generation allows selecting and combining various characteristics, mitigating the existing datasets, which captured narrow slivers of a rich space of workload, and fitting for the dynamic and opportunistic environments that this paper concerns.

We define the following settings to reflect a general situation of volunteer overlays and a general situation of the workload of big data problems.

1. Overlay setting: in the format of (H, DUS, RTT, CR, SP, OI, MRR, RF), the overlay dynamics setting is (*6* tiers, *20/51*, *8*, *30%*, *250 K*, *30*, *20%*, *200*).
2. Workload setting: for *1,400,000 (1.4 M)* map tasks, *280,000 (0.28 M)* reduce tasks (if *20%* MRR is assumed) and the computing load of each map or reduce task of *8000*, the computing workload is *13.44 G* in total.
3. Data setting: for each map or reduce task or a result set of 64 MB, the total amount of data to be processed is about *108 TB* (*89.6 TB* of map + *17.92 TB* of reduce if *20%* MRR is assumed).

4. Misconception and Clarification

This section proposes four common misconceptions about the scalability of volunteer computing for big data problems. The misconceptions are to be clarified with the support of quantitative evaluations based on the dynamic settings, workflow settings and workload settings, and performed on the simulation platform as detailed in Section 3. The clarification is the foundation of the optimization strategies as proposed in Section 5.

4.1. Misconception 1

The scalability of volunteer computing is unlimited, i.e., the more volunteers, the faster computing. This misconception results in a general practice: always welcoming more volunteers for a computing. This misconception is a faulty thought that the overall performance of the whole overlay is proportional to the number of volunteers on the overlay. This misconception is caused by the instinct in real life: wisdom of the crowds, saying that wisdom of the crowds exceeds that of any individual. In real life, taking the average of a large number of responses is able to cancel the effect of the noise that is associated with each individual judgement. However, it is questionable how far this phenomenon can go in the context of volunteer computing.

To quantitatively clarify the misconception, we conduct the following evaluation. The overall performance is evaluated by the overlay, workload and data settings as detailed in Section 3.3 with the initial number of volunteers of *5000*. We increase volunteer numbers by *5000* each time and record the speedup and speedup growth rate accordingly. The evaluation results are shown in Figures 2 and 3.

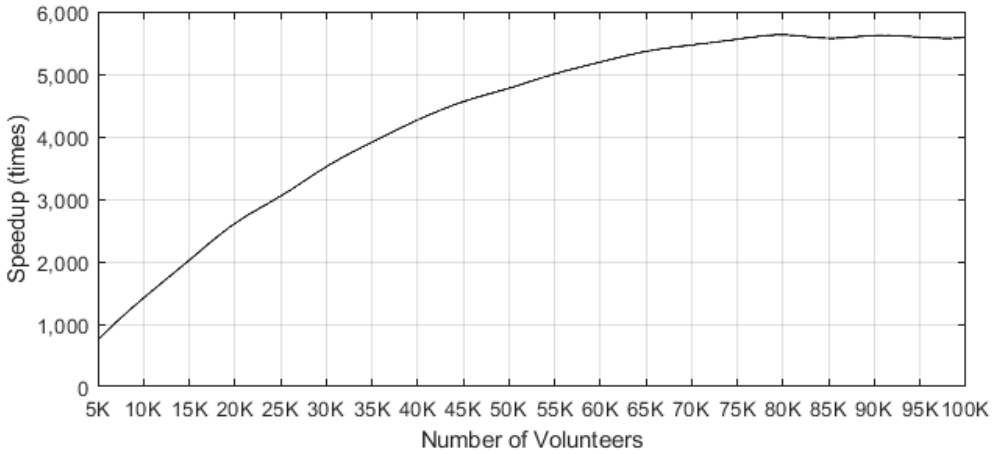

Figure 2. The speedup vs. the size of volunteer overlays.

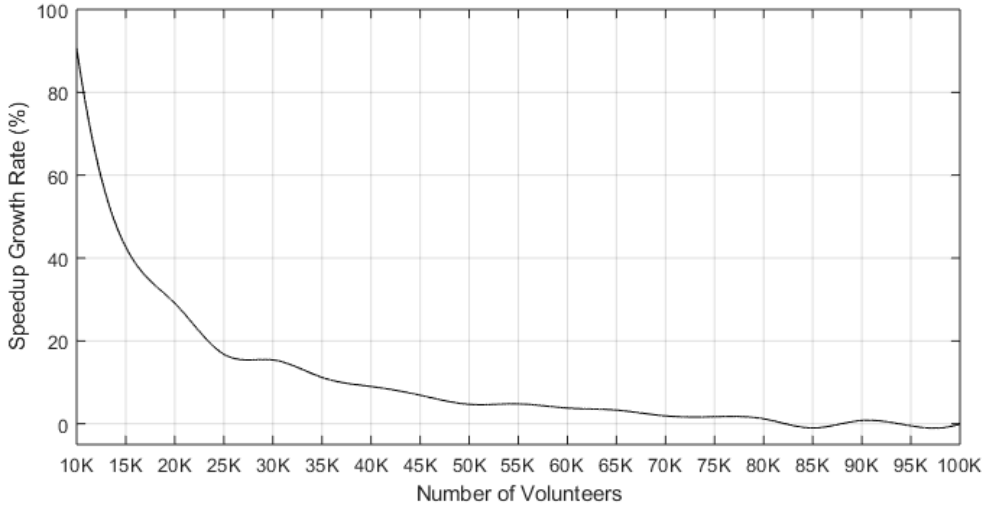

Figure 3. The speedup growth rate in response to overlay size growing.

The growing rate of overlay size is *5000* volunteers each time. That is, each time there are *5000* more volunteers joining the overlay. It is evident that after a certain size, continuously growing overlay size by increasing the number of volunteers brings little benefit for the overall performance as shown in Figures 2 and 3. To clarify the results, in terms of speedup, the overall performance is increasing slowly for the overlay of *40,000* volunteers or more. In terms of speedup growth rate, the performance is improving slower and slower, and it is less than *10%* from the overlay of *40,000* volunteers or more. This concludes that although the overall performance is scalable in response to the growing overlay, but for the same growing size, e.g., *5000* more volunteers each time, the growing brings little benefit after a certain overlay size, e.g., *40,000*. Finally, the scalability converges at the speedup growth rate of *1%* from *70,000* or more volunteer overlays. The working cost increasing on larger overlays is the key reason of the convergence of speedup growth rate. This conclusion is not conflicting with the scalability of volunteer computing for big data processing, but it breaks the misconception that the same number of volunteers brings the same or similar amount of benefit at any time.

4.2. Misconception 2

A volunteer always contributes positively to the overall performance no matter what compute capacity/power it can provide. This misconception results in a general practice: always welcoming any volunteers without considering their compute capacities. This misconception is caused by the illusion when the numbers of tasks and volunteers are both very large, the workload of millions of tasks is naturally balanced between millions of volunteers. The real situation is that once a task is assigned to a volunteer, it is held by the volunteer unless it is completed, or the volunteer leaves or crashes. At that time, the result can be collected, or the task can be reassigned. However, if the volunteer compute capacity is very low and keeps active, it becomes a straggler. Thus, the overall performance is finally controlled by stragglers rather than the fast volunteers. For example, assume that there are two tasks and there are two volunteers: v_1 and v_2, and v_2 is *5* time slower than v_1. If v_1 completes a task in t, the *2* tasks will be complete in *2* t by v_1 only. However, if the *2* tasks are assigned to v_1 and v_2, respectively, the completion time will be *5* t. This situation happens when there are no available tasks for the fast volunteers to do, but at the same time there is a large number of tasks that are held by stragglers. This situation happens when the tasks as a whole have already progressed a large part, then the stragglers have obtained the chance to control the progress of remaining computing. In other words, there exists a turning point in terms of compute capacity, welcoming volunteers that are slower than the turning point will bring little benefit.

To quantitatively clarify the misconception, we conduct the following evaluations. First of all, we do not include churn to cancel possible noises that churn may bring to the evaluation results. We use 6 tiers of compute capacity and *40,000* volunteers as the median point. Other settings on workload and data are the same as given in Section 3.3. We assume that increasing or decreasing the number of volunteers will increase or decrease the tiers of compute capacity of the volunteers proportionally. The rule is that increasing/decreasing the number of volunteers by *1/6* will increase/decrease the heterogeneity of compute capacity for *1* tier. Based on this rule, the *15* rounds of experimental evaluation of heterogeneity are listed as follows in the format: HV_i *(*$tier_i$*; number of volunteers)*, where i is the sequence number of an evaluation pair and $i \in \{1, 2, \ldots, 15\}$.

$HV_1(1;\ 6666)$, $HV_2(2;\ 13{,}333)$, $HV_3(3;\ 20{,}000)$, $HV_4(4;\ 26{,}666)$, $HV_5(5;\ 33{,}333)$, $HV_6(6;\ 40{,}000)$, $HV_7(7;\ 46{,}666)$, $HV_8(8;\ 53{,}333)$, $HV_9(9;\ 60{,}000)$, $HV_{10}(10;\ 66{,}666)$, $HV_{11}(11;\ 73{,}333)$, $HV_{12}(12;\ 80{,}000)$, $HV_{13}(13;\ 86{,}666)$, $HV_{14}(14;\ 93{,}333)$, $HV_{15}(15;\ 100{,}000)$.

The fourteen changes of the *HV* evaluation pairs are listed as follows in the format: $HC_j(HV_k \rightarrow HV_m)$, which means that the jth change is from pair HV_k to HV_m, where $j, k \in \{1, 2, \ldots, 14\} \wedge m \in \{2, 3, \ldots, 15\}$.

$HC_1(HV_1 \rightarrow HV_2)$, $HC_2(HV_2 \rightarrow HV_3)$, $HC_3(HV_3 \rightarrow HV_4)$, $HC_4(HV_4 \rightarrow HV_5)$, $HC_5(HV_5 \rightarrow HV_6)$, $HC_6(HV_6 \rightarrow HV_7)$, $HC_7(HV_7 \rightarrow HV_8)$, $HC_8(HV_8 \rightarrow HV_9)$, $HC_9(HV_9 \rightarrow HV_{10})$, $HC_{10}(HV_{10} \rightarrow HV_{11})$, $HC_{11}(HV_{11} \rightarrow HV_{12})$, $HC_{12}(HV_{12} \rightarrow HV_{13})$, $HC_{13}(HV_{13} \rightarrow HV_{14})$, $HC_{14}(HV_{14} \rightarrow HV_{15})$.

The evaluation results have been reported in Figure 4 for speedup vs. *HV* pairs and in Figure 5 for speedup growth rate vs. change of *HV* pairs.

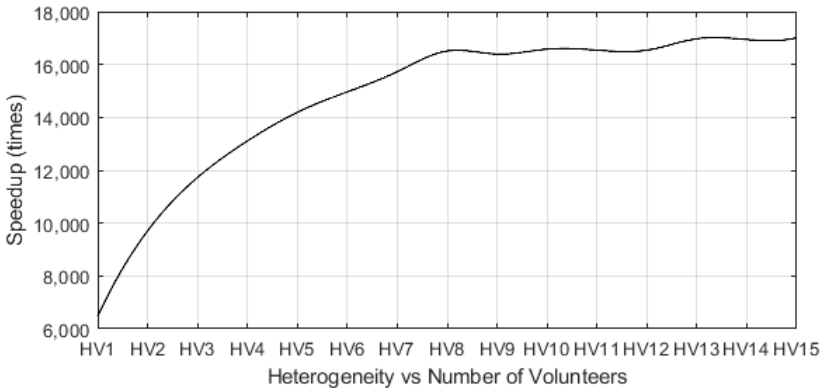

Figure 4. The speedup vs. the evaluation pairs.

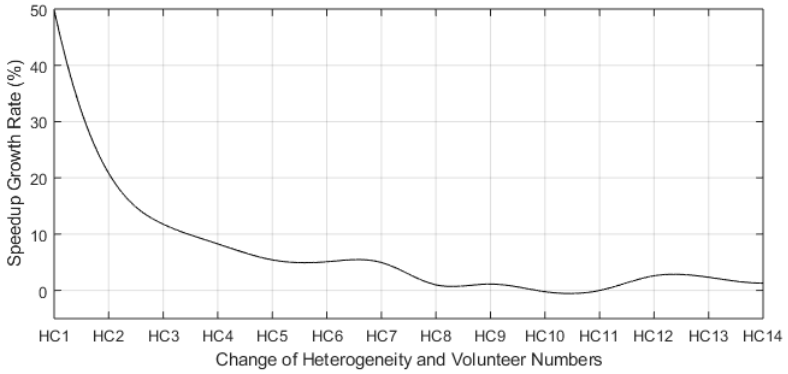

Figure 5. The speedup growth rate vs. change of evaluation pairs.

As shown in Figures 4 and 5, the evaluation results have confirmed that from the number of volunteers of *40,000* and compute capacity of *6 tiers* (HV_6 and HC_5 in the above lists) forward, the speedup is increasing very slow, and the speedup growth rate is going

down to *5%* or less. From *60,000* volunteers and *9-tier* capacity (HV_9 and HC_8 in the above lists) forward, the speedup growth rate converges to around *1.5%*.

To provide another supporting evidence, we record and check:

1. The completion time of a *25%* workload vs. the evaluation pairs as shown in Figure 6, and
2. The weighting of the completion time of a *25%* workload vs. the evaluation pairs as shown in Figure 7.

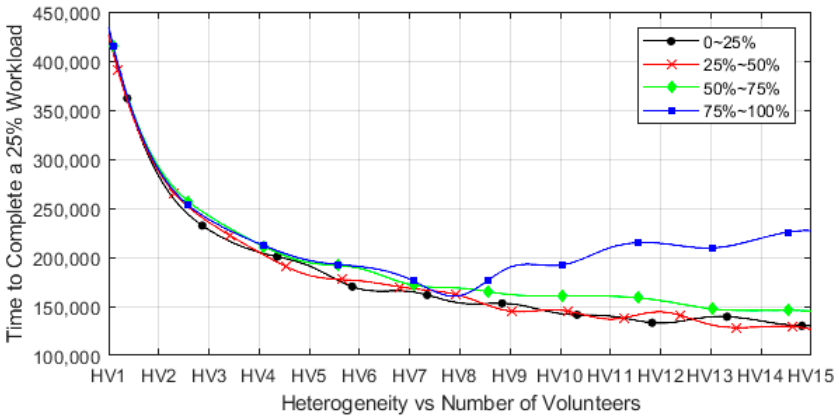

Figure 6. The time to complete a *25%* workload vs. evaluation pairs.

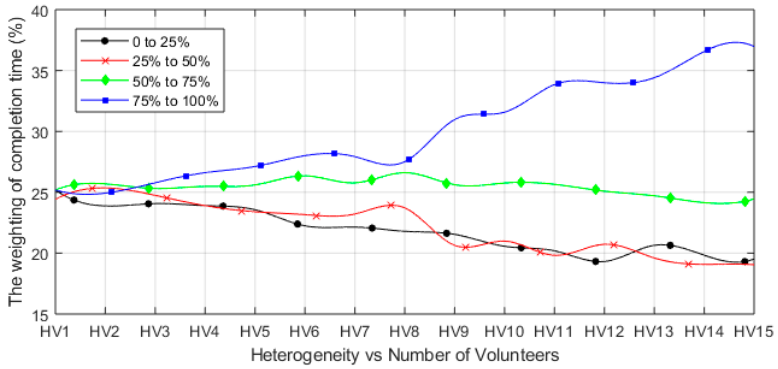

Figure 7. The weighting of the time to complete a 25% workload vs. evaluation pairs.

To check Figure 6, the overall performance is similar for the first (*0~25%*), second (*25~50%*) and third (*50~75%*) quarter of the entire workload. The completion time goes down quickly with the increasing of volunteer numbers and compute capacity tiers till the *6*th evaluation pair HV_6 (*6; 40,000*), which has *40,000* volunteers with *6 tiers* compute capacities. Then, the completion time goes down slowly with the increasing of propositional volunteers and tiers of compute capacity from HV_7 forward. The three tendency lines have demonstrated that for the first three quarters of workload, there are always tasks available for the fast volunteers; more volunteers can result in a faster computing progress. However, volunteers with compute capacity of *7 tiers* or above contribute little to the overall speedup although they are still positive. The situation of the last quarter (*75~100%*) of entire workload is significantly different. For the volunteer numbers up to *40,000* with compute capacity of up to *6 tiers*, the slow volunteers cannot dominate the overall performance because they still perform in an acceptable speed. However, with the increasing of compute capacity tiers from *7 tiers* forward, the slow volunteers begin to dominate the computing progress strongly and in fact slow down the overall performance significantly. They hold the computing tasks to make the fast volunteers starving. Thus, adding more volunteers with compute capacity bigger than *7 tiers* actually contributes negatively to the overall performance. The aforementioned results comply with Figure 7, where for the first, second and third *25%* workload, the weighting of completion time of a *25%* workload is less or no

more than *25%* of the total workload. However, for the last *25%* workload, the weighting of completion time keeps increasing up to *37%* of the total.

The aforementioned are not conflicting with the scalability of volunteer computing for big data processing, but it breaks the misconception that a volunteer always contributes positively to the overall performance. In fact, a slower volunteer could become a straggler and contributes negatively to the overall progress and affects performance in some circumstances.

4.3. Misconception 3

A volunteer always contributes positively to the overall performance no matter what the scale of a computing problem is. This misconception results in a general practice: the size of overlay is not considered for the scale of a given big data problem. This misconception is caused by the illusion that the working cost on an overlay is always trivial compared with the workload of a computing problem. In fact, the lookups of available tasks, the download/upload of tasks/results, and the overlay stabilization in response to the dynamics of volunteers all contribute to the working cost. For example, in Chord DHT protocol, lookups and overlay stabilization incur the working cost of $O(_{log}n)$ and $O(_{log}{}^2n)$, respectively, which are logarithmically proportional to n, the number of volunteers on the overlay. Consequently, given a problem scale, there exists a turning point so that overlay size that is larger than the turning point will bring little benefit for the overall performance. The reason is that the overall working cost is comparable with the workload of the computing problem from that turning point.

To quantitatively clarify the misconception, we conduct the following evaluations. First, we set a reference point in terms of overlay size N (*40,000* volunteers) and workload W (*1,400,000* tasks) by using the overlay setting, the workload setting and the data setting as given in Section 3.3. Then, we conduct the evaluations in two directions: one is to keep the number of volunteers staying on *40,000* but to increase the workload from *20%* to *200%*, symbolized as *W (−80% to +100%)*. The other is to keep workload staying at *20%* but decrease the number of volunteers from *40,000* to *5000*, symbolized as *N (5 K to 40 K)*. The evaluation results are reported in Figure 8 for speedup and in Figure 9 for speedup growth rate.

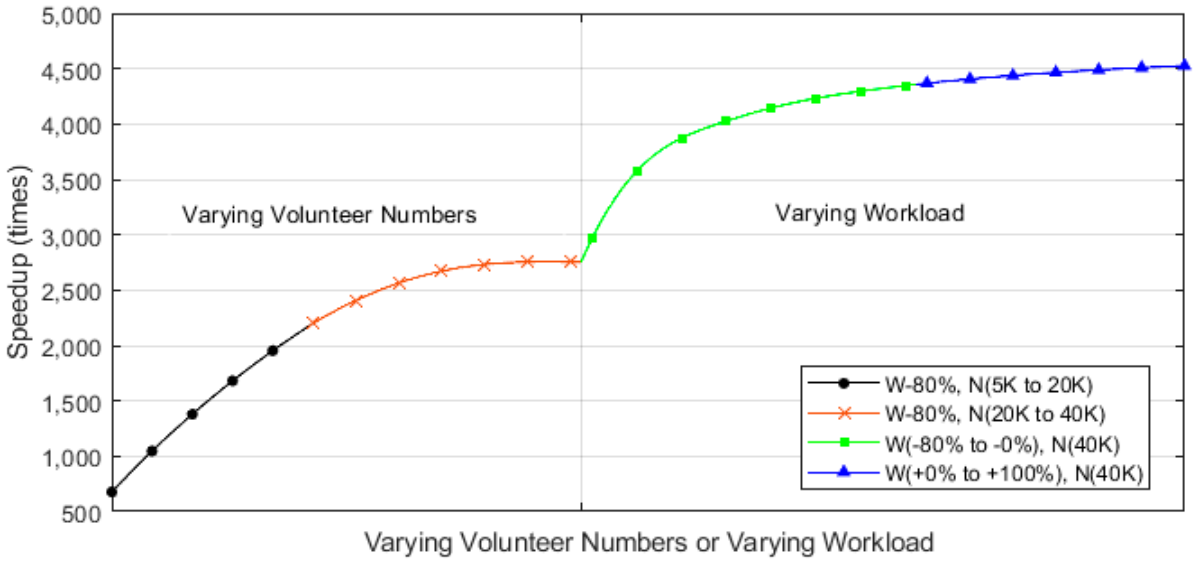

Figure 8. The speedup vs. varying volunteer numbers or varying workload.

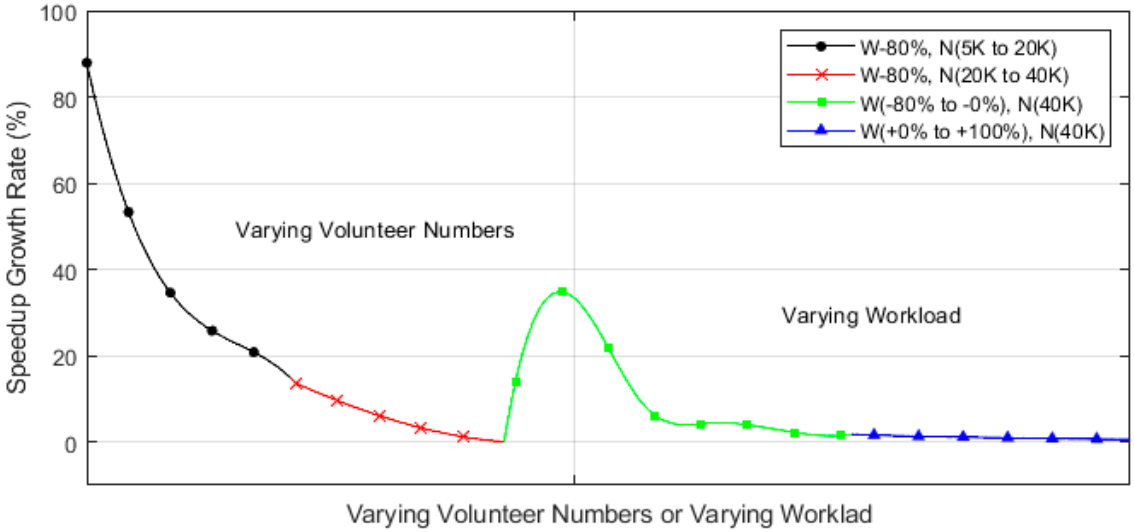

Figure 9. The speedup growth rate vs. varying volunteer numbers or varying workload.

It is evident that for the *20%* of reference workload (symbolized *W-80%*), small overlay from *5000* to *20,000* volunteers speeds up the overall performance from *677* to *2200* times as shown in Figure 8, but the speedup grow rate keeps dropping from *88%* down to *24%* accordingly as shown in Figure 9. The overall speedup increases slower from *2200* to *2752* times for the overlays from *20,000* to *40,000* volunteers as shown in Figure 8, and the speedup growth rate decreases from *24%* to *0.1%* accordingly, as shown in Figure 9. We conclude that for such a *20%* of the reference workload, the working cost on a dynamic overlay of such a size from *20,000* to *40,000* volunteers has cancelled the increase in compute capacity. Furthermore, the speedup of the reference overlay of *40,000* volunteers bounces up from *2752* to *4281* times in response to the workload increasing from *20%* (*W-80%*) to *100%* (*W-0%*) as shown in Figure 8. This performance bounce-back is also reflected by the speedup growth rate bouncing back to *33%* as shown in Figure 9. This resilience of speedup has demonstrated that for the *20%* workload (*W-80%*) to *100%* workload (*W-0%*), the compute capacity of *40,000* volunteers overlay overwhelms the working cost on the overlay of such a size. Although there is still speedup increase from *4281* to *4526* times in response to the workload increasing from *100%* (*W-0%*) to *200%* (*W + 100*) as shown in Figure 8, but the speedup growth rate is decreasing from *33%* down to *3%* and finally converges to around *0.6%* as shown in Figure 9. The reason is that the compute capacity of such an overlay of *40,000* volunteers becomes weaker for such a workload (*100%* to *200%*). This conclusion is not conflicting with the scalability of volunteer computing for big data processing, but it breaks the misconception that the working cost on a dynamic overlay is always trivial compared with the entire workload of computing. There exists an optimal overlay size for the scale of a given computing problem.

4.4. Misconception 4

In terms of overlay structure, the map step and the reduce step are the same for a single round MapReduce application. This misconception results in a general practice: a single overlay is used for the whole MapReduce procedure. This misconception is caused by the illusion that all the volunteers must be on the same overlay so that the global results covering all the input can be obtained in one round. In fact, the MapReduce paradigm requires that all the <*key, value*> pairs with the same key emitted from the map step must be shuffled into the same reduce task if the application is to be completed in one round. That requirement equals that all volunteers of reduce step must be on the same overlay for a global redistribution of the intermediate results emitted from the map step. However, single round of completion does not require/need that all volunteers are on the same single overlay for the map step. Based on the aforementioned analysis, volunteers are unnecessary

to stay on a single plain overlay for the whole MapReduce procedure, i.e., both the map step and reduce step. Thus, we propose separate overlays: map overlay and reduce overlay. Multiple map overlays have no impact on the single round completion requirement for global results. The global results can be obtained in one round as long as the volunteers are on the same single reduce overlay. The rationale of using smaller multiple overlays for map step is to decrease the working cost on a larger overlay, which is logarithmically proportional to the overlay size, i.e., $O(_{log}n)$ for lookup and $O(_{log}{}^{2}n)$ for stabilization.

To quantitatively demonstrate the benefits of using multiple map overlays, we conduct the following evaluations. The basic start point is the overlay, workload and data settings as proposed in Section 3.3. In the three evaluations, the original overlays are a single overlay of *40,000*, *60,000* and *100,000* volunteers, respectively. Each time the number of volunteers on the overlay is halved to form two equal overlays, and accordingly the original workload is halved as well for each separated overlay. As the map overlays are independent and the same in size, we assume that the map step of each separated overlay starts from the same time and finishes at the same time *t*. Thus, the *equivalent speedup* will be calculated as: *the whole workload/t*. The evaluation results are reported in Figure 10 for equivalent speedup and in Figure 11 for equivalent speedup growth rate.

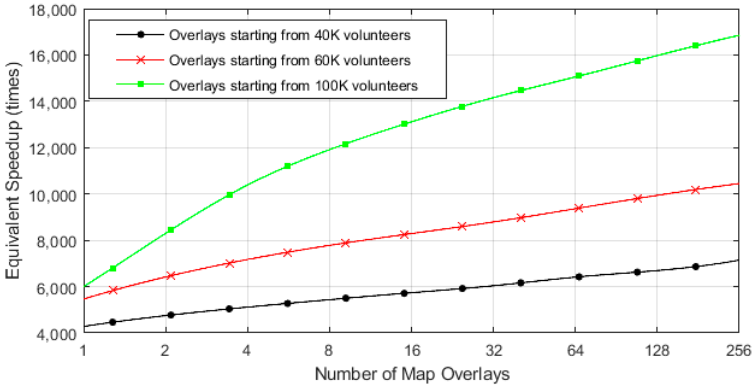

Figure 10. The equivalent speedup vs. the number of map overlays.

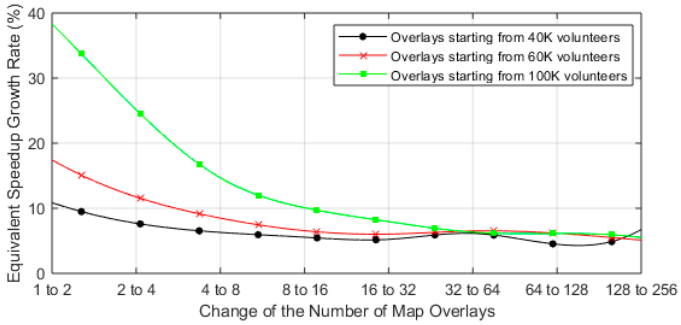

Figure 11. The equivalent speedup growth rate vs. the change of the number of map overlays.

As shown in Figure 10, the original overlay will finally be separated into *256* map overlays with each overlay having *1/256* volunteers for the *1/256* of the original workload. The computing time of each separation keeps decreasing, and the equivalent speedup keeps increasing for the overlay clusters separated from the original overlay. The equivalent speedup growth rate keeps drooping as shown in Figure 11, suggesting that the separation is more efficient for a larger overlay with a higher workload but inefficient for a smaller overlay with a lighter workload. For example, starting from *100,000* volunteer overlay, the first separation (2 map overlays) achieves a *38%* grow rate; the third separation (*8* map overlays) achieves *15%* growth rate. In the evaluations, staring from either *40,000*; *60,000* or *100,000*, the separation to *32* map overlays makes the speedup growth rate converged to around *7%* as shown in Figure 11. The results have confirmed that multiple map overlays bring benefit for the overall performance at the map step. This conclusion breaks the

misconception that volunteers must stay on a single overlay for the whole procedure of map and reduce. It opens the window to use different overlay structures for map or reduce steps.

5. Optimization Strategy

Based on the clarification of misconceptions of Section 4, this section proposes the optimization strategies on the overlay size and organization in response to a given big data problem and the dynamics of volunteers. A case study is detailed in this section to demonstrate the effectiveness of the optimization strategies.

5.1. Optimization Strategy

The optimization strategies that are proposed in this section is based on the assumption and needs the simulation to achieve the optimization goal as given as follows.

1. Assumption: the dynamics of volunteers, the workload and the dataset of a big data problem are given in the format of the three settings in Section 3.3.
2. Simulation: a series of simulations to produce performance results vs. the concerned variables as shown in Figures 2–11 in Section 4.
3. Goal: an overlay structure that can achieve optimal performance for the given big data problem and the dynamics of volunteers.

The optimization consists of five steps, as depicted by Figure 12.

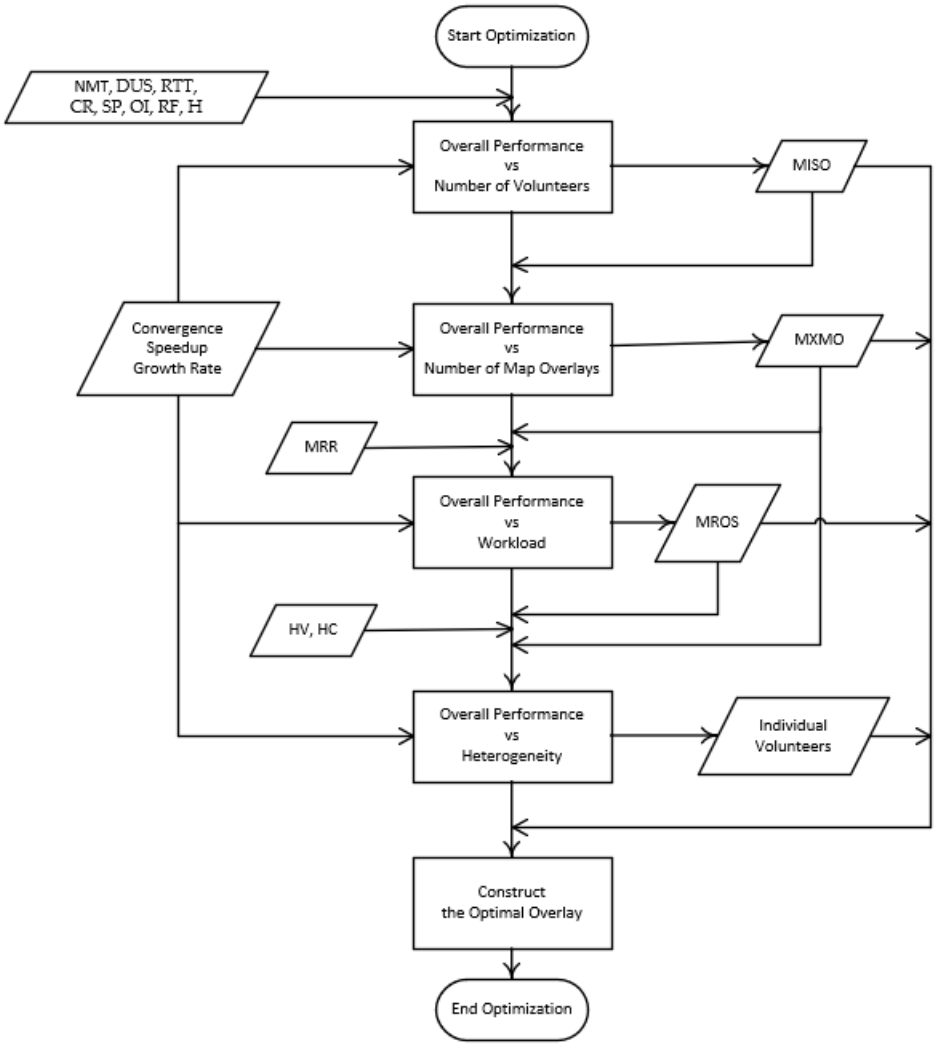

Figure 12. The optimization steps.

The first step of optimization is based on the overall performance vs. the number of volunteers as shown in Figures 2 and 3. Given a convergence speedup growth rate (e.g., *5%*), the minimum overlay size (*MIOS*) that can achieve the convergence growth rate can be determined accordingly. Any overlays that are larger than *MIOS* will achieve a benefit not bigger than the converged growth rate at the cost of growing the same number of volunteers; any overlays that are smaller than *MIOS* can only achieve a smaller overall performance than *MIOS*. Thus, *MIOS* is the preliminary level optimization on overlay size for the given big data problem and the dynamics of volunteers.

The second step of optimization is based on the overall performance vs. the number of map overlays as shown in Figures 10 and 11. Given a convergence speedup growth rate (e.g., *5%*) and use the *MIOS* obtained from the first step as the starting point, the maximum number of map overlays (*MXMO*) can be determined accordingly. Any separations that are more than MXMO will not achieve a bigger benefit than the convergence grow rate, but will increase the risk for fault tolerance for storage because of the fewer volunteers on the overlay; any separations that are less than MXMO can only achieve a smaller overall performance than *MXMO*. Thus, *MXMO* is the second level optimization on overlay size for the map step for the given big data problem and the dynamics of volunteers.

The third step of optimization is based on the overall performance vs. workload as shown in Figures 8 and 9. Given an *MRR* (e.g., *20%*), the workload of reduce step can be determined accordingly. Based on the convergence speedup growth rate (e.g., *5%*) on the workload, the minimum reduce overlay size (*MROS*) can be determined. Any overlays that are larger than *MROS* will achieve a benefit not bigger than the convergence growth rate at the cost of growing the same number of volunteers; any overlays that are smaller than *MROS* can only achieve a smaller overall performance than *MROS*. Thus, the *MROS* is the third level optimization on overlay size for reduce step for the given big data problem and the dynamics of volunteers.

The fourth step optimization is based on the overall performance vs. the heterogeneity of volunteers as shown in Figures 4–7. Given the compute capacities of volunteers and the *MROS* obtained from step 3, chose *MROS* volunteers from the pool of *MXMO* volunteers for compute capacities of *tier 1* and then of *tier 2* and so on. The chosen volunteers will form the reduce overlay; the non-chosen volunteers will leave after the map step. The *MROS* volunteers of the higher tiers will be the fast speedup for the reduce step for the given big data problem and the dynamics of volunteers.

The last step is to use the output of the optimization from the previous four steps, including the minimum overlay size (MIOS), the maximum number of map overlays (MXMO), the minimum reduction in overlay size (MROS) and the selected individual volunteers, to construct an optimal overlay, achieving the maximum speedup against the dynamics of volunteers using the minimum number of volunteers.

5.2. Case Study

This case study is to demonstrate the benefit that the proposed optimization strategies can bring with the support of simulations. Given:

1. A workload *1,400,000* map tasks and an *MRR* of *20%* as in the workload setting in Section 3.3;
2. The volunteer dynamics of *30%* churn and heterogeneity of *6 tiers* as in the overlay setting in Section 3.3;
3. The dataset of *108 TB* as in the data setting in Section 3.3;
4. The preliminary simulation of scalability as shown in Figures 2 and 3.

A *MIOS* of *40,000* volunteers can be determined for an overall speedup of *4267* times with a speedup growth rate not less than *9%*.

For the determined *MIOS* of *40,000* volunteers overlay, a simulation of performance for multiple map overlays starting from *40,000* volunteers can be conducted. Based on the simulation results as shown in Figures 10 and 11, the *MXMO* of *32* map overlays can

be determined for an equivalent overall speedup *6038* (*1,854,767* steps for a workload of *11,200,000,000*) with a speedup growth rate not less than *5%*.

For the given *MRR* of *20%*, the reduce step needs to perform *280,000* tasks. Starting from the *MIOS* of *40,000* volunteers, a simulation on performance vs. number of volunteers can be conducted as shown in Figures 8 and 9. Based on the simulation results, the *MROS* of *20,000* volunteers can be determined for an overall speedup *2200* times with a speedup growth rate not less than *24%*.

For the *MROS* of *20,000* volunteers, chosen *tier 1* to *tier 3* volunteers only from the entire pool of *40,000* volunteers to form the reduce overlay. Revise the overlay setting of Section 3.3 to include *tier 1* to *tier 3* volunteers only and perform a simulation for the overall performance on the reduce step only as shown in Figures 4–7. The result is *5415* (*413,697* steps for a workload of *2,240,000,000*). Thus, the final performance will be as follows:

1. The overall speedup is: *(11,200,000,000 + 2,240,000,000)/(1,854,767 + 413,697) = 5925* times;
2. The overall speedup growth rate is: *((5925 − 4267)/4267) × 100% = 38.86%*;
3. The overall improvement is: *5925 − 4267 = 1658*.

If we assume

1. The compute capacities of volunteers are evenly distributed between tiers, and
2. The compute capacities of volunteers can be evenly distributed into each overlay,

the construction of the overlays is to construct *32* map overlays and *1* reduce overlay. For each of *40,000* volunteers, it is on one of *32* map overlays. For the *20,000 tier 1* to *tier 3* volunteers, they are on the reduce overlay. The other *20,000* volunteers will leave on completion of the map and shuffle steps.

6. Conclusions

The misconceptions on the scalability of volunteer computing for big data processing have been clarified from four aspects. First, always welcoming volunteers onto the same overlay brings little benefit in terms of speedup growth rate. The cause is that the working cost on such a growing overlay cancels the increase in compute capacity that the new volunteers can bring in. Second, the very slow volunteers can contribute to or at least to impair the overall speedup when there are still available tasks for the fast volunteers. However, as the overall progress approaches to the end of computing, the slow volunteers will become stragglers, holding the tasks to starve the fast volunteers so that the overall performance is impaired. Third, there exists an optimal overlay size for a certain problem scale; neither a too small nor a too big overlay achieves the maximum speedup for the given problem scale. Fourth, a single round MapReduce application does not necessarily require a single overlay; multiple overlays can still maintain the one round requirement but bring opportunities for optimization of overall performance. Based on the clarification, the optimization strategies are four folds. The first step is to find a turning point of overlay size in terms of overall performance for a given problem. The second step is to form a certain number of map overlays and keeps a single reduce overlay. The third step is to determine an optimal overlay size for the reduce step in response to the MRR of the given problem. The last step happens before the start of reduce step in order to shrink the reduce overlay to the optimal size by dismissing the unnecessary slow volunteers. This paper has demonstrated that fast simulations help find these optimal points for a given big data problem and the dynamics of volunteers.

Author Contributions: Conceptualization, W.L.; data curation, W.L.; methodology, W.L. and M.T.; software, W.L.; validation, W.L. and M.T.; formal analysis, W.L. and M.T.; investigation, W.L. and M.T.; visualization, W.L.; writing—original draft preparation, W.L.; writing—review and editing, W.L. and M.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Oracle. An Enterprise Architect's Guide to Big Data—Reference Architecture Overview. Oracle Enterprise Architecture White Paper. 2016. Available online: http://www.oracle.com/technetwork/topics/entarch/articles/oea-big-data-guide-1522052.pdf (accessed on 12 March 2021).
2. Sarmenta, L. Volunteer Computing. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2001.
3. ATLAS@Home. Available online: http://lhcathome.web.cern.ch/projects/atlas (accessed on 12 March 2021).
4. Asteroids@home. 2021. Available online: http://asteroidsathome.net/ (accessed on 12 March 2021).
5. Einstein@Home. Available online: https://einsteinathome.org/ (accessed on 12 March 2021).
6. Li, W.; Guo, W.; Li, M. The Impact Factors on the Competence of Big Data Processing. *Int. J. Comput. Appl.* **2020**. [CrossRef]
7. Casado, R. The Three Generations of Big Data Processing. 2013. Available online: https://www.slideshare.net/Datadopter/the-three-generations-of-big-data-processing (accessed on 12 March 2021).
8. Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
9. Stoica, I.; Morris, R.; Liben-Nowell, D.; Karger, D.R.; Kaashoek, M.F.; Dabek, F.; Balakrishnan, H. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Trans. Netw.* **2003**, *11*, 17–32. [CrossRef]
10. Kaffille, S.; Loesing, K. *Open Chord (1.0.4) User's Manual 2007*; The University of Bamberg: Bamberg, Germany, 2007; Available online: https://sourceforge.net/projects/open-chord/ (accessed on 12 March 2021).
11. Fadika, Z.; Govindaraju, M.; Canon, R.; Ramakrishnan, L. Evaluating Hadoop for Data-Intensive Scientific Operations. In Proceedings of the IEEE 5th International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 67–74.
12. Dede, E.; Fadika, Z.; Govindaraju, M.; Ramakrishnan, L. Benchmarking MapReduce Implementations under Different Application Scenarios. *Future Gener. Comput. Syst.* **2014**, *36*, 389–399. [CrossRef]
13. Cheng, D.; Rao, J.; Guo, Y.; Jiang, C.; Zhou, X. Improving Performance of Heterogeneous MapReduce Clusters with Adaptive Task Tuning. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 774–786. [CrossRef]
14. Hadoop. Available online: https://cwiki.apache.org/confluence/display/HADOOP2/ProjectDescription (accessed on 12 March 2021).
15. Jothi, A.; Indumathy, P. Increasing Performance of Parallel and Distributed Systems in High Performance Computing using Weight Based Approach. In Proceedings of the International Conference on Circuits, Power and Computing Technologies, Nagercoil, India, 19–20 March 2015.
16. Yildiz, O.; Ibrahim, S.; Antoniu, G. Enabling Fast Failure Recovery in Shared Hadoop Clusters: Towards Failure-aware Scheduling. *Future Gener. Comput. Syst.* **2017**, *74*, 208–219. [CrossRef]
17. Singh, S.; Garg, R.; Mishra, P.K. Observations on Factors Affecting Performance of MapReduce based Apriori on Hadoop Cluster. In Proceedings of the International Conference on Computing, Communication and Automation, Greater Noida, India, 29–30 April 2016; pp. 87–94.
18. Ardagna, D.; Bernardi, S.; Gianniti, E.; Aliabadi, S.; Perez-Palacin, D.; Requeno, J. Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well-Formed Nets. In Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing, Granada, Spain, 14–16 December 2016; pp. 599–613.
19. Perarnau, S.; Sato, M. Victim Selection and Distributed Work Stealing Performance: A Case Study. In Proceedings of the 28th IEEE International Symposium on Parallel and Distributed Processing, Phoenix, AZ, USA, 19–23 May 2014; pp. 659–668.
20. Vu, T.T.; Derbel, B. Link-Heterogeneous Work Stealing. In Proceedings of the 4th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, USA, 26–29 May 2014; pp. 354–363.
21. Zhang, X.; Wu, Y.; Zhao, C. MrHeter: Improving MapReduce Performance in Heterogeneous Environments. *Clust. Comput.* **2016**, *19*, 1691–1701. [CrossRef]
22. Li, W.; Guo, W. The Optimization Potential of Volunteer Computing for Compute or Data Intensive Applications. *J. Commun.* **2019**, *14*, 971–979. [CrossRef]
23. Zhang, X.; Qin, Y.; Yuen, C.; Jayasinghe, L.; Liu, X. Time-Series Regeneration with Convolutional Recurrent Generative Adversarial Network for Remaining Useful Life Estimation. *IEEE Trans. Ind. Inform.* **2021**, *17*, 6820–6831. [CrossRef]
24. Chen, Y.; Ganapathi, A.; Griffith, R.; Katz, R. The Case for Evaluating MapReduce Performance Using Workload Suites. In Proceedings of the 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, Singapore, 25–27 July 2011; pp. 390–399.