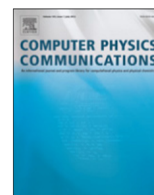
Computer Physics Communications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc



MaMR: High-performance MapReduce programming model for material cloud applications

Weipeng Jing [a], Danyu Tong [a], Yangang Wang [b,*], Jingyuan Wang [c], Yaqiu Liu [a], Peng Zhao [a]

[a] College of Information and Computer Engineering, Northeast Forestry University, Harbin, China
[b] Computer Network Information Center, Chinese Academy of Sciences, Beijing, China
[c] School of Computer Science and Engineering, Beihang University, Beijing, China

A R T I C L E I N F O

Article history:
Received 20 February 2016
Received in revised form
26 May 2016
Accepted 2 July 2016
Available online xxxx

Keywords:
Materials
Programming model
MapReduce
BSP
Merge phase

A B S T R A C T

With the increasing data size in materials science, existing programming models no longer satisfy the application requirements. MapReduce is a programming model that enables the easy development of scalable parallel applications to process big data on cloud computing systems. However, this model does not directly support the processing of multiple related data, and the processing performance does not reflect the advantages of cloud computing. To enhance the capability of workflow applications in material data processing, we defined a programming model for material cloud applications that supports multiple different Map and Reduce functions running concurrently based on hybrid share-memory BSP called MaMR. An optimized data sharing strategy to supply the shared data to the different Map and Reduce stages was also designed. We added a new merge phase to MapReduce that can efficiently merge data from the map and reduce modules. Experiments showed that the model and framework present effective performance improvements compared to previous work.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

In recent years, a large number of known and hypothetical materials have been studied, such as batteries, catalysts, and the stable structures of solid materials, so the amount of calculated data increases exponentially with time [1]. Thus, big data presents a huge challenge to the computing disciplines.

To improve computing speed, a large number of computing tasks in materials science must be moved from traditional High-Performance Computing (HPC) to High-Throughput Computing (HTC) and Many-Task Computing (MTC) platforms based on big data, such as the widely used cloud computing systems [2] and grid computing [3].

As is known, cloud computing, which should provide different layers of service to users, is constituted by large-scale distributed computers and various resources such as CPUs and storage. Meanwhile, different types of services are also offered, such as software [4]. In cloud computing systems, multiple virtual machines (VMs) are run on a single physical computer, for a homogeneous result [5].

It is more convenient to develop and deploy applications through the cloud computing platform, Therefore, in this paper, we adopt this approach to process large amounts of material data. We use cloud computing's powerful computation and storage capacity to effectively solve the problem of large-scale material data in the process of analytical calculations. To fit the material high-performance computing needs of materials science, different models have been proposed to maximize the performance. Meanwhile, to provide greater flexibility and higher parallel efficiency, the challenges to the programming model should be faced [6]. The MapReduce programming model has been widely used in large-scale and data-intensive applications, such as Google and Amazon [7,8]. The other most successful programming model is Microsoft's Dryad [9]. Yahoo also has similar infrastructures. Hadoop is an open-source implementation of MapReduce, and it has already been applied to various fields because of the reliability and scalability of the parallel programming framework in the MapReduce model [10]. In Hadoop and MapReduce, the input data are split into chunks of size 64 M, and each task is allocated to a VM. Therefore, we can take the computation nodes as the local modes.

However, traditional programming models cannot adapt to material data calculations, and Hadoop's frequent reading and

* Corresponding author.
E-mail addresses: weipeng.jing@outlook.com (W. Jing), wangyg@sccas.cn (Y. Wang).

http://dx.doi.org/10.1016/j.cpc.2016.07.015
0010-4655/© 2016 Elsevier B.V. All rights reserved.

Please cite this article in press as: W. Jing, et al., MaMR: High-performance MapReduce programming model for material cloud applications, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.07.015

writing become the bottleneck of the cloud system. First, in a cloud computing system, many computing jobs are running in a single physical computer because the data nodes in Hadoop are deployed in virtual machines. How to avoid I/O resource competition and reduce I/O overhead is a big problem in the programming models for cloud computing.

Second, the MapReduce model can handle multiple applications, but a large amount of data is repeatedly read and written. Another application might not be able to handle the original data effectively. Thus, the problem becomes even worse when the above two problems must be faced in material cloud computing applications.

Finally, the MapReduce framework does not directly support iterative data analysis applications. Instead, we must complete iterative programs by manually issuing multiple MapReduce jobs [11].

In a cloud computing system, resources are used based on payment. Therefore, it is important to develop a new programming model to support multiple iterative tasks. However, because the data are split into several chunks, we must first improve the parallelization in Hadoop. Meanwhile, we must develop a realistic model to adapt to material high performance computing.

Thus, research on related programming model issues has been regarded as one of the most important research areas for MapReduce [6,12,13]. The Bulk Synchronous Parallel (BSP) is used for parallel bridges in programming language, and the BSP model has been widely used in different research fields [12]. However, this programming model does not solve the problems of a heterogeneous distributed system [11]. More importantly, there are no improvements for the BSP model for use in cloud computing systems because cloud computing systems with many virtual CPUs provide users with virtual machines to use.

Though sufficiently generic to perform many tasks, the MapReduce framework is best at handling homogeneous datasets. However, there are some modified MapReduce models. HaLoop [14] is a modified programming model of the Hadoop MapReduce framework that is proposed to serve data mining, web ranking, and graph analysis applications. HaLoop not only extends MapReduce with programming support for iterative applications but also dramatically improves the efficiency by making the task scheduler loop-aware and adding various caching mechanisms. Map-Reduce-Merge [15] adds a Merge phase that can efficiently merge data already partitioned and sorted (or hashed) by the map and reduce modules. In this paper, we focus on material data sharing on the MapReduce stage and on improving the task scheduling performance.

At the same time, some other research efforts addressed the performance in data streaming environments. Such as Nova [16], Pig Latin [17], S4 [18] can deals with continuous arrival of streaming data.

To improve the capability of data streaming applications in cloud computing systems, the programming model called CloudFlow [19] is proposed. CloudFlow is built on top of MapReduce and is highly suitable for handling shared data in cloud computing systems. With this inspiration, we proposed a streaming data programming model based on MapReduce for materials science high-performance computing and data processing.

The above observation motivates the design of MaMR as a new programming model for materials science applications, which is designed for performance predictability and data sharing to improve its availability while retaining the simplicity of traditional MapReduce. We show that the MaMR model can obtain good speedups on cloud computing systems; meanwhile, it has good performance predictability. MaMR offers the following advantages:

(1) We defined a programming model for materials science cloud applications. This model includes multiple Map and Reduce functions running concurrently. MaMR uses a hybrid shared-memory BSP model to improve parallel efficiency, allowing full use of the VMs in cloud computing systems.

(2) We designed an optimized data sharing strategy that supplies the shared data to different Map and Reduce stages. Meanwhile, we further provide multi-copies of the output to reduce the shuffle overhead.

(3) We add a new Merge phase to Map-Reduce that can efficiently merge data already partitioned and sorted (or hashed) by the Map and Reduce modules.

(4) We conduct a theoretical analysis of implementing the MaMR framework, and the experimental results show that the proposed model and framework can effectively improve the performance compared to previous work.

The rest of the paper is organized as follows: Section 2 introduces MapReduce and gives its design principles and features. We describe the MaMR framework for materials science cloud applications in Section 3. Multi-copies of the output to the Reduce stage are applied to enhance this programming model, which is described in Section 4. In Section 5, we describe the addition of a new merge phase to MapReduce. The experimental results are presented in Section 6. Finally, we summarize our technical contributions and describe our plans for future work.

2. Mapreduce

The Hadoop Map-Reduce programming model and its underlying Hadoop File System (HDFS) [20] are designed to support search engines, as reflected in their parallel processing speed and convenient features. According to [21], this model has also been heavily applied within data-intensive applications such as machine learning.

A MapReduce program consists of two primitives, Map and Reduce. The Map function is applied to an individual input record from HDFS to compute a set of intermediate key/value pairs. For each key, Reduce works on the list of all values with this key. The conclusion is also written to the HDFS node. An overview is shown in Fig. 1. In contrast to the traditional parallel processing model, MapReduce has good flexibility and fault tolerance. The features and principles are described below.

- **A**. *High performance in data processing*
  The MapReduce model can effectively improve the efficiency and achieve high parallelism along with high performance simply by deploying a large number of VMs. More importantly, the MapReduce model can provide a simple framework to enable automatic data processing by users. A world-record sorting experiment has shown that MapReduce can obtain high performance when used on thousands of VMs [22].
- **B**. *Simplified*
  In the MapReduce programming model, the Developers can focus on task processing using the MapReduce interface, without worrying about such issues as implementing memory management, programming synchronization, and network delay. We take only the map and reduce the interfaces to complete our data processing.
- **C**. *Distributed Sorting Framework*: MapReduce is essentially a 2-phase parallel sorter similar to the one in NOW [23]. These phases include a practitioner function that partitions mapper outputs to reducer inputs, a sort-by-key function that sorts reducer inputs based on keys, and a group-by-key function that groups sorted key/value pairs with the same key into a single key/value pair incorporating the shared key and all of the values.

Please cite this article in press as: W. Jing, et al., MaMR: High-performance MapReduce programming model for material cloud applications, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.07.015

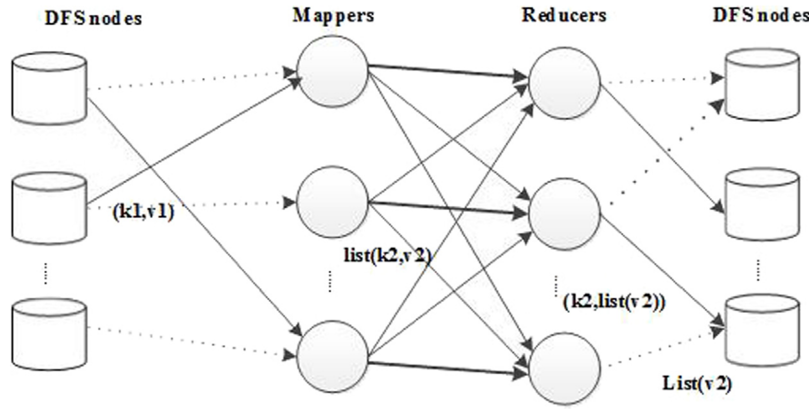


Fig. 1. MapReduce overview.

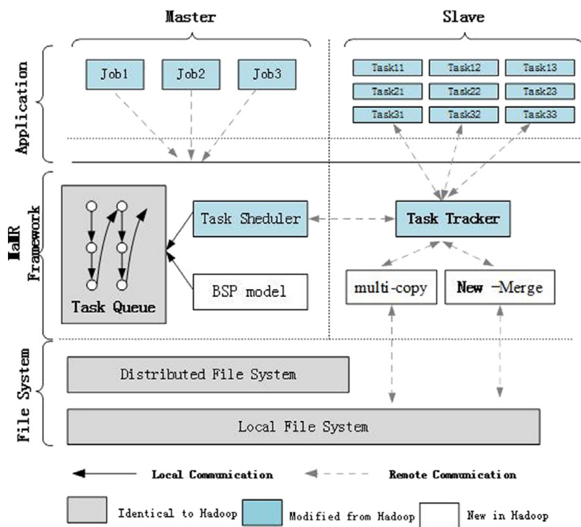


Fig. 2. Architecture of the MaMR framework.

3. MaMR framework

In this paper, we have proposed an improved MapReduce programming model. Fig. 2 illustrates the architecture of the MaMR framework that can adapt to analysing large material data. There are three layers, the application, MaMR framework, and file system.

- **A.** *Architecture*

  In the MaMR framework, the material application job is sorted into a task queue. The master takes the highest task to schedule from the queue. The BSP model is used to predict the VM performance in a cloud computing system. The task schedulers include multi-copy and merge stages. In the application layer, the job is decomposed into a set of Map functions and Reduce functions.

  In this paper, we consider only the case of material analysis. The task is divided into some uncorrelated independent subtasks for scheduling, and these tasks will be assigned to m VMs to perform. The cloud computing system is modelled by a set of m VMs, where n is the number of material computing tasks, $(m < n)$. For given $T(n) = \{t_1, t_2, \dots, t_n\}$ $n \in N$, define a task set. Now, $t_i (i = 1, 2, \dots, n)$ is the No. i task in the set $T(n)$, $t_i = (t^i_{id}, t^i_{mi}, t^i_{file}, t^i_{fee}, t^i_{deadline}, t^i_{memory}, t^i_{submit})$, where t^i_{id} is the unique identification number of t_i; t^i_{mi} is the size of t_i, namely the number of millions of instructions (MI); t^i_{file} is the program file size of t_i; t^i_{file} is the *user′s* desired fee for t_i, which the user pays based on the *task′s* QoS requirements; $t^i_{deadline}$ is the *user′s* desired deadline for t_i; $t^i_{deadline}$ is the memory requirement of t_i; and t^i_{submit} is the submission time of t_i.

  The priority of a task reflects its importance, and in this paper, we consider the fairness and efficiency for the computing users. We give full consideration to the cost of execution and deadlines of tasks, and the task value density and the urgency of the task execution are designed as constraints. Dynamic priority is proposed to achieve high performance.

  Then, as the first step, the value density of task (VDT) is defined as follows:

$$TVD_i = \frac{t^i_{fee}}{t^i_{mi}}. \tag{1}$$

Here, VDT_i indicates the ratio of the *user′s* desired fee to the *task′s* size.

Let t^i_wait be the waiting time of a task and t^i_left the time remaining. Then, the urgency of task execution time (UTET) is as follows:

$$UTE_i = \frac{t^i_{wait}}{1 + t^i_{left}}. \tag{2}$$

Obviously, with increasing waiting time, the remaining time will decrease accordingly, while UTET will increase rapidly, which will satisfy the time constraints of tasks and improve the successful completion ratio of tasks completed before the deadline, thus reflecting the dynamic characteristics of priority.

To establish the dynamic priority, we need to normalize VDT_i and $UTET_i$. For this purpose, we have proposed the Z-score method, based on the mean of the original data and standard deviation, which is used in data normalization.

$$st_{ik} = (z_{ik} - \overline{z_k})/\delta_k. \tag{3}$$

Now, we define some new parameters. Let st_{ik} be the normalization result of the No. k priority factor of t_i, Where δ_k is the standard deviation of the No. k priority factor, calculated by $\delta_k = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(z_{ik} - \overline{z_k})}$, $(k = 1, 2)$. We define st_{ik} as the element in the $n \times 2$ order matrix Z, denoted as $Z = \{TVD \quad UTE\}$, while $\overline{z_k}$ is the average of the No. k column of matrix Z, which is calculated by $\overline{z_k} = \frac{1}{n}\sum_{i=1}^{n} z_{ik}$.

Let $DP(t_i)$ denote the dynamic priority of t_i, as defined in Eq. (4):

$$DP(t_i) = \omega_1 \times st_{i1} + \omega_2 \times st_{i2} \tag{4}$$

where $\omega_1, \omega_2 \in [0, 1]$ are the weighting factors and satisfy the equation $\omega_1 + \omega_2 = 1$.

Fig. 2 shows the use of the distributed file system HDFS on top of the local file system in Hadoop. In the MaMR model, the data

Please cite this article in press as: W. Jing, et al., MaMR: High-performance MapReduce programming model for material cloud applications, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.07.015

are saved to the distributed file system in the VMs. MapReduce processes the input data and reads/writes the data from/to the HDFS.

In MaMR, we define multiple heterogeneous Map and Reduce functions to analyse the material computing model. We formulate the representation as follows:

Map: $(k_m, v_m) \rightarrow \{(k'_m, v'_m)\}, m \in [1, M]$
Reduce: $\{\{(k'_{(i,r)}, [v'_{(i,r)}])\}, \ldots, \{(k'_{(i,r)}, [v'_{(i,r)}])\}\}(i, r), \ldots, (j, r) \in [1, M], r \in [1, R]$.

In this model, there are M Map Functions and R Reduce Functions to perform. The Reduce Functions are different from the traditional MapReduce programming model in that they take as input several key/value pairs from multiple key–value pair outputs of the Map Functions. We define (i, r) or (j, r) as a mapping from a tuple to a number between 1 and M, where r represents the index of the Reduce Function. The symbol pairs (i, r) and (j, r) represent the first and last Map Function that feed the r th Reduce Function.

- **B**. *BSP model*

We propose the BSP model to attain proper speedups on cloud computing systems. The BSP model includes three interdependent components: the processor/memory pair, network point-to-point communication, and the synchronization barrier.

Thus, in the BSP model, each step consists of a sequence of super-steps divided into three ordered phases. We define three parameters, p, g, and l, for measurement. The number of processor/memory pairs is given by p; and g is defined as the network throughput rate; and l is the global synchronization time. We define the cost as follows:

$$T = \sum_{s=1}^{S} \max_{1 \le i \le p} \omega_i^s + \sum_{s=1}^{S} \max_{1 \le i \le p} h_i^s \times g + l \times s \quad (5)$$

where the computing time of processor i is given by ω_i; h_i is the data needed to be communicated and s is the total number of super-steps.

However, we can see the original BSP Library, such as BSPlib. In BSPlib, we only used a single CPU node in computing. Therefore the model cannot be used directly in Hadoop because the virtual machines run in a physical machine and are not provided to users directly.

To improve the parallel efficiency, a hybrid of distributed-memory BSP is used in MapReduce. Each computing node in the VMs can work in parallel and communicate through a network or IO system. Using the BSP model, the communication data will be significantly reduced.

In every VM, the shared-memory BSP model employs the bulk of threads to accomplish task execution. Therefore, we use this BSP memory model to pass messages, and the communication messages are incorporated into a class BSPMP. A BulkTracker mechanism is designed: we use the send() function to transmit data and the bspRemove() function to remove messages, as implied by the java socket.

In BulkTracker, we take a thread as a server socket. When communication occurs, the thread begin to listen for channel communication. If there are data transmissions, the socket creates an Http connection and sends the data. When the data are received, they are allowed into the queue and can be obtained by the BulkTracker in the next super-step.

In the MaMR programming model, the data node can use a shared-memory model to finish computing tasks. In each VM, every thread uses the memory model to avoid the I/O competition. However, the synchronization phase will use a hierarchical approach, which will reduce the synchronization time. Therefore, we can see that MaMR achieves good speed speedup and scalability.

4. Multi-copy strategy

In this section, we execute a multi-copy strategy. For example, we begin with a simple example of processing wood molecules, which will be continued into the subsequent sections, to show how the Map, Reduce, and Merge modules work together. There are two datasets in this example: the wood molecules and the wood structure. The wood molecule's key attribute is the molecule id (mol-id), and the others are packed into a mol info value. The wood structure's key is the structure id (stru-id), and the others are packed into a stru info value. One example query is to join these two datasets and compute the numbers of molecules.

Before these two datasets are joined by a merger, a pair of mappers and reducers first handle the datasets. We can see the complete data flow in Fig. 3. On the left hand side, a mapper reads the molecule and the numbers for each molecule. A reducer then sums up these mol-numbers for every molecule and sorts them by mol-id. On the right hand side, a mapper reads the wood structure and structure numbers. A reducer then sorts this stru-id. Finally, a merger matches the output records from the two reducers by mol-id using the sort-merge algorithm, applying a department-based approach.

When the two Map-Reduce tasks are finished, a merger task places their intermediate outputs in BSP memory, and thus, we will describe the details of the major merge components in the subsequent sections.

In the Definition and Terms (1)–(10) below, we list the major contributions of the multi-copy strategy we utilize, followed by proving the correctness and uniqueness of the output key/value pair.

(1) Let $\{Map\} = \{Map_0, \ldots, Map_{M-1}\}$ be the set of M Map functions.
(2) Let $\{Re\} = \{Re_0, \ldots, Re_{R-1}\}$ be the set of R Reduce functions.
(3) Let $Out - in\{Map_m, Re_r\} = 1$ be the bipartite graph that represents the dependency (or links) between (the output of) the Map Functions and (the input of) the Reduce Functions.
(4) Relevant Map Functions for Map Function Map_m associated with Reduce Function Re_r.

$$\forall Map_i \in list\{Map_m, Re_r\}, Out - in\{Map_m, Re_r\} : \{Map_i\} = 1 \wedge Out - in\{Map_m, Re_r\} : \{Map_i\} = 1$$
$$\forall Map_i \notin \{Map\} - \{Map_i\}, Out - in\{Map_m, Re_r\} = 0.$$

(5) All relevant Map Functions for Map Function

$$list(Map_m) : \{Map_i\}, \exists Re_r \in S_R,$$
$$Out - in(Map_m, Re_r) = 1 Out - in(Map_i, Re_m) = 1.$$

(6) Irrelevant Map Functions for Map Function Map_m associated with Reduce function Re_r.

$$no - list(Map_m, Re_r) : \{Map_i\}, \{Map\} - list(Map_m, Re_r) - Map_m.$$

(7) Irrelevant Map Functions for Map Function m:

$$no - list(Map_m) : \{Map_i\}, \{Map\} - list(Map_m) - Map_m.$$

(8) Let (k_m, v_m) be a key and value pair of the output of map function Map_m.
(9) The shuffling space is M-dimension discrete space. The length of dimension j is u_j.
(10) f is the function that maps a key–value pair to a set of M tuples.

$$f : (k_m, v_m) \rightarrow \{(i_1, \ldots, i_M)\}, i_j \in \{0, 1, \ldots, u_j - 1\}$$

u_j is the number of Reduce units in dimension j of the shuffling space. The set of M-tuples is determined as follows:

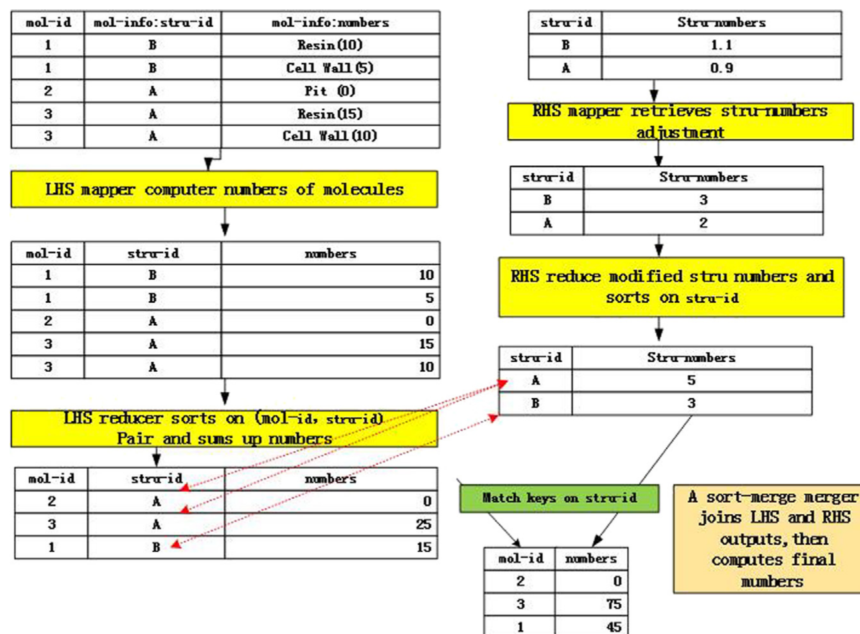


Fig. 3. Model example.

$$i_m = h(k_m)\%u_m$$
$$i_j = \text{all of } \{0, 1, \ldots, u_j - 1\} \text{ if } Map_j \in list(Map_m, Re_r)$$
$$i_j = g_j \text{ if } Map_j \in list(Map_m, Re_r).$$

g is another hash function that takes a dimension index and returns a value in $\{0, 1, \ldots, u_j - 1\}$. We need to prove the following: Suppose $Map_m \in \{Map_m\}, \forall \mathrm{Re}_r \in \{\mathrm{Re}_r\}, list(Map_m, \mathrm{Re}_r) = \{Map_0, Map_1, \ldots, Map_{m'}\}$.

Then $f(k_m, v_m) \cap f(k_0, v_0) \cap f(k_1, v_1) \cap \cdots f(k_{m'}, v_{m'}) \cap = 1$.

Lemma 1. *If $list(Map_m, \mathrm{Re}_r) = \{Map_0, Map_1, \ldots, Map_{m'}\}$ then*

$$list(Map_0, \mathrm{Re}_r) = \{Map_m, Map_1, \ldots, Map_{m'}\}$$
$$list(Map_1, \mathrm{Re}_r) = \{Map_m, Map_1, \ldots, Map_{m'}\},$$
$$\vdots$$
$$list(Map_{m'}, \mathrm{Re}_r) = \{Map_m, Map_1, \ldots, Map_{m'}\}.$$

Proof. (1) For (k_t, v_t), by definition of the f function, it can be shuffled to a Cell Set, denoted by

$$S_t = h(k_t)\%u_t$$
$$\{S_j\} = \{0, 1, \ldots, u_j - 1\}$$
$$S_j = g(j)$$
$$ifMap_j \in no - list\{Map_t\}$$
$$t \in \{m, 0, 1, \ldots, (m')\}.$$

As concluded above, $(i_1, \ldots, i_{M-1}) \in CS$. Thus, the proof is complete.

5. Map-Reduce-Merge

The Map-Reduce-Merge extension [15] supports various join predicates, but it requires fundamental changes to MapReduce and how it is used. It not only adds a new Merge phase but also requires the user to write code that is explicitly aware of the distributed nature of the implementation.

To describe how the MaMR programming model processes wood molecule data relationships, PC-Hschema [24] and its No. 2 query [25] are used.

We take nested query as the inquire method, and at the end, the material data must be ordered by different parallel computing node. If the condition for the 5-way join can fit the data order, so the nested query is only method to obtain the minimum supply cost. There are 5-way join in Algorithm 1, because it is essentially the same as the outer join, its logic can be processed during executing the outer one. In MaMR model, we use four 2-way joins for the overall 5-way join. The join tree of this execution plan is shown in Fig. 4. We use the SQL query to implement the Reduce-Merge framework. There are region and nation data in this stage, the parallel join is not required in Map, but the Reduce and Merge is must exist. In fact, share-memory can be read to the queue.

In the MaMR Reduce stage, there are five tables, *p_ps*, *r_c*, *r_c_p*, *M_ps_r_c_p*, *F_ps_r_c_p*. Fig. 4 shows the region and nation joined into *n_r*, and different *n_r* are joined by *r_c_p*.

In the join tree, part and partsupp are joined into a temporary table called *p_ps*. In parallel, Table *r_c* is then joined with the supplier into *r_c_p*. Later, *p_ps* and *r_c_p* are joined into *M_ps_r_c_p*. Once these four 2-way joins are performed to develop the overall 5-way join, *F_ps_r_c_p* is processed by two Map-Reduce tasks. The final Map-Reduce task is simply a sorter for the order by clause.

In Fig. 4, we replace the join with a different Map, and the Merge stage is simpler than a MapReduce model. The example shown in Fig. 4 includes 4 mappers, 6 reducers, and 4 mergers.

Algorithm 1 Map-Reduce-Merge:

```
1   Partition Selector PS;
2   Left Processor LP;
3   Right Processor RP;
4   Merger merger;
5   Iterator Manager IM;
6   int merger Number;
7   int  left Reducer Numbers;
8   int right Reduce Numbers;
9   select and filter left and right reducer outputs for this merger
10  PS.select(merger Number, left Reducer Numbers, right Reduce Numbers);
11  Configurable Iterator left = initiated parameter
12  Reduce outputs by left Reducer Numbers*/
13  Configurable Iterator right = initiated parameter
14  Reduce outputs by right Reducer Numbers*/
15  while(material parameter=true)
```

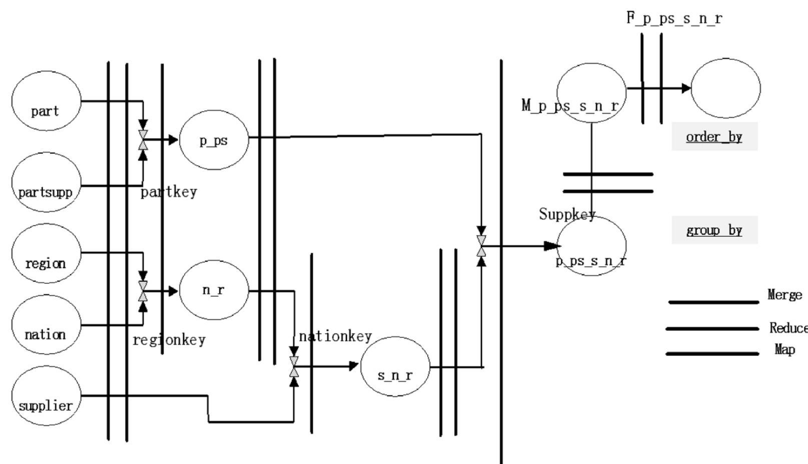


Fig. 4. Join tree for TPC-H Query.

```
16   pair<Map,Reduce>=Order(Map(left), Map(right));
17   if (first.array[Map] && second.array[Map])
18   break
19   end if
20   if ( first .array[Map])
21   LP.process(left value, left Map);
22   end if
23   if (second.array[Map])
24   RP .process(right value, right Map);
25   end if
26   if (more. array[Map] && more. array[Map])
27   merger. (left value, left Map,right value, right Map)
28   end if
29   pair<Map ,Reduce> =Order(Map(left), Map(right));
30   if (Numbers.first)
31   left++;
32   end if
33   if (Numbers.second)
34   right++;
35   end if
36   end while
```

6. Experiment

We set up Hadoop on four I450-G10 tower-style servers made by Sugon (InterXeon E5-2670 8-core CPU 2.2 GHZ 8 GB RAM). To make full use of these servers, we virtualize them into 16 VMs (hosts) using XenServer6.2. The hosts are the slaves of the cluster, while a PC (HP Compaq dx 2308) is the master of the cluster. It is built up on the base of Cenos6.4 final (kernel 2.6.32) using Hadoop 2.3, and the total capacity of HDFS is 3.34 T. We use Timber Data, data for specimens of 30 varieties of timber, for the air-dried density modulus.

- **A**. *performance predictability of MaMR* MaMR has the advantage of performance predictability. We take advantage of the communication efficiency between different data nodes, and the synchronization latency is ignored. We can thus easily know the cost of the application. We design a novel benchmarking program.
  We evaluate the cost of the MaMR computing time and theoretical time. The matrix multiply data are set by Timber Data. This experiment tests the shared-memory model and the BSP model, using 2 nodes, 4 nodes, and 8 nodes to evaluate the computing time.
  Fig. 5 shows the results, in which the MaMR model does not lead to accurate predictions of the execution time. The discrepancy is determined to occur for the following reasons. First, on the MapReduce model, the parallel parameters are average values, but in the MaMR model, the parameters are different at every step. Second, while there is communication between different data nodes in the IPC socket, the material data application testing does not consider the connection or disconnection time. Therefore, the communication is lower than the theoretical value. Finally, the shared-memory model requires a complex scheduling system, while MapReduce has a prior scheduling algorithm, allowing the BSP model to achieve good performance.
- **B**. *Speedup and scalability of MaMR*
  We also evaluate the speedup and scalability of MaMR. We first fix the size of the data set, applying all data nodes. We compare MaMR with Hadoop MapReduce and MPI. Fig. 6(a) shows that MaMR has a higher speedup than the MapReduce and MPI programming models because MaMR takes advantage of the BSP model and the flexible Map and Reduce stages. Fig. 6(b) shows that the MaMR model is substantially superior, especially as the data sets increase in size.
- **C**. *Effectiveness of MaMR*
  In Fig. 7, we present two benchmark applications that use timber data as input data. Fig. 7((a) (b)) shows the different input data sizes for the timber material benchmark. The execution time shown, which is small to moderate, includes the shared data copy time for MaMR.
  Importantly, in both applications, the execution time results demonstrate a clear advantage of MaMR over MapReduce. In the one-job case, regardless of how large the cloud computing system is, we can see the advantage of MaMR because MaMR can utilize more parallel jobs for execution. We can also see the performance difference in the data nodes.
  Fig. 7(a) shows the job performance of the traditional MapReduce, from 1 to 3. MaMR has a lower execution time, taking 500 s to run three concurrent jobs, while MapReduce, conversely, requires 800 s. Similar results also apply to the same configuration in Fig. 7(b). MR requires 6000 s for the three jobs on four VMs at moderate data size, while MaMR takes 1500 s instead. We can see that as the amount of data increases, the performance advantages of MaMR become more obvious. This benefit is better illustrated for larger input data sizes. In Fig. 7(a) and (b), the execution time of MR for the three-job case is very close across different cloud computing system sizes.
- **D**. *Performance of MaMR*
  The coarse histograms (1–100 buckets) show that the max-reducer-input data can be analysed in relation to the timber data, which was not the case for finer-grained histograms. In

Please cite this article in press as: W. Jing, et al., MaMR: High-performance MapReduce programming model for material cloud applications, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.07.015

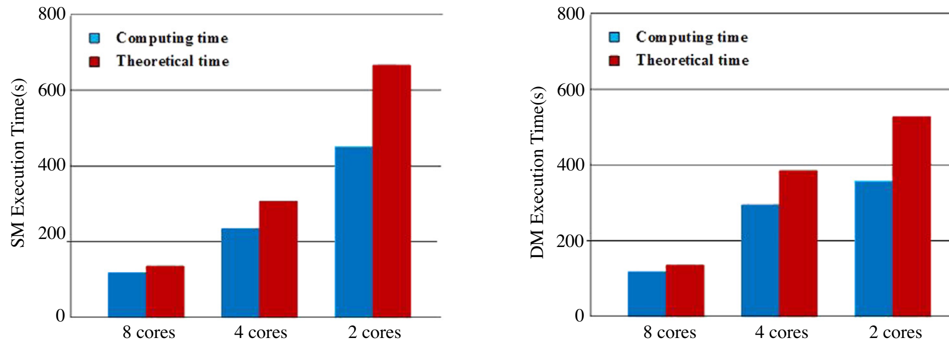
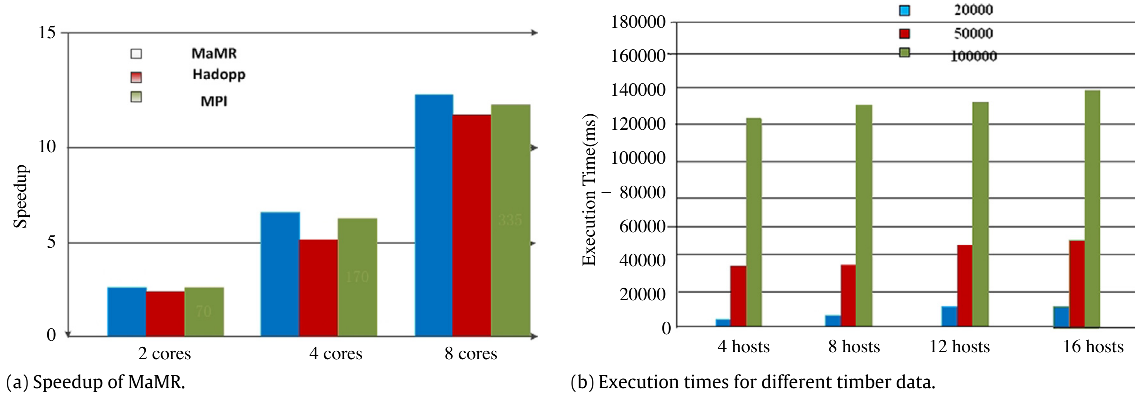


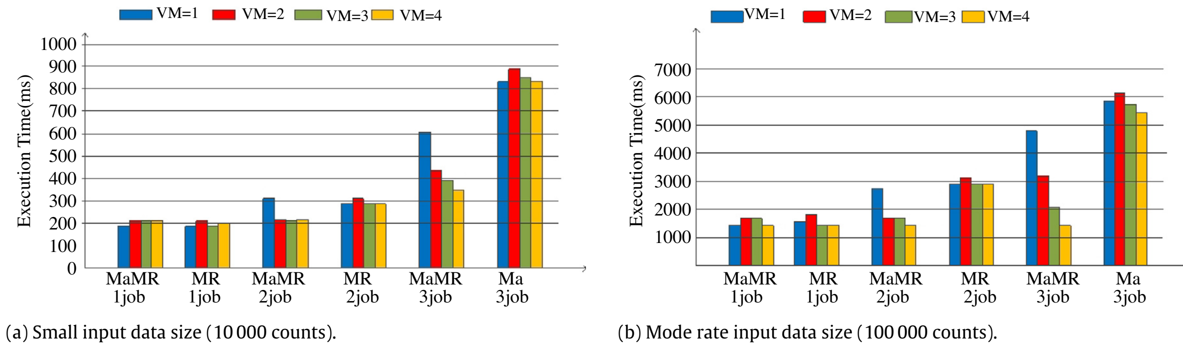
Fig. 5. Time cost using shared-memory and BSP memory approaches.



(a) Speedup of MaMR.

(b) Execution times for different timber data.

Fig. 6. Speedup and scalability.



(a) Small input data size (10 000 counts).

(b) Mode rate input data size (100 000 counts).

Fig. 7. Execution time (seconds) of MaMR running on timer data benchmark.

this simulation, the size of the buckets reflects the output data nodes. Fig. 8 shows that the max-reducer execution time is substantially reduced compared to that for MapReduce. We can see that the high reduce stage with varying histogram granularity is higher than the rest. Based on the different job execution times, at least 30 reducers were used in the simulation. We can see that the MaMR model achieved its goal of balancing input timber data.

Fig. 9 shows that the max-reducer-input steps for the MaMR programming model conform to the application of the timber data. In this simulation, we take 1 bucket data point to correspond to a different Reduce stage. It is clear that despite the high Reduce stage, MaMR is better balanced than MapReduce with more coarse-grained histograms. Fig. 9 shows that the input-size-dominated joins of the MapReduce job completion time track the max-reducer-input trend almost perfectly.

Fig. 10 shows that MaMR does not include a pre-processing stage. In the simulation, we take the average run times of 10 executions and record the results. The run time of the job was less than 15.12%. The MapReduce job performance is no worse than that of MaMR, but based on finding the approximate quintiles and then counting the number of records per quintile range, MaMR does have the best Reduce stage time.

- **E.** *Output-size of MaMR*

Fig. 11 shows that the max-reducer-output steps for the MaMR programming model are consistent with the timber data application. Because of the high Reduce and Merge stage, the MaMR model achieves almost perfect output balancing compared to the MapReduce model. Because, in the timber data sets, we use a random size data to evaluate the performance, the Map-Reduce-Merge framework from the MaMR model can handle more imbalance for high Reduce stages than in the previous experiment.

Because the number of output tuples in a region is difficult to estimate, the high Reduce stage must use the Map-Reduce-Merge framework to handle this problem in the MaMR programming model. A low Reduce stage, as in the traditional

Please cite this article in press as: W. Jing, et al., MaMR: High-performance MapReduce programming model for material cloud applications, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.07.015

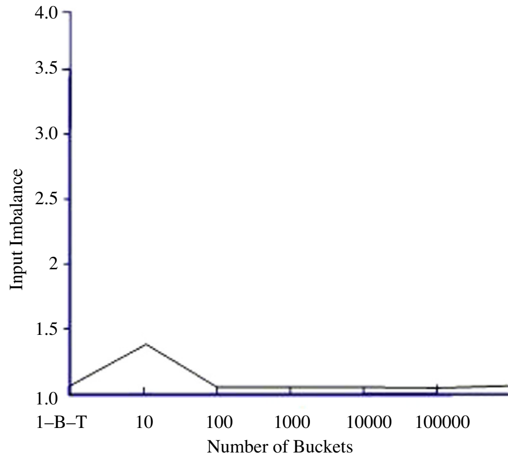


Fig. 8. Input imbalance.

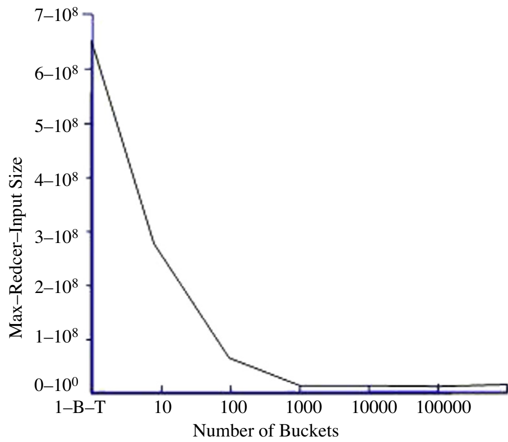


Fig. 9. Max-reducer-input.

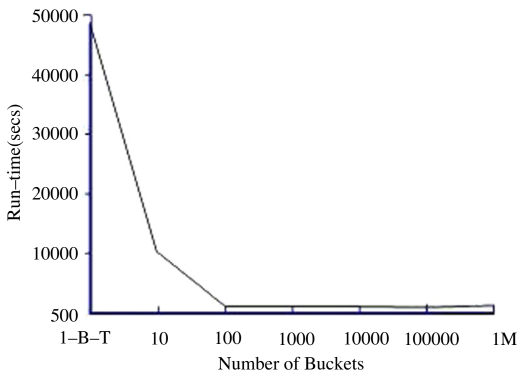


Fig. 10. Run-time by number of buckets.

MapReduce, cannot do this because a tuple can only be assigned to the appropriate histogram bucket, not any random bucket. We can see from Fig. 12 that with increasing number of buckets, the conclusion takes exactly the same form as in Fig. 11. Fig. 13 shows that the job output imbalance for the MaMR programming model has the clear advantage. Additionally, because the join is output-size dominated, the minimizing max-reducer-output size is not considered. The maximum standard deviation between the job completion times was 3.56%. The input duplication rates for histograms with 1, 10, 100, 1000, and 5951 buckets are 7.5, 4.2, 1.5, 1.0, and 1.1, respectively.

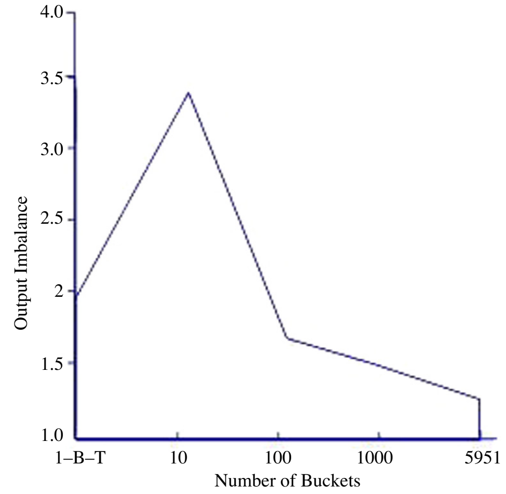


Fig. 11. Output imbalance by number of buckets.

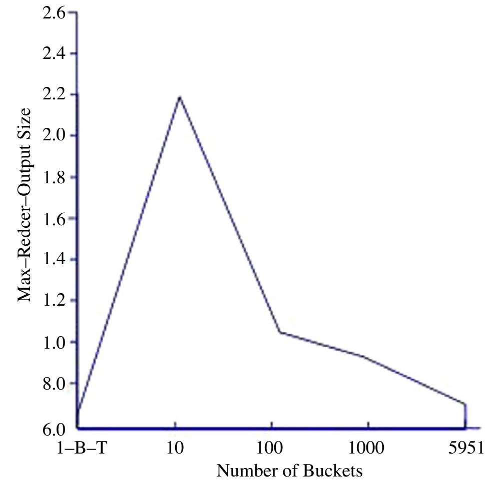


Fig. 12. Max-reducer-output by number of buckets.

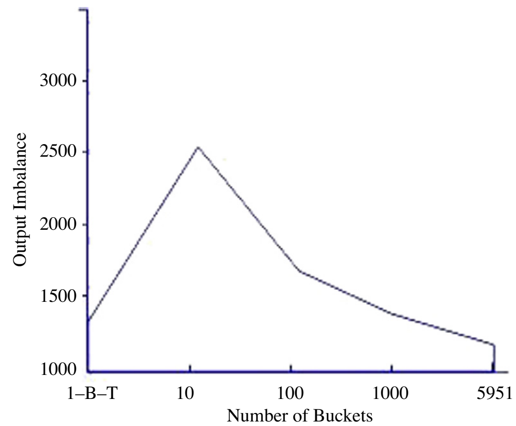


Fig. 13. Output imbalance for number of buckets.

7. Conclusions and futurework

In this paper, we have defined a programming model for material cloud applications that supports multiple different Map and Reduce functions running in parallel. MaMR uses a hybrid

Please cite this article in press as: W. Jing, et al., MaMR: High-performance MapReduce programming model for material cloud applications, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.07.015

ARTICLE IN PRESS

shared-memory BSP model that can make full use of the data nodes in a cloud computing system. We have designed an optimized data-sharing strategy using the BSP model to support the shared data for Map and Reduce. Meanwhile, we further provide multi-copies of the output to reduce the shuffle overhead. We add a new Merge phase to Map-Reduce that can efficiently merge data already partitioned and sorted (or hashed) by the map and reduce modules.

In future work, we will explore this new method to improve the parallel efficiency. Currently, more large cloud computing systems should be used to test and verify the MaMR model. The advantages of the programming model should be further amplified by more material data.

Acknowledgments

The work described in this paper is supported by the Fundamental Research Funds for the Central Universities (2572014EB05-4), Doctoral Fund of the China Ministry of Education (20120062110012), Special Foundation for Science and Technology Innovate Talents of Harbin (2015RAYXJ005) and National Natural Science Foundation of China (No. 31370565) and NCET-12-0809.

References

[1] Luca M. Ghiringhelli, Jan Vybiral, Sergey V Levchenko, Claudia Draxl, Matthias Scheffler, Phys. Rev. Lett. 114 (2015) 105–108.
[2] H. Jin, T. Cortes, R. Buyya, High Performance Mass Storage and Parallel I/O:Technologies and Applications, Wiley, 2001.
[3] I. Foster, I. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 2002.
[4] iaodong Liu, Weiqin Tong, Fu ZhiRen, Liao WenZhao, Int. J. Grid Distrib. Comput. 6 (1) (2013) 87–97.
[5] Y. Yu, M. Isard, D. Fetterly, DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-level Language, USENIX Association, 2008.
[6] Lin JiaChun, Leu FangYie, Chen Yin-ping, Comput. J. 59 (5) (2016) 701–714.
[7] Yang JiJiang, Li JianQiang, Niu Yun, Future Gener. Comput. Syst. 29 (2013) 739–750.
[8] T. Jie, H. Marten, A. Streit, S.U. Khan, J. Kolodziej, D. Chen, The 26th IEEE International Parallel and Distributed Processing Symposium, IPDPS, Shanghai, China, 2012.
[9] M. Iiard, M. Budiu, Y. Yuan, Oper. Syst. Rev. 41 (2007) 59–72.
[10] R. Moraveji, J. Taheri, M.R.H. Farahabady, N.B. Rizvandi, A.Y. Zomaya, The Proceedings of the ACM/IEEE 13th International Conference on Grid Computing, Grid 12, September 2012, pp. 95–103.
[11] O. Bonorden, 2007 IEEE International Parallel and Distributed Processing Symposium, 2007.
[12] L.G. Valiant, Commun. ACM 33 (1990) 103–111.
[13] M.A.H. Hassan, M. Bamha, A Parallel Processing of 'Group-by Join' Queries on Shared Nothing Machines, Springer-Verlag New York Inc., 2008.
[14] Y. Bu, B. Howe, M. Balazinska, M.D. Ernst, Proc. of the 36th International Conference on Very Large Data Bases, VLDB 10, September 2010, pp. 11–17.
[15] H. Yang, A. Dasdan, R. Hsiao, D.S. Parker, Proc. of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD 07, June 2007, pp. 1029–1040.
[16] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V.B.N. Rao, V. Sankarasubramanian, S. Seth, C. Tian, T. ZiCornell, X. Wang, Proc. of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD 11, June 2011, pp. 1081–1090.
[17] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins, Proc. of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 08, June 2008, pp. 1099–1110.
[18] L. Neumeyer, B. Robbins, A. Nair, A. Kesari, Proc. of the International Workshop on Knowledge Discovery Using Cloud and Distributed Computing Platforms, KDCloud 10, December 2010, pp. 170–177.
[19] I. Fan Zhang, Qutaibah M. Malluhi, Tamer Elsayed, Samee U. Khanc, Keqin Li, Albert Y. Zomaya, Future Gener. Comput. Syst. 51 (2015) 98–110.
[20] S. Ghemawat, H. Gobio, S.-T. Leung, SOSP, 2003, pp. 29–43.
[21] J. Dean, S. Ghemawat, OSDI, 2004, pp. 137–150.
[22] J. Gray, Sort Benchmark, 2006 http://research.microsoft.com/barc/SortBenchmark/.
[23] A.C. Arpaci-Dusseau, et al., SIGMOD 1997, 1997, pp. 243–254.
[24] TPC. TPC-H. http://www.tpc.org/tpch/default.asp.
[25] Mahout. http://lucene.apache.org/mahout/, 2010 (accessed 07.07.2010).

Please cite this article in press as: W. Jing, et al., MaMR: High-performance MapReduce programming model for material cloud applications, Computer Physics Communications (2016), http://dx.doi.org/10.1016/j.cpc.2016.07.015