IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 July 2022, accepted 14 July 2022, date of publication 25 July 2022, date of current version 1 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3193770

RESEARCH ARTICLE

The Performance Optimization of Big Data Processing by Adaptive MapReduce Workflow

WEI LI[1] **AND MAOLIN TANG**[2]
[1]School of Engineering and Technology, Central Queensland University, Rockhampton, QLD 4702, Australia
[2]School of Computer Science, Queensland University of Technology, Brisbane, QLD 4001, Australia

Corresponding author: Wei Li (w.li@cqu.edu.au)

This work was supported by the Research Translation Fund, Central Queensland University, Australia.

ABSTRACT The discussion context of this paper is big data processing of MapReduce by volunteer computing in dynamic and opportunistic environments. This paper conducts a series of simulations to explore the relationship between the overall performance of volunteer overlays responding to different workload of big data problems. The discovery from the simulations includes some optimization points in overlay size, going over which by adding more volunteers brings little benefit for the overall performance. Based on the discovery of optimization points, this paper proposes a bootstrapping protocol, which can adapt volunteers into variable-sizes overlays, enabling workflow of single-round MapReduce or multiple-round MapReduce, and a single or multiple overlays for each round. The variable overlays aim to create adaptive workflow during MapReduce processing, so that the optimization points can be caught. As another benefit, the unnecessary computing-capacities can be released during computing when the optimization points are reached. The case study shows a few optimization workflows that are formed by the proposed bootstrapping protocol to process the big data cases. The workflows lead to the optimization points and dynamically balance the workload at the same time. The experiment results have demonstrated that the optimization strategies have either achieved *36%* or *71%* higher performance than the plain MapReduce workflow and minimized the use of computing resources by releasing *12.5%* to *75%* volunteers during computing, where the original plain MapReduce must hold all the volunteers to the end of computing. The extensibility of the simulation parameterization to more diverse real-world applications have been clarified.

INDEX TERMS Big data, optimization, MapReduce, volunteer computing, simulation.

I. INTRODUCTION

Volunteer Computing (VC) [1], that is, harnessing millions of commodity computing resources together to cope with large scale compute-intensive or data-intensive problems, has been successfully applied to scientific projects such as ATLAS@Home [2], Asteroids@Home [3] and Einstein@Home [4]. The success of VC comes from the fact that it is a practical and cheaper way of distributed computing. To support this statement, we need to check the needs and conditions of data processing. Nowadays data, either of scientific projects to produce answers to scientific hypotheses or generated by business transactions or social events, have become a big size. Furthermore, 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. In short, big data fusion and analysis has become the pillars to support scientific research and business operations or business intelligence [5]. The condition of big data processing is a high-performance computing environment. To compute the global result, big data is too big to be processed by a single commodity computer in a reasonable amount time. Ideally, big data is to be processed by a dedicated data center, where a high-performance cluster consists of reliable computing nodes connected by fast networks, for real time data streaming or data warehousing. Unfortunately, small or medium business or scientific projects do not have the conditions to satisfy the needs. They are unable to invest such a data

The associate editor coordinating the review of this manuscript and approving it for publication was Bilal Alatas.

center. Thus, we conclude that for many small or medium business or scientific projects, they have to make use existing commodity computers to process big data in distributed environments. When this situation is extended to the Internet scale, the donated compute-cycles can be used to build such distributed environments. Consequently, to make use of commodity desktops or laptops or donated cycles from the Internet, we have to face a challenge of dynamics and opportunism, unreliability and heterogeneity rooted in such distributed environments. In addition, for big data processing in distributed environments, MapReduce [6] has been a successful programming paradigm, which is used for the discussions of this paper.

The first issue to make use of the volunteered computing-capacities is to ensure the reliability of the whole volunteer overlay though a volunteer can be unreliable as an individual. In this area, *Distributed Hash Table* (*DHT*) ensures the Internet-wide reliability of the overlay that is formed by dynamic and opportunistic volunteers. However, the reliability comes with the cost of stabilization when volunteers join, leave or crash. In addition, looking up a data item on the overlay incurs certain cost vs the size of an overlay. For example, the lookup and stabilization of DHT protocol Chord [7] are $O(_{log}n)$ and $O(^{2}_{log}n)$ respectively, where n is the number of volunteers on an overlay. The cost is meant that the thought that the more volunteers, the faster computing is just a misconception. The working cost will eventually consume more computing-capacities than the real tasks consume on a big-size overlay. Thus, increasing overlay size does not always bring benefits.

The existing work has explored one or two of volunteer features or dynamics. The existing work has not well explored the misconception that the more volunteers, the faster computing. The existing work has not well explored if a single round or multiple round MapReduce differs in performance in a fully dynamic environment. Our work in this paper places big data processing in a fully distributed, dynamic and opportunistic environment. Our work explores the working cost of big-size volunteer overlays and searches for performance convergence points. To make full use of the computing potentials of available volunteers, this work proposes single round with multiple overlays MapReduce and multiple round MapReduce to adapt and balance workload to the dynamics of volunteers. A bootstrapping protocol has been proposed to implement the adaptiveness. The evaluation results have confirmed the effectiveness and adaptiveness of the models and demonstrated the optimization in terms of improved performance or using fewer computing resources.

To introduce more of our work in this paper, the goal of this paper is optimal use of available volunteers to achieve high performance of big data processing, avoiding over consuming computing-capacities by the working cost (stabilization for volunteer dynamics, replication to cope with data loss and data integrity). For the goal, our research methodology is firstly modelling the dynamics and opportunism of volunteers. The modelling needs to represent the heterogeneity, unreliability, and randomness of volunteers. Accordingly, the first contribution of this paper is the parameterization of MapReduce features, volunteer dynamics and heterogeneity, networking patterns, big data workload and volunteer overlay scalability strategies. Second, the research methodology includes discovering the relationship between the workload of big data problems and the working cost on the overlay. Accordingly, the second contribution of this paper is a series of simulations. The analysis of simulation results is to discover the optimal overlay size for a given workload by a single plain and one-round MapReduce. Going over that size, the growing of more volunteers for the overlay will bring little benefit for the overall performance. Third, the research methodology includes modelling of workflow for multiple-round MapReduce and multiple-overlays MapReduce. The workflows aim to catch the optimization points during computing for achieving high performance. Accordingly, the third contribution of this paper is to propose a bootstrapping protocol. The protocol is to adapt volunteers into overlays, which vary in response to the change of workload during computing. The protocol is to organize overlays in series or in parallel or their combinations. The adaptive workflow is to catch the optimization points and to release compute-resources when they are no longer in need. The contribution of this paper also includes a physical implementation of a big data processing platform, which is built on the Open Chord APIs [8] and supports all the aforementioned volunteer dynamics and the bootstrapping protocol.

The case study of this paper provides experimental results to confirm that the proposed bootstrapping protocol can form adaptive workflow for the given big data problem. The workflows can either achieve a higher performance than that of plain MapReduce workflow or release computing-capacities during computing without affecting the performance or achieve both of the above.

The organization of this paper is as follows: related work is reviewed in Section II. The discussion context of this research is presented in Section III. Section IV prepares the settings of simulations and the measurement of performance. The rationale of the extensibility of simulation parameterization to more diverse real-world scenarios has been provided. In the first part of Section V, the discovery of optimization points is presented based on the simulation results. The second part proposes the bootstrapping protocol for the construction of adaptive workflow. A study case is detailed in Section VI to demonstrate the effectiveness of bootstrapping protocol to achieve the optimization points. Section VII concludes that optimal use of volunteers can achieve high performance without overusing resources.

II. RELATED WORK

The related work for optimizing MapReduce performance is mainly data locality and tuning of Hadoop [9] parameters in reliable and homogeneous cluster environments. Prabhu et al [10] reported a simple way to find optimal values for the three categories of *180* parameters: the core-related,

MapReduce-related, and HDFS-related. The tuning was fully based on trial and error. On a cluster of three homogeneous nodes and using the web log data of *2.1GB*, they reported that tuning some values of a certain number of parameters improved about *32%* for the performance comparing with the default settings of Hadoop. Targeting a real-world application: big data analytics in smart electrical grid, Khan *et al.* [11] proposed an enhanced parallel detrended fluctuation analysis (EPDFA) algorithm. EPDFA was built on Hadoop, which parameters were optimized by their proposed Gene Expression Programming. A cluster of *8* VMs was set as the testbed. The datasets consist of *8.6* million to *86.4* million samples. The test results of EPDFA outperformed their previous work DFA and PDFA for *26* times and *16* times in terms of speedup. Chen *et al.* [12] took effort to classify what parameters of Hadoop were important for overall performance. On a cluster of two nodes, they used benchmarks Sort, Kmeans and Wordcount on a *16GB* dataset to identify what parameters were CPU-bound, I/O bound or both. When testing a parameter, other parameters' values were kept constant. The metrics of test was execution time, and the importance of a parameter was quantified as root mean square of several test results deviated from the average result. If the quantified importance was greater than a user-set threshold, it was treated as important. The drawback of the method was that the setting of a threshold was subjective. Consequently, the applicability of the experimental results was not generic. Pattanshetti and Attar [13] selected three Apache big data platforms: Hadoop [9], Spark [14] and Storm [15] to study performance improvement by tuning hundreds of configurable parameters. The goal was to identify the optimal values for the parameters. The metrics was the drop of execution time for Hadoop and Spark and the gain of processed tuples for Storm. While tuning a parameter, other parameters were kept constant at their default values. The testbed consisted of a cluster of *4* nodes and a dataset of *100GB* for Wordcount and Terasort applications, and the experimental results showed significant improvement. They concluded that heuristic optimal values brought critical improvement to the overall performance. However, how the fine tuning was conducted or what parameters were critical was not described in their paper. The research focus of Htay and Phyu [16] was the tuning of Hadoop parameters as well. Their effort was to find optimal concurrent containers per node to improve performance in terms of Map Stage Elapsed Time (MSET). The following three parameters were selected because *HDFS block size* determined the *number of map tasks*, which subsequently determined the *optimal concurrent containers per node*. Experimental studies were conducted with a CPU-bound benchmark Wordcount, and I/O bound benchmarks Sort and Terasort. The dataset was *15GB*, and the testbed was a single VM. The optimal number of concurrent containers per node was *1* for both Sort and Terasort and *4* for Wordcount. Correspondingly, TeraSort improved *52%* and *59%*, Sort improved *47%* and *64%* and Wordcount improved *47%* and *64%* in terms of MSET for HDFS block size of *128MB* and *256MB* respectively. The method of Htay and Phyu [16] were very specific to a particular application and not effective to apply to other applications without case-by-case trial and error.

Lee *et al.* [17] termed the traditional map-only locality of Hadoop as shallow data locality. Then they proposed Deep Data Locality (DDL) to all stages of MapReduce. As a part of DDL, block-based locality was replicating data blocks to the mapper node, which would become reducer nodes. The other part, key based DDL, made use of ETL (extract, transform and load) operations that must be done before MapReduce starts to create files for different keys. The results of ETL were the data blocks that have homogeneous keys loaded to the same mappers. The proposed methods were tested on cloud, by a simulator and on a hardware implementation. The dataset was up to *120GB*, the benchmark was Wordcount and Terasort, and the cluster was up to *10* slave nodes in homogeneous architecture. The experimental results showed performance improvement up to *34%* comparing with the default Hadoop data locality. Eldouh *et al.* [18] had a similar goal of improving MapReduce performance by reducing the overhead caused by the shuffle stage of MapReduce. The application scenario of their work was SQL queries. To achieve the goal, they proposed to use TF-IDF (Term Frequency-Inverse Document Frequency) algorithm to calculate queries similarities. They used K-means algorithm to partition the related queries into clusters. An enhancement to HDFS was to co-locate the related data files in the same nodes. The TPC-H benchmark was used with up to *800M* records for a test on a cluster of *6* homogeneous nodes. They declared the experimental results of an improvement of *27%* in data locality and *40%* in execution time comparing with the Hadoop default. The dynamics and heterogeneity of computing nodes in larger networks were not discussed by their paper. Gandomi *et al.* [19] combined two existing techniques: dynamic job prioritizing and data localization to form a hybrid scheduling algorithm, aiming at increasing data locality rate, and decreasing completion time. The proposed schedulers were evaluated on a Hadoop cluster of one master node and *20* slave nodes, which had homogeneous architecture with stable and fast network connections. The dataset of the evaluation was *6.4GB* in *64MB* blocks. Using Wordcount benchmark, the proposed scheduler outperformed Hadoop FIFO scheduler but was similar to Hadoop Fair scheduler in both data locality rate and completion time; using Terasort benchmark, the proposed scheduler outperformed both Hadoop FIFO scheduler and Hadoop Fair scheduler in both data locality rate and completion time. The proposed schedulers were not evaluated in either dynamic or heterogeneous environments. Alanazi *et al.* [20] proposed to give priority to the jobs with the minimum data size and response time for job scheduling. To achieve that, they proposed an artificial neural network to predict resource usage and running jobs by Hadoop data nodes. They added an aggregator node to the HDFS of Hadoop to assign jobs among the data nodes, making the name node for tracking the aggregator nodes

only. They used a virtual cluster on Amazon EC2 and S3 for evaluations. The proposed model was compared with native Hadoop and other two approaches DGNS and iShufe using the Terasort benchmark for *1TB* dataset. The evaluation results showed better performance of the proposed method than the compared ones in both throughput and response time. The proposed model was not used for either heterogeneous or dynamic computing environments.

There exist a few related works of optimizing MapReduce performance in either dynamic or heterogeneous environments. Zhang *et al.* [21] treated heterogeneous nodes differently: fast nodes or straggler nodes. Their model allowed faster nodes to steal some work from the stragglers, which could slow down the completion of map step, preventing the reduce step from starting. Their study was used for heterogeneous environments but not applied to dynamic or unreliable environments. Yildiz *et al.* [22] coped with failure recovery by prioritizing tasks and allowing the tasks to pre-empt other tasks. The key idea was to alleviate the impact of node failure and achieve a reduction in overall completion time. In a small cluster of *19* nodes of *8*-core Intel Xeon CPUs connected by a *10Gbps* Ethernet network, they demonstrated the effectiveness of the model. Their work just focused on node failure without touching heterogeneity. Shu and Wu [23] proposed a set of models to classify VM types, to form workflow, and to represent resource requirements and time and financial cost of jobs. The goal was to minimize workflow makespan under a given budget constraint when the big data was processing in the public clouds. The evaluations were performed on a cluster of *20* VMs that were classified into *5* categories with homogeneous VMs in the same categories but heterogeneous VMs between categories. The workload was between 0.06×10^{15} and 2.16×10^{15} CPU cycles. The scheduler partitioned each MapReduce jobs into a certain number of homogeneous tasks and executed then on a selected set of VMs. The performance superiority of the proposed approach was demonstrated in terms of makespan, financial budget, workflow size, heterogeneity of VMs and workflow structure comparing with other existing algorithms GGB, GR, CPG. The heterogeneity was considered in terms of VM classification and corresponding scheduling, but dynamics of computing nodes was not modelled by their work.

The following three articles are the latest work related to this paper. J. C. S. Dos Anjos et al [24] developed a hybrid model of cloud as the task management (data distribution and resource allocation etc.) center and volunteer computers as the edge computing resources only. The goal of the hybrid model is to reduce the budget of big data processing using volunteer computing at a free monetary cost. Their models explored the relationship between workload, number of machines and load balancing for the optimized use of computing and storage resources and minimizing data transfer. A similarity of their model to our model of this paper is that their model deals with volunteers' crash, which is one of the dynamic factors that our model optimizes performance for. Another similarity is that both their and our models used generic tasks abstracted by data size, chunk size, number of tasks and computing intensity of tasks. The strength of their paper is to use cloud environments hosted in the Grid'5000 data center, simulating the cloud and volatile volunteer computers for the model evaluation. The test results showed a cost decrease down to *57.14%* compared to cloud computing only. Our model of this paper differs with their models from three aspects. First, our models are fully distributed and self-adaptive solution to task management without needing to use centralized task managers. Second, our models explore wider and stronger dynamics of volunteers such as different bandwidths for volunteers and not only crash but also join and leave of volunteers. Third, our models optimize performance using multiple round MapReduce, single round MapReduce with multiple map overlays with/without load balancing. The same research group [25] explored using the edge computing power of the IoT to improve the network latency in real-time big data streaming and processing. Their model accumulated micro-batches, which sizes were dynamically adjusted, to send instead of sending a single data item in each message. That reduced a set of data items to a single network latency. Their model introduced an adaptive method to distribute incoming streams to the stream processing framework. The micro-batches received at the stream processing framework were assigned to the machines based on dynamically updated CPU and memory states. In terms of the two ways of big data processing, our work in this paper is more related to [24] as both optimize batch processing. Readers would read the original paper [25] for dynamic micro-batch size adjusting and adaptive data partition if they work on stream processing.

The similarity between Gonzalo *et al.* [26] and our work of this paper is that both believe that the working cost (stabilization of volunteer dynamics, replication to cope with data loss and data integrity) on volunteer overlays prevents the overall performance from continuously improving but makes it converge. Thus, both believe continuously increasing volunteers to an already big-size overlay brings little benefit. However, the ways of optimizing the overall performance of a volunteer cluster are different between [26] and ours in this paper. Gonzalo *et al.* [26] tried to predict volunteer behaviors and then classify volunteers as high availability and low availability nodes. The high availability nodes were always allocated with tasks. However, the low availability nodes were not fully ignored but used as the complimentary computing resources to avoid overcrowding the high availability nodes. Their methods were quantitatively confirmed for the effectiveness by comparing to related work. When predicting volunteer availability is not always accurate or practical in a dynamic environment, our work of this paper allows any volunteers to participate a computing. In our model, the low availability nodes are always allowed to contribute to the overall performance. However, once the low availability nodes leave or crash, they are automatically bypassed as their unfinished tasks are checkpointed and picked up by other volunteers. In addition, our work avoids to use of big-size

overlays, where the working cost is comparable with the real task computing.

Our previous works [27], [28] researched on the clarification of the misconceptions of big data processing in dynamic and opportunistic environments [27] and the important insights to the impact factors and their strength of volunteer dynamics on big data processing [28]. Our previous works aim at paving a way to future optimization or avoidance of potential bottlenecks for big data processing, which is the work of this paper.

As reviewed of the existing work in this section, optimizing performance of MapReduce big data processing has not been well explored for decentralized or fully distributed environments to minimize monetary cost and maximize computing potential of volunteers. To deal with the gap, our work of this paper contributes to the field research in the following aspects.

1. Model a fully dynamic and opportunistic volunteer computing environment for big data processing.
2. Parameterize an extensible simulation framework to adapt the current application scenarios and be ready for future more diverse scenarios.
3. Explore the relationship between the working cost (stabilization for volunteer dynamics, replication to cope with data loss and data integrity) and peer overlay scalability to find the optimal overlay size for a given big data problem.
4. Propose a bootstrapping protocol to adapt volunteers into single round MapReduce or multiple round MapReduce to catch the optimization points.
5. Evaluate the performance of the proposed models and analyze the evaluation results to guide volunteer computing for big data processing in different scenarios.

III. THE DISCUSSION CONTEXT

This section presents the discussion context including the workflow of single-round or multiple-round MapReduce paradigm and result locality and globality. The dynamics of volunteers and the cost for guaranteeing the reliability of volunteer overlays are discussed. The integration of MapReduce paradigm and DTH paradigm is also discussed in the context of volunteer dynamics.

A. MAPREDUCE: PARADIGM AND WORKFLOW OF SINGLE OR MULTIPLE ROUNDS

MapReduce paradigm consists of a big number of map tasks and reduce tasks. The original big data, which must be formatted in the form of $<key, value>$ pairs, are divided into a big number of datasets and assigned to the map tasks as input. That is, each map task just processes a few subsets of the original big dataset. The local result of each map task will be shuffled into the reduce tasks as input. Depending on shuffling, the MapReduce workflow could be one-round or may incur multiple-rounds of map and reduce steps. Only when the condition, a round of shuffling ensures that all the $<key, value>$ pairs with the same key are assigned to the same reduce task, is satisfied, the results that are produced by the reduce tasks are global. If one-round of shuffling makes the condition satisfied, it results in a one-round MapReduce workflow as shown in Figure 1 (a) and (b). If multiple-rounds of shuffling together make the condition satisfied, it incurs a multiple-round MapReduce workflow as shown in Figure 1 (c). Furthermore, only when global results are produced, the MapReduce computing workflow ends. More formal modelling of MapReduce computing on volunteer overlays can be found in our previous work [27].

In MapReduce paradigm, multiple map or reduce tasks are to make use of the parallel computing capacity of an overlay in distributed environments. In the paradigm, map tasks or reduce tasks are computed in parallel on distributed computing nodes. The only synchronization happens when the results of map step need to be shuffled into the reduce tasks at the end of map step. The reduce tasks can start only when they are filled with data at the end of shuffling step. When a single-round MapReduce is the simplest way of the paradigm, it can be extended to multiple rounds in the sense of making use multiple small overlays to lower the working cost: for example, $O(_{log}n)$ for lookup or $O(^{2}_{log}n)$ for stabilization on a single large overlay, where the cost is logarithmically proportional to the number of volunteers n on the overlay [7].

B. OVERLAY: VOLUNTEER DYNAMICS AND GUARANTEE OF RELIABILITY

This paper is to use volunteers to form overlays to commit to MapReduce tasks. Thus, the dedicated part is minimized to start from a list of dedicated bootstrap nodes only. A bootstrap node is a reliable computing node, which is responsible for two duties. First, a bootstrap node traps volunteers to form an overlay. Second, a bootstrap node needs to upload the original map or reduce tasks to the overlay. Apart from the bootstrap nodes, all others are volunteers on the overlay. On the dynamics and opportunism of volunteers, a volunteer may leave before finishing a task or may crash while computing. Thus, the guarantee of reliability of the entire overlay is essentially necessary when the volunteers behave dynamics individually. For this concern, DHT protocols such as Chord [7] satisfy this requirement.

In addition, DHT has a natural match for MapReduce because both use the data format of $<key, value>$ pairs. Integrating the properties of MapReduce and DHT, a single-round MapReduce paradigm workflow is proposed as follows. Multiple-round MapReduce workflow is just a merger of multiple single-round MapReduce workflow as shown in Figure 1 (c).

In *map* step, a volunteer looks up a map task $<mk_{i0}, mt_i>$ on the overlay, where mk_{i0} is the key to task mt_i, $i \in \{1, 2, \ldots, m\}$ and m is the number of map tasks. The map task needs to be changed to $<mk_{i1}, mt_i, mts_i>$, where mts_i is the timestamp of the task. The volunteer downloads and executes mt_i. When the map task is in execution, the volunteer will need to update the timestamp mts_i in a regular

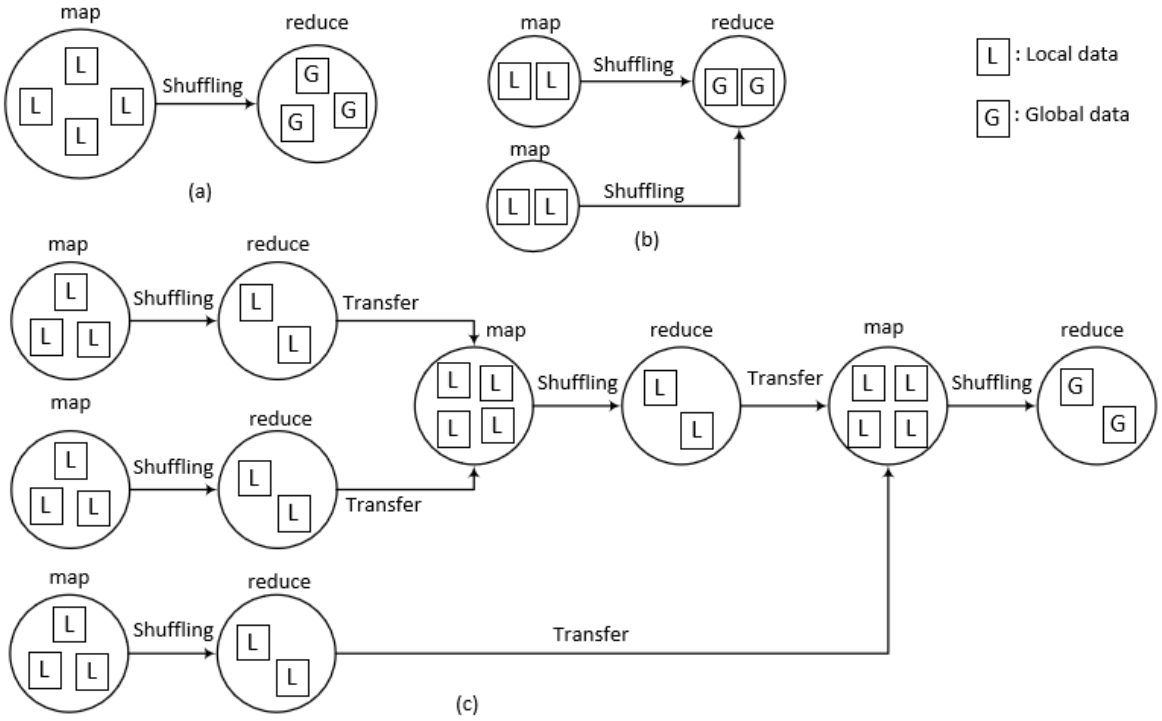

FIGURE 1. Single-round and multiple-rounds of MapReduce workflow vs result locality and globality. (a) one-round MapReduce with one map overlay and one reduce overlay, (b) one-round MapReduce with multiple map overlays and one reduce overlay, and (c) multiple-round MapReduce with multiple map overlays and multiple reduce overlays for the result locality and globality.

time interval *ui*. If the volunteer leaves during computing, the task mt_i is checkpointed and changed back to $<mk_{i0}, mt_i>$. If a task cannot be found by mk_{i0}, the volunteer looks up a map task $<mk_{i1}, mt_i, mts_i>$ that satisfies the condition: *(the current time - mts_i)> ui*, where mk_{i1} is the key. Such a map task was in execution by another volunteer that is treated as crashed already.

The shuffle step redistributes the result set of each map task into several reduce tasks. The shuffling procedure ensures all the *<key, value>* pairs with the same key in the result set will be merged in the form of *<key, a list of values>* and distributed into a single reduce task. The shuffling procedure is performed using a hash function so that the pairs emitted by different map tasks but with the same key can be distributed into the same reduce task. If the map step covers all the original data, the shuffling ensures that the reduce step will produce the global results.

In reduce step, a volunteer looks up a reduce task $<rk_{j0}, rt_j>$ on the overlay, where rk_{j0} is the key to task rt_j, $j \in \{1, 2, \ldots, r\}$ and r is the number of reduce tasks. The reduce task needs to be changed to $<rk_{j1}, rt_j, rts_j>$, where rts_j is the timestamp of the task. The volunteer downloads and executes rt_j. When the reduce task is in execution, the volunteer will need to update the timestamp rts_j in a regular time interval *ui*. If the volunteer leaves during computing, the task rt_j is checkpointed and changed back to $<rk_{j0}, rt_j>$. If a task cannot be found by rk_{j0}, the volunteer looks up a reduce task $<rk_{j1}, rt_j, rts_j>$ that satisfies the condition: *(the current time - rts_i)> ui*, where rk_{j1} is the key. Such a reduce task was in execution by another volunteer that is treated as crashed already.

Based on the integration of DHT and MapReduce paradigm as proposed as above, the dynamics of volunteers are tolerated in terms of the uncompleted tasks of dynamic (left or crashed) volunteers can be collected by other active volunteers. Thus, the MapReduce is guaranteed to finish as long as there are active volunteers on the overlay.

IV. THE EXTENSIBILITY OF SIMULATION SCENARIOS

This section formalizes the settings of volunteer dynamics, workload of big data problems, networking cost and MapReduce paradigm features into several simulation parameters and patterns. The quantitative measurement of overall performance of big data processing is defined in this section. This section describes how the parameterization of the impact factors produces extensibility for the application scenarios of big data processing, and how the extensibility could be achieved through two ways. An example setting is used to support the rationale behind the extensibility.

A. SETTINGS AND MEASUREMENT OF SIMULATION

The volunteer dynamics and opportunism can be simulated by discrete and random events rather than mathematically modelled [29]. The modelling of volunteer dynamics is to inject random events for volunteer join, leave and crash. Dynamics injection is to form a random pattern for volunteers to join an overlay and leave or crash on the overlay. A volunteer v_i joins an overlay at time jt_i=*random* $[i \times C_1, (i+1) \times C_1 - 1]$, where C_1 is a constant, representing the injection interval, $i \in \{0, 1, 2, \ldots, n-1\}$ and n is the total number of volunteers. For example, if n=*40,000* and C_1=*20*, the first volunteer

joins randomly between *0* and *19*, the second volunteer joins randomly between *20* and *39* and so on.

The leave or crash of volunteers is determined by several factors. First, the number of churn volunteers $cv{=}n{\times}CR$, where n is the total number of volunteers, is determined by a dynamic factor CR, representing *churn rate*. For example, if $n{=}40{,}000$ and $CR{=}30\%$, there will be *12,000* volunteers to commit churn (leave or crash) during computing. To evenly distribute the churn among all volunteers, the volunteers can be divided into cv domains, of which each domain contains $D{=}1/CR$ volunteers including a single churn volunteer. The churn volunteer v_k in domain j can be determined by $k{=}random[j{\times}D,\ (j{+}1){\times}D\text{-}1]$, where $j \in \{0, 1, 2, \ldots, cv\text{-}1\}$. If we define other two dynamic factors, we can fully simulate a churn window for each volunteer.

1. *Start Position (SP)* is a factor to reflect how long a volunteer has been working on the overlay before committing dynamics.
2. *Occurrence Interval (OI)* is a factor to reflect the time-window in which a volunteer could commit churn.

Based on the factors, the leave or crash time of a churn volunteer v_k will be $lct_k{=}random[jt_k{+}SP, jt_k{+}SP{+}OI]$. That is, after joining the overlay at jt_k, the volunteer needs to stay on the overlay for *SP* long and then can commit churn randomly in the time-window of a width of *OI*. Furthermore, if k is an odd number, the volunteer commits leave. Otherwise, it commits crash. Visually jt_i and lct_k will behave dynamics as shown in Figure 2, where the ascending line represents joins (jt_i); the descending line represents leaves or crashes(lct_k).

The other factors that reflect the features of big data processing on volunteer overlays include:

1. *Heterogeneity(H)* is a factor to reflect the computing capacity of volunteers. For evaluation purposes, we assume the base capacity as *tier-1*. Thus, a *tier*-2 volunteer is 2-times slower.
2. *Download/Upload Speed (DUS)* is a factor to reflect the internet speed of a volunteer. If we use a moderate internet speed of *25/10* Mbps as an example, the *DUS* is *20/51* seconds for download/upload of a *64MB* data.
3. *Round Trip Time (RTT)* is a factor to reflect the time to establish/close an internet connection between two volunteers. An *RTT* of *8* time-unit is a conservative setting for a moderate internet speed.
4. *Map-to-Reduce Ratio (MRR)* is a factor to reflect data compression/expansion features of a MapReduce application. For example, a *MRR* of *20%* decreases the workload and data scale of reduce tasks to *20%* of map tasks. When data aggregation ($input{>}output$) and data summary ($input{\gg}output$) are the most common MapReduce applications, data transformation ($input{\approx}output$) and data expansion ($input{<}output$) are still some applications [30].
5. *Redistribution Factor (RF)* is a factor to reflect the difference between the keys of the result set that is emitted by a map task. For example, a *RF* of *200* means the $<key, value>$ pairs of a result set produced by a map task has *200* different keys. The result set needs to be redistributed into *200* reduce tasks in the shuffle step so that the globalization of results can be achieved by the reduce step.

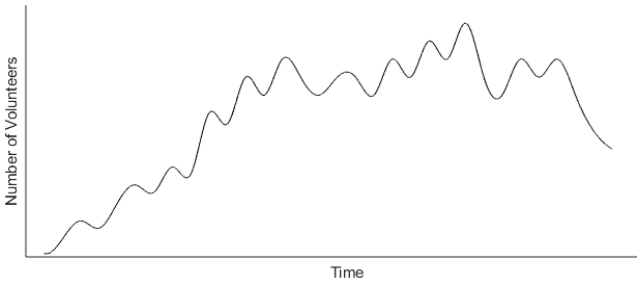

FIGURE 2. The simulation of dynamics of volunteers.

Integrating the above impact factors, the overlay setting *(6-tiers, 20/51, 8, 30%, 250K, 30, 20%, 200)* in the format of *(H, DUS, RTT, CR, SP, OI, MRR, RF)* is to reflect a very general situation. The overlay scales are set to start from *5,000* volunteers with *5,000* more volunteers each time growing up to *120,000* volunteers. To simulate other scenarios, we only need to change the values in the pattern *(H, DUS, RTT, CR, SP, OI, MRR, RF)*.

To reflect a big data problem, the workload setting is randomly choosing e.g., *1,400,000* (*1.4M*) map tasks. With an *MRR* of *20%*, the number of reduce tasks are *280,000*. The computing load of each map or reduce task is random set e.g., *8,000*. Thus, the total computing workload is *13.44G*. The unit of computing load is omitted because this setting is contrasting the competence between different strategies. The unit can be seconds, hours, or days.

The *64MB* is very common for a dataset of a map or a reduce task or a result set [24]. Based on the number of map and reduce tasks, the data setting is about *108TB* ($1{,}400{,}000{\times}64MB{=}89.6TB$ of map plus *17.92TB* of reduce if *20% MRR* is assumed) to be processed.

The two measurements that are used to make quantitative investigations of the overall performance and how the overall performance changes are *Speedup* and *Speedup Growth Rate*. Speedup is used to measure the overall performance of a volunteer overlay on the given workload of a problem and the given dynamics of volunteers. Speedup Growth Rate is to measure how the speedup changes between two overlays, which have different values on the impact factors.

When both reflect the overall performance of overlays, the speedup growth rate is finer than the speedup. The speedup growth rate can check what factors have a stronger impact or whether performance scalability converges to a specific value.

B. THE EXTENSIBILITY OF SIMULATION SCENARIOS

It is shown in Figure 3 how the impact factors/parameters determine simulation scenarios. The big data analysis is processing key-value pairs if it is MapReduce-based. Depending

on the application scenarios, the map to reduce relationship could be data aggregation (*input*>*output*), data summary (*input*≫*output*), data transformation (*input*≈*output*) and data expansion (*input*<*output*) [30]. Among them, the first two are the most common cases. To reflect the feature, a factor/parameter named *Map-to-Reduce Ratio (MRR)* is sufficient to represent all the above four application scenarios. MapReduce is also featured by the globality of a map result set. The lists of key-value pairs in a map result set should be by the shuffle step to redistribute into a few of reduce tasks so that the key-value pairs with the same keys can aggregate into a single reduce task. Thus, the result set of each reduce task will be globalized. To reflect this feature, a factor/parameter named *Redistribution Factor (RF)* is sufficient to represent the number of reduce tasks that the result set of a map task should be redistributed for the globalization of results in the next step.

Many dynamic volunteers behave randomly and freely on leaving and crashing on the volunteer overlay. On the other hand, there should be many volunteers, who would stay on the overlay till the entire big data processing completes. Thus, a *Churn Rate (CR)* factor/parameter is used to reflect the volunteers who behave dynamics gracefully or ungracefully. Furthermore, the random leave or crash of dynamic volunteers are supposed to happen in a time window since a volunteer starts working on the overlay. Thus, a volunteer commits leave or crash randomly in a time window that can be represented by the factors/parameters *Start Position (SP)* and *Occurrence Interval (OI)*.

Volunteers' computers are different in memory and computational power. If we assume that an operating system would not crash for a big job on a small memory machine but does it slowly because of data exchange between the memory and the external storage, we can generalize a volunteer's multi-factor computing power to a single factor/parameter named *Computing Capacity (CC)* in this paper. To reflect the heterogeneity of volunteers, the computing capacities of volunteers are tiered. That is, if we assume that the base capacity is *1*, a volunteer's computing capacity will be a times ($>$, $=$ or $<$ *1*) of the base capacity. Furthermore, the download/upload speed are different for volunteers' networks. Thus, a factor *Download/Upload Speed (DUS)* is used to reflect the download or upload time for a certain *Size of Data Set (SDS)*. The *64MB* or *128MB* data sets are the commonly used sizes for big data processing.

For Chord-based communication [7], [8], each volunteer sees the overlay rather than individual peers. A volunteer needs to download data sets and upload result sets. Thus, the aforementioned *DUS* is an impact factor on communication.

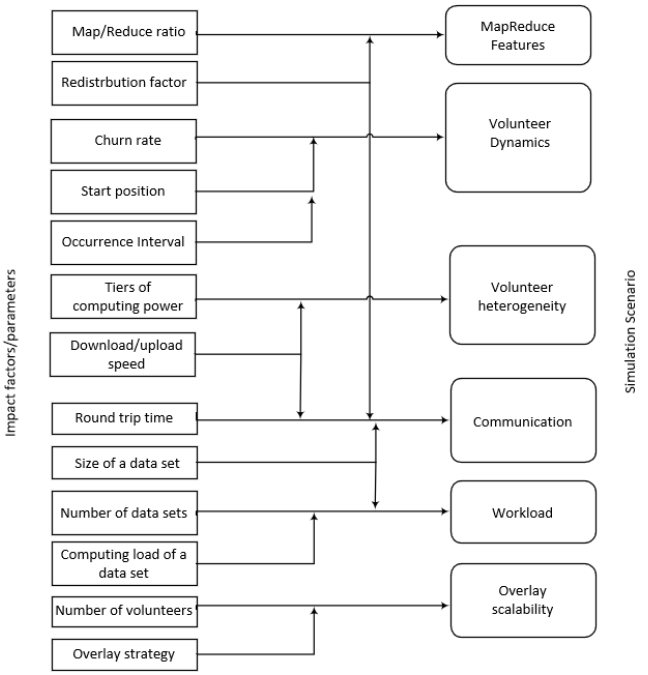

FIGURE 3. The relationship between the impact factors and simulation scenarios.

The open and close of connections before or after communication constitutes a *Round Trip Time (RTT)*, which is also a factor/parameter to reflect communication. In addition, the aforementioned redistribution factor *RF* has an impact on the communication because the bigger a *RF* is, the more communication links are needed, and more data packets are sent.

When a big data is divided into many data sets, the *Number of Data Sets (NDS)* is a factor to have impact on the workload. Once the size of a data set is certain, the *Computing Load of a Data Set (CLDS)* can be determined on a base-capacity volunteer, and thus on any volunteers. Furthermore, the number of map result sets equals the number of map data sets, and the number of reduce data sets equals $NDS \times MRR$. Thus, the entire workload of the big data processing is determined.

As the last scenario-related matter, the overlay scalability is determined by the *Number of Volunteers (NV)* and how the volunteers are organized into overlays, that is, the *Overlay Strategies*.

The above parameterization of simulation impact factors has covered a variety of aspects of big data processing, including MapReduce features, volunteer dynamics, volunteer heterogeneity, communication, big data workload and overlay scalability. The parameterization is ready for extending the application scenarios in two ways.

$$Speedup = \frac{The\ total\ time\ to\ complete\ the\ entire\ problem\ by\ a\ volunteer\ overlay}{The\ total\ time\ to\ complete\ the\ entire\ problem\ by\ a\ single\ volunteer\ of\ tier-1} \quad (1)$$

$$Speedup\ Growth\ Rate = \frac{The\ speedup\ of\ value2 - The\ speedup\ of\ value1}{The\ speedupof\ value1} \times 100\% \quad (2)$$

```
<factors>
   <factor>
      <featureID>MapReduce</featureID>
      <mapreduceRatio>20%</mapreduceRatio>
      <redistributionFactor>200</redistributionFactor>
   </factor>
   <factor>
      <featureID>VolunteerDynamics</featureID>
      <churnRate>30%</churnRate>
      <startPosition>250K</startPosition>
      <occurrenceInterval>30</occurrenceInterval>
   </factor>
   <factor>
      <featureID>VolunteerHeterogeniety</featureID>
      <tiers>6</tiers>
      <downloadUploadSpeed>20/51</downloadUploadSpeed>
   </factor>
   <factor>
      <featureID>Communication</featureID>
      <roundTripTime>8</roundTripTime>
   </factor>
   <factor>
      <featureID>Workload</featureID>
      <datSetSize>64MB</dataSetSize>
      <numberDataSet>1.4M</numberDataSet>
      <dataSetComputingLoad>8000</dataSetComputingLoad>
   </factor>
   <factor>
      <featureID>OverlayScalability</featureID>
      <volunteerNumbers>120K</volunteerNumbers>
      <overlayStrategy>1</overlayStrategy>
   </factor>
</factors>
```

FIGURE 4. An example of application scenario determined by the impact factors.

1. To extend the simulation scenarios for a newer factor, the developer needs to include an xml *<factor/>* tag in the *Application Scenario Properties (ASP)* file. Accordingly, a newer function module is to be implemented and added to the simulator framework.
2. To hold the existing factors but to extend to another scenario with different parameter values, the end user would reset the settings of the factors/parameters. For example, to simulate an application scenario for processing a big data, the scenario part of the ASP file is set as shown in Figure 4.

The parameter settings in Figure 4 represent a simulation scenario that processes an original data set of *89.6TB*. The big data is divided into *1.4* million data sets of *64MB* per data set. Each data set needs *8,000* time-units of computing load. The big data will be processed by *120,000* volunteers, who form overlays based on *Strategy-1* (detailed in Section VI-A) for the processing. There are *30%* volunteers committing churn during the computing, and the churn happens randomly in a time window with the start position *250k* (since a volunteer joins the overlay) for an occurrence interval of *30*. The volunteers' computing power is *6* tiers and the upload/download speed of a *64MB* data set is *20/51* time-units. This is a data summary application with *20%* map-to-reduce ratio. Each local map result set will be redistributed into *200* reduce tasks for the final computing and globalization.

V. OPTIMISATION STRATEGIES

We have implemented a simulator to support adaptive MapReduce workflow on the Open Chord APIs [8]. On the basis of the simulations as given in the settings of Section IV, this section presents a detailed analysis of the overall performance of volunteer overlays in different sizes responding to different workload. The analysis aims discovering possible optimization points, over which growing overlay size will bring little benefit. A bootstrapping protocol is proposed in this section to form adaptive workflow, which can catch the optimization points to achieve high performance and not to overuse volunteers at the same time.

A. SEARCH FOR POSSIBLE OPTIMISATION POINTS

Based on the settings in Section IV, the sample results have been produced to demonstrate the performance scalability in terms of speedup and speedup growth rate vs the number of volunteers as shown in Figure 5 and Figure 6. To clarify the results, we choose a typical point: overlay of *55K* volunteers. To contrast speedup with speedup growth rate, the overlay with *55K* volunteers achieves *5,004* times speedup with speedup growth rate of *4.8%* higher than the overlay of *50K* volunteers. Before this point, the speedup is smaller than *5,004* but the speedup growth rate is greater than *4.8%*; after this point, the speedup is greater than *5,004* but the speedup growth rate is smaller than *4.8%*. This contrast of scalability vs number of volunteers has demonstrated that for the given workload and volunteer dynamics, simply increasing volunteers more than a certain number, e.g., *55K* in this evaluation, brings little benefit, e.g., less than *4.8%* speedup growth rate for every *5,000* more volunteers in this evaluation. The reason is that the working cost (stabilization for volunteer dynamics, replication to cope with data loss and data integrity) on an overlay is proportional to the number of volunteers on the overlay, which behave dynamics. This result is also supported by the computational complexity of performance and stabilization of Chord protocol [7], which the lookup cost is $O(_{log}n)$ and the stabilization of overlay on volunteer dynamics is $O(^{2}_{log}n)$, where n is number of volunteers on the overlay. Thus, the working cost of a larger overlay will finally cancel/weaken the computing-capacity it brings real tasks.

To scrutinize the relationship between performance scalability and other impact factors, we extend the evaluations to explore the relationship between performance and varying workload or varying overlay sizes. On the basis of the settings in Section IV, we introduce the following variables as shown in Table 1, where W represents the workload of *1.4M* tasks as in the settings in Section IV. The settings explore how the performance varies on a varying workload (such as *0.02W* to *0.2W*) on an overlay (such as *5K* volunteers) or on a workload (such as *0.2W*) on varying overlays (such as *5K* to *20K* volunteers).

Based on the settings, the sample results have been produced to demonstrate the impact on performance scalability that is caused by the potential relationship between

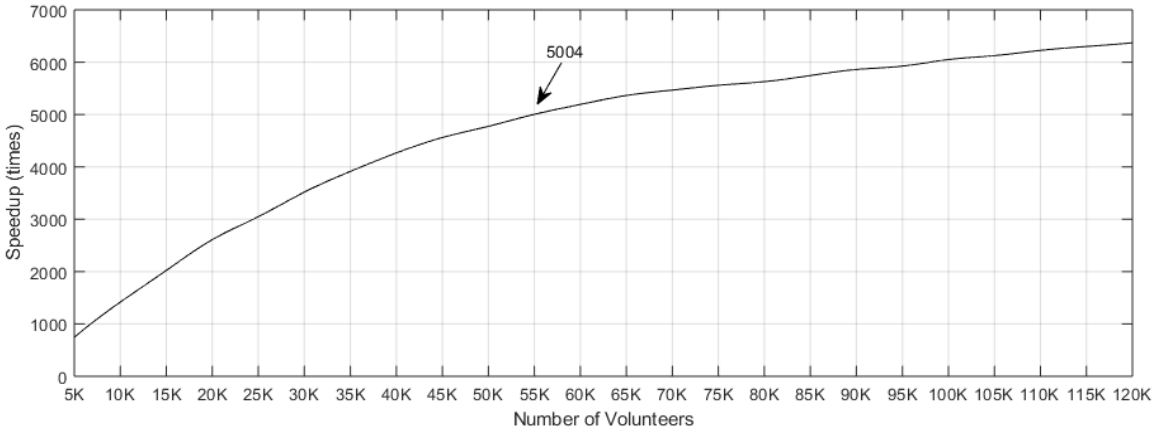

FIGURE 5. The performance scalability (speedup) vs the number of volunteers.

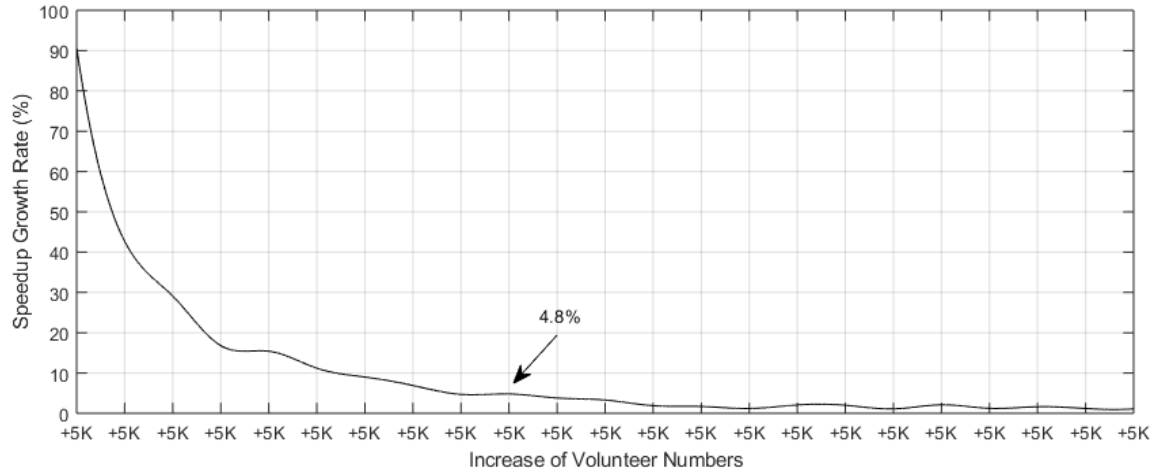

FIGURE 6. The performance scalability (speedup growth rate) vs volunteer number growing.

TABLE 1. The relationship settings between varying workload and varying overlay size.

Relationship (workload plus number of volunteers)	Workload (×1.4M)	Overlay Size (×5,000 volunteers)
(0.02~0.2)W+(5K)N	0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18, 0.2	1
0.2W+(5K~40K)N	0.2	1, 2, 3, 4, 5, 6, 7, 8
(0.2~2)W+(40K)N	0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2	8
2W+(40K~80K)N	2	8, 9, 10, 11, 12, 13, 14, 15, 16

workload and overlay size. The performance scalability in terms of speedup and speedup growth rate are shown in Figure 7 and Figure 8. Finding the optimization points needs to contrast the results in the four figures of Figure 5 to Figure 8.

1) SPEEDUP GROWTH RATE CONVERGENCE

For a certain workload, continuingly increasing overlay size is not continuingly beneficial as much as proportional to the overlay growing size. That feature can be observed on Figure 5 and Figure 6, where the workload is a single *W* and the overlay size is from *5K* growing to *120K*. The speedup is climbing slowly for the same size of overlay growing and the speedup growth rate keeps dropping and finally converges to about *1.2%*. When this observation is double-checked by varying workload vs varying overlay sizes, the second and fourth part (counting from left to right) of Figure 7 and Figure 8, where overlay sizes grow for the same workload, show the same feature. Thus, depending on satisfaction, the first optimization point can be found if a single-round MapReduce is required. This point is in the scalability of the full workload for the satisfaction. For example, for the workload of *W* and the satisfaction is *4.8%* (Figure 6) speedup growth rate, the optimization point will be *55K* (Figure 5) volunteers overlay, which can achieve a speedup of *5,004* times (Figure 5). As another example, for the workload of *2W* and the satisfaction is *3%* speedup growth rate, the optimization point will be *70K* volunteers overlay (Point-1 in Figure 7), which can achieve a speedup of *6,312* times. For overlays more than *70K*, they can achieve a higher speedup.

However, they can only achieve a lower speedup growth rate (i.e., satisfaction).

2) MULTIPLE OVERLAYS

There are three observations with the relationship evaluations in Figure 7 and Figure 8. First, for a given overlay size, there is an optimal workload in terms of satisfaction. For example, if the satisfaction is not less than *10%* speedup growth rate and the overlay size is *5K*, the optimal workload is *0.16W* (*Point-4* in Figure 7). As another example, if the satisfaction is not less than *3%* speedup growth rate and the overlay size is *40K*, the optimal workload is *W* (*Point-2* in Figure 7). Second, for a given workload, there is an optimal overlay size in terms of satisfaction. For example, if the satisfaction is not less than *13%* and the workload is *0.2W*, the optimal overlay size is *25K* (*Point-3* in Figure 7). As another example, if the satisfaction is not less than *3%* and the workload is *2W*, the optimal overlay size is *70K* (*Point-1* in Figure 7). Third, for a given workload, increasing overlay size makes speedup growth rate bounce high as shown by *Point-1*, *Point-2* and *Point-3* in Figure 8 and then drops again. In addition, the bounce height is greater for a smaller overlay than a larger overlay. For example, the bounce heights in descending order are *10K*, *40K* and *50K* volunteer overlays as shown by Point-1, Point-2, and Point-3 in Figure 8. It implies that increasing the same number of volunteers on a larger overlay brings less benefit than on a smaller overlay in terms of speedup growth rate. All the above three observations have demonstrated possible optimization points if splitting workload and overlays. Splitting workload and using smaller multiple overlays can result in:

- A multiple-round MapReduce.
- A single-round MapReduce with multiple map overlays and a single reduce overlay.
- A combination of the above.

3) COMPUTING-CAPACITY SELECTIVITY

Using multiple smaller overlays can bring an incidental benefit: providing selectivity of volunteers in terms of computing-capacity. This situation is particularly suitable for data aggregation and data summary application [30], where *data input*>*data output* or *data input*≫*data output*. In the situations, the optimal overlay size for reduce step could be much smaller than the overlay size of map step. As a result, the volunteers of higher computing-capacity can be selected from a large pool of available volunteers. For example, based on the simulation results in Figure 7 and 8, for a workload of *W* and the number of map tasks is *1,400,000*, the optimal overlay size of map step will be *40K* volunteers if a single round MapReduce is used. If *MRR* is *20%*, the workload is *280,000* for reduce tasks. For such as a workload, the optimal overlay size for reduce step will be *20K*. Thus, if the original pool of volunteers is of *40K* with computing-capacity of *tier-1* to *tier-6* evenly, the volunteers of *tier-1* to *tier-3* can be selected for the reduce step, and the volunteers of *tier-4* to *tier-6* can be released once the map step finishes.

B. BOOTSTRAPPING PROTOCOL

The proposed architecture of big data processing is formed by decentralized multiple overlays. The only dedicated requirement is a few of dedicated computing nodes as the bootstrapping nodes to form the overlays. In addition, the dedicated nodes are responsible for uploading the original map tasks (computing logic filled with data) and reduce tasks (only computing logic without data). Otherwise, all of others are volunteers, who behaves dynamics or opportunism during computing. An overlay is guaranteed reliable by Chord [7] DHT protocol although individual volunteers are dynamic or unreliable. The multiple overlays are interconnected by the bootstrapping protocol to enable an adaptive structure for MapReduce as follows.

1. MapReduce paradigm can be a single-round or multiple-rounds processing in a self-organizing way without any centralized control.
2. A round of MapReduce can use the same overlay or two different overlays for the map step or reduce step.
3. A round of MapReduce can use multiple overlays for map step but a single overlay for reduce step.
4. The life-cycle listeners can monitor the start, progress and completion of the map step or reduce step.
5. The number of volunteers on an overlay is adapted without impacting the MapReduce paradigm.

The bootstrapping protocol works in a fully distributed mode as proposed as follows.

When a volunteer v_i joins, it retrieves a list of bootstrap nodes: $bs_{i,1}, bs_{i,2}, \ldots, bs_{i,m}$, where $i \in \{1, 2, \ldots, n\}$ and n is the number of bootstrap nodes. The following are the constraints on a bootstrap list.

1. The number of bootstrap nodes on a list must be a positive even number, i.e., $m=2k$ and $k \in N_1$.
2. Each bootstrap node bs is a dedicated computing node, representing a volunteer overlay that is formed by bootstrapping volunteers from it.
3. The bootstrap nodes $bs_{i,1}, bs_{i,2}, \ldots, bs_{i,m}$ are m dedicated computing nodes. The node $bs_{i,1}$ is responsible for uploading the original map tasks on its overlay. The uploading pace depends on the number of volunteers on the current overlay. The rule is to balance task availability and fault tolerance of storage. For example:
   - Uploading tasks of the double number of volunteers on the overlay if the number of volunteers is less than *100*.
   - Uploading tasks of the triple number of volunteers on the overlay if the number of volunteers is between *100* and *1,000*.
   - Uploading all tasks if the number of volunteers is greater than *1,000*.
4. For any two contiguous nodes $bs_{i,j}$ and $bs_{i,j+1}$ on the list and $j \in \{1, 3, \ldots, m\text{-}1\}$, they represent a round of

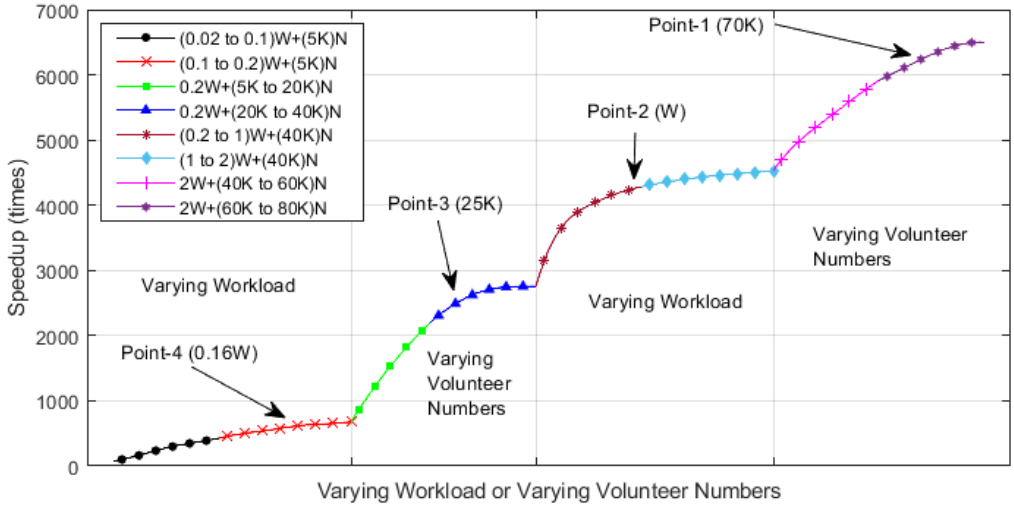

FIGURE 7. The performance scalability (speedup) vs varying workload or varying volunteer numbers.

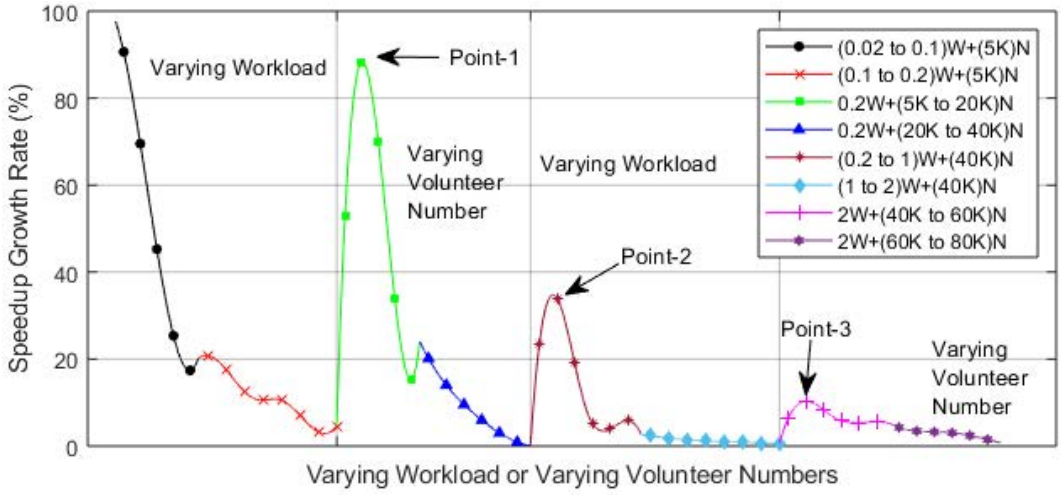

FIGURE 8. The performance scalability (speedup growth rate) vs varying workload or varying volunteer numbers.

MapReduce, where $bs_{i,j}$ represents an overlay for map tasks and $bs_{i,j+1}$ represents an overlay for reduce tasks.

5. For any two contiguous node $bs_{i,j}$ and $bs_{i,j+1}$ on the list and $j \in \{1, 2, \ldots, m\text{-}1\}$, if $bs_{i,j} = bs_{i,j+1}$, the map step and reduce step use the same overlay. If $bs_{i,j} \neq bs_{i,j+1}$, the map and reduce step use different overlays.
6. Node v_i always downloads map tasks from $bs_{i,j}$ overlay to perform and shuffles the reults to $bs_{i,j+1}$ if $j \in \{1, 3, \ldots, m\text{-}1\}$.
7. Node v_i always downloads reduce tasks from $bs_{i,j}$ overlay to perform and simply uploads results to $bs_{i,j+1}$ if $j \in \{2, 4, \ldots, m\text{-}2\}$.
8. Node v_i downloads reduce tasks from $bs_{i,m}$ to perform and simply uploads the results to8$bs_{i,m}$ as well.

Some overlay examples that can be formed by the bootstrapping protocol are given as follows.

If a list of $\{bs_1, bs_2\}$ returns to every volunteer, this is a typical single-round MapReduce. In this structure as shown in Figure 9 (a), normally $bs_1 = bs_2$, that is, a single overlay for both map (including shuffle) step and reduce step.

If a volunteer retrieves one of the two lists: $\{bs_{11}, bs_{12}, bs_3, bs_4\}$, $\{bs_{21}, bs_{22}, bs_3, bs_4\}$, this is a two-round MapReduce. In this structure as shown in Figure 9 (b), normally $bs_{11} = bs_{12}$, $bs_{21} = bs_{22}$ and $bs_3 = bs_4$. That is, there are three overlays; two overlays are for the first-round MapReduce and one overlay is for the second-round MapReduce.

If a volunteer retrieves one of the following lists: $\{bs_{11}, bs_{12}, bs_3, bs_4, bs_5, bs_6\}$, $\{bs_{21}, bs_{22}, bs_3, bs_4, bs_5, bs_6\}$, $\{bs_{31}, bs_{32}, bs_5, bs_6\}$, this is a three-round MapReduce. In this structure as shown in Figure 9 (c), normally $bs_{11} = bs_{12}$, $bs_{21} = bs_{22}$, $bs_{31} = bs_{32}$, $bs_3 = bs_4$ *and* $bs_5 = bs_6$. That is, there are five overlays; three overlays are for the first-round MapReduce, one overlay is for the second-round MapReduce and one overlay is for the third-round MapReduce.

If a volunteer retrieves one of the following lists: $\{bs_{11}, bs_2\}$, $\{bs_{21}, bs_2\}$, $\ldots$, $\{bs_{i1}, bs_2\}$, where $i \in \{1, 2, \ldots\}$, this

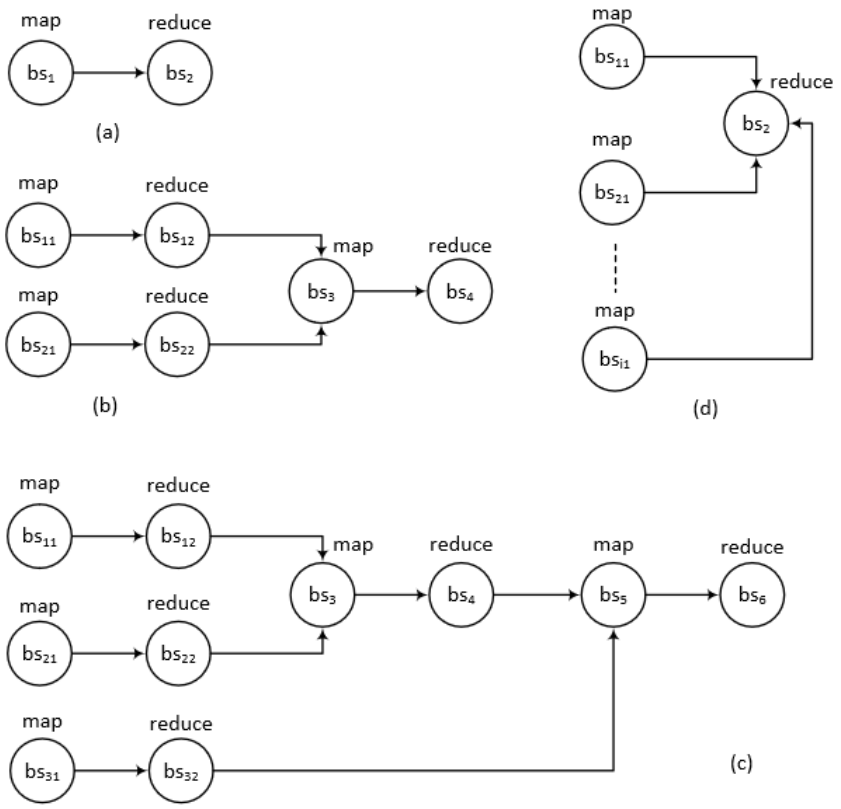

FIGURE 9. Examples of overlay structure for single or multiple rounds of MapReduce. (a) a single overlay for both map step and reduce step, (b) two overlays for the first-round MapReduce and one overlay for the second-round MapReduce, (c) three overlays for the first-round MapReduce, one overlay for the second-round MapReduce and one overlay for the third-round MapReduce, and (d) multiple overlays for map tasks and a single overlay for reduce tasks in a single round MapReduce.

a single-round MapReduce. In this structure as shown in Figure 9 (d), there are $i+1$ overlays. When $i \geq 2$, multiple (i) overlays are for map tasks and a single overlay is for reduce tasks.

To facilitate monitoring the progress of a map step or a reduce step on an overlay, a progress marker *pm* is attached to a bootstrap node *bs*, denoted as $<bs, pm>$ as a global object synchronized and shared between all volunteers who bootstrap at *bs*. The initial value of *pm* is the total number of tasks (either for map or reduce step) on the overlay. Once a volunteer completes a task and the result has been successfully uploaded to the overlay of *bs*, the value of *pm* is counted *1* less. When $pm>0$, there are tasks available or running on the overlay of *bs*. If $pm=0$, the map step or reduce step performed on the overlay of *bs* is completed.

Depending on varying workload in different steps of a MapReduce round or in different rounds of MapReduce, a volunteer may be necessary for an overlay but unnecessary for another overlay in terms of computing the corresponding workload. Because a volunteer can upload results to an overlay only where it stays on, the volunteer must be on the overlay of next step computing. If the volunteer is not necessarily to involve next step computing, it will need to leave after uploading all the results it has produced and before the start of next computing.

The control of volunteer numbers on an overlay *bs* can be achieved by registering a listener *lnr()* to the progress marker *pm* of the overlay *bs*. The use of listener *lnr()* is as follows. First, a volunteer v_i needs to join every overly identified by the bootstraps: $bs_{i,1}, bs_{i,2}, \ldots, bs_{i,m}$ on the given list. Assume that v_i is computing on $bs_{i,j}$ and uploading results onto $bs_{i,j+1}$, where $j \in \{1, 2, \ldots, m-1\}$. The *lnr()* will be triggered on the event that *pm* of $bs_{i,j}$ is *0*. Depending on each volunteer v_i, the logic of *lnr()* is:

if (leave) exit $\boldsymbol{bs_{i,j+1}}$
else {stop any actions on $\boldsymbol{bs_{i,j}}$**; start actions on** $\boldsymbol{bs_{i,j+1}}$**}**

C. SUMMARY

First in Section V-A, the thought that the more volunteer the faster computing is clarified as a misconception. For a certain big data problem, a simulation is necessary before real computing to determine an optimal overlay size, where the performance converges. Furthermore, given satisfaction (e.g., a speedup growth rate), it is confirmed that for a certain problem scale, there is an optimal overlay size; for a certain overlay size, there is an optimal problem scale. That confirmation suggests that group volunteers into multiple volunteer overlays instead of using the entire available volunteers as a single overlay would benefit more for performance. To group volunteers, the traditional single round MapReduce is remodeled as

adaptive multiple round MapReduce or multiple overlays for map step. Based on the discoveries, a fully distributed bootstrapping protocol is proposed in Section V-B. The protocol can adapt the available volunteers to the given problem scale by trapping them into multiple round MapReduce, which can catch the optimal performance points.

VI. THE CASE STUDY OF OPTIMIZATION

In this section, we use the adaptiveness that the bootstrapping protocol provides to form some optimization strategies and compare their competence.

We set a reference workload of *1,400,000* map tasks, an *MRR* of *20%* and each task of *8,000* compute-intensity as stated in Section IV. This reference workload is denoted as W. To compare the competence of the optimization strategies, we first set a non-optimized case of workload of *2W* and overlay size *80K* as shown in Figure 10. The simulation result of such an overlay is *6,502* times speedup.

A. STRATEGY 1

This strategy is two-round MapReduce. In the first round, the workload of *2W* is assigned to two overlays, of which each has a size of *40K* volunteers with a workload of W as shown in Figure 11. In the second round, the map task is *0.4W* and reduce task is *0.16W*. Thus, the overlay size has changed to *20K* (*tier-1* and *tier-2*) and *10K* (*tier-1* only) respectively. The design motivation of this strategy is based on the analytical results as stated in Section V-A and includes:

1. Using small overlays to lower the working cost
2. Using volunteers of high tier computing-capacity for the small overlays

The benefit of this strategy includes:

1. Not holding the *80K* volunteers all the time; releasing volunteers (*40K*, *20K* and *10K* gradually) during computing.
2. Maintaining a similar speedup of the full-size overlay of *80K* volunteers though multiple-round incurs a greater overall workload than *2W*.

B. STRATEGY 2

This strategy is one-round MapReduce with multiple map overlays and a single reduce overlay. The workload of *2W* is divided into *64* small units, of which each is of *W/32* workload. Each unit is assigned to one of the *64* map overlays, of which each has *1.25K* (*tier-1* to *tier-6*) volunteers. The single reduce overlay is *20K* (*tier-1* and *tier-2*) volunteers. This strategy is shown in Figure 12. The design motivation of this strategy is based on the analytical results as stated in Section 5-A and includes:

1. Using small overlays to lower the working cost
2. Using volunteers of high tier computing-capacity for the small overlays
3. Using the highest growth rate of small overlays

The benefit of this strategy includes:

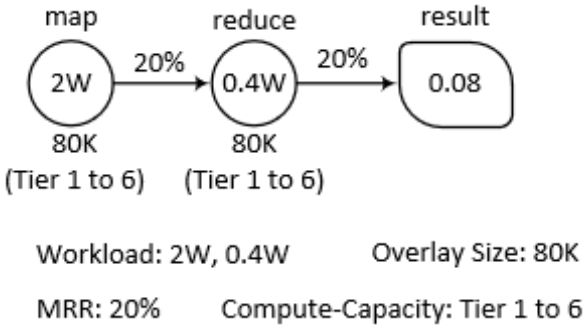

FIGURE 10. The workflow of a plain one-round MapReduce.

1. Not holding the *80K* volunteers all the time; releasing volunteers (*60K*) during computing.
2. Producing a higher speedup than the full-size overlay of *80K* volunteers.

C. STRATEGY 3

This strategy is an improvement of one-round MapReduce of *Strategy 2* in terms of multiple map overlays and a single reduce overlay. First, the volunteers are grouped by computing-capacities into *6* groups so that a volunteer belongs to a single group only. For example, a *tier-2* volunteer belongs to *Group-2* only. The overall computing-capacity of the *6* tiers is $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} = \frac{147}{60}$. So, the ratios of the computing-capacity of each tier vs the overall computing-capacity are: $\frac{60}{147}$, $\frac{30}{147}$, $\frac{20}{147}$, $\frac{15}{147}$, $\frac{12}{147}$, $\frac{10}{147}$. Based on the ratios, the entire workload *2W* assigned to each tier is: *0.82W*, *0.41W*, *0.27W*, *0.2W*, *0.16W* and *0.14W*. The *6* group volunteers are divided into *60* map overlays, of which each group is evenly divided into *10* overlays. Thus, for each map overlay of *1.34K* volunteers in *Group-1*, it is assigned *0.082W* workload. Similarly, *0.041W*, *0.027W*, *0.02W*, *0.016W* and *0.014W* are assigned to each map overlay in *Group-2* to *Group-6* respectively.

The single reduce overlay is *20K* (*tier-1* and *tier-2*). This strategy is shown in Figure 13. The design motivation of this strategy is based on the analytical results as stated in Section V-A and includes:

1. Using small overlays to lower the working cost
2. Using volunteers of high tier computing-capacity for the small overlays
3. Using the highest growth rate of small overlays
4. Workload balancing in accordance with the computing-capacity of volunteers.

The benefit of this strategy includes:

1. Not holding the *80K* volunteers all the time; releasing volunteers (*60K*) during computing.
2. Producing a higher speedup than the flat map overlays as used in *Strategy-2*.

The original plain MapReduce and the three optimization strategies are compared in Table 2 by the simulation results. The plain single-round MapReduce workflow has achieved a speedup of *6,502* times without releasing any volunteers dur-

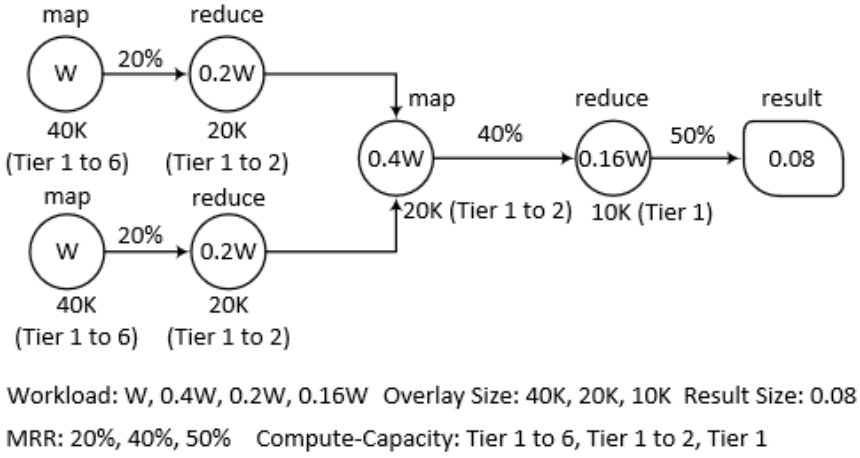

FIGURE 11. The workflow of Strategy-1 of a multi-round MapReduce.

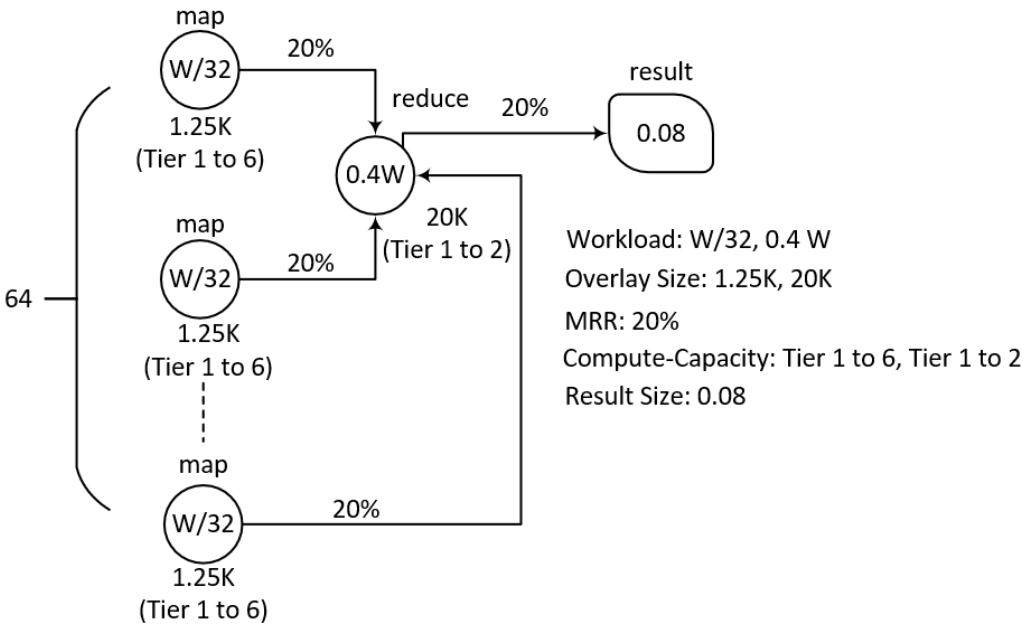

FIGURE 12. The workflow of Strategy-2 of multiple map overlays and a single reduce overlay.

TABLE 2. The performance comparison of different strategies.

Workflow	Speedup	Speedup Growth	Total Steps	Release of Volunteers				
				In Steps	Overall Progress	Number Released	Percentage Released	Tier Released
Plain	6,502	-	4,134,072	-	-	-	-	-
Strategy-1	6,500	-	4,135,384	2,615,921	63%	40K	50%	3, 4, 5, 6
				28,69,923	69%	20K	25%	2
				3,970,334	96%	10K	12.5%	1, 2
Strategy-2	8,846	36%	3,038,498	2,495,490	82%	60K	75%	3, 4, 5, 6
Strategy-3	11,152	71%	2,410,301	1,867,293	77%	60K	75%	3, 4, 5, 6

ing computing. Comparing with the plain workflow, the two-round workflow (*Strategy-1*) has achieved a similar speedup of *6,500* times to the plain workflow. The strategy has gradually released a total of *70K* volunteers at three different steps: *2,615,921* (overall progress *63%*) and *28,69,923* (overall progress *69%*) and *3,970,334* (overall progress *96%*) during computing. The *Strategy-2* of *64* map overlays and a single reduce overlay has achieved a higher speedup of *8,846* times, and a growth rate *36%* higher than the plain workflow. The strategy has released a total of *60K* volunteers at step *2,495,490* (overall progress *82%*) during computing. The *Strategy-3* of *60* map overlays and a single reduce overlay

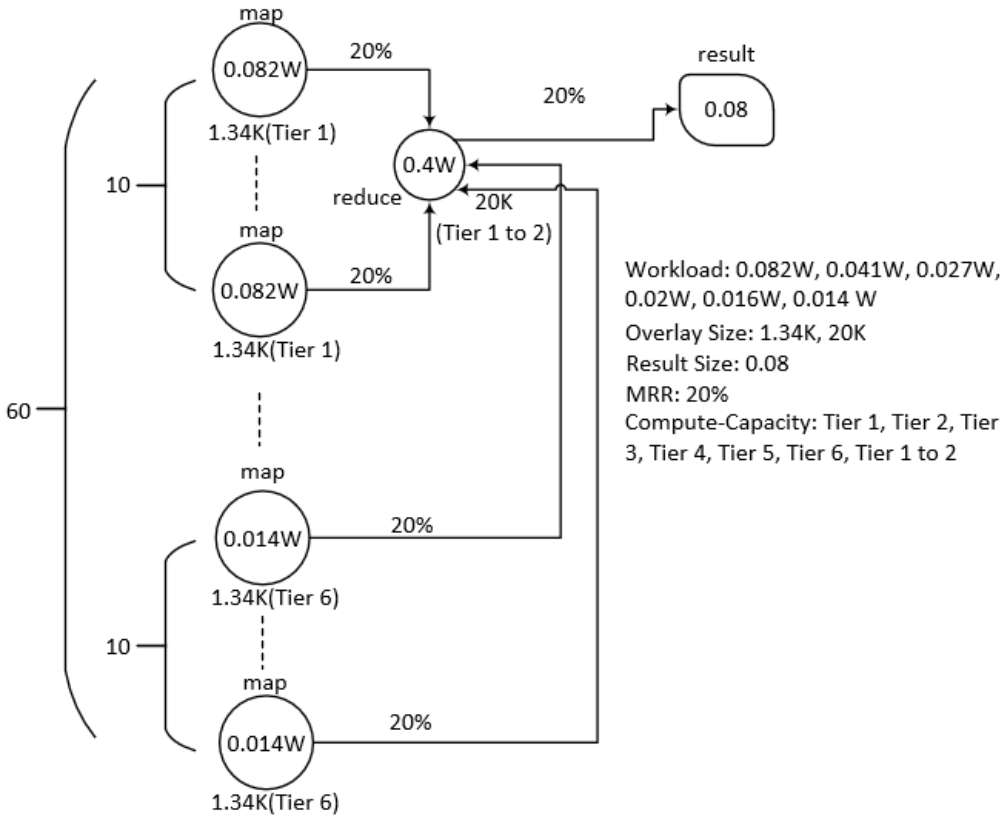

FIGURE 13. The workflow of Strategy-3 with workload balancing among multiple map overlays.

with workload balancing has achieved a higher speedup of *11,152* times, and a growth rate *71%* higher than the plain workflow.

The strategy has released a total of *60K* volunteers at step *1,867,293* (overall progress *77%*) during computing. Although *Strategy-2* and *Strategy-3* are similar in overlay structure, the embedded workload balancing makes *Strategy-3* achieve a higher speedup than *Strategy-2*.

VII. CONCLUSION

Our work in this paper places MapReduce big data processing in a fully dynamic and opportunistic environment, where volunteers are of heterogeneous computing power and subject to join, leave and crash freely. The first discovery of this work is that for a certain workload, there are an optimal overlay size to achieve a satisfactory performance. Growing over that size, the performance improvement is little, e.g., the performance growth rate is less than *3%* for each more *5K* volunteers. The second discovery of this work is that for the same growing size of overlay, e.g., *5K* more volunteers, the performance improvement is higher, e.g., *20%* for small overlays, e.g., *15K* volunteers than that e.g., *5%* of large overlays, e.g., *50K* volunteers. Based on the discoveries, the first optimization is to use multiple small overlays instead of a single large overlay to lower overlay maintaining/stabilizing cost when computing local and intermediate results (Map step). The second optimization is to use fewer selected volunteers of higher computing power to achieve higher performance when computing the global results (Reduce step) with lighter workload. At the same time, it can release certain computing resources for other computing. A bootstrapping protocol has been proposed and implemented to achieve the optimization goals by trapping volunteers to compute single round MapReduce or multiple round MapReduce through sequential overlays or parallel overlays or any combinations of them. The case study uses the bootstrapping protocol to construct three optimization strategies for computing *2,800,000* tasks using *80K* volunteers. The first strategy is a multiple round MapReduce, which produces similar speedup to the original plain MapReduce, but releases *50%*, *25%* and *12.5%* volunteers at the overall progress of *63%*, *69%* and *96%*. The second strategy is a single round MapReduce with multiple map overlays and a single reduce overlay, which produces a *36%* speedup growth higher than the original plain MapReduce, and releases *75%* volunteers at the overall progress of *82%*. The third strategy is the workload balanced version of the second strategy, which produces a *71%* speedup growth higher than the original plain MapReduce, and releases *75%* volunteers at the overall progress of *77%*. An important advantage of the simulation parameterization is the extensibility for more diverse real-world scenarios, existing or newer in the future. The nearest future work is to develop the optimization API library into

a visual and automatic tool and share the relevant test data sets. The software kit can guide volunteer computing for big data processing by producing optimization strategies when workload, dataset scale, and volunteers' features are given as parameters.

REFERENCES

[1] L. Sarmenta, "Volunteer Computing," Ph.D. thesis, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2001.
[2] (2022). *ATLAS@Home*. [Online]. Available: http://lhcathome.web.cern.ch/projects/atlas
[3] (2022). *Asteroids@Home*. [Online]. Available: http://asteroidsathome.net/
[4] (2022). *Einstein@Home*. [Online]. Available: https://einsteinathome.org/
[5] Oracle. (2016). *An Enterprise Architect's Guide to Big Data—Reference Architecture Overview, Oracle Enterprise Architecture White Paper*. [Online]. Available: http://www.oracle.com/technetwork/topics/entarch/articles/oea-big-data-guide-1522052.pdf
[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[7] 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 Trans. Netw.*, vol. 11, no. 1, pp. 17–32, Feb. 2003.
[8] S. Kaffille and K. Loesing, "Open chord (1.0.4) user's manual," Fac. Inf. Syst. Appl. Comput. Sci., Univ. Bamberg, Germany, Tech. Rep. 74, 2007. [Online]. Available: https://sourceforge.net/projects/open-chord/
[9] (2022). *Hadoop*. [Online]. Available: https://cwiki.apache.org/confluence/display/HADOOP2/ProjectDescription
[10] S. Prabhu, A. P. Rodrigues, G. Prasad, and H. R. Nagesh, "Performance enhancement of Hadoop MapReduce framework for analyzing BigData," in *Proc. IEEE Int. Conf. Electr., Comput. Commun. Technol. (ICECCT)*, Mar. 2015, pp. 1–8.
[11] M. Khan, Z. Huang, M. Li, G. A. Taylor, P. M. Ashton, and M. Khan, "Optimizing Hadoop performance for big data analytics in smart grid," *Math. Problems Eng.*, vol. 2017, pp. 1–11, Nov. 2017.
[12] X. Chen, Y. Liang, G. R. Li, C. Chen, and S. Y. Liu, "Optimizing performance of Hadoop with parameter tuning," in *Proc. ITM Web Conf.*, vol. 12, 2017, p. 03040.
[13] T. Pattanshetti and V. Attar, "Parameter tuning of big data platforms for performance optimization," *J. Inf. Optim. Sci.*, vol. 41, no. 2, pp. 403–410, Feb. 2020.
[14] (2022). *Apache Spark*. [Online]. Available: https://spark.apache.org/
[15] (2022). *Apache Storm*. [Online]. Available: https://storm.apache.org/
[16] T. T. Htay and S. Phyu, "Improving the performance of Hadoop MapReduce applications via optimization of concurrent containers per node," in *Proc. IEEE Conf. Comput. Appl. (ICCA)*, Feb. 2020, pp. 1–5.
[17] S. Lee, J.-Y. Jo, and Y. Kim, "Hadoop performance analysis model with deep data locality," *Information*, vol. 10, no. 7, p. 222, Jun. 2019.
[18] A. Eldouh, H. Elkadi, and M. Khafagy, "Reducing data shuffling and improving MapReduce performance using enhanced data locality," in *Proc. IASTEM Int. Conf.*, 2019, pp. 58–64.
[19] A. Gandomi, M. Reshadi, A. Movaghar, and A. Khademzadeh, "HybSMRP: A hybrid scheduling algorithm in Hadoop MapReduce framework," *J. Big Data*, vol. 6, no. 1, pp. 1–16, Dec. 2019.
[20] R. Alanazi, F. Alhazmi, H. Chung, and Y. Nah, "A multi-optimization technique for improvement of Hadoop performance with a dynamic job execution method based on artificial neural network," *Social Netw. Comput. Sci.*, vol. 1, no. 3, p. 184, May 2020.
[21] X. Zhang, Y. Wu, and C. Zhao, "MrHeter: Improving MapReduce performance in heterogeneous environments," *Cluster Comput.*, vol. 19, no. 4, pp. 1691–1701, Dec. 2016.
[22] O. Yildiz, S. Ibrahim, and G. Antoniu, "Enabling fast failure recovery in shared Hadoop clusters: Towards failure-aware scheduling," *Future Gener. Comput. Syst.*, vol. 74, pp. 208–219, Sep. 2017.
[23] T. Shu and C. Q. Wu, "Performance optimization of Hadoop workflows in public clouds through adaptive task partitioning," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2017, pp. 1–9.
[24] J. C. S. Dos Anjos, K. J. Matteussi, P. R. R. De Souza, G. J. A. Grabher, G. A. Borges, J. L. V. Barbosa, G. V. Gonzalez, V. R. Q. Leithardt, and C. F. R. Geyer, "Data processing model to perform big data analytics in hybrid infrastructures," *IEEE Access*, vol. 8, pp. 170281–170294, 2020, doi: 10.1109/ACCESS.2020.3023344.
[25] P. R. R. De Souza, K. J. Matteussi, A. D. S. Veith, B. F. Zanchetta, V. R. Q. Leithardt, A. L. Murciego, E. P. De Freitas, J. C. S. D. Anjos, and C. F. R. Geyer, "Boosting big data streaming applications in clouds with BurstFlow," *IEEE Access*, vol. 8, pp. 219124–219136, 2020, doi: 10.1109/ACCESS.2020.3042739.
[26] S. Gonzalo, J. M. Marquès, A. García-Villoria, J. Panadero, and L. Calvet, "CLARA: A novel clustering-based resource-allocation mechanism for exploiting low-availability complementarities of voluntarily contributed nodes," *Future Gener. Comput. Syst.*, vol. 128, pp. 248–264, Mar. 2022, doi: 10.1016/j.future.2021.10.002.
[27] W. Li and M. Tang, "The optimization strategies on clarification of the misconceptions of big data processing in dynamic and opportunistic environments," *Big Data Cognit. Comput.*, vol. 5, no. 3, p. 38, Aug. 2021, doi: 10.3390/bdcc5030038.
[28] W. Li, and W. Guo, "The experimental study of performance impairment of big data processing in dynamic and opportunistic environments," *J. Commun.*, vol. 15, no. 11, pp. 776–789, 2020, doi: 10.12720/jcm.15.11.776-789.
[29] W. Li and W. Guo, "Work stealing based volunteer computing coordination in P2P environments," *J. Commun.*, vol. 12, no. 10, pp. 557–564, 2017.
[30] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating MapReduce performance using workload suites," in *Proc. IEEE 19th Annu. Int. Symp. Model., Anal., Simul. Comput. Telecommun. Syst.*, Jul. 2011, pp. 390–399.

WEI LI received the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, China. He is currently with the School of Engineering and Technology, Central Queensland University, Australia. His research interests include dynamic software architecture, P2P volunteer computing, and big data analytics. He has been a Peer-Reviewer of several international journals, including IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, *Journal of Systems and Software* (Elsevier), and *Journal of Software Maintenance and Evolution: Research and Practice* (John Wiley & Sons). He is a program committee member of more than 30 international conferences.

MAOLIN TANG received the B.E. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, the M.E. degree in computer science from Chongqing University, Chongqing, China, and the Ph.D. degree in computer systems engineering from Edith Cowan University, Perth, Australia.

He is a Faculty Member with the School of Computer Science, Queensland University of Technology, Brisbane, Australia. His current research interests include evolutionary computation, nature-inspired computation, and their applications in optimization, planning, placement, and scheduling. He has published over 100 papers in prestigious journals and international conference proceedings, including IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, IEEE TRANSACTIONS ON SYSTEM, MAN, AND CYBERNETICS—PART B, IEEE TRANSACTIONS ON CYBERNETICS, IEEE TRANSACTIONS ON SERVICES COMPUTING, and IEEE TRANSACTIONS ON CLOUD COMPUTING.