The Experimental Study of Performance Impairment of Big Data Processing in Dynamic and Opportunistic Environments

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

Abstract —In contrast to HPC clusters, when big data is processing in a distributed, particularly dynamic and opportunistic environment, the overall performance must be impaired and even bottlenecked by the dynamics of overlay and the opportunism of computing nodes. The dynamics and opportunism are caused by churn and unreliability of a generic distributed environment, and they cannot be ignored or avoided. Understanding impact factors, their impact strength and the relevance between these impacts is the foundation of potential optimization. This paper derives the research background, methodology and results by reasoning the necessity of distributed environments for big data processing, scrutinizing the dynamics and opportunism of distributed environments, classifying impact factors, proposing evaluation metrics and carrying out a series of intensive experiments. The result analysis of this paper provides important insights to the impact strength of the factors and the relevance of impact across the factors. The production of the results aims at paving a way to future optimization or avoidance of potential bottlenecks for big data processing in distributed environments.

Index Terms—Big data, performance impairment, dynamic environment, MapReduce, scalability

I. Introduction

There are two certainties in modern data processing and analysis. First, the data itself becomes so called big data. Big data is produced by business transactions and social events and accumulated through the operating of an enterprise. The production of big data can also be extended to the areas where data is continuously collected by sensors, cameras or telescopes. Big data is either a scientific project's resources to produce the answers to scientific hypotheses [1] or the pillars of business intelligence [2] in support of smart decision and prediction of business trade. Second, although it is difficult to define how big a big datum should be, it is certainly too big to be processed by a single commodity computer in a reasonable amount of time. From the aforementioned reasons we have derived that a high-performance cluster is needed to process big data. However, at the same time we have also proposed a question: whether a scientific project or an enterprise can either have or access a dedicated computing center. The answer is negative for either of them. Scientific projects such as ATLAS@Home [3], Asteroids@home [4] and Einstein@Home [5] by using volunteer computing [6], [7] are an evidence of shifting big data processing to distributed and heterogeneous environments. In addition, not every enterprise, particularly small or medium enterprises can afford to invest and maintain a dedicated data center. Consequently, we come to another derivation: it is necessary to make use of the existing corporate computers of an enterprise or a community or volunteer computers on the Internet to construct a distributed and parallel computing environment for big data processing. Unfortunately, to make use of such a convenience, we have to tolerate the heterogeneity, dynamics, opportunism and unreliability of computing nodes. Heterogeneous computing nodes could have big difference in compute- or storage-capacity. Dynamic nodes may join in or leave from the overlay at any time. Unreliable nodes may crash on the overlay at any time. Working on dynamic and opportunistic overlay will bring many uncertainties. For example, some questions include how churn impairs the overall performance and whether the impaired performance can be tolerated; whether continuously accommodating more computing nodes benefits for the overall performance; what the key factors are to impair the overall performance significantly.

In this paper we propose eight impact factors to cover the aspects including the compute-capacity and heterogeneity of computing nodes, the communication cost of working on the overlay, the churn of computing nodes and the big data application itself. To compare the intra- and inter-factor impacts and to answer the aforementioned questions, we need a reference system:

- A clean and ideal computing environment that grows linear speedup proportionally to the increase of computing nodes. The ideal environment allows any single factor's or combination of factors' injection to test impact.
- A spectrum of representative values for each factor that is able to produce weak, medium and strong impact on the overall performance and to avoid extreme situations, either all too weak or all too strong impact.
- A couple of cross-factor reference points that is able to vary a single factor but to stabilize all other factors,

Manuscript received June 19, 2020; revised September 6, 2020.
Corresponding author email: w.li@cqu.edu.au.
doi:10.12720/jcm.15.11.776-789

exposing the factor's impact and making the evaluation results trustable.
- A number of reference units to record the impact.

This paper proposes an experimental prototype that is able to integrate the aforementioned impact factors and the reference system. The prototype platform can be configured to represent a dynamic overlay that allows opportunistic and unreliable nodes as an individual but guarantees the reliability of the overlay as a whole under the dynamics.

When we place the host platform at the bottom and the impact factors on the top, we will need an algorithm to operate big data in the middle. Big data processing is still an open issue and there is a number of processing paradigms [8]. Among them, MapReduce [9] was proposed by Google from IT industries and has been the most successful model in both industry and academia. The main reason that we prefer to use MapReduce as the big data processing paradigm is the key feature of <*key, value*> pairs form of data representation. To clarify this feature of MapReduce, in the first step *map*, a number of independent map tasks with the input data in the form of <*key, value*> pairs is executed. The original input data of a map task is a set *IM* in the form of <*key, value*> pairs. There exist $<key_1, value_1>$, $<key_2, value_2> \in IM$ and $key_1=key_2$. A map task operates *IM* by using a predefined computing logic to sort out all the <*key, value*> pairs in *IM* so that for any $<key_1, value_{11}>$, $<key_2, value_{21}> \in OM$, $key_1 \neq key_2$, where *OM* is the output data set of the map task. On the completion of map step, however, there exist $<key_1, value_{12}> \in OM_1$, $<key_2, value_{22}> \in OM_2$ and $key_1=key_2$, where OM_1 and OM_2 are *2* different output data sets of *2* different map tasks. The second step *shuffle* is responsible for breaking every result set of the map step and redistributing the <*key, value*> pairs to a number of reduce tasks so that for any $<key_1, value_{13}> \in OM_1$ and $<key_2, value_{23}> \in OM_2$ and $key_1=key_2$, $<key_1, value_{13}>$ and $<key_2, value_{23}>$ must be redistributed to the same reduce task. The shuffle step involves a lot of data exchange. The third step *reduce* is the independent computing of reduce tasks and still produces data sets in the form of <*key, value*> pairs without any intra-or inter-task pairs having the same key. This feature coheres with a successful dynamic internet overlay protocol Distributed Hash Table (DHT) in terms of <*key, value*> pair data format. For the high coherence with the data processing layer, the proposed platform is to be based on the DHT protocol Chord [10] and to be implemented on top of Open Chord APIs [11].

The goal of this paper aims at confirming how the impact factors impair the overall performance of big data processing in dynamic and opportunistic environments, sorting out the impact strength of the factors and identifying the key factor having strong impact.

The methodology of this paper consists of the following steps.
- Identify and classify impact factors that exist in dynamic and opportunistic environments and impair the performance of big data processing.
- Propose a reference system to inject impact factors, evaluate impact, and measure the impact strength on the running big data applications.
- Cohere MapReduce big data paradigm with DHT to unite the computing paradigm and supporting platform.
- Generalise domain applications of big data and dynamics of overlay to construct an application-independent MapReduce case and a random churn pattern.
- Conduct intensive experiments to produce trustable results and carry out impact strength and cross-factor relevance analysis.

The contributions of this paper include:
- A united framework of impact factors, MapReduce paradigm and supporting platform.
- The realisation of the cross-factor comparability.
- The potential use of the result analysis for balancing workload and overlay size, computing nodes selection and avoidance of extremely impaired performance.

The organization of this paper is as follows: related work is reviewed in Section II. The impact factors are proposed and classified, and the churn patterns are detailed in Section III. Section IV proposes an application-independent study case of MapReduce paradigm. In Section V, a reference system, including computing environment, spectrums of reference values, cross-factor reference points and reference units, is detailed. In Section VI, the workflow of MapReduce on dynamic and opportunistic overlays is explained. Section VII presents experiments, results and analysis. Section VIII concludes the study by confirming the impact strength and the relevance of factors and proposing optimization potentials.

II. Related Work

The existing work tried to improve MapReduce performance by three ways or their combinations: framework modification; communication-aware or CPU capacity-aware load balancing; optimization of Hadoop parameter configuration.

Singh *et al.* [12] stated that the computing environments could not be always homogenous. They demonstrated that heterogeneity was a key impact factor and workload balancing was the critical way to make full use of resources and maximize the efficiency in heterogeneous environments. They classified impact factors as algorithm-specific factors (filtered transactions and data structure) and cluster-specific factors (speculative execution, performance of physical nodes versus virtual nodes, distribution of data blocks, and parallelism control with input split size). They criticized that Hadoop [13] load balancing was not optimized for some situations as data locality might be able to overwhelm load balancing and slow down the overall performance. Some optimization strategies they used included: the filtered transactions as a combiner to

maximize the intra-node processing before shuffle; disabling speculative execution to increase efficiency by sacrificing redundancy or fault tolerance; removing slow nodes to decrease the number of nodes but to improve the overall performance. Integration of MapReduce-based Apriori (a very efficient data analyzer) and Hadoop to maximize efficiency was their optimization goal but was still an open issue with their work.

Cheng *et al*. [14] proposed *adaptive task tuning* to target load balancing. The core of the method was a genetic algorithm, which was able to monitor task execution in heterogeneous environments. The method profiled computing nodes, clustered as worker nodes and assigned tasks based on the profiles. The method was automated for collecting performance of workers and adjusting task balancing in the course of computing. They finely tuned the *109* parameters of Hadoop and MapReduce at the cluster level, job level and task level. The method was evaluated on physical clusters (*3* nodes) and virtual clusters (*24* virtual machines) on cloud by using *3* CPU- or I/O-intensive applications: *wordcount* [15], *grep* [16] and *Terasort* [17] of a dataset of *900GB* with a reducer out of mapper ratio of *0.4%* to *3.5%*. The results were compared with default Hadoop and other *2* optimizers *Starfish* and *Rules-of-Thumb*. The results demonstrated a *31%*, *20%* and *14%* improvement on physical clusters and *23%*, *11%* and *16%* improvement on virtual clusters. In general, the genetic algorithm of [14] needs multiple rounds for the optimization, so the assumption of the method is that MapReduce, particularly the map step, needs multiple rounds. However, most of the real-world applications are suitable for one round MapReduce because of the application properties or the purpose of minimizing data exchange.

Spivak and Nasonov [18] tried to improve Hadoop MapReduce performance by using data preloading and data placement. When CPU was in intensive computing, but IO was light, preloading data from network parallelized the preparation of next tasks and the current computing. By data placement, which was based on the assumption that faster nodes would be able to process more data, data were distributed to computing nodes in ratio to their CPU-capacity. A *5*-nodes heterogeneous cluster was used as a testbed to process *10GB*, *15GB* and *25GB* datasets. By using *readData* MapReduce task, the evaluation showed that the preloading strategy outperformed standard Hadoop MapReduce. By using three applications: *grep*, *wordcount* and *readData*, the data placement strategy was shown outperforming the standard Hadoop MapReduce's locality strategy. The method was applicable to heterogeneous network but inapplicable to dynamic networks, where nodes might join in, leave from or crash on the network at any time, making preloading or placement of data invalid.

There are some overlaps between the research goals of Chen *et al*. [19] and ours of this paper. Chen et al. compared two production traces from Facebook and Yahoo and clarified that both traces contained jobs performing data aggregation (*input>output*), data expansion (*input<output*), data transformation (*input≈output*) and data summary (*input>>output*). They stated that the existing benchmarks captured narrow slivers of a rich space of workload. They described the importance of constructing a workload suite. They proposed a representative workload suite to allow selecting and combing various characteristics such as job submission time and arrival pattern, data size & ratio and presented a framework to generate and execute the workloads. They demonstrated that the running workloads helped cluster operators understand the system configuration and identify systems bottlenecks.

Han and Lu [20] aimed at building a big data benchmark suite, which was truly representative and comprehensive instead of application- or system-specific, to compare performance, energy efficiency and cost effectiveness. They composed a data generator to produce synthetic datasets while preserving the *4V* (volume, variety, velocity and veracity) properties of big data. They composed a test generator to produce benchmarking tests consisting of operations and workload patterns. The strength of [20] was the classification of *10* commonly used big data benchmarks in terms of *4Vs* and the classification of benchmarking techniques of the *10* benchmarks in terms of workload types (online, offline or real time) and software stacks (Hadoop, NoSQL, DBMS, real time or offline analytics systems). However, any quantitative study cases or evaluations of the proposed benchmark suit were not presented in [20].

Lee *et al*. [21] stated that *79%* of a job was I/O intensive and reducing HDFS I/O within MapReduce job was the most effective approach to enhancing performance. They stated that *26%* to *70%* of running time of *188,000* MapReduce jobs of Facebook was the shuffle phase. Combiners, which performed partial merging of intermediate data before sending to reducers, decreased the amount of intermediate results saving substantial network cost. They utilized an in-memory NoSQL system: Redis as a cache layer for both input data and intermediate results to improve I/O performance of Hadoop DFS [13]. They utilized in-node combiners to take *2* benefits over in-mapper combiners: minimizing the total number of emitted results and executing combing functions in a separate thread. The test environment of [21] included *4* physical nodes, a *12GB* (*20* files) dataset of random Twitter messages, *2* applications: *word count* and *computing relational status between Twitter users*. By using in-memory cache, they demonstrated that the completion time of map was decreased *14%*; the whole job completion time was decreased *23%*. On *word count* application, the use of in-node combiners decreased the input size of reducers up to *50%* and job completion time was decreased up to *30%*. On computing relationship of Twitter users, combiners decreased *3%* map output and *6%* completion time because the number of intermediate pairs with the same key generated within a map task was decreased. However, their work was limited on only one impact factor for I/O performance when there were

multiple factors interacting with each other in distributed environments.

Ardagna *et al*. [22] was to target capacity planning at the design-time in capturing dynamic resource assignment in the big data scheduler of Hadoop and YARN [13]. They stated that when appropriate size of cluster was important to predict the budget to run Hadoop for an application in public clouds, the execution time of a MapReduce job was unknown. They proposed Petri-net like models QN (Queuing Network) and SWN (Stochastic Well Formed Net) to provide design-time performance analysis, aiming at estimating MapReduce job execution time in Hadoop clusters managed by YARN. The test environment of [22] included Amazon EC2 (*120* CPUs supporting *240* containers) and an Italian supercomputing center CINECA (*120* cores with a container). The datasets were *250GB* and *1TB* and the number of map and reduce tasks were *4,1560* and *1,1009*. The experiments of [22] demonstrated that the accuracy between the simulation and actual measurement was high in terms of the relative error of QN being *32%* and SWN being *14%*.

Dede *et al*. [23] studied extensively on the impact caused by the heterogeneity, unreliability or unstable computing nodes. They compared their own implementation: LEMO-MR with the other two existing implementations of MapReduce: Hadoop and Twister. On the three testing environments: a heterogeneous cluster, a homogeneous but load-imbalanced cluster and a cluster with unreliable nodes, they demonstrated the difference of these implementations in processing data-intensive, CPU-intensive and memory-intensive applications. Among others, they stated that cluster heterogeneity must be strongly considered. That conclusion matches our quantitative results in this paper that heterogeneity strongly impairs performance of big data processing running on commodity computers.

Both our work of this paper and Zhang *et al*. [24] regard the poor performance of big data processing as the equity task allocation, without considering the heterogeneity of computing nodes. Zhang *et al*. [24] stated that the straggler nodes slowed down the overall progress of map or reduce steps, particularly for the reduce step, which could not start until the slowest node completed the map task and exchanged the data to the reduce tasks. They optimized the situation by allowing faster nodes to steal some work from the stragglers. In addition to the heterogeneity, our work of this paper has identified and scrutinized other key factors in the unreliable and opportunistic environments, which are important for potential optimization.

III. Impact Factors

The study background of this paper is derived by the logic that the available application data is too big to be processed by a single commodity computer in a reasonable amount of response time. The big data must be processed in parallel by a distributed computing environment. The impact factors are derived by the logic that a distributed computing environment is a dynamic environment. The performance of such a dynamic computing environment can be impaired by factors coming from computing nodes, communication networking and/or the dynamics of overlay. In addition, MapReduce applications are different. They may perform data aggregation, data expansion, data transformation or data summary [19], having potential impact on the overall performance as well. To scrutinize performance issues, the impact factors can be classified into *4* categories.

Computing Nodes: the *heterogeneity (H)* of computing nodes refers to the difference of compute-capacity between computing nodes. When all the computing nodes are treated equally by task assignment, the slow nodes will impair the overall performance because they hold the assigned tasks to make the fast nodes idle.

Networking: cooperation between distributed computing nodes mainly incurs two types of communication cost. First, any communication between a pair of computing nodes needs *Round Trip Time (RTT)* for a node to lookup another node and establish connection for communication and close the connection on the completion of communication. Second, downloading a task or uploading a result set is the other cost of communication. Thus, *Download/Upload Speed (DUS)* is recognized as another impact factor.

Applications: the structure of MapReduce applications brings two factors to apply impact on the overall performance. *Map/Reduce Ratio (MRR)* refers to how much the overall work has been shrunk by the map step. For example, if an *MRR* is *20%*, the number of reduce tasks will be shrunk as *20%* of the number of map tasks. *MRR* will impair the overall performance of MapReduce in terms of not only the overall computing load but also the communication cost. For an *MMR* of *20%*, the cost of uploading map results and downloading reduce tasks will be shrunk to *20%* and the cost of uploading results of reduce tasks will be shrunk to *4% (20%×20%)*. Obviously if *MMR* changes to *40%*, the above cost will be doubled. *Redistribution Factor (RF)* refers to how many reduce tasks that a map result set will be redistributed into by the shuffle step. For example, if *RF* is *100*, it means the result set of a map task will be redistributed into *100* reduce tasks. When the cost of uploading a result set or downloading a reduce task is dependent on *MRR*, *RF* impairs the overall performance through *RTT*. For the same *DUS*, a bigger *RF* incurs more lookups, connections and disconnections, of which each needs a full *RTT*.

Dynamics and Opportunism: churn refers to the join, leave or crash of computing nodes in the course of computing. Churn represents the opportunistic features in volunteer computing [7], where a computing node may join or leave a computing without any responsibility or dedication. Churn is the most complex factor to impair

the overall performance. The churn factor is divided into *3* refined factors for a scrutiny.

- *Churn Rate (CR)* refers to how many nodes behave dynamically and unreliably. For example, a *30%* churn rate refers to that *30%* of the total nodes will leave or crash randomly during the course of computing.
- *Start Position (SP)* refers to how long after it joins the computing, a churn node would start to commit churn.
- *Occurrence Interval (OI)* refers to the time period from the start position in which the churn node could commit churn randomly.

The first factor *CR* reflects a churn feature for all computing nodes; the second and third factor *SP* and *OI* are the position and size of a churn window that is special for each churn node. The three factors together determine the dynamic, opportunistic and random features of a computing node.

To study the impact of churn, we construct a churn pattern to behave the proposed dynamic and opportunistic features. The churn pattern will be injected into the map step once and the reduce step once. The injections make the whole course of computing be affected by the dynamics and opportunism of the overlay. The construction of churn pattern is as follows and the constructed churn pattern is illustrated in Fig. 1.

- Computing nodes join the overlay sequentially but randomly.
- When the joined computing nodes reache a certain number, churn starts happening.
- A node commits churn randomly in its own churn window that is determined by its own *SP* and *OI*.
- A churn node can leave from or crash on the overlay and the chance of leave or crash is the same for each churn node.
- When *CR* is reached, the churn stops and the left overlay is stable to the end of computing.

IV. A Generic Study Case of MapReduce

We propose a generic study case of MapReduce application considering the following conditions. The application scenario complies with MapReduce paradigm but is independent of any real-world applications. This generality allows us to test with any computing and communication intensity that are not restricted by any specific applications. Based on this rule, the consideration of determining the number of map tasks (*NMT*) and the computing load of each task (*CLET*) is that the scale of overall computing load must be big enough as a big data problem. In this study case, *NMT* is randomly chosen as *1,400,000 (1.4M)* and *CLET* is randomly chosen as *8,000 (8K)* so that the overall computing load for map tasks is *11,200,000,000 (11.2G)*. The unit of this computing load can be seconds, minutes or hours etc. The size of each map task or reduce task *(SEMRT)* is chosen as *64MB*, which is a common task size of real world applications such as ATLAS@Home [3], Asteroids@home [4] or Einstein@Home [5] and it is one of data block size of Hadoop Distributed File System (HDFS) [13]. The size of the result set of a map task or a reduce task *(SRMRT)* is determined by Map/Reduce Ratio (*MRR*). Generally, in real world applications, *MRR* is much smaller than *1*, e.g. *20%* and the Redistribution Factor (*RF*) is not high, e.g. *100*. As a result of certain *MRR* and *RF*, the communication cost and the computing load of reduce tasks can be determined. The calculation of the communication cost and computing load of the aforementioned generic study case is as follows as shown in Table I. The redistribution factor *RF* is applied to map step only because there are no any overlapped keys between different reduce result sets. *RF* is not to influence reduce step and each reduce task just uploads a reduce set as a whole onto the overlay.

V. Reference System

To study the impact caused by the factors, we first need an ideal computing environment as a reference in order to check how far a distributed and dynamic computing environment goes away from it in terms of performance. Second, as we deal with multiple factors and each factor varies values, we need a reference point to settle all the factors expect the factor we want to study in a round. Finally, we need a number of reference units to record the impact strength.

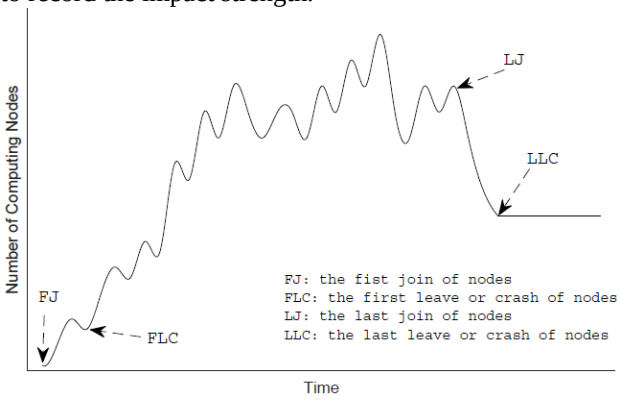

Fig. 1. The churn patterns

Table I: The Setting of a Generic Study Case of MapReduce

Scenario Variables	Values
The number of map tasks (NMT)	1400000 (1.4M)
The lookup times of map tasks	1400000 (1.4M)
The times of downloading map tasks	1400000 (1.4M)
The size of a map or reduce task or a map or reduce result set	64MB
The Redistribution Factor (RF)	100
The computing load of each map or reduce task	8000 (8K)
The computing load of map tasks	1400000 ×8000=11200000000 (11.2G)

The times of uploading map result sets	1400000×100=140000000 (140M)
The upload of results of map tasks	1400000×20%×64MB=17920000MB≈18TB
The Map/Reduce Ratio (MRR)	20%
The number of reduce tasks (NRT)	1400000×20%=280000 (0.28M)
The lookup times of reduce tasks	280000 (0.28M)
The times of downloading reduce tasks	280000 (0.28M)
The computing load of reduce tasks	280000×8000=2240000000 (2.24G)
The upload of results of reduce tasks	280000x20%×64MB=358400000MB≈3.58TB
The upload times of reduce result sets	280000×20%=56000
The overall data size to be processed	1400000×64+280000×64=107520000MB≈108TB

A. Ideal Computing Environment

An ideal computing environment is proposed to provide the fastest speedup through distributed computing nodes for the proposed generic MapReduce application in Section IV. The ideal computing environment satisfies the following conditions.

- All computing nodes are reliable in the course of computing. Thus, the tasks can be pre-assigned to nodes before computing and be located during the course of computing.
- All computing nodes are homogeneous in compute-capacity, providing the maximum computing efficiency without incurring additional cost for optimization scheduling.
- All computing nodes are connected by high speed networks. Thus, for a map or reduce task or a result set of *64MB* or *128MB*, the download or upload cost is trivial.

Based on these conditions, the ideal computing environment is free from the impact of heterogeneity, communication or churn. The ideal computing environment is used to produce the highest performance so that any other environments can be compared, showing the distances they go away from it in terms of overall performance.

B. Reference Values

To compare the impact of different factors, each factor needs a set of representative values. The representative values of a factor should be able to differentiate degrees of impact caused by the factor. The setting of representative values follows the rule that the spectrum of values of any factor should cover a variety of conditions but avoid the extreme conditions such as all light impact or all strong impact. In addition, to compare the impact between different factors, the representative values of a factor cannot go extreme to override the impact of another factor. The change of values, reflecting the change of conditions, should go evenly in the spectrum. Based on the considerations, a spectrum of *5* values is set for each impact factor for the study of this paper. The representative values of each factor are clarified as follows.

The heterogeneity (*H*) of compute-capacity of computing nodes is generalized as tiers. The base tier is *Tier-1* being the fastest compute-capacity. Thus, a *Tier-2* computing node is *2* times slower than a *Tier-1* node. The spectrum of the *5* values of *H* is set as *2-tiers*, *4-tiers*, *6-tiers*, *8-tiers* and *10-tiers* to represent a wide range of heterogeneous computing nodes. The consideration of *8*-times difference is reasonably sound to represent the heterogeneity of real-world commodity computers in terms of compute-capacity.

The download/upload cost depends on network speed and the data size. The network speed depends on the internet service standards of a country. For example, the *5* download/upload speed (in *Mbps*) tiers: *12/1*, *25/5*, *25/10*, *50/20* and *100/40* are provided by Australia National Broadband Network (NBN). In Monsalve et al. [25] study of data-intensive project ATLAS@Home [3] or Einstein@Home [5], *40MB* or *100MB* were treated as a large file for input or output of a computing task. The data block size of big data processing model Hadoop [13] is either *64MB* or *128MB*. In the experimental study of this paper, when the size of a map task is chosen as *64MB*, the download/upload speed (*DUS*) could be set as *5* tiers: *5/13, 10/26, 20/51, 20/102 and 43/512* corresponding to the *5* tiers of the NBN speed. The round-trip time (*RTT*) can be set as *4*, *6*, *8*, *10* and *12* to represent the connection and disconnection time before and after the data exchange between two computing nodes. This *RTT* range is to ensure that any slow network is able to finish a handshake preparing for the data exchange. For the ideal computing environment, its network speed is the fastest of *1000Mbps (100MB/s)* of a modern LAN. Thus, for a *64MB* task or a result set, the download or upload time is less than *1* second. Because in the setting of experiments, download or upload time is a non-zero integer, it is set as *1*. For transferring a data or result set, the TCP connection and disconnection together is set as *1* for the same reason.

The churn rate (*CR*) can be set as *10%*, *20%*, *30%*, *40%* and *50%* to represent how many computing nodes behave dynamic and opportunistic features on the overlay. The maximum of *50%* is large enough to represent the opportunism of volunteer nodes in real world situation. The start position (*SP*) can be set as *450K*, *350K*, *250K*, *150K* and *50K* to represent how long a node stays on the overlay before start to commit churn. The occurrence interval (*OI*) can be set as *50*, *40*, *30*, *20* and *10* to represent how wide of the churn window that a node can randomly commit churn. The combined *SP* and *OI* are able to provide a small churn window (*10*), a medium churn window (*30*) and a big churn window (*50*), of which each can be placed in the front (*50K*), middle (*250K*) or on the rear (*450K*) of the course of computing for each computing node.

The Map/Reduce ratio (*MRR*) can be set as *20%, 40%, 60%, 80%* and *100%* to represent how much the number of tasks is shrunk by map step. The best situation is that the number of reduce tasks is shrunk to *20%* of the number of map tasks, representing data aggregation; the worst situation is that the number of reduce tasks is not shrunk at all and is the same as the number of map tasks, representing data transformation. The redistribution factor *RF* can be set as *100, 150, 200, 250* and *300* to represent data exchange intensity. The best situation of *100* represent most of the cases of real-world MapReduce applications, while the worst situation of *300* can still accommodate specific MapReduce cases.

C. Cross-Factor Reference Points

To evaluate the impact of a factor, we need to vary the factor's values but keep the values of all other factors as constant (reference). A Cross-Factor Reference Point (*CFRP*) is proposed in the form of *(H, DUS, RTT, CR, SP, OI, MRR, RF)* to include a complete cross-factor value. To make the study result trustable, we choose *2* cross-factor reference points $CFRP_1$ and $CFRP_2$ as defined as follows and as shown in Table II, where *FV* represents factor value. $CFRP_1$ is at the beginning and $CFRP_2$ is in the middle of the spectrum of cross-factor values.

$CFRP_1$*: (2, 5/13, 4, 10%, 450K, 50, 20%, 100)*

$CFRP_2$*: (6, 20/51, 8, 30%, 250K, 30, 60%, 200)*

The use of a *CFRP* is to evaluate the impact of a factor, vary the factor's values along the row of the factor but keep all other factors of the *CFRP* as constant. For example, if we evaluate the impact of *H* by $CFRP_1$, we vary *H* from *2* to *10* for $CFRP_1$ as:

((2, 4, 6, 8, 10), 5/13, 4, 10%, 450K, 50, 20%, 100)

As another example, to evaluate the impact of *CR* by $CFRP_2$, we vary *CR* from *10%* to *50%* for $CFRP_2$ as:

(6, 20/51, 8, (10%, 20%, 30%, 40%, 50%), 250K, 30, 60%, 200)

We predict that when visualizing the evaluation results of all factors in a single diagram, all the results are expected to diverge from $CFRP_1$ for $CFRP_1$ based evaluation or go through $CFRP_2$ for $CFRP_2$ based evaluation.

D. Reference Unit

We propose to use *Speedup* as the reference unit to measure the overall performance of a distributed computing environment or overlay.

$$Speedup = \frac{The\ total\ time\ to\ complete\ the\ entire\ problem\ by\ a\ single\ tier{-}1\ computer}{The\ total\ time\ to\ complete\ the\ entire\ problem\ by\ a\ computing\ environment}$$

Along with speedup as a reference unit, we propose other two reference units to compare the impact of different factors or the different degrees of impact by the factor values of a single factor. We propose to use the *Speedup Distance* away from the ideal environment to measure the impact of a factor and to compare impact strength between different factors.

$$Speedup\ Distance = the\ speedup\ of\ the\ deal\ computing\ environment - the\ speedup\ of\ a\ computing\ environment$$

We propose to use the *Speedup Growth Rate* to compare the change rate of speedup upon the value changes of a factor.

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

VI. The Experimental Model

An experimental model is needed for the impact evaluation of the factors. The experimental model is proposed to be able to create the dynamics and opportunism for distributed computing environments as stated in Section III. The MapReduce study case as proposed in Section IV can be executed on the model. The model can enable the reference systems as proposed in Section V fully record the factor impact by using the reference units. The experimental model is proposed to be built on the following conditions.

- The base model is the ideal computing environment. An impact factor can be injected individually or any combination of impact factors can be injected as a group.
- When churn is injected, a MapReduce application on the dynamic overlay will incur lookup cost of $O(_{log}n)$ and stabilisation cost of $O(_{log}{}^2n)$, where *n* is the number of current computing nodes on the overlay.

Based on the above conditions, the workflow of a MapReduce application on the ideal environment or on a dynamic environment is shown in Fig. 2 and clarified as follows:

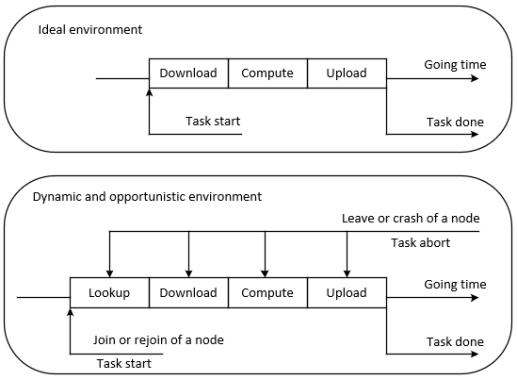

Fig. 2. The MapReduce workflow of the ideal environment and dynamic environments

In the ideal environment, each computing node performs *3* steps for a map task or a reduce task. A task is pre-located on the network. A task is downloaded from the network and computed by a computing node and then the result set of the task is uploaded. Each computing node behaves reliably during the *3* steps and a task can always be completed in one assignment. Each computing node repeats these *3* steps until all the tasks are completed, i.e. the whole MapReduce application is completed.

TABLE II: THE FACTOR VALUES AND CROSS-FACTOR REFERENCE POINT

Impact Factor		Factor Values				
		FV_1 ($CFRP_1$)	FV_2	FV_3 ($CFRP_2$)	FV_4	FV_5
Heterogeneity (H)		2-tiers	4-tiers	6-tiers	8-tiers	10-tiers
Communication Cost	Download/Upload Speed (DUS) for 64MB	5/13	10/26	20/51	20/102	43/512
	Round Trip Time (RTT)	4	6	8	10	12
Churn	Churn Rate (CR)	10%	20%	30%	40%	50%
	Start Position (SP)	450K	350K	250K	150K	50K
	Occurrence Interval (OI)	50	40	30	20	10
Application	Map/Reduce Ratio (MRR)	20%	40%	60%	80%	100%
	Redistribution Factor (RF)	100	150	200	250	300

In the dynamic and opportunistic environment, a task cannot be pre-located because of churn. The only guarantee is that the task is always on the overlay. A computing node on such an overlay must look up a task before downloading it. The lookup starts when a computing node joins the overlay or when a computing node has completed the current task and asks for another task. This turnover procedure is termed as *rejoin* of a computing node. A computing node can leave from or crash on any of the *4* steps: looking up a task, downloading the task, computing the task or uploading the result set of the task. The difference between leave and crash is that the uncompleted task of a leaving node is checkpointed and will be picked up by another computing node, but the uncompleted task of a crashed node will be totally restarted from the beginning by another computing node.

The model has been designed on the theoretical model of Chord [10]. The model is able to simulate the opportunistic features of computing nodes on a dynamic overlay. The model has been implemented on the open source API of Open Chord [11]. The model offers the ability to test the impact of an individual factor or any combination of factors.

The MapReduce model has been implemented as an application-neutral structure. A map or reduce task is self-contained object consisting of code and data container. The data container of a map task is prefilled before the start of computing, while the data container of a reduce task is filled by the results of map tasks by the shuffle step of MapReduce in the course of computing. A map task is ready to start at the beginning of the course of computing, while a reduce task is ready to start on the completion of all map tasks. The whole computing is finished when all reduce tasks are completed.

VII. THE EVALUATION RESULTS AND ANALYSIS

By using the cross-factor reference points $CFRP_1$ and $CFRP_2$, the basic evaluation has been conducted for *80* round tests to expose the impact of every factor value for each impact factor. The analysis of the fundamental results is presented in Section VII-A and VII-B. Based on the analysis, extended evaluations have been conducted to further scrutinize some impact factors in Section VII-C to VII-E.

A. The Impact on Map Step

When the impact measured in *speedup distance* of all *8* factors are compared in a single line graph as shown in Fig. 3 for $CFRP_1$ and in Fig. 4 for $CFRP_2$, it is demonstrated that the impact of *4* factors (defined as *Group A* consisting of *H*, *RF*, *RTT* and *CR*) varies much severe than the other *5* factors (defined as *Group B*, consisting of *DUS*, *SP*, *OI* and *MRR*). The results by $CFRP_1$ and $CFRP_2$ are consistent because *Group A* or *Group B* classified by both $CFRP_1$ and $CFRP_2$ includes the same impact factors. Another observation is that *Group A* factors impair the overall performance propositionally to their factor values, while *Group B* factors impair the overall performance to similar extents by both small factor values and big factor values. The other observation is that within *Group A* factors, the impact strength goes up by *CR*, *RTT*, *RF* and *H* and it is consistent for both $CFRP_1$ and $CFRP_2$. To further scrutinize *Group B* factors, they are separated and scaled into another line graph as shown in Fig. 5 for $CFRP_1$ and in Fig. 6 for $CFRP_2$.

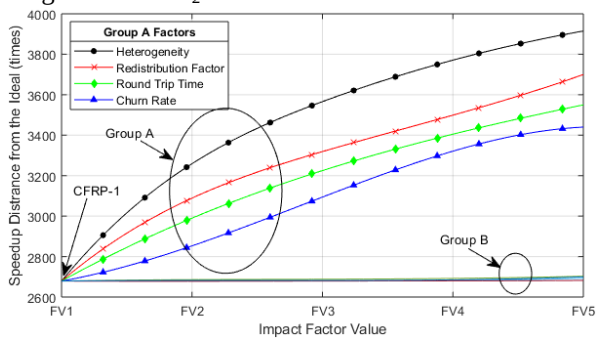

Fig. 3. Map step: The impact in speedup distance classified by CFRP1

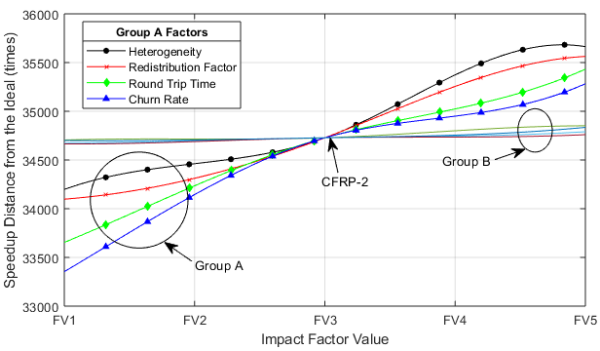

Fig. 4. Map step: The impact in speedup distance classified by CFRP2

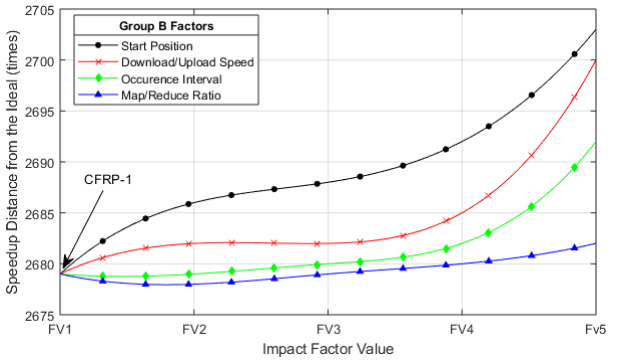

Fig. 5. Map step: The impact in speedup distance of Group B factors classified by CFRP1

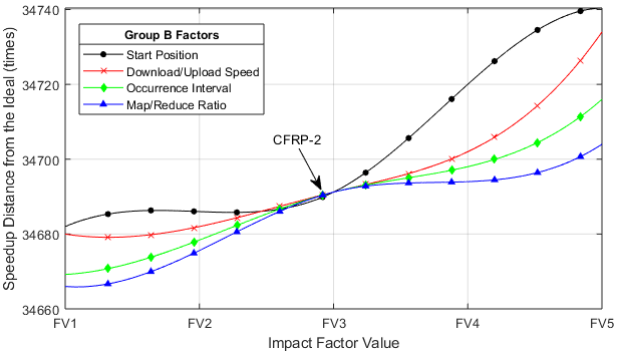

Fig. 6. Map step: The impact in speedup distance of Group B factors classified by CFRP2

The impact measured in speedup distance of each factor in *Group B* is constantly high for both small and big values of that factor. The impact strength of *Group B* factors is closer in terms of at most *21* times in difference classified by $CFRP_1$ and at most *91* times in difference classified by $CFRP_2$ between these factors. For the *4* factors in *Group B*, the impact strength goes up by *MRR*, *OI*, *DUS* and *SP* and is consistent for both $CFRP_1$ and $CFRP_2$.

B. The Impact on Reduce Step

The redistribution factor *RF* is always *1* for the reduce step because there is no any shuffle requirement for reduce results by MapReduce paradigm. Thus, the impact measured in *speedup distance* of all other *7* factors are compared in a single line graph as shown in Fig. 7 for $CFRP_1$ and in Fig. 8 for $CFRP_2$. It is demonstrated that the impact of *4* factors (defined as *Group A* consisting of *H*, *RTT*, *CR* and *MRR*) varies much severe than the other *3* factors (defined as *Group B* consisting of *SP*, *DUS* and *OI*). The results by $CFRP_1$ and $CFRP_2$ are consistent because *Group A* or *Group B* classified by both $CFRP_1$ and $CFRP_2$ includes the same impact factors. Another observation is that *Group A* factors impairs the overall performance propositionally to their factor values, while *Group B* factors impair the overall performance to similar extents by both small factor values and big factor values. The other observation is that within *Group A* factors, the impact strength goes up by *CR*, *RTT* and *H* and it is consistent for both $CFRP_1$ and $CFRP_2$. However, it is different from map step in Fig. 3 and Fig. 4, *MRR* is an exception of *Group A* factors for reduce step; its impact strength goes down with the increase of factor values. To further scrutinize this exception, an extended study has been conducted in Section VII-C. To further scrutinize *Group B* factors, they are separated and scaled into another line graph as shown in Fig. 9 for $CFRP_1$ and in Fig. 10 for $CFRP_2$.

The impact measured in speedup distance of each factor in *Group B* is constantly high for both small and big values of that factor. The impact of *Group B* factors is closer in terms of at most *137* times in difference classified by $CFRP_1$ and at most *195* times in difference classified by $CFR.P_2$ between these factors. For the *3* factor values, the impact strength goes up by *OI*, *DUS* and *SP* and it is consistent for both $CFRP_1$ and $CFRP_2$.

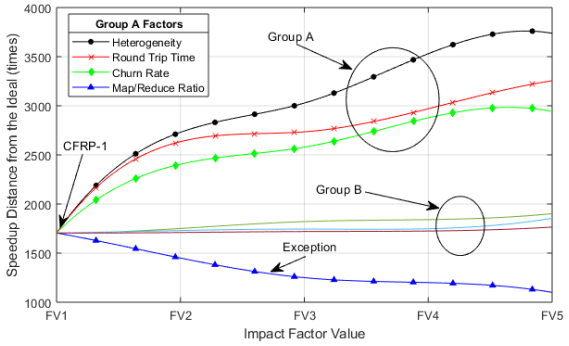

Fig. 7. Reduce step: The impact in speedup distance classified by CFRP1

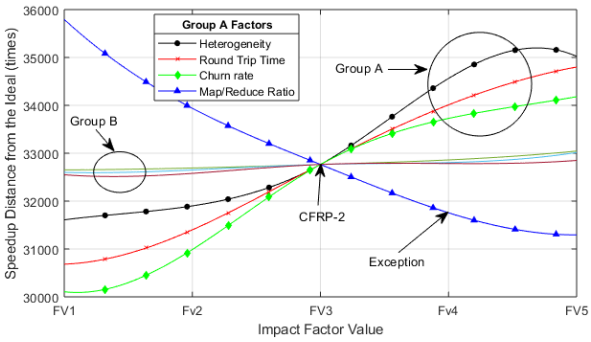

Fig. 8. Reduce step: The impact in speedup distance classified by CFRP2

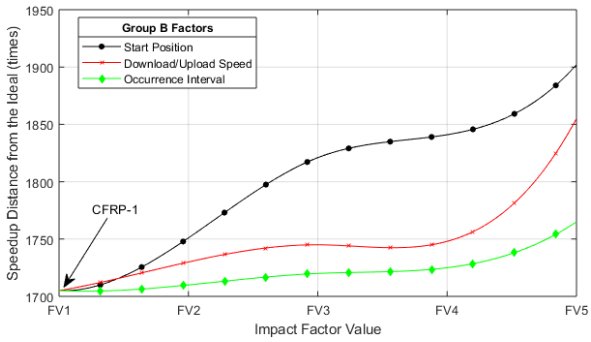

Fig. 9. Reduce step: the impact in speedup distance of Group B factors classified by CFRP1

C. The Exception Caused by MRR

The impact and their strength of *Group A* and *Group B* factors are consistent for both map and reduce steps

except the inconsistence caused by *MRR* between map and reduce steps. In map step, *MRR* as a *Group B* factor has continuous impact but slightly increased the impact strength with the increase of factor values (Fig. 5 and Fig. 6). However, it is observed for reduce step that the impact strength has been decreased with the increase of *MRR* from *20%* going up to *100%* (Fig. 7 and Fig. 8). In fact, when *MRR* increases, the number of lookups, the number of downloads and uploads, and the overall computing load are all increased for reduce step. Based on the fact, it seems there is no reason to see the decrease of impact strength with the increase of *MRR*. To further check this issue, we have conducted an enhanced experiment: *single factor response to MRR*, which is based on the *ideal conditions* (Section V-A) to inject just one impact factor each time to check its responses to *MRR* changes. That is, individual test is conducted for *Ideal+H(6 tiers)*, *Ideal+RTT(8)*, *Ideal+DUS(20/51)* and *Ideal+Churn(30% of CR, 250K of SP, 30 of OI)*. The *3* churn factors are treated and injected as a single factor because they cannot exist independently as explained in Section III. The selection of the factor values is based on $CFRP_2$ and the measurement of these tests is *speedup* and *speedup growth rate* against *MRR*.

When all the individual injections are compared in a single line graph as shown in Fig. 11, the factors are classified by their impact into *2* groups: *Group A* and *Group B*. The *Group B* factors behave no responding to *MRR* or responding to *MRR* negatively and slightly with the maximum difference of *342* times as shown in Fig. 12. *Group A* consists of *Heterogeneity* and *Churn*, both of which respond positively to *MRR* and that response is what we are aiming to find. To explain the positive responding to *MRR* by heterogeneity, we confirm that slow computing nodes impair the overall performance more effectively for smaller *MRR*. For example, assume that by a smaller *MRR*, there are *2* reduce tasks. Assume that there are *2* computing nodes: *Node A* and *Node B*, and *Node B* is *4* time slower than *Node A*. If each of the *2* reduce tasks has the same computing load *cl*, the overall speedup will be *2cl/t*, where *t* is the time that *Node B* completes its task. If there are *5* reduce tasks by another bigger *MRR* for the *2* nodes, the overall speedup will be *5cl/t*. Consequently, heterogeneity impairs the overall performance more effectively for a small *MRR*.

The situation of positive response by churn is more complex and needs a scrutiny. Qualitatively, churn brings a dynamic overlay while all other factors build certain networks in terms of number of computing nodes and the overall compute-capacity of the overlay. In addition, on dynamic overlay, the cost of lookup of a task is $O(_{log}n)$ depending on the number of computing nodes on the overlay when the lookup happens. To further explain why the exception shows the overlay of size of *40,000* nodes with *30%* random churn performs well for a relatively larger workload caused by a larger *MRR*, we predict: a dynamic overlay varying size in a range performs well for a range of workload correspondingly. To confirm this prediction, we have conducted another enhanced experiment by expending the previous *MRR* experiment to *2* directions: one is to keep the number of computing nodes staying on *40,000* for *30%* churn but increase *MRR* from *100%* to *200%*, representing data expansion applications [19]. The other is to keep *MRR* staying at *20%* but decrease the number of computing nodes from *40,000* to *5,000* for *30%* churn. The experimental results are shown in Fig. 13 for *speedup* and in Fig. 14 for *speedup growth rate*.

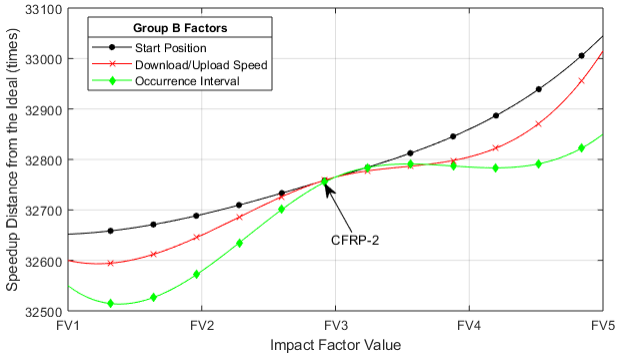

Fig. 10. Reduce step: the impact in speedup distance of Group B factors classified by CFRP2

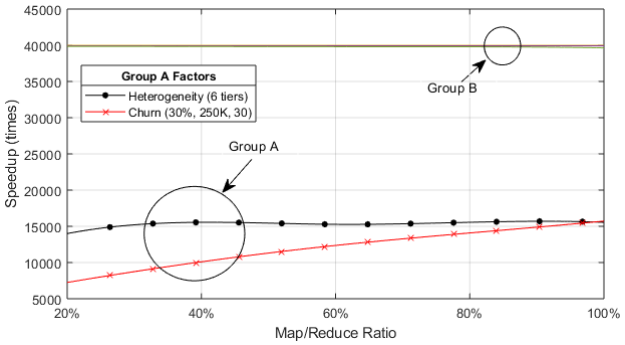

Fig. 11. Impact of single factor injection

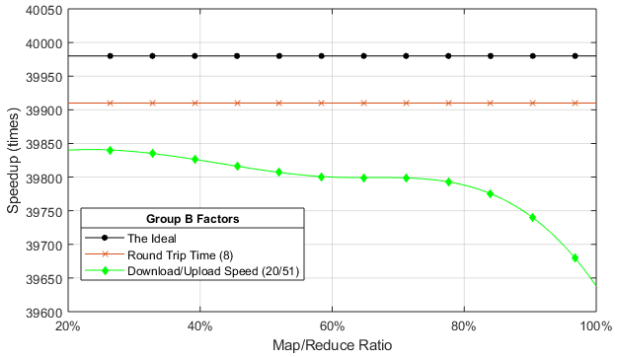

Fig. 12. Impact of single factor injection of Group B factors

It is evident that for the workload caused by a *20% MRR*, small overlay from *5,000* to *20,000* computing nodes speeds up the overall performance from *1430* to *4146* times as shown in Fig. 13 but the speedup grow rate keeps dropping from *92%* down to *1%* correspondingly as shown in Fig. 14. The overall speedup keeps flat for the overlays from *20,000* to *40,000* computing nodes as shown in Fig. 13 and the speedup growth rate fluctuates between *-1%* and *1%* correspondingly as shown in Fig.

14. We conclude that for such a workload caused by a *20%* MRR, the cost (maintenance of reliable overlay on unreliable computing nodes, lookup and download of tasks and upload of results) to work on a dynamic overlay of such a size from *20,000* to *40,000* nodes offsets/counteracts the increase of compute-capacity. Furthermore, the speedup of *40,000* nodes overlay increases again from *4187* to *8685* times in responding to *MRR* change from *20%* to *100%*. This resilience of speedup shows that for these sizes of workload, the compute-capacity of *40,000* overlay overwhelms the working cost on the overlay of such a size. This is the reason of the exceptions in Fig. 7 and Fig. 8 of Section VII-B. However, we can see the resilience of speedup growth rate *44%* keeps dropping to *6%*, responding to the increase of workload as shown in Fig. 14. Although there is still speedup increase from *8685* to *9847* times in responding to *MRR* change from *120%* to *200%*, but the speedup growth rate is very small from *5%* to *1%* correspondingly. The reason is that the compute-capacity of such size overlay (*40,000*) becomes weaker for such workload. Thus, to confirm our prediction, balancing between cost and compute-capacity, overlay of *20,000* nodes, is more suitable for the *MRR* of *20%* and overlay of *40,000* nodes is more suitable for the *MRR* of *100%* in the settings of this experiment.

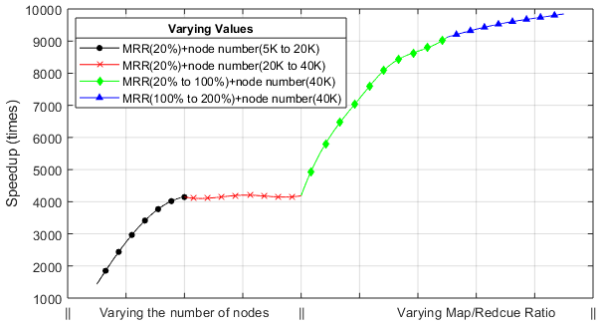

Fig. 13. Speedup responding to varying number of computing nodes or workload caused by MRR

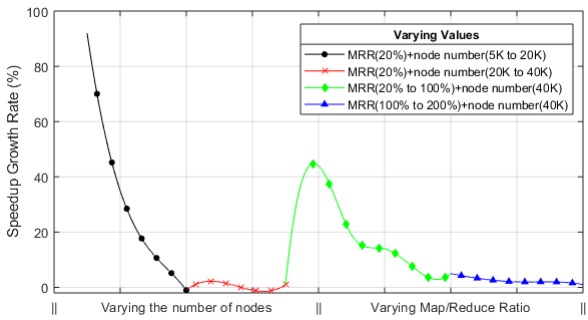

Fig. 14. Speedup growth rate responding to varying number of computing nodes or workload caused by MRR

D. Map Separation and Reduce Union Overlay

An enhanced experiment has been conducted to further confirm that the cost of working on growing overlay will be growing correspondingly and will finally impair the overall performance of the overlay. To evaluate that prediction, we set a full-size map task and a reduce task of *20%*, i.e. *MRR* is *20%*. The other factor values are of *CFRP*$_2$ as defined in Table II of Section V-C. The overlay is growing by 5,000 more computing nodes for each test and the overall performance is measured by *speedup* and *speedup growth rate*. On the basis of the confirmation, we propose to use *Map Separation and Reduce Union Overlay* to improve the situation of working cost on growing overlay.

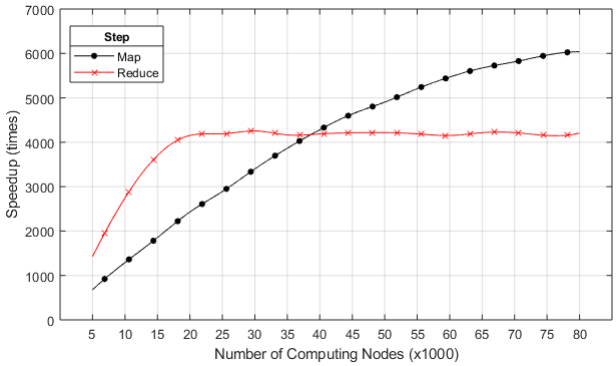

Fig. 15. The speedup in response to overlay growing

It is evident that after a certain size, continuously growing overlay by increasing the number of computing nodes will bring little benefit for the overall performance as shown in Fig. 15 and Fig. 16. To clarify the results, for map step, the speedup growth rate will be less than *5%* for the overlay of *60,000* nodes or more. For reduce step, the speedup growth rate is flat for the overlay of *25,000* nodes or more. This concludes that a growing overlay cannot keep bringing more benefit for the overall performance.

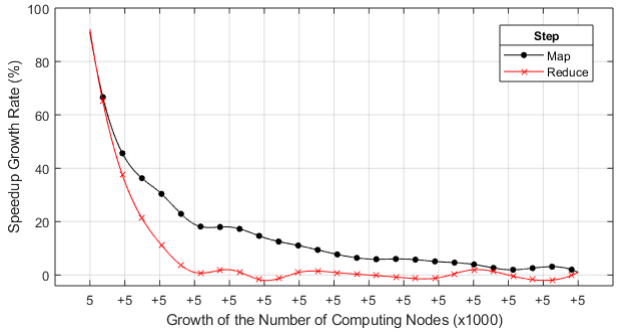

Fig. 16. The speedup growth rate in response to overlay growing

We have observed two facts of MapReduce paradigm. First, map tasks are independent with each other, thus the size of an overlay matters for the storage of map tasks in terms of fault tolerance but is not a matter for computing. Second, if the whole computing of a MapReduce application must be completed in a single map/reduce round, the shuffle step requires all the reduce nodes (reducers) to be on a single overlay. Based on the two facts, the Map Separation and Reduce Union Overlay is proposed as follows:

- Each computing node is on *2* overlays, one is a small size *Map Overlay* and the other is the unique *Reduce Overlay*.

- Each computing node keeps computing the map tasks on its own overlay and upload the results through the shuffle step to the reduce overlay
- Once the map step finishes, the reduce overlay can shrink to an optimal size based on *MRR* as analysed in Section VII-C.
- Each computing node on the reduce overlay keeps computing the reduce tasks till the completion of the whole computing.

To confirm the optimization features of the aforementioned separation and union overlay, an enhanced experiment has been conducted to halve both the overlay size and workload each time for the computing of map tasks. The separation continues by following the conditions:

- The original overlay is a single overlay.
- Each time the number of computing nodes of an overlay is halved to form two overlays and accordingly the original workload is halved as well for each separated overlay.
- Each separated overlay remains the same features as defined by the *8* factors, e.g. the computing capacity, the churn pattern etc as defined in Section III and Section V-B.
- The size of each separated overlay cannot be too small. Otherwise it may affect the fault tolerance features for the storage of tasks.

Based on the aforementioned conditions, we assume that the map step of each separated overlay finishes at the same time *t*. Thus, the *equivalent speedup* will be calculated as: *the whole workload/t*. The experimental results are shown in Fig. 17 for *equivalent speedup* and in Fig. 18 for *equivalent speedup growth rate*.

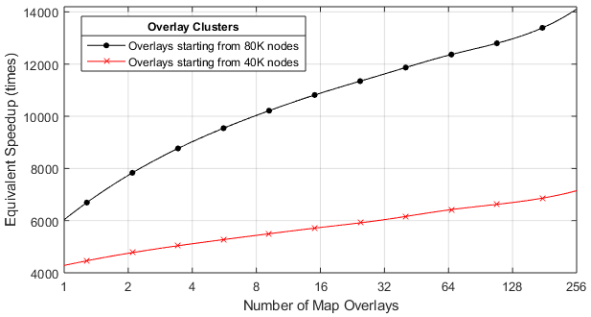

Fig. 17. The equivalent speedup in response to the number of map overlays

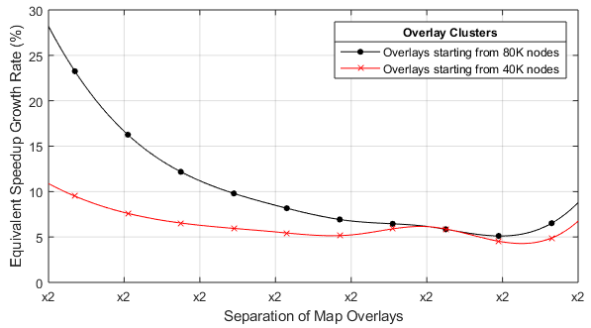

Fig. 18. The equivalent speedup growth rate in response to the separation of map overlays

In this evaluation as shown in Fig. 17, the original overlay of *40,000* or *80,000* nodes will finally be separated into *256* map overlays with each overlay having *156* nodes or *312* nodes 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 of *40,000* or *80,000* nodes. The equivalent speedup of the overlay of *80,000* nodes is higher than that of *40,000* nodes overlay simply because the overall compute-capacity of the former is higher than that of the latter. The equivalent speedup growth rate keeps drooping as shown in Fig. 18, 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.

E. Heterogeneity Reduction

The previously presented experimental results in Section VII-A and Section V-B have showed that heterogeneity of compute-capacity is a key impact factor responsible for severe performance impairment. We predict that growing overlay size will bring little benefit if the growing brings more heterogeneous nodes at the same time. To confirm this prediction, we conduct another enhanced experiment by setting the heterogeneity of compute-capacity to be proportional to the number of computing nodes. To evaluate the relationship between heterogeneity and overlay size, we inject the experimental settings of heterogeneity on top of the ideal computing environment as defined in Section V-A, aiming at removing the interference of dynamics of the overlay.

- The initial overlay size is *6666* computing nodes of *Tier-1* compute-capacity. The workload of *1,400,000* map tasks of each having *8,000* computing-load are used as the reference point.
- Increasing or decreasing the number of computing nodes will increase or decreasing the tiers of compute-capacity proportionally. The rule is that growing/decreasing the number of computing nodes by *1/6* will increase/decrease the heterogeneity of compute-capacity for *1* tier.

The *15* experimental pairs of the number of computing nodes vs the tiers of compute-capacity are listed as follows in the format: P_i*(tier, number of nodes)*, where *i* is the sequence number of a pair.

$P_1(1, 6666)$, $P_2(2, 13333)$, $P_3(3, 20000)$, $P_4(4, 26666)$, $P_5(5,33333)$, $P_6(6, 40000)$, $P_7(7, 46666)$, $P_8(8, 53333)$, $P_9(9, 60000)$, $P_{10}(10, 66666)$, $P_{11}(11, 73333)$, $P_{12}(12, 80000)$, $P_{13}(13, 86666)$, $P_{14}(14, 93333)$, $P_{15}(15, 100000)$

The change of pairs are listed as follows in the format $C_i(P_j \rightarrow P_k)$, which means that the *ith* change is from pair P_j to P_k, where $i, j \in \{1, 2, ..., 14\} \wedge k \in \{2, 3, ..., 15\}$.

$C_1(P_1 \rightarrow P_2)$, $C_2(P_2 \rightarrow P_3)$, $C_3(P_3 \rightarrow P_4)$, $C_4(P_4 \rightarrow P_5)$, $C_5(P_5 \rightarrow P_6)$, $C_6(P_6 \rightarrow P_7)$, $C_7(P_7 \rightarrow P_8)$, $C_8(P_8 \rightarrow P_9)$,

$C_9(P_9 \rightarrow P_{10})$, $C_{10}(P_{10} \rightarrow P_{11})$, $C_{11}(P_{11} \rightarrow P_{12})$, $C_{12}(P_{12} \rightarrow P_{13})$, $C_{13}(P_{13} \rightarrow P_{14})$, $C_{14}(P_{14} \rightarrow P_{15})$

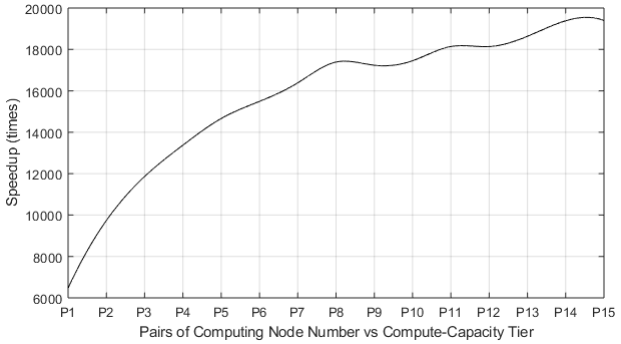

Fig. 19. The speedup in response to the proportional number of computing nodes and tiers of compute-capacity

The evaluation results have been reported in Fig. 19 for speedup vs pairs of tier and compute-capacity and in Fig. 20 for speedup growth rate vs change of pairs. The evaluation results have confirmed that from the number of computing nodes of *40,000* and compute-capacity of *6*-tiers (P_6 and C_6 in the above list) forward, the speedup growth rate is less than *6%* and going flat when increasing the number of computing nodes and tiers of compute-capacity proportionally from that point. This result has demonstrated that even in a non-dynamic environment, computing nodes with compute-capacity lower than a certain level will not speed up but even slow the overall performance. The reason is that the slow nodes still hold the assigned tasks when fast nodes are available.

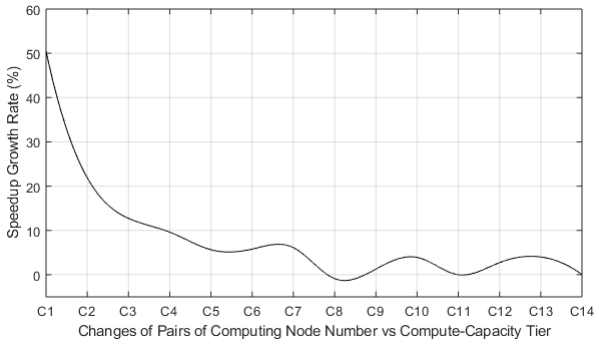

Fig. 20. The speedup growth rate in response to the proportional change of the number of computing nodes and the tiers of compute-capacity

VIII. Conclusions and Future Work

A series of experiments has been conducted to compare the impact strength of the *8* factors: heterogeneity, download/upload speed, round trip time, churn rate, start position, occurrence interval, map/reduce ratio and redistribution factor to cover the impact from compute-capacity, communication, dynamics of network and applications themselves. To make these factors comparable, a spectrum of *5* representative values has been chosen for each factor. To make the results trustable, *2* reference points at the beginning of the spectrum and in the middle of the spectrum have been chosen to inspect consistence between the results.

The evaluation results have demonstrated that for map step the factors can be classified into *2* groups: *Group A* in ascending impact strength of churn rate, round trip time, redistribution factor and heterogeneity, and *Group B* in ascending impact strength of map/reduce ratio, occurrence interval, download/upload speed and start position. Any factors in *Group A* vary their impact much more severe than any factors in *Group B*. For reduce step with an exception, the factors can still be classified into *Group A* and *Group B* having the same intra- and inter-group impact strength as those of map step. The exception is map/reduce ratio, which incurs further evaluation results that a certain size of overlay produces optimal efficiency for a certain level of workload. A possible optimization that has been confirmed for balancing the working cost on an overlay and the compute-capacity of the overlay is to separate the overlay to multiple overlays for map step but retain a single overlay for reduce step. The final conclusive result is that heterogeneity is the strongest factor among the *8* impact factors for both map and reduce steps. Given a certain workload, increasing the number of computing nodes with more heterogeneous compute-capacities will not be useful for the overall performance.

Based on the confirmation of the impact strength of the factors and the relationship between the impact strength, our future work is to explore the optimization of overlay construction to answer the question: given a workload in terms of number of map and reduce tasks, data size and computing load of each task, what overlay in terms of heterogeneity, communication cost and churn will be optimized for the overall performance requirements.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Wei Li conducted the research and experiments and wrote the paper. William Guo reviewed the paper with revision comments. All authors had approved the final version.

References

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

[2] 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

[3] ATLAS@Home 2020. [Online]. Available: http://lhcathome.web.cern.ch/projects/atlas

[4] Asteroids@home 2020. [Online]. Available: http://asteroidsathome.net/
[5] Einstein@Home 2020. [Online]. Available: https://einsteinathome.org/
[6] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proc. 5th IEEE/ACM International Conference on Grid Computing*, 2004, pp. 4-10.
[7] L. Sarmenta, "Volunteer computing," PhD thesis, Massachusetts Institute of Technology, 2001.
[8] R. Casado. (2013). The Three Generations of Big Data Processing. [Online]. Available: https://www.slideshare.net/Datadopter/the-three-generations-of-big-data-processing
[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.
[10] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17-32, 2003.
[11] S. Kaffille and K. Loesing. (2007). Open Chord (1.0.4) User's Manual, The University of Bamberg, Germany. [Online]. Available: https://sourceforge.net/projects/open-chord/
[12] S. Singh, R. Garg, and P. K. Mishra, "Observations on factors affecting performance of mapreduce based apriori on hadoop cluster," in *Proc. International Conference on Computing, Communication and Automation*, 2016, pp. 87-94.
[13] Hadoop Project. 2020. [Online]. Available: https://cwiki.apache.org/confluence/display/HADOOP2/ProjectDescription
[14] D. Cheng, J. Rao, Y. Guo, C. Jiang, and X. Zhou, "Improving performance of heterogeneous mapreduce clusters with adaptive task tuning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 774-786, 2017.
[15] Apache Software Foundation 2020a, Wordcount Example. [Online]. Available: https://cwiki.apache.org/confluence/display/HADOOP2/WordCount
[16] Apache Software Foundation. (2020). Grep Example, [Online]. Available: https://cwiki.apache.org/confluence/display/HADOOP2/Grep
[17] Apache Software Foundation. (2020). Terasort Example, [Online]. Available: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/examples/terasort/package-summary.html
[18] A. Spivak, and D. Nasonov, "Data preloading and data placement for MapReduce performance improving," *Procedia Computer Science*, vol. 101, pp. 379-387, 2016.
[19] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating MapReduce performance using workload suites," in *Proc. 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2011, pp. 390-399.
[20] R. Han and X. Lu, "On big data benchmarking," *Lecture Notes in Computer Science*, vol. 8807, pp. 3-18, 2014.
[21] W. H. Lee, H. G. Jun, and H. J. Kim, "Hadoop mapreduce performance enhancement using in-node combiners," *International Journal of Computer Science & Information Technology*, vol. 7, no. 5, pp. 1-17, 2015.
[22] D. Ardagna, S. Bernardi, E. Gianniti, S. Aliabadi, D. Perez-Palacin, and J. Requeno, "Modeling performance of hadoop applications: A journey from queueing networks to stochastic well formed nets," in *Proc. International Conference on Algorithms and Architectures for Parallel Processing*, 2016, pp. 599-613.
[23] E. Dede, Z. Fadika, M. Govindaraju, and L. Ramakrishnan, "Benchmarking mapreduce implementations under different application scenarios," *Future Generation Computer Systems*, vol. 36, pp. 389-399, 2014.
[24] X. Zhang, Y. Wu, and C. Zhao, "MrHeter: Improving MapReduce performance in heterogeneous environments," *Cluster Computing*, vol. 19, no. 4, pp. 1691-1701, 2016.
[25] S. Monsalve, F. Carballeira, and A. Calderón, "A new volunteer computing model for data - intensive applications," *Concurrency and Computation Practice and Experience*, vol. 29, no. 24, 2017.

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

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