A New Approach to the Cloud-based Heterogeneous MapReduce Placement Problem

Xiaoyong Xu and Maolin Tang, *Senior Member, IEEE*

Abstract—Guaranteeing Quality of Service (QoS) with minimum computation cost is the most important objective of cloud-based MapReduce computations. Minimizing the total computation cost of cloud-based MapReduce computations is done through MapReduce placement optimization. MapReduce placement optimization approaches can be classified into two categories: *homogeneous MapReduce placement optimization* and *heterogeneous MapReduce placement optimization*. It is generally believed that heterogeneous MapReduce placement optimization is more effective than homogeneous MapReduce placement optimization in reducing the total running cost of cloud-based MapReduce computations. This paper proposes a new approach to the heterogeneous MapReduce placement optimization problem. In this new approach, the heterogeneous MapReduce placement optimization problem is transformed into a constrained combinatorial optimization problem and is solved by an innovative constructive algorithm. Experimental results show that the running cost of the cloud-based MapReduce computation platform using this new approach is $24.3\% - 44.0\%$ lower than that using the most popular homogeneous MapReduce placement approach, and $2.0\% - 36.2\%$ lower than that using the heterogeneous MapReduce placement approach not considering the spare resources from the existing MapReduce computations. The experimental results have also demonstrated the good scalability of this new approach.

Index Terms—MapReduce, cloud-based MapReduce computation, MapReduce placement, combinatorial optimization, constructive algorithm

1 Introduction

MapReduce is a parallel programming model for processing large data sets, and it has been widely applied in many commercial and scientific applications, such as data mining [1], bioinformatics [2], machine learning [3], and web indexing [4]. A MapReduce computation is typically broken down into a number of map tasks and reduce tasks, which are respectively executed by two kinds of basic computing units called *mappers* and *reducers*. Both mappers and reducers are called *workers* in this paper.

MapReduce was originally proposed for parallel computation in a cluster which consists of a set of connected computers. The objectives of cluster-based MapReduce computations usually focus on minimizing execution time [5] [6] [7] [8], maximizing cluster utilization [9] [10], and so on. However, in cloud-based MapReduce computation, the most important objective is to guarantee the Quality of Service (QoS) of cloud-based MapReduce computations with minimum cost of using virtual machines (VMs). To guarantee the QoS, the required number of workers must be placed on a selected set of VMs such that the resource requirements of each worker must be met and the total cost of using the VMs is minimum. This is so-called *MapReduce Placement Problem* (MRPP) in cloud-based MapReduce computations.

The approaches to the MRPP can be classified into two categories: *homogeneous MapReduce placement optimization* and *heterogeneous MapReduce placement optimization*. The homogeneous MapReduce placement optimization approaches usually place the workers on a set of homogeneous VMs and place the same number of workers on each of the VMs. Since this category of approaches are easy to implement, most of the existing approaches to the MRPP belong to this category [11] [12] [13] [14] [15] [16] [17] [18]. Very recently, a heterogeneous MapReduce placement optimization approach was proposed [19]. The proposed approach can utilize heterogeneous VMs and place different numbers of workers on different VMs. It showed that the proposed heterogeneous MapReduce placement optimization approach is more cost-effective than those homogeneous MapReduce placement optimization approaches. However, the proposed approach did not reuse those VMs used by old MapReduce computations. This paper presents a new heterogeneous MapReduce placement optimization approach that considers not only new VMs of various types, but also the spare CPU and memory capacities of existing VMs.

• *X. Xu and M. Tang are with the School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane, Australia, 4000*

E-mail: {x21.xu, m.tang}@qut.edu.au

Here is a simple example to illustrate how heterogeneous MapReduce placement optimization potentially outperforms homogeneous MapReduce placement optimization. In this example, it is assumed that there are only two types of VMs, *small VMs* and *large VMs*, and that the capacity of a VM is measured by the number of CPUs (or cores). Each of the small VM contains three CPUs and its price is $4/hour; each of the large VM contains six CPUs and its price is $6/hour. Let's say there is one new MapReduce computation which requires four identical workers, each of which needs two CPUs. If we adopt homogeneous MapReduce placement, we would have to use either four small VMs or two large VMs. The total costs of using the VMS would be $16/hour and $12/hour, respectively. However, if we adopt heterogeneous MapReduce placement, we would need only one small VMs and one large VMs, and the total cost of using the VMs is only $10/hour. It can be seen from this simple example that heterogeneous MapReduce placement has potential to cut the total cost of cloud-based MapReduce computations. In this example, we did not reuse the VMS used by existing MapReduce computations. If there is an existing MapReduce computation when a new MapReduce computation comes, and the existing MapReduce computation is using a VM which has two spare CPUs, then we only one new large VM to accommodate the new MapReduce computation, and the total extra cost would be only $6/hour.

In this paper, we will propose a new heterogeneous approach to the MRPP in cloud-based MapReduce computations, which has more potentials to reuse VMs than existing heterogeneous approaches, and therefore can further reduce the total running cost of cloud-based MapReduce computations. We will formulate the MRPP into a constrained combinatorial optimization problem and prove the constrained combinatorial optimization problem is NP-complete. We will also propose a new constructive algorithm for the constrained combinatorial optimization problem, and evaluate the new constructive algorithm.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 is the problem formulation and the proof of the problem being NP-complete. Section 4 presents the new constructive algorithm. Section 5 evaluates the new constructive algorithm and Section 6 concludes the research and discusses future research work on the MRPP.

2 Related Work

In this section we review the researches on the MapReduce placement problem in non-cloud-based MapReduce computations and cloud-based MapReduce computations.

2.1 Non-cloud-based MapReduce Placement

There are a number of researches on the MapReduce placement problem in non-cloud-based MapReduce computations. Zaharia et al. [5] developed a scheduler called LATE for Hadoop MapReduce deployed on heterogeneous environments to reduce job execution time. They followed a default Hadoop configuration for the worker placement on computing nodes [20]. Lin et al. [6] studied the adaptive task and data scheduling algorithms in MOON, a MapReduce implementation under the volunteer computing environment, and improved the MapReduce performance on execution time under the environment with the volatility of resources and high rate of node unavailability. Like the above research, they also did not care about how to assign slots to slave nodes, just implementing the default Hadoop settings. Wolf et al. [7] proposed a slot allocation scheduling optimizer to provide minimum number of slots to MapReduce workloads. This optimizer aimed at optimizing some metrics like execution time while ensuring the same minimum slot guarantees as in HFS, and maximum slot guarantees as well. In their work, the slot assignment on computing nodes followed a default configuration. Kc and Anyanwu [21] developed a constraint-based Hadoop scheduler based on a job execution cost model to meet the deadline constraints specified by users. Also, they used the default Hadoop configuration, placing two mappers/reducers on every node. Verma et al. [22] developed an automatic resource inference and allocation framework for MapReduce to meet job deadlines. With regard to MapReduce placement, they just adopted a simple way in which a fixed number of workers is assigned to each node.

Unlike the above researches which use the default Hadoop configurations, Herodotou et al. [8] studied how to find the optimum settings of parameters in MapReduce programs to optimize job execution time by means of a self-tuning system named Starfish. The placement of task slots on computing nodes were automatically determined by that system. Polo et al. [9] presented a resource-aware scheduler for MapReduce multi-job workloads. The slot on each node could be dynamically adjusted by leveraging the resource consumptions of different jobs, so as to maximize the resource utilization of the cluster. Later, Polo et al. [23] studied the deadline-based management for MapReduce workloads based on the same assignment technology of task slots, but the aim was to ensure the deadline meeting of jobs. Wang et al. [10] proposed an automatic control mechanism for the dynamical assignment of task slots on each computing node. Using their mechanism, the cluster-wide resource utilization was improved.

All above researches study the MapReduce placement problems in non-cloud environments. The objective of these problems are usually to improve the

cluster utilization, to reduce the execution time, or to meet the deadline. In addition, in their problems, the total number of machines and the types of machines, which can be used for MapReduce computations, are given beforehand.

The MRPP studied in this paper is totally different from those problems in non-cloud environments. Different from the objectives of those problems in non-cloud environments, the objective of the MRPP is to minimize the monetary cost of using VMs. Also, unlike those problems in non-cloud environments, in the MRPP, the total number and types of the VMs, which will be used for MapReduce computations, are unknown in advance. Instead, in the MRPP, we need to select the VM types and the number of the VMs of each selected type, and to determine the placement of workers on selected VMs. Therefore, the aforementioned MapReduce placement approaches for non-cloud-based MapReduce computations cannot be used to address the problem studied in this paper.

2.2 Cloud-based MapReduce Placement

There are also some researches on the MapReduce placement problems in cloud computing environment. Tian et al. [11] studied how to minimize financial charge for a single MapReduce job while meeting a time deadline. In their proposed approach the same number of workers are placed on the same type of slave nodes. Abdelbaky et al. [12] studied proposed an objective-driven scheduler which minimized the required number of VMs to meet the deadline constraint for MapReduce-CometCloud. In their scheduler, each VM could only load one mapper or reducer, although the VMs were heterogeneous. Hwang and Kim [13] studied the resource provisioning problem for MapReduce in the cloud, which aimed at minimizing the monetary cost of VMs while meeting the deadline constraints. They paid more attention on the placement of the VMs on physical machines, while for the problem of mapper/reducer placement on VMs, they adopted a simple way in which a fixed number of mappers/reducers were assigned to each VM. Lama et al. [14] proposed an automated job provisioning system for Hadoop MapReduce. This system could automatically configure the number of VMs to achieve QoS goals while minimizing the incurred cost. But they did not study how to optimize the mappers/reduecrs on the VMs. Alternatively, they assigned mappers/reduers to the VMs following a basic rule like one mapper and one reducer to a small VM while two mappers and two reducers to a medium VM. Chen et al. [15] built up a cost function modeling the relationship among execution time, input size, and available cloud resource, and solved a problem aiming at meet deadline requirements with minimum monetary cost. Just like previous researches, they studied the optimum number of VMs rather than the placement optimization of mappers/reducers on VMs. With regard to the placement, they placed the same number of mappers/reducers on one type of VMs.

Unlike the above researches using the simple rules of assigning workers to VMs, Herodotou et al. [16] used a more exact method to address the MapReduce placement issue. They developed a system named Elastisizer included in Starfish to answer the cluster sizing problems for the MapReduce operated on cloud platforms. This system could tell MapReduce users the best VM type from multiple types provided by public clouds and the optimum number of the VMs of that VM type. However, the cluster sizing problems were different the MRPP, since the constraints of the cluster sizing problems were meeting the desired requirements on execution time or cost whereas the constraint of the MRPP was satisfying the resource requirements of all the workers to be placed. Thus, their approach could not be used to address the MRPP. Cardosa et al. [17] studied how to place the VMs for MapReduce computations on physical machines with minimum energy costs. Their problem was similar to the MRPP, but the MRPP was more complicated. The physical machines or bins were identical in their problem whereas multiple types of VMs or bins were considered in the MRPP. Also, the number of the bins in their problem was definite, while that number in the MRPP was infinite. Thus, their algorithm could not be used to address the MRPP immediately. Palanisamy et al. [18] proposed a cost-effective resource provisioning model called Cura for cloud-based MapReduce, which could help MapReduce users to decide the right VM type and size for every job by leveraging Starfish to analyze the job profiles.

The MapReduce placement approaches involved in the above researches almost can be categorized into homogeneous MapReduce placement optimization, as they usually assigned workers on homogeneous VMs or follow homogeneous configurations of worker numbers on each VM. These approaches are totally different from the heterogeneous MapReduce placement optimization approach proposed in this paper, which allows using heterogeneous VMs and heterogeneous placement on each used VM.

We have given a preliminary work [19] on the MapReduce placement problem in cloud-based MapReduce. However, the MRPP in this paper is different from that problem in [19]. In our preliminary work, the existing resources (VMs) have not been considered, and the algorithm could not make good use of the existing resources to further reduce the cost of the cloud-based MapReduce computations. However, in this paper, the issue about how to utilize the existing resources has been addressed.

Besides, there are also some researches on other

placement problems similar to the MRPP in cloud computing, like the VM placement problems. Most of them took the problems as bin-packing ones, and adopted or modified bin-packing algorithms, like first-fit-decreasing [24] [25], best-fit-decreasing [26], set covering [27] [28], or other algorithms [29] [30], to solve them. But the MRPP is more complicated than their problems, as the MRPP considers both multiple types of bins (VMs) and multiple resource constraints whereas their problems just considered either of them. Thus, their algorithms also can not be used for the MRPP immediately.

3 Problem Formulation

There could be different types of cloud-based MapReduce implementations. In this research it is assumed that the cloud-based MapReduce computation platform is built on top of a set of VMs of various types rented from a public cloud, and the cloud-based MapReduce computation platform can perform multiple MapReduce computations concurrently and new MapReduce computations may arrive and existing MapReduce computations may finish and go at any time. In order to minimize the ongoing running cost of the cloud-based MapReduce computation platform, we should minimize the running cost of the cloud-based MapReduce computation platform at any time.

In order to minimize its running cost, the cloud-based MapReduce may use a number of different types of VMs which have different capacities and prices. Thus, a fundamental problem is to find which types of VMs should be rented, the numbers of instances of each selected VM type and the placement of the mappers and reducers (workers) on those rented VMs such that the total cost of renting the VMs is minimum while guaranteeing the QoS of the cloud-based MapReduce computation platform at any time.

There are two situations where the MapReduce placement of the cloud-based MapReduce computation platform may need to be reorganized. One is when some existing MapReduce computations are finishing; another is when some new MapReduce computations coming. In the former situation, the MapReduce placement problem can be transformed into a VM consolidation problem, which has been intensively investigated for many years and many efficient approaches have been proposed [31] [32] [33] [34]. Thus, this paper focuses on the MapReduce placement problem in the latter situation, which is formulated in the following.

It is assumed that there are n new MapReduce computations arriving and n' existing MapReduce computations when the MapReduce placement is carried out. In order to guarantee the QoS of the i^{th} new MapReduce computation ($1 \leq i \leq n$), at least t_i^M mappers and t_i^R reducers need to be provided for the map/reduce tasks of the new MapReduce computation, and need to be placed on VMs where their resource requirements are met. The mappers and reducers provided for the new MapReduce computations are respectively expressed by two tuples, $< M_i^{CPU}, M_i^{Mem} >$ and $< R_i^{CPU}, R_i^{Mem} >$, where M_i^{CPU} and M_i^{Mem} are the CPU and memory requirements of the map tasks of the i^{th} new MapReduce computation ($1 \leq i \leq n$), and R_i^{CPU} and R_i^{Mem} are the CPU and memory requirements of the reduce tasks of of the i^{th} new MapReduce computation.

The mappers and reducers provided for the existing MapReduce computations are respectively expressed by two tuples, $< {M'}_i^{CPU}, {M'}_i^{Mem} >$ and $< {R'}_i^{CPU}, {R'}_i^{Mem} >$, where ${M'}^{CPU}_i$ and ${M'}_i^{Mem}$ are the CPU and memory requirements of the map tasks of the i^{th} existing MapReduce computation ($1 \leq i \leq n'$), and ${R'}_i^{CPU}$ and ${R'}_i^{Mem}$ are the CPU and memory requirements of the reduce tasks of the i^{th} existing MapReduce computation.

All the mappers/reducers of the MapReduce computations are required to be placed on VMs, and the VMs that can be used include a set of new VMs, denoted by $\mathcal{V}$, rented from the public cloud, and a set of existing VMs, denoted by $\mathcal{V}'$, which are being used by existing MapReduce computations and have some spare resources. The new VMs can be classified into m types in terms of their resource capacities and prices and the existing VMs can be classified into m' types in terms of their spare resource capacities, and $\mathcal{V} = \bigcup_{j=1}^{m} \mathcal{V}_j$, $\mathcal{V}' = \bigcup_{j=1}^{m'} \mathcal{V}'_j$, where $\mathcal{V}_j$ is a multiset of new VMs of type j and $\mathcal{V}'_j$ is a multiset of existing VMs of type j.

In addition, let v_k be an instance of the VMs to be used in the MapReduce placement, where $v_k \in \mathcal{V} \cup \mathcal{V}'$, $1 \leq k \leq |\mathcal{V}| + |\mathcal{V}'|$, and v_k has a CPU capacity, v_k^{CPU} and a memory capacity v_k^{Mem}. Let

$$v_k^s =< x_{k1}^M, x_{k2}^M, \cdots, x_{kn}^M, x_{k1}^R, x_{k2}^R, \cdots, x_{kn}^R >$$

be the assignment of the mappers and reducers of the new MapReduce computations to $v_k \in \mathcal{V} \cup \mathcal{V}'$, where x_{ki}^M and x_{ki}^R are the numbers of the mappers and reducers of the i^{th} new MapReduce computation assigned to v_k and $1 \leq i \leq n$; and let

$$v_k^{s'} =< c_{k1}^M, c_{k2}^M, \cdots, c_{kn'}^M, c_{k1}^R, c_{k2}^R, \cdots, c_{kn'}^R >$$

be the assignment of the mappers and reducers of the existing MapReduce computations to $v_k \in \mathcal{V}'$, where c_{ki}^M and c_{ki}^R are the numbers of the mappers and reducers of the i^{th} existing MapReduce computation assigned to v_k.

Given the entire set of existing VMs that have spare resources, $\mathcal{V}'$, and the placement of the mappers and reducers of the n' existing MapReduce computations on the VMs in $\mathcal{V}'$, the MRPP is to find a set of new VMs, $\mathcal{V}$, and placements of the mappers and reducers of the n new MapReduce computations on all the new

and existing VMs such that the total cost of those new VMs is minimal, that is,

$$\min \sum_{j=1}^{m} p_j \cdot |\mathcal{V}_j| \quad (1)$$

subject to

$$\sum_{k=1}^{|\mathcal{V}|+|\mathcal{V}'|} x_{ki}^{M} = t_i^{M}, 1 \leq i \leq n \quad (2)$$

$$\sum_{k=1}^{|\mathcal{V}|+|\mathcal{V}'|} x_{ki}^{R} = t_i^{R}, 1 \leq i \leq n \quad (3)$$

$$\sum_{i=1}^{n}(x_{ki}^{M} \cdot M_i^{CPU} + x_{ki}^{R} \cdot R_i^{CPU}) \leq v_k^{CPU}, \forall v_k \in \mathcal{V}_j \quad (4)$$

$$\sum_{i=1}^{n}(x_{ki}^{M} \cdot M_i^{Mem} + x_{ki}^{R} \cdot R_i^{Mem}) \leq v_k^{Mem}, \forall v_k \in \mathcal{V}_j \quad (5)$$

$$\sum_{i=1}^{n}(x_{ki}^{M} \cdot M_i^{CPU} + x_{ki}^{R} \cdot R_i^{CPU}) + \sum_{i=1}^{n'}(c_{ki}^{M} \cdot {M'}_i^{CPU} + c_{ki}^{R} \cdot {R'}_i^{CPU}) \leq v_k^{CPU}, \forall v_k \in {\mathcal{V}'}_j \quad (6)$$

$$\sum_{i=1}^{n}(x_{ki}^{M} \cdot M_i^{Mem} + x_{ki}^{R} \cdot R_i^{Mem}) + \sum_{i=1}^{n'}(c_{ki}^{M} {M'}_i^{Mem} + c_{ki}^{R} {R'}_i^{Mem}) \leq v_k^{Mem}, \forall v_k \in {\mathcal{V}'}_j \quad (7)$$

In the above problem formulation, p_j is the price of the j^{th} type of VM. Constraints (2) and (3) ensure the required numbers of mappers and reduces of all the new MapReduce computations are placed on the VMs; constraints (4) and (5) make sure the total CPU and memory requirements of the mappers/reducers on a new VM do not exceed its CPU and memory capacities; constraints (6) and (7) guarantee the total CPU and memory requirements of the mappers/reducers of the new MapReduce computations and the existing MapReduce computations on an existing VM do not exceed its CPU and memory capacities.

The MRPP is NP-complete, and the proof for this is presented by the following theorem.

Theorem 3.1. *The MRPP is NP-complete.*

Proof: The MRPP is a special case of the classical bin packing problem [35] where the workers are objects and the VMs are containers, and the volume of an object (worker) is its CPU requirement and the volume of a container (VM) is the VM's CPU capacity. Let the memory requirement of all the objects be r^M which is a constant, and the memory capacity of a container (VM) be $N * r^M$, where N is the total number of objects (workers). Then, the packing is only constrained by the VM's CPU capacity, but not the VM's memory capacity. In addition, let the cost of each VM be the same, amounting to one dollar. Thus, in this special case, the MRPP can be transformed into the classical bin packing problem: given a set of objects (workers), how to pack these objects into the minimum number of containers (VMs). Since the classical bin packing problem is NP-complete [35], the MRPP is also NP-complete. □

4 A New Algorithm for the MRPP

Since the MRPP is NP-complete and the size of the MRPP is usually large, it is not feasible to adopt an optimum algorithm to solve it as it would lead to the explosion in its search space. Therefore, we propose an approximation algorithm for the MRPP.

The approximation algorithm is basically a constructive algorithm, which is broken down into two consecutive procedures: *placement pattern generation* and *MRPP solution building*. The first procedure is used to generates a small set of placement patterns; the second procedure is used to find a combination of the placement patterns that form a solution to the MRPP with a minimum total cost for using VMs.

A *placement pattern* for a type of VM is a combination of workers of various types that can be placed on that type of VM satisfying the capacity constraints of that type of VM. A placement pattern is said to be feasible if the total CPU and memory requirements of those workers that are placed on that type of VM do not exceed the CPU and memory requirements of that type of VM, respectively. The details about the placement patterns generation procedure and the MRPP solution building procedure are discussed in the following subsections.

4.1 Placement Pattern Generation Procedure

The basic idea behind the placement pattern generation procedure is to use an FFD-based algorithm to generate a set of placement patterns for each type of VM, where the VM is a container and there are many instances of the container, and the workers are objects that need to be put into the multiple containers. *Algorithm* 1 describes a procedure that generates a set of placement patterns for a particular type of VM.

The input of Algorithm 1 is the entire multiset of workers, $\mathcal{W}$, which are needed to be placed on multiple instances of the j^{th} type of VM. The output of the algorithm is a set of placement patterns for the j^{th} type of VM, $\overline{\mathcal{S}}_j$.

In order to make the algorithm more efficient, first of all, the algorithm sorts out those workers which cannot be put into any of the containers because their 'size' is bigger than that of any container, which is done by checking if their resource requirements exceed the capacity of the container in steps 2-6 of the algorithm. Then, the algorithm iterates q times (steps 7-22) of a variant of the FFD algorithm, namely

Algorithm 1 Generating a set of placement patterns for j^{th} type of VM

1: $\mathcal{W}_j = \emptyset$, $\overline{\mathcal{S}}_j = \emptyset$;
2: **for** $i = 1$ to $|\mathcal{W}|$ **do**
3: **if** the CPU/memory requirement of the worker $w_i \in \mathcal{W}$ does not exceed the CPU/memory capacity of a VM of the j^{th} type **then**
4: $\mathcal{W}_j = \mathcal{W}_j \cup \{w_i\}$;
5: **end if**
6: **end for**
7: **for** $k = 1$ to q **do**
8: **for** $i = 1$ to $|\mathcal{W}_j|$ **do**
9: $S_i = \emptyset$;
10: **end for**
11: randomly generate a sequence of the workers in $\mathcal{W}_j$, L;
12: **while** $L \neq \emptyset$ **do**
13: get the first worker w from L;
14: put w into the first VM container that can accommodate it;
15: remove w from L;
16: **end while**
17: **for** $i = 1$ to $|\mathcal{W}_j|$ **do**
18: **if** $S_i \neq \emptyset$; **then**
19: $\overline{\mathcal{S}}_j = \overline{\mathcal{S}}_j \cup S_i$
20: **end if**
21: **end for**
22: **end for**
23: output $\overline{\mathcal{S}}_j$;

random FFD algorithm (steps 8-16) in which the order of the objects (workers) is randomly generated, rather than in descending order by their 'size'. The reason behind that is that we wanted the procedure to generate different placement patterns in each of the iterations. The total number of containers used in the random FFD algorithm is $|\mathcal{W}_j|$, which is enough to accommodate all the objects (workers). Thus, after the packing process of the random FFD algorithm there could be some containers which are empty. Thus, we need to get rid of those empty containers (steps 17-21). Each of the non-empty containers, S_i, gives a placement pattern for the j^{th} type of VM, and all the placement patterns generated in the q iterations are stored in $\overline{\mathcal{S}}_j$.

Algorithm 2 Placement pattern generation

1: $\overline{\mathcal{S}} = \emptyset$;
2: **for** $j = 1$ to $m + m'$ **do**
3: use $Algorithm\ 1$ to generate a set of placement patterns for the j^{th} type of VM, $\overline{\mathcal{S}}_j$;
4: $\overline{\mathcal{S}} = \overline{\mathcal{S}} \cup \overline{\mathcal{S}}_j$;
5: **end for**
6: output $\overline{\mathcal{S}}$;

The placement pattern generation procedure is described in *Algorithm 2*. The input is the entire multiset of the workers needed to be placed, $\mathcal{W}$, and the output is a set of placement patterns for all m types of VMs, $\overline{\mathcal{S}}$.

Algorithm 2 iterates $m+m'$ times (steps 2-5), where m is the total number of types of VMs and m' is the total number of types of existing VMs. It should be noted that we categorize the existing VMs with the same spare CPU and memory capacities into the same type. In each iteration, Algorithm 2 invokes Algorithm 1 to generate a set of placement patterns for one type of VM, $\overline{\mathcal{S}}_j$ (step 3), and then merges those placement patterns stored in $\overline{\mathcal{S}}_j$ into $\overline{\mathcal{S}}$ (step 4). Finally, it outputs $\overline{\mathcal{S}}$.

4.2 MRPP Solution Building Procedure

After using the above placement pattern generation procedure to find a set of feasible placement patterns for all types of VMs, the MRPP solution building procedure is used to find the best combination of the placement patterns in $\overline{\mathcal{S}}$ to form a solution to the MRPP.

From the computational point of view, the MRPP solution building problem is a constrained combinatorial optimization problem. Considering that the total number of feasible placement patterns are not huge, however, we transform the MRPP solution building problem into a Mixed Integer Programming (MIP) [36] problem as follows:

A placement pattern can be expressed by an N-tuple:

$$s_j^k =< x_{jk}^1, x_{jk}^2, \cdots, x_{jk}^i, \cdots, x_{jk}^N >$$

where s_j^k is the k^{th} placement pattern of the j^{th} type of VM, x_{jk}^i is the number of workers of the i^{th} type used in the placement pattern, and N is the total number of different types of workers. It should be noted that the workers with the same CPU and memory requirements are categorized into the same type. The objective of the MIP problem is

$$\min Z = \sum_{j=1}^{m} \sum_{k=1}^{|\overline{\mathcal{S}}_j|} p_j \cdot y_j^k \tag{8}$$

subject to

$$\sum_{j=1}^{m+m'} \sum_{k=1}^{|\overline{\mathcal{S}}_j|} x_{jk}^i \cdot y_j^k \geq |\mathcal{W}_i|, 1 \leq i \leq N \tag{9}$$

$$\sum_{k=1}^{|\overline{\mathcal{S}}_j|} y_j^k \leq N_j, m < j \leq m + m' \tag{10}$$

$$y_j^k \geq 0, 1 \leq k \leq |\overline{\mathcal{S}}_j|, 1 \leq j \leq m + m' \tag{11}$$

In Eq. (8), Z is the total cost of all the VMs needed in the MapReduce placement, y_j^k is the decision variable

representing the number of the placement pattern, s_j^k, used in the MapReduce placement, p_j is the price of the VM of the j^{th} type, $|\overline{\mathcal{S}}_j|$ denotes the total number of the placement patterns for the j^{th} VM type, which is generated in the placement pattern generation procedure. The constraint (9) ensures the required number of the workers of every type involved in the MapReduce computations are assigned to one of the VMs. The constraint (10) makes sure the number of each type of existing VMs used in the solution do not exceed its available number, where N_j is the maximum available number of one type of existing VMs. The constraint (11) ensures all variables must be non-negative integers.

It should be noted that the total number of workers in the MIP solution could be more than the total number of workers required to be placed in the MRPP because of the relaxed constraint (9). Therefore, we need to remove those redundant workers from the MIP solution before the MIP solution can be used for the MRPP.

5 EVALUATION

The evaluation of our new approach is done through two experiments. The first experiment is to test the performance of our new constructive algorithm (NCA). In the experiment, we compare NCA with three baseline algorithms in terms of the cost of the MapReduce placements generated by the algorithms for a set of test instances of various characteristics.

One of the baseline algorithms is the most popular algorithm for HOMOgeneous MapReduce placement (HOMO) presented in [18]. HOMO selects a suitable type of VM among multiple types of VM and then assigns the same number of workers to multiple instances of the selected type of VM. A second baseline algorithm is an FFD-based MapReduce placement algorithm (FFD-based). The FFD-based algorithm picks workers in a decreasing order by their resource requirements and places them in a first-fit fashion. Details about this algorithm can be found in [37]. A third baseline algorithm is the original constructive algorithm (OCA) presented in [19]. Both the FFD-based algorithm and NCA reuse those spare resources on existing VMs whereas HOMO and OCA do not. All the algorithms except for MONO are designed for heterogenous MapReduce placement.

The second experiment is to test the scalability of NCA, which is done by observing how the computation time of NCA increases when the size of the test problems increases.

Both of the experiments were conducted on a laptop with an Intel Core i7-3520M CPU (2.90 GHz) and 8 GBs of RAM. All the VMs used in the experiments were generated by VMware Workstation 10.0.0 [38], and were deployed on 12 HP workstations (32 Intel Xeon 2.40 GHz CPUs and 320 GB memory) interconnected via a Gigabit Ethernet network. Hadoop 0.20.2 [39] was used to run the MapReduce benchmarks and Ganglia [40] was used to monitor the resource consumption during runtime. All of the algorithms used in the experiments were implemented in C#. The solver for the MIP in the MRPP solution building procedure is CPLEX (12.5.1.0) [41].

5.1 Construction of test instances

In the evaluation, we selected two benchmarks for MapReduce computations from a popular MapReduce benchmark suit, namely *HiBench* [42], and used the benchmarks to construct a number of test instances of different sizes, each of which was used as a test problem in the experiments. One benchmark was *TeraSort*, a standard MapReduce sort benchmark; another was *WordCount*, an application that counts the number of occurrences of each word in a text file.

Each test instance had three inputs: the number of MapReduce computations, the number of workers in each of the MapReduce computations, and the information about existing VMs. The types of VMS used in the experiments are shown in TABLE 1.

TABLE 1
The VM types used in the experiments

VM Type	CPUs (#Cores)	Mem (GB)	Cost ($)
m1 small	1	1.7	0.06
m1 medium	2	3.75	0.12
m1 large	4	7.5	0.24
m1 xlarge	8	14.7	0.48
m2 xlarge	6.5	17.1	0.41
m2 2xlarge	13	34.2	0.82
c1 medium	5	1.7	0.145
c1 xlarge	20	7	0.58

When constructing test instances, we needed to know the resource requirements of workers (mappers and reducers), which was done by experiments. TABLE 2 shows the resource requirements of the workers for the two benchmarks with different input sizes. The resource requirements shown in the table is the average results of 10 runs.

Using the information shown in TABLE 2, we used the following methods to construct more and large-size test instances. It was assumed that the CPU and memory requirements of mappers were uniformly distributed in the interval $[a, b]$, where a was the observed minimum amount of resource requirement and b was the observed maximum amount of resource requirement. Thus, the CPU requirement for a test instance with a fixed input size was randomly picked up between a and b.

The CPU and memory requirements of reducers were generated in different way, which will be elaborated below. It was observed that the requirements for CPU and memory were in proportional to the input size of the MapReduce computation. Thus, to generate the CPU and memory requirements for reducers, we

TABLE 2
The resource requirements of the workers with different input sizes

	Input Size (GB)	Mapper		Reducer	
		CPUs (#Cores)	Mem (GB)	CPUs (#Cores)	Mem (GB)
TeraSort	2	[1.5,1.8]	[0.1,0.2]	1.12	0.9
	4	[1.5,1.8]	[0.1,0.2]	1.32	1.3
	6	[1.5,1.8]	[0.1,0.2]	1.4	1.65
	8	[1.5,1.8]	[0.1,0.2]	1.52	1.8
	10	[1.5,1.8]	[0.1,0.2]	1.68	2
WordCount	4	[1.7,1.9]	[0.3,0.4]	0.68	0.15
	8	[1.7,1.9]	[0.3,0.4]	0.85	0.4
	12	[1.7,1.9]	[0.3,0.4]	1.08	0.59
	16	[1.7,1.9]	[0.3,0.4]	1.2	0.7
	20	[1.7,1.9]	[0.3,0.4]	1.29	0.85

firstly applied the following four linear regressions to find the relationship between the CPU and memory requirements and the MapReduce computation input size:

$$y_c^{ts} = 0.066x + 1.012 \quad (12)$$
$$y_m^{ts} = 0.135x + 0.72 \quad (13)$$
$$y_c^{wc} = 0.0393x + 0.549 \quad (14)$$
$$y_m^{wc} = 0.0425x + 0.028 \quad (15)$$

where x is the input size, y_c^{ts} (y_m^{ts}) represents the requirement for CPU (memory) of the reducers for TeraSort, y_c^{wc} (y_m^{wc}) denotes the requirement for CPU (memory) of the reducers for WordCount.

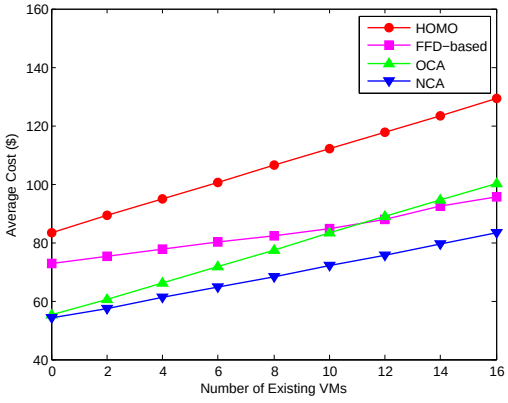
Given any input size x, which was uniformly distributed in the interval $[10 - 120]$, we calculated the resource requirements for reducers using the above equations.

5.2 Experiments and Results

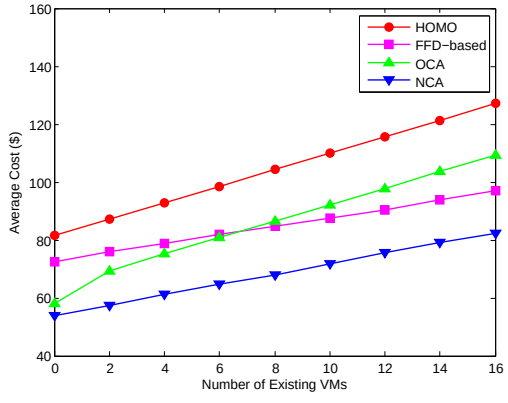
In the experiments, we used HOMO, the FFD-based algorithm, OCA and NCA to solve each of the test instances. Because of the stochastic nature of OCA and NCA, we repeatedly used them to solve each of the test instances for 20 times and used the averages of the 20 runs to compare with the other two algorithms. The maximum time for solving the MIP problem in the MRPP solution building phase of OCA and NCA was set to 30 seconds, following the suggestion in [28]. The parameter q used in Algorithm 2 was fixed to 10 after a number of trials.

Fig. 1 (a) and (b) show how the costs of using VMs varied when the four algorithms were used to solve the test instances of TeraSort and WordCount, respectively. In the experiments, the number of existing VMs varied from 0 to 18, the number of worker types was fixed at 24, and the number of workers of each type was fixed at 20. It was assumed in the experiments that the remaining resource on each existing VM was 50% of the total resource.

It can be seen from Fig. 1 that when the number of existing VMs was zero, or there was no existing VMs,



(a) TeraSort

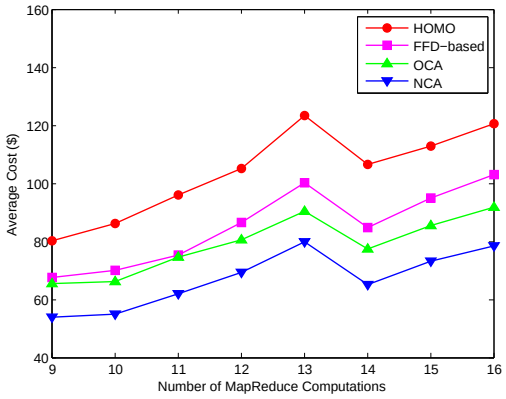


(b) WordCount

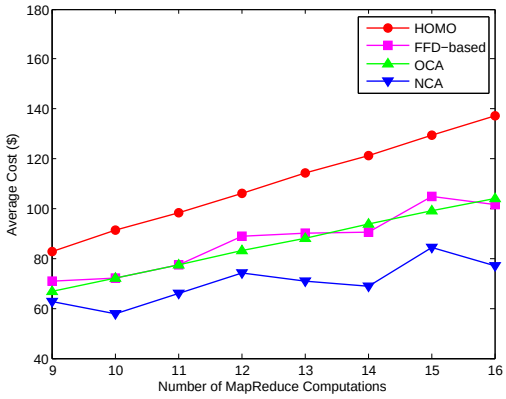
Fig. 1. The comparison of the four algorithms on the cost of using VMs when the number of existing VMs varied

the cost of the MapReduce placement generated by NCA was 35.1% less than that of HOMO, 25.8% less than that of the FFD-based algorithm and 2.0% less than that of OCA for those test instances of TeraSort, and 33.7% less than that of HOMO, 25.6% less than that of the FFD-based algorithm and 7.1% less than that of OCA for those test instances of WordCount. It can be also seen from Fig. 1 that when there were existing VMs, the cost of the MapReduce placement generated by NCA was 35.5%−35.7% less than that of

HOMO, $12.9\% - 23.6\%$ less than that of the FFD-based algorithm and $4.7\% - 17.0\%$ less than that of OCA for those test instances of TeraSort, and $34.0\% - 35.1\%$ less than that of HOMO, $15.1\% - 24.2\%$ less than that of the FFD-based algorithm and $17.1\% - 24.6\%$ less than that of OCA for those test instances of WordCount.



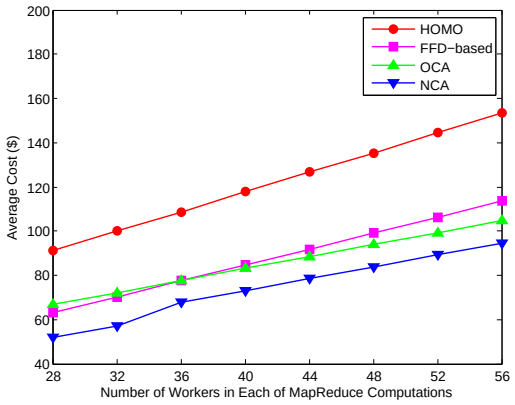
(a) TeraSort



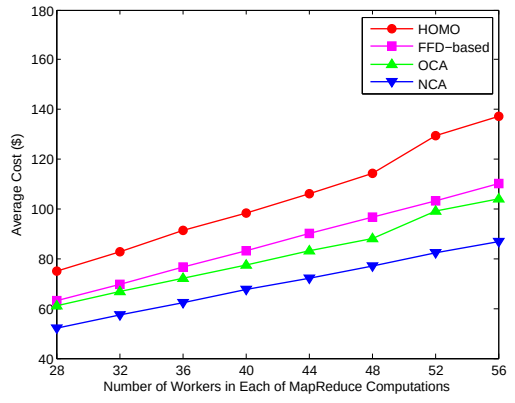
(b) WordCount

Fig. 2. The comparison of the four algorithms on the cost of using VMs when the number of MapReduce computations varied

Fig 2 (a) and (b) compare the costs of the MapReduce placement solutions generated by the four algorithms for TeraSort and WordCount, respectively, when the number of MapReduce computations varied from 9 to 16. In the experiments, the number of workers in each of the MapReduce computations was fixed at 40, the number of the existing VMs of each type was fixed at 10, and the remaining resource on each existing VM was 50% of the total resource. For the test instances of TeraSort, the cost of the MapReduce placement generated by NCA was $32.8\% - 39.1\%$ less than that of HOMO, $17.5\% - 27.0\%$ less than that of the FFD-based algorithm, and $13.2\% - 21.4\%$ less than that of OCA. For the test instances of WordCount, the cost of the MapReduce placement generated by NCA was $24.3\% - 44.0\%$ less than that of HOMO, $12.8\% - 31.9\%$ less than that of the FFD-based algorithm, and $6.2\% - 36.2\%$ less than that of OCA.



(a) TeraSort

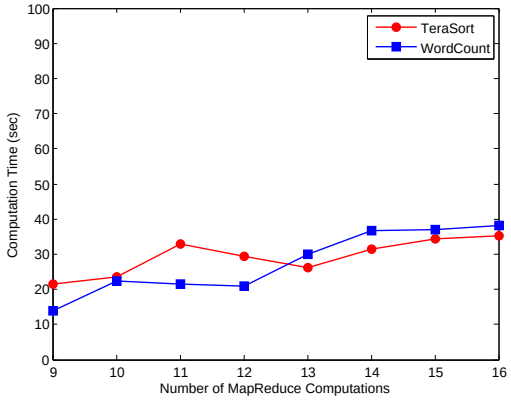


(b) WordCount

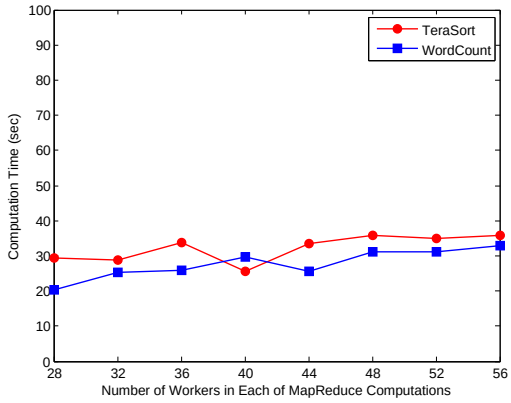
Fig. 3. The comparison of the four algorithms on the cost of using VMs when the number of workers in each of the MapReduce computations varied

Fig 3 (a) and (b) is the comparison of the cost of the MapReduce placement solutions generated by the four algorithms for TeraSort and WordCount, respectively, when the number of workers in each of the MapReduce computations varied from 28 to 56. In the experiments, the number of MapReduce computations was fixed at 12, the number of the existing VMs of each type was fixed at 10, and the remaining resource on each existing VM was 50% of the total resource. For the test instances of TeraSort, the cost of the MapReduce placement generated by NCA was $37.6\% - 43.1\%$ less than that of HOMO, $12.7\% - 18.4\%$ less than that of the FFD-based algorithm, and $10.8\% - 28.9\%$ less than that of OCA. For the test instances of WordCount, the cost of the MapReduce placement generated by NCA was $30.8\% - 36.8\%$ less than that of HOMO, $17.7\% - 21.2\%$ less than that of the FFD-based algorithm, and $12.6\% - 16.8\%$ less than that of OCA.

Fig. 4 displays the experiments on the scalability of NCA. In the experiments, the number of the existing VMs of each type was fixed at 10 and the remaining resource on each existing VM was 50% of the total resource.



(a)



(b)

Fig. 4. The scalability of NCA

Fig. 4 (a) shows how the computation times of NCA changed with the number of MapReduce computations when NCA was used to solve TeraSort and WordCount problems and Fig. 4 (b) displays how the computation times of NCA changed with the number of workers in each of the MapReduce computations when NCA was used to solve TeraSort and WordCount problems. It can be seen from Fig. 4 that the computation time of NCA increased linearly when the number of MapReduce computations increased, and that the computation time of NCA did not change significantly when the number of workers in each of the MapReduce computations varied.

In summary, NCA always had better performance than all the three baseline algorithms for all the tested problems. In addition, it was demonstrated the good scalability of NCA.

6 Conclusion and Future Work

This paper has proposed a new approach to the cloud-based heterogeneous MapReduce placement problem, and has evaluated the new approach by experiments. The experimental results have shown that the running cost of the cloud-based MapReduce computation platform using this new approach is $24.3\% - 44.0\%$ lower than that using the most popular homogeneous MapReduce placement approach, $12.7\% - 31.9\%$ lower than that using the FFD-based algorithm, and $2.0\% - 36.2\%$ lower than that using the heterogeneous MapReduce placement approach not considering the spare resources from the existing MapReduce computations. The experimental results have also demonstrated the good scalability of this new approach.

In this research it was assumed that all the workers of new cloud-based MapReduce computations must be placed on VMs and start processing map/reduce tasks immediately in order to make sure that the cloud-based MapReduce computations can be done before their deadlines. Thus, we did not consider the scheduling of the workers, or consecutive placement of the workers. It is conjectured that by considering consecutive placement, the utilization of VMs can be further improved and therefore the total running cost of the cloud-based MapReduce computation platform can be further reduced. In addition, it is suggested that problem semantics might be used to further improve the performance of the constructive algorithm [43]. Thus, in the future we will investigate how to use problem semantics in the constructive algorithm to further improve its performance.

Acknowledgments

This research was funded by the State Scholarship Fund of China Scholarships Council (CSC) and the CSC Top-Up Scholarship of Queensland University of Technology.

The authors would like to thank the associate editor and reviewers for their valuable comments and suggestions.

References

[1] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *Cloud Computing*, ser. Lecture Notes in Computer Science, M. Jaatun, G. Zhao, and C. Rong, Eds. Springer Berlin Heidelberg, 2009, vol. 5931, pp. 674–679.

[2] A. Matsunaga, M. Tsugawa, and J. Fortes, "Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications," in *Proc. IEEE 4th Int. Conf. eScience*, 2008, pp. 222–229.

[3] Q. He, T. Shang, F. Zhuang, and Z. Shi, "Parallel extreme learning machine for regression based on mapreduce," *Neurocomputing*, vol. 102, pp. 52 – 58, 2013.

[4] M. Husain, L. Khan, M. Kantarcioglu, and B. Thuraisingham, "Data intensive query processing for large rdf graphs using cloud computing tools," in *Proc. IEEE 3rd Int. Conf. Cloud Computing*, July 2010, pp. 1–10.

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments." in *OSDI*, vol. 8, no. 4, 2008, p. 7.

[6] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang, "Moon: Mapreduce on opportunistic environments," in *Proc. ACM 19th Int. Symposium on High Performance Distributed Computing*, 2010, pp. 95–106.

[7] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin, "FLEX: A slot allocation scheduling optimizer for MapReduce workloads," in *Middleware 2010*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6452, pp. 1–20.

[8] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of mapreduce programs," *Proc. Int. Conf. VLDB Endowment*, vol. 4, no. 11, pp. 1111–1122, 2011.
[9] C. Polo, Jord?and Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and A. Eduard, "Resource-aware adaptive scheduling for MapReduce clusters," in *Middleware 2011*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 7049, pp. 187–207.
[10] K. Wang, B. Tan, J. Shi, and B. Yang, "Automatic task slots assignment in hadoop MapReduce," in *Proc. 1st Workshop Architectures and Systems for Big Data*, ser. ASBD '11, New York, NY, USA, 2011, pp. 24–29.
[11] F. Tian and K. Chen, "Towards optimal resource provisioning for running MapReduce programs in public clouds," in *Proc. IEEE 4th Int. Conf. Cloud Computing*, 2011, pp. 155–162.
[12] M. AbdelBaky, H. Kim, I. Rodero, and M. Parashar, "Accelerating MapReduce analytics using CometCloud," in *Proc. IEEE 5th Int. Conf. Cloud Computing (CLOUD)*, 2012, pp. 447–454.
[13] E. Hwang and K. H. Kim, "Minimizing cost of virtual machines for deadline-constrained MapReduce applications in the cloud," in *Proc. ACM/IEEE 13th Int'l Conf. Grid Computing (GRID)*, 2012, pp. 130–138.
[14] P. Lama and X. Zhou, "Aroma: Automated resource allocation and configuration of MapReduce environment in the cloud," in *Proc. ACM 9th Int. Conf. Autonomic computing*, 2012, pp. 63–72.
[15] K. Chen, J. Powers, S. Guo, and F. Tian, "Cresp: Towards optimal resource provisioning for mapreduce computing in public clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1403–1412, June 2014.
[16] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics," in *Proc. ACM 2nd Symposium on Cloud Computing*, 2011, p. 18.
[17] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, "Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1737–1751, Dec 2012.
[18] B. Palanisamy, A. Singh, and L. Liu, "Cost-effective resource provisioning for mapreduce in a cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2014.
[19] X. Xu and M. Tang, "A more efficient and effective heuristic algorithm for the MapReduce placement problem in cloud computing," in *Proc. IEEE 7th Int. Conf. Cloud Computing*, 2014, to be published.
[20] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2009.
[21] K. Kc and K. Anyanwu, "Scheduling hadoop jobs to meet deadlines," in *Proc. IEEE 2nd Int. Conf. Cloud Computing Technology and Science (CloudCom)*, Nov 2010, pp. 388–392.
[22] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic resource inference and allocation for MapReduce environments," in *Proc. ACM 8th Int'l Conf. Autonomic Computing*, 2011, pp. 235–244.
[23] J. Polo, Y. Becerra, D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, "Deadline-based MapReduce workload management," *IEEE Trans. Network and Service Management*, vol. 10, no. 2, pp. 231–244, June 2013.
[24] A. Verma, P. Ahuja, and A. Neogi, "pmapper: power and migration cost aware application placement in virtualized systems," in *Middleware 2008*. Springer, 2008, pp. 243–264.
[25] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating heuristics for virtual machines consolidation," *Microsoft Research, MSR-TR-2011-9*, 2011.
[26] "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755 – 768, 2012, special Section: Energy efficiency in large-scale distributed systems.
[27] M. Monaci and P. Toth, "A set-covering-based heuristic approach for bin-packing problems," *INFORMS Journal on Computing*, vol. 18, no. 1, pp. 71–85, 2006.
[28] M. Haouari and M. Serairi, "Heuristics for the variable sized bin-packing problem," *Computers and Operations Research*, vol. 36, no. 10, pp. 2877–2884, 2009.
[29] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proc. 2008 Conf. Power aware computing and systems*, vol. 10. San Diego, California, 2008.
[30] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Locality-aware resource allocation for mapreduce in a cloud," in *Proc. 2011 Int. Conf. High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 58:1–58:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063462
[31] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE 2010 INFOCOM*, 2010, pp. 1–9.
[32] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Green Computing and Communications (GreenCom), Proc. 2010 IEEE/ACM Int. Conf. Cyber, Physical and Social Computing (CPSCom)*. IEEE, 2010, pp. 179–188.
[33] G. Wu, M. Tang, Y.-C. Tian, and W. Li, "Energy-efficient virtual machine placement in data centers by genetic algorithm," in *Neural Information Processing*. Springer, 2012, pp. 315–323.
[34] M. Tang and S. Pan, "A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers," *Neural Processing Letters*, pp. 1–11, 2014.
[35] H. Dyckhoff, "A typology of cutting and packing problems," *European Journal of Operational Research*, vol. 44, no. 2, pp. 145 – 159, 1990.
[36] L. A. Wolsey, *Mixed Integer Programming*. Wiley Encyclopedia of Computer Science and Engineering, 2008.
[37] J. Kang and S. Park, "Algorithms for the variable sized bin packing problem," *European Journal of Operational Research*, vol. 147, no. 2, pp. 365–372, 2003.
[38] VMware, "VMware homepage." [Online]. Available: http://www.vmware.com
[39] Hadoop, "Hadoop releases." [Online]. Available: http://hadoop.apache.org/releases.html
[40] Ganglia, "Ganglia monitoring system." [Online]. Available: http://ganglia.sourceforge.net/
[41] CPLEX, "IBM CPLEX Optimizer." [Online]. Available: http://ibm.com/software/commerce/optimization/cplex-optimizer/
[42] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the MapReduce-based data analysis," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE, 2010, pp. 41–51.
[43] C. Bellettini, M. Camilli, L. Capra, and M. Monga, "Distributed ctl model checking in the cloud," *arXiv preprint arXiv:1310.6670*, 2013.

Xiaoyong Xu received a Master degree in Management Science and Engineering from Nanjing Tech. University in 2012. He is currently a PhD student at the School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane, Australia. His current research interests include cloud computing and big data.

Maolin Tang (M'04—SM'10) received a B.E. degree from the Huazhong University of Science and Technology, Wuhan, China, a M.E. degree from Chongqing University, Chongqing, China, both in computer science, and a Ph.D. degree in computer systems engineering from Edith Cowan University, Perth, Australia.

He is a Senior Lecturer with the School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane, QLD, Australia. His current research interests include evolutionary computation and cloud computing, in particular applications of evolutionary computation in cloud computing. He has published over 80 refereed papers in prestigious journals and international conference proceedings, including IEEE Transactions on System, Man and Cybernetics – Part B, IEEE Transactions on Cybernetics, IEEE Transactions on Intelligent Transportation Systems, and Future Generation Computer Systems.

Dr. Tang has been a Program Committee Member of a number of international conferences, and has co-chaired an international workshop and two special sessions in international conferences.