Sustainable Computing: Informatics and Systems 21 (2019) 80–89

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

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

Energy efficiency optimization in big data processing platform by improving resources utilization

Jie Song [a,*], Zhongyi Ma [a], Richard Thomas [b], Ge Yu [c]

[a] Software College, Northeastern University, Shenyang, China
[b] School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Queensland, Australia
[c] School of Computer Science and Engineering, Northeastern University, Shenyang, China

ARTICLE INFO

Article history:
Received 17 February 2017
Received in revised form 14 July 2018
Accepted 21 November 2018
Available online 24 November 2018

Keywords:
Big data processing platform
Energy efficiency
Resource utilization
Resource ratio
Resource allocation
Task scheduling
Green computing

ABSTRACT

Big data creates tremendous value for humanity, but also places a heavy burden on the environment. Energy consumption of computing platforms, especially big data processing platforms, cannot be ignored, and optimization of energy usage is imperative. To the out-of-core data processing, this paper proposes that the energy efficiency of a big data processing platform is closely related to the utilization of its computing resources; and an efficient resource allocation strategy for data processing tasks improves the platform's resources utilization. To improve the resources utilization, different resources are allocated according to a task-related Best Resource Ratio (*BRR* for short), such as "*cpu*, *disk*, *network* in the ratio of 1:2:4", rather than the resource's quantity, such as "*cpu* = 1 GHz, *network* = 20MB/s". We deduce the *BRR* of data processing tasks, and design a resource ratio based approach (R^2), which includes a task scheduling algorithm and resource allocation algorithm, for energy efficiency optimization. Experiments show that the R^2 approach can improve energy efficiency by 10%, 10% and 6% compared to *FIFO*, *Capacity* and *Fair* schedulers respectively, by maximizing resource utilization of a 12-node MapReduce platform.

1. Introduction

Green computing is a hot research topic, and how to optimize the energy efficiency of a computer system has a long history of research. This paper studies energy efficiency optimization on big data processing platforms. A big data processing platform is by definition the computing platform for processing big data. Generally, a computing platform is a combination of hardware system, operating system, middleware and runtime libraries. A big data processing platform has a computing cluster as the hardware system, share-nothing as its architecture and a distributed computing framework as its middleware system. The computing framework mentioned in this paper refers to out-of-core frameworks, such as Hadoop, Dremel, or Pregel, but not for in-memory frameworks, such as Spark or GraphX. The framework divides the algorithm into different phases, such as *Map* and *Reduce* in a MapReduce platform. In a phase, tasks are distributed to nodes and executed in parallel. It is pipelines of work executed across server nodes.

Essentially, there is a hardware stack that processes the data, and the data only moves along the pipeline, which includes disks, CPUs, network cards, etc. Each of these hardware components has a different rate at which data can be processed (i.e., processing throughput), which could be theoretically normalized in terms of bytes processed per second. An optimal resource allocation should take into account the data processing rate of each component, otherwise some components will be underutilized, leading to a sub-optimal resource allocation strategy. For example, if there are only 400 KB/s of disk bandwidth available, there is no point in allocating 10 CPU cores for a given task, as most of the cores will be sitting idle. By applying this technique, the idle energy consumption is minimized. This is the basic knowledge of our research.

In computing, performance per watt is a measure of the energy efficiency of a particular computer architecture or computer hardware [1], such as MIPS/W (Millions of Instructions per Second per Watt) for energy efficiency of mainframe systems. Following the hardware definition, energy efficiency of a software is also as performance (in operations/second) per watt (consumed by the resources allocated to the software). Comparing with the definition of energy efficiency metrics in prior works [2,3], we defined the energy efficiency of a data processing platform or a data processing task as the ratio of the data processing performance and the energy consumed by the allocated resources. The quantitative definition is given in Definition 4 in Section 3. In this paper, energy efficiency is abbreviated to *EE*. We optimize the *EE* of big data pro-

* Corresponding author.
E-mail address: songjie@mail.neu.edu.cn (J. Song).

https://doi.org/10.1016/j.suscom.2018.11.011
2210-5379/

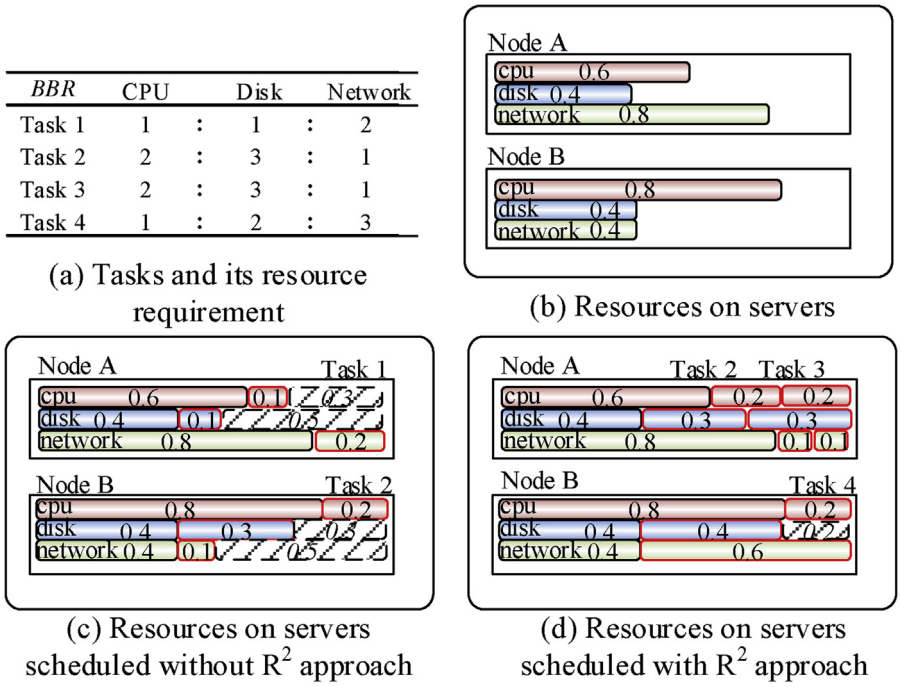

Fig. 1. A brief example of R^2 approach to improve the resources utilization.

cessing platforms by improving their resource utilization. Firstly, we clarify the following concepts:

(1) The computing resources of a node, such as CPU frequency or cores, and I/O bandwidth of disk and network, are allocated to the task to process data, and consequently also consume energy. There are many different tasks synchronously running on a node. Resources of the node are allocated to each task independently, and isolated in a runtime environment such as a slot in Hadoop.
(2) An *EE* of a data processing platform or a data processing task is the ratio of processing performance and energy consumed by computation resouces, which is proportional to the ratio of "the data processing performance" and "the energy consumed by its resources". According to this, the *EE* is optimal when there is no idle resouces, for all enegy consumed by resources are used in data processing.
(3) During the execution of a task, the allocated resources may be idle. For example, a CPU is suspended while waiting for I/O operations or network transmission. Let a computer component represent a resource. There is no component whose standby power is zero. Ideally, let the power of a component at time t be $p(t)=k\omega(t)+p_0$, $\omega(t)$ is resource utilization at time t, and k is a positive constant, p_0 is standby power. When the component is full-loaded, $\omega(t)=1$, $p(t)=k+p_0$; when the component is partly idle, energy wasted during period T is $wasted\_energy = consumed\_energy - utilization() \times full\_loaded\_energy = \int_T[\omega(t)k+p_0]dt - \int_T\omega(t)(k+p_0)dt = p_0\int_T[1-\omega(t)]dt$. Therefore, the idle resources waste energy, which is defined as idle energy consumption.
(4) If there are no idle resources, all the consumed energy contributes to the data processing and there is no idle energy consumption, then the *EE* of the task is optimal. Idle resources may not be eliminated but can be reduced by an efficient resource allocation strategy under the condition of the tasks are sufficient.

The existing research about *EE* optimization are "workload concentration" and "idle nodes power-off". But in big data processing platforms, data is stored in nodes. Nodes should be älways-onänd are expensive to switch off for the following reasons: **Firstly**, it is seldom that a node is totally idle because it continuously provides a data service; **Secondly**, the intermediate results of a task are stored on the local disk for further processing; **Thirdly**, synchronization of tasks, such as *Map* and *Reduce* phases in a MapReduce platform, requires always-on nodes, otherwise all the other nodes are blocked and waiting for their startup. Thus ÖN-OFFälgorithms [4,5] are difficult to be applied in big data processing platforms. In this paper, under the condition of sufficient tasks, we take the computing resources into consideration. If the resources of a task are fully utilized, the *EE* of the task is maximized; and if the *EE* of every task on a node is optimal, the *EE* of the node is optimal. To optimize the *EE* of the platform, we should try to ensure that a proper amount of resources are allocated to every running task, and minimize the idle resources in the platform.

As previous definition, the *EE* of a task is "performance/energy". On one hand, allocating more resources to a task may not improve its *EE* because both performance and energy consumption are increased. On the other hand, unreasonable resource allocation leads to resource redundancy, with the consequent issue of idle energy consumption. Therefore, we argue that when the amount of resources allocated to a task satisfies a certain proportion, *EE* is a constant no matter how much resources are allocated. Moreover, there is an optimal ratio, which is defined as the Best Resource Ratio (*BRR* for short), that results in the highest *EE*. This paper proves the existence of the *BRR* by formal deduction and experimental verification. We then designed the resource ratio based approach (R^2), which includes task scheduling and resource allocation algorithms.

Fig. 1 shows an example of the *BRR* and R^2 approach. The four tasks in a queue are shown in Fig.1-(a); and the available resources on both node *A* and node *B* are shown in Fig. 1-(b); For the traditional scheduling approach such as FIFO, task 1 is selected fristly and allocated to server *A*, and task 2 is allocated to server *B*, therefore, both node *A* and node *B* produce much idle resources due to the unmatched resource requirement, as shown in Fig.1-(c);

For R^2 approach, as shown in Fig. 1-(d), according to the available resources and *BRR* of tasks, task 1 is not scheduled because its *BBR* is not matched the remained resources of both node *A* and *B*, while task 2 and 3 are scheduled to node *A*, and task 4 is scheduled to node *B*, according to the available resources and *BRR* of tasks; afterwards, the resources of node *A* and *B* are allocated to their tasks, respectively. Using such approach, there is a minimum idle resources in each node, and each task satisfies its resources requirements. Fig. 1 is a brief example about the proposed approach, for instance, resources of node *A* are re-allocated to the three tasks according to their *BBR*s after scheduling (the same as node *B*), such strategy are abbreviated in Fig.1.

Based on the previous description, the following three questions are studied for *EE* optimization: **Firstly**, what is the criterion of resource allocation for a task? **Secondly**, which task should be scheduled next when a node is available? **Thirdly**, how to allocate the resources in a node to the tasks on the node. Obviously the first question is the critical one. Resource allocation is a classic topic in different environments with different objectives, such as performance, fairness and adaptability. Aiming to optimize *EE* of the platform, the main innovation of this paper is *BRR* based task scheduling and resource allocation, rather than resource quantities based scheduling and allocation.

The structure of the paper is as follows: Section 2 introduces related work; Section 3 introduces the resource and *EE* model; Section 4 deduces the *BRR* of a task theoretically; Section 5 proposes *BRR* based task scheduling and resource allocation algorithms; Section 6 proves the existence of the *BRR* and shows the optimization effect of the R^2 approach; Section 7 discuss the application scenario of *BRR*. Section 8 concludes the paper and proposes future work.

2. Related works

A task's *EE* is the ratio of its performance to its energy consumption, so we must consider the balance between performance and energy consumption. There are two different opinions about this balance in current research. Prior work [6–8] argue that energy consumption and performance are different objectives. Following this opinion, optimization is a trade-off since improving performance brings with it higher energy consumption, whereas reducing energy consumption necessarily causes lower performance. However, Tsirogiannis et al. [9] and Wang et al. [10]. argue that the optimization of energy consumption is consistent with that of performance, which is the reason why the best energy-saving system is also the one with the best performance. Conceptually the former conforms to hardware design, that is, faster equipment has higher power consumption; while the latter accords with softwares design that aims to save energy by reducing execution time. The different opinions come from different assumptions. The former does not take idle energy consumption into consideration, and the latter neglects the instability of computer power. For example, suppose a computer has two modes: green mode has the lowest power consumption, such as 50 W, and the worst performance, while performance mode has the highest power consumption, such as 100 W, and the best performance. Then a task running in green mode for 2 min, consumes as much energy as a task running in performance mode for 1 min. However, in a big data processing platform, the energy consumption is more dominated by idle energy consumption than by the dynamic computer power [11].

The energy consumption optimization approaches are categorized into hardware optimization and software optimization. The hardware optimization includes the components "on-off" algorithm [4,5], which observes the workload of each component, and turn it off when it is idle. In addition, performance scaling techniques, such as dynamic voltage and frequency scaling (DVFS) [12] adjust hardware performance dynamically according to its power, providing high-power mode for high performance, and low-power mode for energy saving. Software optimization is used in both stand-alone [13] and cluster environments [14]. Some software approaches focus on the characteristics of tasks or a task queue instead of "workload concentration" and reduce idle energy consumption by improving resources utilization. Venkatesh et al. [15] adjust the CPU frequency according to its utilization to optimize energy consumption; Wang et al. [16] designed a task scheduling algorithm for improving resource utilization; Yong et al. [17] propose a dynamic slots mechanism to save energy and improve performance by increasing CPU utilization. These task-specific optimization approaches improve task *EE* through fully utilizing the allocated resources, and they mainly consider the quantities of one or several specific resources. In this paper, we consider the ratio of several resources, hence differing from those approaches.

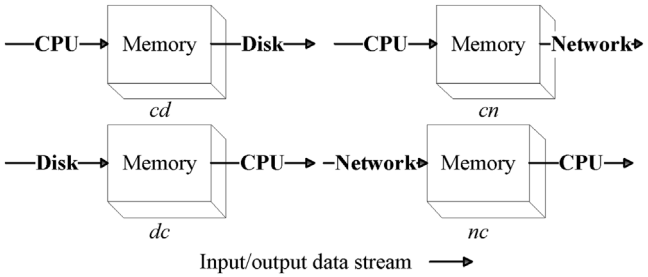

Fig. 2. The phase types.

In an application co-location environment, resource contention and sharing are related technologies for performance or energy consumption optimization. For CPU and disk resource, Manousakis et al. [18] take the data-access into consideration, they proposed an energy profiling tool for task-parallel programs, by which the data locality and memory access patterns could be concluded, and further avoid the resources contention by a task scheduler. For CPU and memory resource, Dauwe et al. [19] study the multicore processors system, and proposed a method of predicting the application performance and energy consumption under application co-location environment. For CPU and network resource, Leal [20] presented performance based scheduling and self-adjusting resource sharing strategy that reduce computing power and communication bandwidth, and hence reduce algorithm's total communication cost and power consumption. The main idea is to restrict to the maximum the number of needed messages per job scheduled. Xu et al. [21] presented a task scheduling algorithm on heterogeneous system with multiple priority queues. The Communication to Computation Ratio (CCR) can be used to indicate whether a task graph is communication-intensive or computation-intensive. For tasks, CCR is computed to determine what kind of resource is needed. With CCR, multiple priority queues scheduling algorithm provide better support for heterogeneous system by supporting various tasks. All these works took CPU resource into consideration because of which is the default resource for any application. Beside CPU, another resource is characterized and modeled for contention. In our approach, we focus on a data processing task, in which all data is processed by CPU and via memory. We divide a task into parallel phases. Each phase is associated with memory, CPU and another resource. Beside memory, the other two resources are as a data producer and a consumer (see Fig. 2). We characterized the required resources of a task by the aggravation on the required resources of its phases. Consequentialy, all resources are considered.

Some works study the contention of all resources by dividing tasks into CPU-intensive, disk-intensive, I/O-intensive, memory-intensive and network-intensive [22]. For eample, Zhu et al. [23] propose a scheduling strategy called green scheduling that makes

the tasks, whose resource requirements are complementary, execute in parallel. For example, the strategy increases the possibility of CPU intensive tasks and I/O intensive tasks running simultaneously. Similarly, Sheikhalishahi et al. [24] divided the workload into compute, data, memory, communication intensive, and then proposed autonomic resource contention-aware scheduling that reduce resource contention by sharing one resource among multiple tasks simultaneously. These works lack the quantified analysis about resources utilization, and not propose the concept of resource ratio. In our paper, the *BRR* model quantitatively defines what complementary resources are required by tasks. Unlike "resource intensive", "resource ratio" shows how much multiple resources are proportionally required by a task. For a task, the best resource allocation is not the maximal resource allocation but the allocation in an appropriate resource proportion, and typical variation of energy per task with utilization (of a single resource) can be expected to result in a concave curve. The best resource ration should be the objective of resource management.

In our approach, the *BRR* of tasks is deduced, and then the task scheduling algorithm selects a suitable task from the task queue and schedules it to a suitable node, while the resource allocation algorithm allocates resources to the task according to its *BRR*. It aims to improve resource utilization instead of maximizing or balancing the resource quantity, which is the essential difference to traditional approaches.

3. Resources and energy efficiency model

Resources in computer systems mainly include hardware resources, network resources, software resources and data (content) resources, and the first two consume energy directly, through usage of the central processor unit (CPU), memory, external memory, I/O devices, network adapters, etc. In this paper, computing resources are the abstraction of hardware and network resources. If the cluster is built using virtualization technology, resources are the physical hardware assigned to the virtual node.

Definition 1 *(Resource quantity).* Resource quantity is a measurement of the quantity of computing resources allocated to a task or consumed by a task. Quantity of both allocated and consumed resources is time relevant. At time t, there are n resources for a task, the notations are shown as follows:

(1) $r_i(t)$ is the resource quantity of the i-th resource consumed by the task;

(2) $r_i'(t)$ is the resource quantity of the i-th resource allocated to the task;

(3) $\Delta r_i(t) = r_i'(t) - r_i(t)$ is the idle resource quantity of the i-th resource allocated to the task;

(4) $|r_i|$ is the total quantity of the i-th resource in the node;

The quantity of different resources should have a uniform unit of measurement to allow the calculation of ratios. We define the unit of resource quantity to be MB/s. For a data processing task, the disk resource quantity $r_d(t)$ and the network resource quantity $r_n(t)$ represent bandwidth of disk and network, respectively. The CPU resource quantity $r_c(t)$ represent how much data being processed by the benchmarked data processing task. The well-known linear search is an algorithm for finding a particular value in a list that checks each element in sequence until the desired element is found or the list is exhausted. It is chosen as the benchmarked data processing task because its linear complexity can easily be compared by other algorithms. The CPU resource quantity follows definition 2.

Definition 2 *(CPU resource quantity).* Let $F(t)$ be CPU frequency (GHz), and $W(t)$ be CPU utilization (dimensionless) at time t. Then CPU resource quantity at time t is defined as $C{\cdot}F(t){\cdot}W(t)$, whose unit is MB/s. The unit of coefficient C is MB/(s·GHz), while C MB/(s·GHz) means a 1 GHz CPU can search C MB data per second using a linear search algorithm.

Please notice that the "linear search" is a benchmarking to quantified CPU resource. Definition 2 is a measurement to CPU resource, not a measurement to algorithm. With definition 2, we can illustrate that: a CPU resource is 400 Mb/s, which means it can linearly search 400Mb data per second, or we allocate 100MB/s CPU resource to a sorting task, which means "the computation power of linearly searching 100MB data per second" is to the task.

Definition 3 *(Throughput).* Given a task, the throughput of the i-th resource, denoted as $v_i(t)$, is the speed of the data being processed via the i-th resource (MB/s), on condition that $r_i'(t)$ is allocated and $r_i(t)$ is consumed at time t.

For example, for a task at time t, CPU throughput $v_c(t)$ is the data volume being processed by CPU on condition that $r_c'(t)$ is allocated and that $r_c(t)$ is consumed; disk throughput $v_d(t)$ is the data volume being read from or written to the disk on condition that $r_d'(t)$ is allocated and $r_d(t)$ is consumed; and the same for network throughput $v_n(t)$.

For a task, throughput of a resource $v(t)$ and consumed resource $r(t)$ are not equal. On one hand, CPU throughput $v_c(t)$ is not equal to consumed CPU resource $r_c(t)$, because the complexity of the task may not be the same as that of linear search algorithm. On the other hand, big data processing algorithms are out-of-core algorithms, and the processed data is stored not in random-access memory but sequential-access disk, so that disk throughput $v_d(t)$ and network throughput $v_n(t)$ are not equal to consumed disk resource $r_d(t)$ and consumed network throughput $r_n(t)$, respectively, for the size of data item which is processed by the task is not equal to the size of data block which is retrieved from the disk and network. For CPU resource, let θ_c be the average times of same data item being processed; For disk (or network resource), let $\theta_d(\theta_n)$ be the ratio of data items size and data block size. Then:

$$v(t) = f(r){\cdot}r(t)\ f(r) = \theta_c \text{ if } r \text{ is CPU resource, } f(r) = \theta_d(\theta_n) \text{ if } r \text{ is disk (or network) resource} \quad (1)$$

The *EE* of a task, is the ratio of "processed data volume" and "energy consumed by the allocated resources". Resource allocation is exclusive so that energy consumption is for allocated resources not just for consumed resources, this then includes idle energy consumption. For the instantaneous value of *EE*, the "processed data volume" is the task throughput, and "energy consumed by the allocated resources" is "power × resource occupancy (%)".

Definition 4 *(Energy efficiency (EE) of a task).* For a task at time t, the energy efficiency $\eta(t)$ is the ratio of the throughput $V(t)$ and the energy $E(t)$ consumed by the allocated resources.

$$n(t) = \sum_{i=1}^{n} n_i(t) = \sum_{i=1}^{n} \frac{v_i(t)}{p_i(t) \cdot r_i'(t) / |r|} = \sum_{i=1}^{n} \lambda_i(t) f_i(t) \cdot \frac{r_i(t)}{r_i'(t)} (\lambda_i(t) = \frac{|r_i|}{p_i(t)} \quad (2)$$

In (2), $p_i(t)$ is the power of the i-th component at time t. The unit of *EE* is MB/Joule. Because $p_i(t)$ is hardware specific, $r_i(t)/r_i'(t)$, which represents resources utilization, is the element to be optimized in this paper. It is impossible that n resources are all somehow idle because the bottleneck resource is used up firstly. For example, in an I/O intensive task, the CPU may be blocked by I/O operations, while the I/O resources are fully utilized and CPU resources are idle.

The maximum value of $r_i(t)/r_i'(t)$ is 1 where $r_i(t)=r_i'(t)$. Therefore, maximum *EE* occurs at time t on condition that every quantity of allocated resources is equal to that of the corresponding consumed resources. *BRR* can be determined through quantity of consumed resources.

Definition 5 *(Best resource ratio (BRR)).* For a task, its resource ratio at time t is the ratio of each allocated resource quantity. If there are n resources, and let $a(t)$ be the resource ratio, then

$$a(t) = r1'(t) : r2'(t) : \ldots : rn'(t) \tag{3}$$

The *BRR* of a task is simply defined as above, but there remain three challenges to be solved: (1) definitions 1–5 are resource and *EE* models of time t, it is a real-time model, task scheduling and resource allocation cannot be triggered in real-time to satisfy the *BRR*. (2) Theoretically, *BRR* is a ratio of consumed resources, but it is infeasible to test *BRR* of tasks by various combinations of resource allocation. An approach to deduce *BRR* is required. (3) Memory is a special resource particularly in big data processing platforms. Out-of-core algorithms treat memory as a buffer rather than data storage, thus the memory allocated to tasks is never idle. The size of allocated memory affects the utilization of other resources. Therefore, memory should be treated as a restriction of *BRR*, but not an element of *BRR*. To solve the above three challenges, we will deduce *BRR* (CPU, disk and network) under the restriction of memory in section 4.

4. Best resource ratio

On one hand, it is too simple to treat an entire task with a single resource model since the task contains different phases which process data in different ways; On the other hand, the resource model at time t is also too transient since the resource requirements do not change so frequently. Therefore, we divide a task into fine-grained phases according to its resources utilization, and then deduce the *BRR* of the task through analyzing each phase. In terms of data processing, a phase is an operation during which two resources exchange data through memory, and a task is the sequence of phases. In terms of throughput, a phase is a duration in which task has a stable throughput. In terms of resource, a phase is the abstraction of the resource's utilization regularity.

Definition 6 *(Phases and phase type).* In a data processing task, a phase is the most fine-grained form of computing model. A phase is a producer-consumer model, in this case one resource provides data while another resource consumes data. The memory is the buffer for data exchange. The buffer supports the non-conforming speed (throughput) of providing data and consuming data. In a phase, data provided by a "data provider" streams into memory, while data consumed by a "data consumer" streams out of memory. There are four types of producer-consumer models, and we define them as phase types: *cd*, *cn*, *dc* and *nc* (see Fig.2). The phase types are "models", and the phases are "instances".

The phase types are used to make it easier to deduce the *BRR* of a phase. A phase h can be expressed by an ordered pair $h =< \vec{r}, \longleftarrow r >$. $\vec{r}$ is the resource (data provider) that puts data into the memory by speed $\vec{v}$; $\longleftarrow r$ is the resource (data consumer) that gets data out of the memory by speed $\longleftarrow v$. Two resources process data is different throughput because there is a buffer (memory) between them. Phases are the partitions of a task's execution duration. In section 3, functions $r(t)$, $v(t)$ and $a(t)$ at time t are converted to the functions $r(k)$, $v(k)$ and $a(k)$ at phase k. Moreover, all the notations are for the k-th phase in this section, thus we abbreviate the function argument k to simplify the equations.

Definition 7 *(τ function).* Given the k-th phase $h =< \vec{r}, \longleftarrow r >$ of a task, the τ function returns the execution time of this phase.

$$T = \tau(h) = \tau(\vec{r}, \longleftarrow r) \tag{4}$$

Suppose that ε is the size of memory which is allocated to a task exclusively, and ε_0 is the size of data actually stored in memory, thus for any phase $\varepsilon_0 \le \varepsilon$. All phase types require the CPU resource and at least one of the disk and network resources. Memory is a data buffer (read or write) for the resource with the largest throughput. Let M be the size of processed data, and the execution time of a phase is the time that un-buffered resource processes M data set:

$$T = \frac{M}{min(\vec{v}, \longleftarrow v)} \tag{5}$$

According to (4) and (5)

$$\tau(\vec{r}, \longleftarrow r) = \frac{M}{[f(\vec{r})\cdot\vec{r}]^{\alpha}\cdot[f(\longleftarrow r)\cdot\longleftarrow r]^{1-\alpha}} \begin{cases} \vec{v} < \longleftarrow v, \alpha = 1 \\ \vec{v} \ge \longleftarrow v, \alpha = 0 \end{cases} \tag{6}$$

The basic idea of deducing the *BRR* of phases is to take advantage of the data relationship between data producers and consumers. If there is no memory cache, the throughput of the data producer is equal to that of the data consumer. The memory is fixed-size storage, the data producer is blocked when the memory buffer is full and the data consumer is blocked when the memory buffer is empty.

Suppose phase $h =< \vec{r}, \longleftarrow r >$, the speed (throughput) of producing data (inflow to memory) and consuming data (outflow from memory) is $< \vec{v}, \longleftarrow v >$, and the execution time is T. When (7) is satisfied, resources cannot be blocked and there are no idle resources:

$$T\cdot(\vec{v} - \longleftarrow v) \in [-\varepsilon_0, \varepsilon] \tag{7}$$

If $T\cdot(\vec{v} - \longleftarrow v) > \varepsilon$, the data producer (inflow to memory) is blocked and idle. Similarly, if $T\cdot(\vec{v} - \longleftarrow v) < -\varepsilon_0$, the data consumer (outflow from memory) is blocked and idle. According to the definition of the τ function and the relationship between resource throughput and resource quantities, (7) is transformed to (8).

$$\frac{M\cdot[f(\vec{r})\cdot\vec{r} - f(\longleftarrow r)\cdot\longleftarrow r]}{[f(\vec{r})\cdot\vec{r}]^{\alpha}\cdot[f(\longleftarrow r)\cdot\longleftarrow r]^{1-\alpha}} \in [-\varepsilon_0, \varepsilon] \quad \begin{cases} \vec{v} < \longleftarrow v, \alpha = 1 \\ \vec{v} \ge \longleftarrow v, \alpha = 0 \end{cases} \tag{8}$$

The *BRR*, which is closely related to the size of allocated memory, has multiple possible values. Let $\varepsilon_0 = \varepsilon$, we take boundary conditions into account, so that the memory is filled when it is the data output buffer, and is empty when it is the data input buffer. Then:

When $\alpha = 1$ in Eq. (8)

$$\frac{M\cdot\left[f\left(\vec{r}\right)\cdot\vec{r} - f(\longleftarrow r)\cdot\longleftarrow r\right]}{f\left(\vec{r}\right)\cdot\vec{r}} = -\varepsilon \Rightarrow$$

$$BBR = \frac{\vec{r}}{\longleftarrow r} = \frac{M\cdot f(\longleftarrow r)}{M\cdot f\left(\vec{r}\right) + \varepsilon} \tag{9}$$

When $\alpha = 0$ in Eq. (8)

$$\frac{M\cdot[f(\vec{r})\cdot\vec{r} - f(\longleftarrow r)\cdot\longleftarrow r]}{f(\longleftarrow r)\cdot\longleftarrow r} = \varepsilon \quad \Rightarrow$$

$$BBR = \frac{\vec{r}}{\longleftarrow r} = \frac{M\cdot f(\longleftarrow r) + \varepsilon}{M\cdot f(\vec{r})} \tag{10}$$

The *BRR* of a task is in form of "r_c:r_d:r_n". Let a task contain s phases, and the *BRRs* of them are "$r_c{}^1$:$r_d{}^1$:$r_n{}^1$", "$r_c{}^2$:$r_d{}^2$:$r_n{}^2$", ..., "$r_c{}^s$:$r_d{}^s$:$r_n{}^s$". For deducing *BRR* easily, assume there are only two resources in each phase, with the CPU resource being the essential one, and disk or network resources are secondary. Therefore, for each k-th phase, $r_d{}^k$ or $r_n{}^k$ is zero. Actually, phases are executed in parallel. The aggregation on phases' *BRRs* aims at taking parallel execution into account. For example, the parallel "1:1:0″ and "1:0:1″ can be aggregated into "2:1:1″. We aggregate the same

resource elements of *n BRRs*, for example $r_c{}^1, r_c{}^2, \ldots, r_c{}^s$, to the sum of them.

5. Task scheduling and resource allocation

This section introduces Resource Ratio (R^2) based task scheduling and resource allocation. The R^2 approach allocates resources to the selected tasks to satisfy their *BRR*. The R^2 approach could be adopted in big data processing platform where: *BRR*s of tasks are known, and there is a sufficient number of tasks.

Definition 8 *(Task scheduling).* The task scheduling algorithm selects the optimal task from task queue when a node requests a new task. The selected task is one whose *BRR* is proportionally congruent with the unallocated resources in the node. Let vector $r = \{r_1, r_2, \ldots, r_n\}$ be the unallocated resources in the node, and vector $a = \{a_1, a_2, \ldots, a_n\}$ be the *BRR* of a task. Cosine similarity of the two vectors, a and r, is adopted.

After scheduling, both the selected tasks from the task queue and the incomplete tasks on the node are allocated (or re-allocate) the node's resources.

Definition 9 *(Resource allocation).* Let n resources be allocated (or re-allocated) to the m tasks on the node, the $n \times m$ matrix $A = [a_{ij}]_{n\times m}$ represents the *BRR* of m tasks, and the $n \times 1$ column-matrix $r = [r_i]_{n\times 1}$ represents the unallocated resources. Resource allocation is defined as assigning r to m tasks resulting in the matrix $r' = [r_{ij}']_{n\times m}$ which satisfies following conditions:

Condition (1): $\forall j \in [1,m]$, $r_{1j}' : r_{2j}' : \ldots : r_{nj}' = a_{1j} : a_{2j} : \ldots : a_{nj}$

Condition (2): Minimize Δ, and $\Delta = \sum_{i=1}^{n} \left(r_i - \sum_{j}^{m} r_{ij}' \right)$

Condition (3): When $m>1$, to minimize σ and $\sigma = \frac{1}{m}\sum_{j=1}^{m} \left(\sum_{i=1}^{n} r_{ij}' - \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{n} r_{ij}' \right)^2$

Condition (1) means that resources allocated to each task satisfy its *BRR*. Condition (2) means that the unallocated resources are minimized. Condition (3) means that the resources allocated to each task are balanced in order to ensuring the fairness. We define "unitary resource allocation" when $m = 1$, and "plural resource allocation" when $m>1$.

Solving the result matrix r' of unitary resource allocation is simple. Without loss of generality, let *BRR* of the task be $a_1 : a_2 : \ldots : a_n$, and $x = \min(r_1/a_1, r_2/a_2, \ldots, r_n/a_n)$, then the optimal allocation is $r' = x \cdot a$. It is easy to prove that the $x \cdot a$ satisfies conditions (1)–(3).

Solving the result matrix r' of plural resource allocation is complex. We consider the condition (1) firstly. If there is a matrix $x = [x_j]_{1\times m}$ satisfying $A \cdot x = r$, then the resources allocation x satisfies *BRR* of every task without unallocated resources. Besides, the i-th resource allocated to the task j is as much as $a_{ij} \times x_j$. The matrix expression of $A \cdot x = r$ is as follows:

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \vdots & \vdots & \ldots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{mn} \end{bmatrix} \times \begin{bmatrix} x_1 & x_2 & \ldots & x_m \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix} \tag{11}$$

The matrix x is the solution of the "m-varieties non-homogeneous linear equation":

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots a_{1m}x_m = r_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots a_{2m}x_m = r_2 \\ \qquad\qquad \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots a_{nm}x_m = r_n \end{cases} \tag{12}$$

The optimal resources allocation is matrix $r' = [r_{ij}']_{n\times m} = [a_{ij} \times x_j]_{n\times m}$ if a valid solution for (12) exists. The process of solving (12) is introduced in Appendix A.

6. Experiments

We employed the MapReduce platform to verify the *BRR* model and R^2 approach. MapReduce is a big data processing platform running in a cluster environment. This section compares the *EE* optimization of R^2 with other approaches and analyse the overhead of R^2. The testbed is as follows:

6.1. Setup

We evaluate the *EE* optimization effect through the R^2 approach in the MapReduce platform. The experiment is prepared as follows:

(1) The five use cases shown in Table 1 are simplified to Map tasks for highlighting resource allocation;
(2) The five use cases are categorized into two types: CPU-intensive tasks and I/O-intensive tasks;
(3) Tasks are issued by clients continually. There are sufficient tasks in the task queue.
(4) There are 4 slots on each node for 4 tasks running together.
(5) Tasks are randomly generated from the five use cases, but the proportion of CPU-intensive tasks to all tasks, denoted as parameter p, is adjustable. In the experiments, p is 0, 1/3, 1/2, 2/3 and 1;
(6) For comparison, we ensure that the five use cases are of similar scale by adjusting their input data size. It means the execution time of single task of each use case is roughly equal in a stand-alone environment. Therefore, the effects of the resource allocation and task scheduling algorithms are highlighted in the multi-tasks environment.
(7) It is difficult to distinguish the *EE*s of different tasks, so *EE* of the platform is evaluated. It is a ratio of "number of executed tasks" and "energy consumption of the platform".

We implement the R^2 scheduler in Hadoop because the task scheduler in Hadoop is a pluggable module. In addition, the "slot", in which the *Map* task of Hadoop is executed, is a Java process in the operating system. Linux provides comprehensive solutions for processes management. We compare the following schedulers (including resource management module):

(1) *FIFO* (First-in, First-out): It is the default scheduler in Hadoop. It chooses tasks according to the arriving time. The resources allocated to each tasks are equal.
(2) *Capacity*: It supports multiple queues, each of which has tasks with special resource requirements. When scheduling, the queue is selected firstly according to its resource features, and then tasks are selected according to the *FIFO* scheduling policy from the queue. For resource allocation, slots occupy equal-sized memory, while CPU, disk, and network resources, which are managed by the Linux system, are not restricted.
(3) *Fair*: Fair scheduling is a method of assigning resources to tasks, such that all tasks get, on average, an equal share of resources

Table 1
Experiment environment.

Item	Description
Node	1 master node, 11 computing nodes, homogeneous computers of Tsinghua Tongfang Z900, CPU Intel i5-2300 2.80 GHz, 8GB memory, 1TB hard disk.
Operation System	CentOS 5.6, Linux 2.6.18 core, On-demand governor
MapReduce version	Hadoop 1.0.2
Programming Environment	JavaSE 6
Power Measurement	According to the previous study [25]
Resources restriction	Cpulimit [26] restricts CPU resource, and Cgroup [27] restricts disk and network resources
MapReduce Use Cases and Approximate *BRR* (cpu: disk: network)	CPU intensive task: *π-calculation* [28] (1:∞^{-1}:∞^{-1});*Json-Parser*(1:2:∞^{-1}), *WordCount* [29] (1:4:1) I/O intensive task: *Descendant-query* [30] (1:15:8); *Map-Side-Join* [31] (1:10:12) ∞^{-1} represents a very small number
Parameter	*BRR*: theoratical deduction
Collection	CPU, disk, network utilization: Linux *sar* instructions

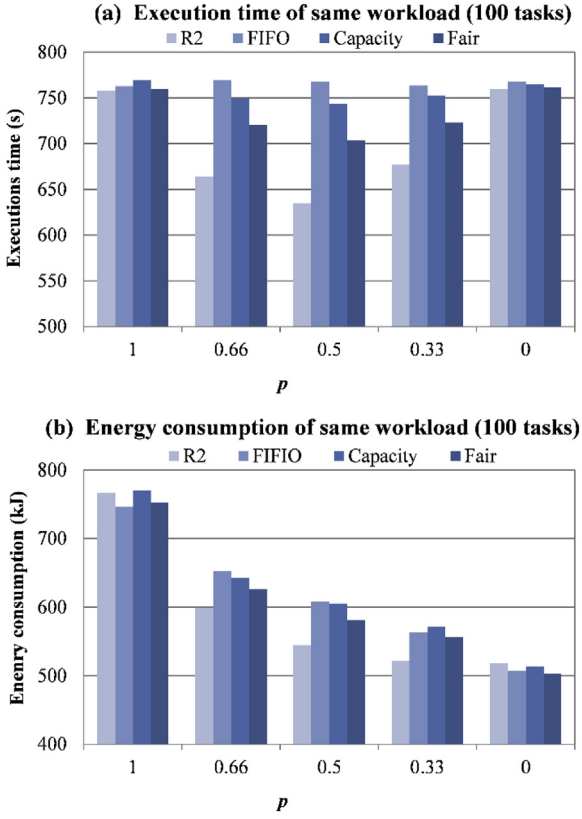

Fig. 3. Comparison of execution time and energy consumption of 100 tasks in different *p* value.

over time. It is also a resource quantity based scheduling. The tasks, whose required resource quantities are varied, are allocated with the minimum amount of resources at first from the resource pool to ensure that each task can survive. When a slot is available and some resources are idle, the task which requires for most resources is executed first.

(4) R^2: The *BRR* based task scheduling algorithm and resource allocation algorithm. We extend the slots management of Hadoop, so that memory is restricted by the Java virtual machine, CPU resource are restricted by *cpulimit*, and disk and network resources are restricted by *cgroup*.

Again, the time to complete all tasks, no matter what kind of use case they are, is nearly equal in a stand-alone environment. In the experiment, we set the number of tasks for each experiment to be 100, the *EE* is the ratio of 100 and energy consumption, thus lower energy consumption represents better energy efficiency. Fig. 3 compares the time to complete tasks and energy consumption with different *p* values executed with the four schedulers.

6.2. Time and energy

Fig. 3-(a) shows the following conclusions:

(1) When *p* is 1 or 0, there are only CPU-intensive tasks (*p* = 1) or I/O-intensive tasks (*p* = 0), any resource quantity or resource ratio based scheduling does not work because all tasks compete for the same resources. Due to the tasks being of similar scale, the time to complete the tasks is approximately equal.
(2) When *p* is 0.66, 0.5 or 0.33, time to complete tasks under *FIFO* is the worst because it takes no resource requirements into consideration and the possibility of resource competition is high.
(3) When *p* is 0.66, 0.5 or 0.33, time to complete tasks under R^2 is better than that of *Fair* and *Capacity* because R^2 is based on the resource ratio and the latter two are based on resource quantity. Sometimes the small quantity of resources, which cannot be allocated by the *Fair* and *Capacity* schedulers, can be fully utilized by a task in R^2. The R^2 scheduler has finer granularity than resource quantity based schedulers.
(4) The time to complete tasks with an even mix of tasks (*p* = 0.5) is much better than that of the other two (*p* = 0.66 or 0.33) under R^2, *Fair* and *Capacity*. What's more, R^2 improves the performance more than *Fair* and *Capacity* for the reasons described above. The *Fair* scheduler benefits performance more than *Capacity* because *Capacity* schedules the task according to the resource requirement, but lacks resource restriction so that tasks on the same node compete for resources. On the contrary, *Fair* schedules the task according to the required and available resource, consequently competition is infrequent.

As shown in Fig. 3-(b), comparison of energy consumption under various *p* values are consistent with that of execution time. However, the relationship between executed time and energy consumption is not proportional because computer power usage is not stable, but changes with the computer's working state [32]. By CPU frequency scaling technology [33], the computer power usage in the CPU-busy state is higher than in the CPU-idle state. This leads to the following conclusions:

(1) When *p* is 1 or 0, the energy consumption under all schedulers is similar. Energy consumption of CPU intensive tasks (*p* = 1) is much higher than that of I/O intensive tasks (*p* = 0) despite the tasks taking a similar amount of time to complete. This is because CPU intensive tasks place a higher demand on the CPU than I/O intensive task, which causes the CPU to run at a higher frequency and consume more power.
(2) When *p* is 0.66, 0.5 or 0.33, the energy consumption under R^2 is less than that under *Fair* and *Capacity*, but the optimization

effects on energy consumption is not as significant as with the time to complete tasks. This is because R^2 reduces the amount of time that resources are idle and consequently the computer power is higher, which partly counterbalances the optimization to reduce energy consumption.

(3) Due to the lack of CPU intensive tasks, the energy consumption under *FIFO* reduces from $p = 1$ to $p = 0$ because the time to complete the tasks does not change but the computer power usage is lower.

6.3. Optimization

Fig. 4 compares the optimization effect on time to complete tasks and energy consumption (the same with *EE*) under R^2 with those under *FIFO* (a), *Capacity* (b) and *Fair* (c). The cases when $p = 1$ or 0 are not included. In the best situation ($p = 0.5$), R^2 reduce execution time by 17% and energy consumption by 10% in comparison with *FIFO*; R^2 reduce execution time by 14% and energy consumption by 10% in comparison with *Capacity*; R^2 reduce execution time by 10% and energy consumption by 6% in comparison with *Fair*. Thus, the optimization effect of R^2 is evident. In fact, the energy consumption optimization effect should be more significant if the hardware does not provide a power-saving strategy when it is idle.

6.4. Resource utilization

In order to discuss how resources utilization is improved by R^2, Fig. 5 shows the real-time usage (%) of CPU, disk and network of a computing node in the platform. To improve readability, only R^2, *FIFO*, and *Capacity* are compared in the Fig. 4. We set $p = 0.5$, and the time axis is trimmed to 600 s.

In Fig. 5, the horizontal axis represents time and the vertical axis represents real-time resource usage. It can be clearly observed that resource usage of R^2 is higher and more stable. Compared with R^2, resource usage of *FIFO* is sometimes deficient, for example from 280 to 300 s the CPU and disk usage are lower but the network usage is extremely higher, and a similar situation also appears at about 450 s.

6.5. Cost analysis

The overhead of R^2 consists of three parts: *BRR* dedution, task scheduling and resource re-allocation. *BRR* dedution is based on the prior knowledge of a task. We statically analyze the data processing tasks and deduce their *BRR*. On one hand, the additional experiments are performed to collect the throughput of each phase, on the other hand, the reasonable approximation is also accepted. However, the cost *BRR* deduction is not a runtime cost.

The cost of task scheduling, according to the definition 8, is calculation the cosine similarity of the 3-dimensions vectors. The cost of resource re-allocation consists two parts: the calculation of solving Eq. (12), and the implementation of resource restriction. The former one is calculation on matrix, and the later is re-configuration on virtual machine or Hadoop slot [17]. All the calculation mentioned above is performed in parallel. Comparing the out-of-core big data processing task, the cost of task scheduling and resource re-allocation is negligible. The experimental results in Figs. 3 and 4 also show the performance advantage of R^2 with its overhead, even comparing the lightweight *FIFO*, *Capacity* and *Fair* scheduler

7. Application scenario

R^2 approach is suitable for the mixed and sufficient workload, such as the mix of CPU-intensive, disk-intensive and network-intensive workload. It is shown in Section 6.2 that the diversity

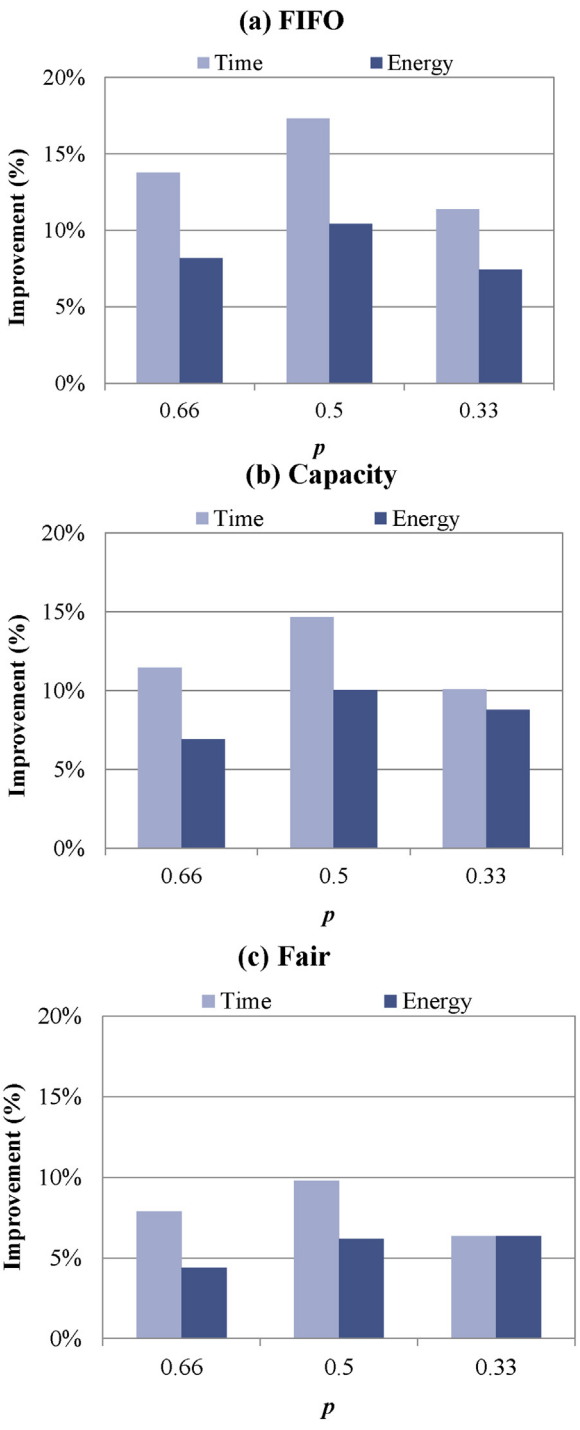

Fig. 4. Optimization effect on execution time and energy consumption (100 tasks).

of resource requirement contributes to solving the resource contention.

The experiments prove the advantages of R^2 approach in homogeneous cluster. As far as the heterogeneous cluster is concerned, R^2 approach is theoretically suitable but the advantages remain to be stuied. For the positive reason, the diversity of resource quantity among nodes in a heterogeneous cluster benefits to the task scheduling algorithm because a task whose *BRR* has more chances to match a node with the remained resources. For the negative reason, a heterogeneous cluster introduce an issue where the resources may be fully utilized, but they are utilized inefficiently, resulting in worse overall performance. Whereas resource

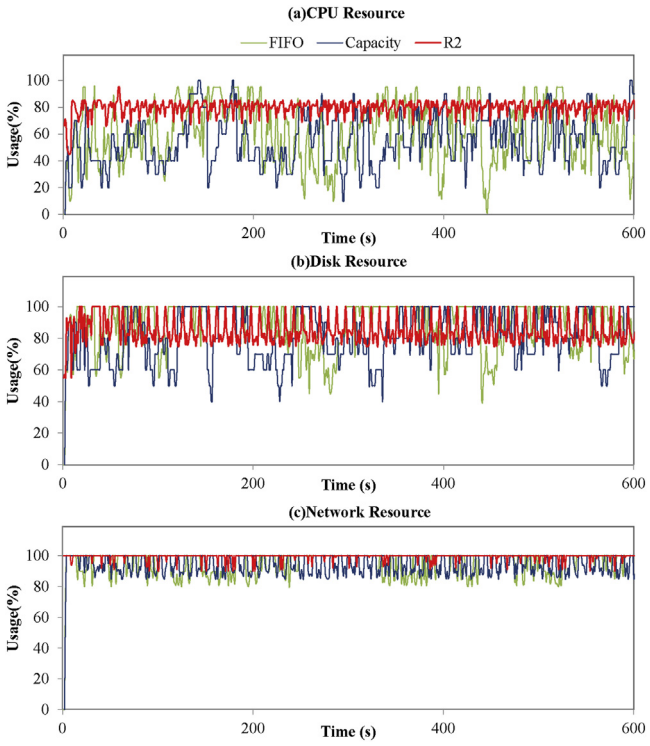

Fig. 5. Comparison of resources utilization with different schedulers(p=0.5).

utilization is not the only criterion for the scheduling. For example, different type of tasks have affinities for which node type they execute more efficiently on, then *BRR* should capture and allow R^2 to efficiently schedule considering these affinities. In this paper, our observations with respect to homogeneous clusters are convinced, however, adapting R^2 approach to the heterogeneous clusters are remained to be challenges.

R^2 approach is suitable for our-of-core data processing tasks in which memory is not a critical resource, but a larger memory is better. In R^2 approach, the memory is the buffer for data exchange. The buffer supports the non-conforming speed (throughput) of providing data and consuming data. Memory is a default resource and not included in *BRR*, during the *BRR* deduction, memory is a parameter in Eq. (9) and (10), during the resource allocation, the memory is allocated to each task averagely. Consequently, the applicable data processing frameworks are out-of-core ones, such as Hadoop, Dremel, or Pregel, but not in-memory ones, such as Spark or GraphX. The MapReduce task is a typical one and be adopted in the experiment.

In Section 6, experiments adopt MapReduce as the computing model because of its simplicity and popularity. However, R^2 approach can also apply to other task types due to both resources and energy efficiency model, *BRR*, task scheduling and resource allocation algorithms are computing model independent. There is no MapReduce specified techniques be mentioned in Sections 3–5. Beside Apache Hadoop, there are also many outstanding frameworks for processing big data, such as Tez [35] and Drill [36]. Tez is aimed at building an application framework which allows for a complex directed-acyclic-graph of tasks for processing data. Drill is designed from the ground up to support high-performance analysis on the semi-structured and rapidly evolving data coming from modern big data applications. R^2 theoretically supports these frameworks. Further, MapReduce is a batch computing model, Tez employs a DAG based computing model, Drill employs MPP based computing model. Due to the variety, integrating R^2 approach with the later two frameworks are our further works.

Finally the aspect of big data processing architecture is discussed, Lambda architecture [34] is a data processing architecture designed to handle massive quantities of data by taking advantage of both batch- and stream-processing methods. Lambda architecture describes a system consisting of three layers: batch processing, speed (or real-time) processing, and a serving layer for responding to queries. R^2 approach can serve for scheduling and energy saving in batch layer.

8. Conclusion

In a big data processing platform, inefficient resource allocation results in poor resource utilization resulting in idle resources which consume energy with no processing benefit, decreasing the *EE* of the platform. *EE* can be optimized by improving resource utilization. For a task, the maximum quantity of allocated resources may not result in the best *EE*. In this paper, we propose the *BRR* based task scheduling and resource allocation algorithms. When the ratio of allocated resources satisfies the task's *BRR*, all resources are fully utilized, so that *EE* is optimal. The main contributions of this paper are as the follows:

(1) Proposal of a resource model, *BRR* and *EE*, which is suitable for data processing tasks. Since *BRR* is a ratio, we unify the units of various resources to MB/s.
(2) Based on the deduction of the *BRR* model, we propose an R^2 approach including task scheduling and resource allocation algorithms for big data processing platforms.
(3) We implemented the *BRR* model and R^2 approach in Hadoop MapReduce, to show the existence of *BRR* and prove that the R^2 approach effectively improves the *EE* for tasks compared with other resource quantity based schedulers.

As mentioned in Section 7, further work includes adapting R^2 approach in the heterogeneous clusters, and integrating it with the different computing frameworks. Furthermore, the deduction of *BRR* should be simplified for practical application of the theory, for example, using computational complexity to simplify parameters in the deduction.

Acknowledgment

This paper is supported by the National Natural Science Foundation of China under Grant No. 61672143, 61433008, U1435216, 61662057, 61502090. The Fundamental Research Funds for the Central Universities under Grant No. N161602003.

Appendix A.

This appendix describes the solving process of Eq. (12) in Section 5. We define that the valid solution x of (12) should satisfy: each element of x is greater than a given positive constant x_0. It means Ax_0 is the minimal resource a task requires. Referring by the rank $R(A)$ of the coefficient matrix A, and the rank $R(A, r)$ of the augmented matrix B=(A, r), (12) has:

Case (1): Unique valid solution iff $R(A) = R(A, r) = m$;
Case (2): Infinite valid solutions iff $R(A) = R(A, r) < m$.
Case (3): No valid solution iff $R(A) < R(A, r)$;
For case (1), the solution is $x = \frac{1}{|A|}A^* r$ and $\Delta = 0$. In addition, Condition (2) is satisfied automatically, and Condition (3) is meaningless due to the unique solution.

For case (2), $\Delta = 0$ and Condition (2) are both satisfied automatically. However, we should find the optimal one from these solutions to satisfy Condition (3). Since the *BRR* of a task is a constant and nor-

malized (see definition 5), and elements in the matrix A are similar, Condition (3) can approximately convert to Condition (3′):

$$\text{Condition (3')}: \text{maximize } \sigma_x = \frac{1}{m}\sum_{j=1}^{m}\left(x_j - \frac{1}{m}\sum_{j=1}^{m}x_j\right)^2.$$

For case (3), we define $Ax \le r$ and $x > x_0$ as (13):

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots a_{1m}x_m \le r_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots a_{2m}x_m \le r_2 \\ \qquad\qquad \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots a_{nm}x_m \le r_n \\ x_1 > x_0 \\ x_2 > x_0 \\ \quad \cdots \\ x_m > x_0 \end{cases} \tag{13}$$

For satisfying the condition (3′) firstly, let $x_1 = x_2 = \ldots = x_m = x'$, then (13) is transformed as follows:

$$\begin{cases} a_{11}x' + a_{12}x' + \cdots a_{1m}x' \le r_1 \\ a_{21}x' + a_{22}x' + \cdots a_{2m}x' \le r_2 \\ \qquad\qquad \cdots \\ a_{n1}x' + a_{n2}x' + \cdots a_{nm}x' \le r_n \\ x' > x_0 \end{cases} \Rightarrow \begin{cases} x'\sum_{j=1}^{m} a_{1j} \le r_1 \\ x'\sum_{j=1}^{m} a_{2j} \le r_2 \\ \qquad \cdots \\ x'\sum_{j=1}^{m} a_{nj} \le r_n \\ x' > x_0 \end{cases} \tag{14}$$

Eq. (14) represents the same issue as the solving of the unitary resource allocation.

References

[1] Performance per watt https://en.wikipedia.org/wiki/Performance_per_watt. 2017.
[2] E. Capra, C. Francalanci, S.A. Slaughter, Measuring application software energy efficiency, IT Prof. 14 (2) (2012) 54–61.
[3] E. Capra, C. Francalanci, S.A. Slaughter, Is software "green"? Application development environments and energy efficiency in open source applications, Inf. Softw. Technol. 54 (1) (2012) 60–71.
[4] A.C. Orgerie, M.D. Assuncao, L. Lefevre, A survey on techniques for improving the energy efficiency of large-scale distributed systems, ACM Comput. Surv. 46 (4) (2014) 47.
[5] F.D. Rossi, M.G. Xavier, C.A.F. De Rose, R.N. Calheiros, R. Buyya, E-eco: performance-aware energy-efficient cloud data center orchestration, J. Netw. Comput. Appl. 78 (2017) 83–96.
[6] T. Mühlbauer, W. Rödiger, R. Seilbeck, A. Kemper, T. Neumann, Heterogeneity-conscious parallel query execution: getting a better mileage while driving faster!, Proceedings of the Tenth International Workshop on Data Management on New Hardware (2014).
[7] A. Roukh, L. Bellatreche, A. Boukorca, S. Bouarar, eco-dmw: eco-design methodology for data warehouses, Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP (2015).
[8] B. Guo, J. Yu, B. Liao, D. Yang, L. Lu, A green framework for DBMS based on energy-aware query optimization and energy-efficient query processing, J. Netw. Comput. Appl. 84 (2017) 118–130.
[9] Y. Zhou, S. Taneja, X. Qin, W.S. Ku, J. Zhang, EDOM: improving energy efficiency of database operations on multicore servers, Future Gener. Comput. Syst. (2017).
[10] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Inf. Sci. 319 (2015) 113–131.
[11] J. Song, T. Li, X. Liu, Z. Zhu, Comparing and analyzing the energy efficiency of cloud database and parallel database, Proc. Adv. Comput. Sci. Eng. Appl. (2012) 989–997.
[12] J. Peraza, A. Tiwari, M. Laurenzano, L. Carrington, A. Snavely, PMaC's green queue: a framework for selecting energy optimal DVFS configurations in large scale MPI applications, Concurr. Comput. Pract. Exp. 28 (2) (2016) 211–231.
[13] H. Zeng, C.S. Ellis, A.R. Lebeck, Experiences in managing energy with ecosystem, IEEE Pervasive Comput. 4 (1) (2005) 62–68.
[14] N.J. Kansal, I. Chana, Energy-aware virtual machine migration for cloud computing-a firefly optimization approach, J. Grid Comput. 14 (2) (2016) 327–345.
[15] A.K. Saha, K. Sambyo, C.T. Bhunia, Integration of DVFS and multi CPUs scaling in a multi-processor, Computer Networks & Automated Verification (EDCAV), 2015 International Conference on Electronic Design (2015).
[16] X. Zhang, B. Hu, J. Jiang, An optimized algorithm for reduce task scheduling, J. Comput. 9 (4) (2014) 794–801.
[17] M. Yong, N. Garegrat, S. Mohan, Towards a resource aware scheduler in hadoop, in: Proc. IEEE International Conference on Web Services, 2009, pp. 102–109.
[18] I. Manousakis, F.S. Zakkak, P. Pratikakis, D.S. Nikolopoulos, TProf: an energy profiler for task-parallel programs, Sustain. Comput. Inform. Syst. 5 (2015) 1–13.
[19] D. Dauwe, E. Jonardi, R.D. Friese, S. Pasricha, A.A. Maciejewski, D.A. Bader, H.J. Siegel, HPC node performance and energy modeling with the co-location of applications, J. Supercomput. 72 (12) (2016) 4771–4809.
[20] K. Leal, Energy efficient scheduling strategies in Federated Grids, Sustain. Comput. Inform. Syst. 9 (2016) 33–41.
[21] Y. Xu, K. Li, J. Hu, K. Li, A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues, Inform. Sci. 270 (2014) 255–287.
[22] F.A. Armenta-Cano, A. Tchernykh, J.M. Cortés-Mendoza, R. Yahyapour, A.Y. Drozdov, P. Bouvry, et al., Min_c: Heterogeneous concentration policy for energy-aware scheduling of jobs with resource contention, Program. Comput. Softw. 43 (3) (2017) 204–215.
[23] T. Zhu, C. Shu, H. Yu, Green scheduling: a scheduling policy for improving the energy efficiency of Fair scheduler, in: Proc. International Conference on Parallel and Distributed Computing, Applications and Technologies, 2011, pp. 319–326.
[24] M. Sheikhalishahi, L. Grandinetti, R.M. Wallace, J.L. Vazquez-Poletti, Autonomic resource contention-aware scheduling, Softw. Pract. Exp. 45 (2) (2015) 161–175.
[25] J. Song, T. Li, Z. Wang, Z. Zhu, Study on energy-consumption regularities of cloud computing systems by a novel evaluation model, Computing 95 (4) (2013) 269–287.
[26] A. Marletta. (2017). CPULIMIT(#46)[Online]. Available:https://github.com/opsengine/cpulimit.
[27] Wikipedia. (2016). Cgroups[Online]. Available:http://en.wikipedia.org/wiki/Cgroups.
[28] Module for Monte Carlo Pi[Online]. Available: http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopimod.html. 2017.
[29] Wordcount Program. Available in Hadoop source distribution: src/examples/org/apache/hadoop/examples/Word-Count. 2017.
[30] Y. Bu, B. Howe, M. Balazinska, M.D. Ernst, The haloop approach to large-scale iterative data analysis, VLDB J. 21 (2) (2012) 169–190.
[31] G. Luo, L. Dong, Adaptive Join Plan Generation in Hadoop, Duke University, Durham NC, USA, 2012.
[32] S. Mittal, Power Management Techniques for Data Centers: a Survey, arXiv preprint arXiv:1404.6681, 2014.
[33] Wikipedia. Dynamic frequency scaling [Online]. Available:http://en.wikipedia.org/wiki/Dynamic_frequency_scaling. 2017.
[34] N. Marz, J. Warren, Big Data: Principles and Best Practices of Scalable Realtime Data Systems, Manning Publications Co., 2015.
[35] Bikas Saha, et al., Apache tez: a unifying framework for modeling and building data processing applications, Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (2015).
[36] M. Hausenblas, J. Nadeau, Apache drill: interactive ad-hoc analysis at scale, Big Data 1 (2) (2013) 100–104.