CSRA: An Efficient Resource Allocation Algorithm in MapReduce Considering Data Skewness

Ling Qi[1,2], Zhuo Tang[1(✉)], Yunchuan Qin[1,2], and Yu Ye[1,2]

[1] College of Information Science and Engineering, Hunan University, Changsha 410082, China
ztang@hnu.edu.cn, qilinghunan@163.com

[2] State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China

Abstract. MapReduce offers a promising programming model for big data processing. One significant issue in practical applications is data skew, its an important reason for the emergence of stragglers which makes the data assigned to each reducer imbalance. This paper presents CSRA, an efficient resource allocation algorithm in MapReduce considering data skew. CSRA aims at reducing the running time and coefficient of variation by reordering the task list and splitting the big clusters. Through thinking over the actual status of tasks, this method largely squares up the resource utilization. After we implement CSRA in Hadoop, the experiments show that CSRA has negligible overhead and can speed up the execution time of some popular applications obviously.

Keywords: MapReduce · Data skew · Splitting

1 Introduction

Large-scale data processing has been gaining more and more attentions in this information society. As a parallel programming model, MapReduce [1] has become a popular tool for distributed data processing. It provides load balancing, data distributing, fault tolerance, resource allocating and job scheduling programming environment for many applications.

MapReduce is a distributed programming framework which allows programmers just to concentrate on the data processing algorithm. Since the parallel controlling works have already been done by the MapReduce system, programmers only need to overwrite map or reduce functions. Hadoop has developed into version 2.0 Yarn [2]. It provides resource management and scheduling service for a lot of applications. But the functions that the system provides are not efficient enough for some problems met by users, for example the problem of data skew.

In the input relations, some $\langle key, value\rangle$ pairs may appear much more often than others, this is called data skew problem. Data skew problem can lead to significantly longer job execution time and lower cluster throughput. What's more, this is invisible and imperceptible for programmers before running the system.

S. Zhang et al. (Eds.): KSEM 2015, LNAI 9403, pp. 651–662, 2015.
DOI: 10.1007/978-3-319-25159-2_59

In this paper, we address a new resource allocation method named Cluster Splitting based Resource Allocation algorithm (CSRA) which focuses on solving the problem of data skew. After studying the source code of Yarn, we know that resource requirements of a task can be described as a 5-tuple: $\langle priority, hostname, capability, containers, relax\_locality\rangle$, denote job priorities, host location of expectation resources, the amount of resources, the number of container and whether relaxation locality. To ranking the tasks properly, we evaluate the process speed considering resources the task consumed and the data finished by each node. By choosing an appropriate host name to allocate resources, the data skew can be avoid largely.

The contributions of this paper include the following:

We put forward an innovative schedule algorithm CSRA to reduce the data skew and improve the resource use efficiency. By considering the resources a task applied, the priority of the task can be calculated much more accurately. At the same time, CSRA implements an innovative approach to balance the workload among the reduce tasks by splitting reduce tasks associated with a single large cluster into multiple. We evaluate the performance of CSRA with some popular and widely used applications. The experimental results show that CSRA can improve the performance of system quite considerably.

The remainder of this paper is organized as follows:

We introduce the background and the causes of data skew in different aspects in Section 2. Section 3 describes task executing time model, task priority model and splitting method used in this paper. In Section 4, we present the implementation details of CSRA. The experimental results and analyses are presented in Section 5. Section 6 concludes the whole paper.

2 Background and Related Works

2.1 Yarn Workflow

Yarn is a new concept that bring up as an upgraded version of MapReduce. The two most important functions that Yarn provides are resource management and scheduling service for many type of applications. There is no more concept of Slots but Container replaced in Yarn. The most typical contribution of Container is that it encapsulates the resources (CPU and Memory two resource categories) on a node. And the node allocates resources based on the amount of resources the task applied.

After users submit an application on Yarn, Application Master will then start. It will run the application in two steps: apply resources for it and then monitor its operation until the entire application is completed.

There are two kinds of applications running on YARN: short application and long application. Short application means some applications which can complete operation and normal exit in a certain time, such as MapReduce task, Tez DAG task. Long application is a kind of never-ending application not surprisingly, usually are some services, such as Storm Service, HBase Service. And as a framework they provide programming interfaces for users.

2.2 Cause of Data Skew

In a Yarn application, data skew may exist in many phases, and the causes of those skews are quite different. Some are caused by the uneven distributed data sets, some are resulted by the node's low efficiency of data executing ability, some are just leaded by the error of the executing code, etc.. After summarizing the paper written by Kwon [3] and Dhawalia [4], we know that there are two typical kinds of data skew which can arise in a Yarn application. We divide them into two categories according to which phase the skew happens.

Sources of Map-side Skew. Users can run arbitrary code as long as it conforms to the MapReduce interface (map or reduce), and typically initialization and cleanup. Such flexibility enabled users to push the boundaries of what map and reduce phases have been designed to do: each map output can depend on a group of input records. Such map task is non-homomorphic. On the other hand, some records require more CPU and Memory to process than others. These expensive records may simply be larger than other records, or the runtime of map may depend on the value of records.

Sources of Reduce-side Skew. As in the case of expensive records processed by map, expensive $\langle key, value\rangle$ pairs can imbalance the runtime of reduce tasks. Since reduce phase operates key groups instead of individual record, the expensive input problem can be more pronounced.

Both kinds of data skew will lead to the inefficiency of system. In this paper, we introduce CSRA to solve the problems caused by data skew.

2.3 Related Works

Based on the advantages MapReduce provides, many data-intensive applications can be easily implemented. And scholars have studied many kinds of advancing methods to improve the performance of MapReduce, such as scheduling to meet deadlines [5], co-scheduling [6] and SkewedJoin in Pig [7]. However, in some papers, users still need to implement their own methods for their specific applications to tackle the data skew problem, such as CloudBurst [8].

Okcan et al. [9] proposes a skew optimization for the join by adding two pre-run sampling and counting jobs. Chen [10] provides a special method to split large clusters in the join and CloudBurst applications (e.g., weighted range partitioning in [3]). Guo [11] and Xu [12] suggest ways to avoid the creation of skew in tasks. The works of Kwon [13] and Guo [14] are based on detecting and mitigating skew dynamically.

All these methods have their advantages, but most of them just focused on one type of solutions. These algorithms can only be used in specific applications and bring non-negligible extra sampling cost. They cannot solve the data skew problem and improve the performance of system as efficiently as the applications need.

3 The Task Executing Time Model for Data Skew

3.1 Data Skewness Model

We know that data skew often comes from the physical properties of objects and hot spots on subsets of the entire domain (e.g., the word frequency appearing on the documents obeys a Zipfian distribution). By varying parameter σ of a data set which following Zipf distribution, we can control the degree of data skew. A common measurement for data skew is the coefficient of variation:

$$COV(\sim X) = \frac{stddev(\sim x)}{mean(\sim x)} \tag{1}$$

where $\sim x$ is a vector that contains the data size processed by each task. $stddev(\sim x)$ is the standard deviation of $\sim x$. Larger coefficient indicates heavier skew.

3.2 Resource Use Efficiency

Here we come up with a new method to divide the resource more efficient by taking the left resource of the node into consideration. The task scheduling algorithm bases on this method performs quite well in some special applications such as Grep, Join, etc..

In this paper, the priority of a task is calculated by executing time and resource use efficiency (RUE) of each node. RUE is decided by the execution time per resource unit (RU) costs when process a certain number of tasks and the efficiency of a node. To calculate RUE, we have to evaluate the process speed and the data finished by each node. Here we suppose RU as a constant value that refers to the sum of CPU and Memory a task applied from a node, defined as Eq. (2):

$$RU = (1 + \alpha \times CPU\_Quantity) \times [1 + (1 - \alpha) \times Mem\_Quantity] \tag{2}$$

where $CPU\_Quantity$ is the unit of CPU, $Mem\_Quantity$ is the unit of Memory. α reflects the tradeoff between CPU and Memory where the value is between 0 and 1. And the value is setted based on the job being tested.

Once a task is scheduled, the resource then is determined and unchangeable. We can get the size of resource (R_apply) and calculate the number of resource unit (N_RU) this task required using Eq. (3):

$$N\_RU = \frac{R\_apply}{RU} \tag{3}$$

As the RUE is a dynamic element, to calculate the value more accurately, we choose a model to predict it which is related to the past station. There are many prediction models in other papers, such as Exponentially Weighted Moving Average (EWMA) [15] and Markov Chain Monte-Carlo Particle Filter (MCMC

PF) [16]. In this paper, we choose the EWMA scheme which can be expressed as Eq. (4):

$$E\_RUE(t) = \beta \times RUE(t) + (1-\beta) \times E\_RUE(t-\bigtriangledown), 0 \leq \beta \leq 1 \tag{4}$$

where $E\_RUE(t)$ stands for estimated resource use efficiency and $RUE(t)$ stands for observed resource use efficiency at time t. β reflects the tradeoff between stability and responsiveness. We set the value of β as 0.2 in this paper. By analyzing the definition and characteristic of RUE, the value of RUE at time t can be calculated as Eq. (5):

$$RUE(t) = \frac{Data\_finish}{(t-t_0) \times N\_RU} \tag{5}$$

where $Data\_finish$ refers to the finished data from current time t to start time t_0. N_RU is the resources being occupied. As we can see, the more data the node output, the bigger will the value of RUE be.

To divide the resources, Hadoop monitors progress tasks need to select an appropriate task who has the highest priority. Here, we use the current executing time of a task represents its priority. As real time process speed of a node will cause calculating delay, we use the average process speed to estimate the executing time like Eq. (6):

$$exe\_time[i] = \frac{Data[i]}{R\_apply \times \overline{RUE}} \tag{6}$$

where R_apply is the resource the task applied from the node. As we can see, the key point of $exe\_time[i]$ is to find a node with the highest $\overline{RUE}$.

3.3 Splitting Large Intermediate Cluster

If applications treat each intermediate cluster with the same $\langle key, value \rangle$ pairs independently in reduce phase, this can be quite improper. Enabling cluster splitting will then have a profound impact on data skew mitigation.

Considering the REU we have analyzed before, we provide an effective cluster splitting strategy and modify the partition decision include both the partition keys and the partition size. That is a partition decision record $(k; s)$. It means that one of the partition point is s of key k. Before reducer reads the partition, the monitor compares the partition size s with $\frac{data}{num\_r \times RUE}$. Eq. (7) shows the splitting data size level.

$$R\_data = \begin{cases} R\_data & ; s \leq \frac{data}{num\_r \times RUE} \\ \frac{data}{num\_r \times RUE} & ; s > \frac{data}{num\_r \times RUE} \end{cases} \tag{7}$$

here num_r is the number of reducers. For a reducer, if there is data in the cluster to be read, it only read $\frac{data}{num\_r \times RUE}$ keys and the remaining keys left for other reducers to process. R_data' means the data that have not been executed calculated as Eq. (8):

$$R\_data' = R\_data - \frac{data}{num\_r \times RUE} \tag{8}$$

4 Resource Allocation Algorithm

4.1 The Task Priority Based on Executing Time

We have introduced a method to estimate the executing time. Here we define a priority model for a task as $P < L, T >$, where L stands for level and T stands for *exe_time*. In this paper, we set $L = 1$ for failed task, which gets the highest priority, $L = 2$ for unscheduled tasks, which gets the average priority, $L = 3$ for speculative executed task, which gets the lowest priority. To improve the speculation performance in heterogeneous environment, we use Eq. (9) as a task's speculative execution priority determination method.

$$specPriority = (1 - progress)/progressRate \tag{9}$$

If there is any node which has free Container to apply for tasks, the tasks waiting online obey the 2-tuples priority determination method $P < L, T >$. The specific process is given in the following Task_Priority algorithm.

Algorithm 1. Task_Priority algorithm

```
Input:
    $N$: the tasks collection;
    $P$: the processors collection.
Output:
    $P < L, T >$: the priority of tasks.
 1: for each $i \in \underline{[1, n]}$ do
 2:     Calculate $\overline{exe\_time[i]}$;
 3:     Get the execution status task $N[i]$;
 4:     if The $N[i]$ is judged to be failed task then
 5:         Set $L[i] = 1$;
 6:     else if $N[i]$ is judged to be unscheduled task then
 7:         Set $L[i] = 2$;
 8:     else if $N[i]$ is judged to be speculative task then
 9:         Set $L[i] = 3$;
10:     end if
11:     Range $L[i]$ with increasing order;
12:     Rearrange the order with the same $L[i]$ in nondecreasing based on the $\overline{exe\_time[i]}$
13:     Update $P < L, T >$
14: end for
15: return $P < L, T >$.
```

Here the executing time of a task is not only determined by the data size and the process speed of node, but also determined by real-time resource utilization of the node which improves the performance of the system quite marked. For example, each node has its own real-time E_RUE. If E_RUE of a node is quite low while it has enough spare resource for other tasks to apply. Once it applies the tasks successfully, compared with other nodes whose E_RUE in real-time are much higher, it will lead to a long execution time. So we should judge if the apply should be accepted.

4.2 Resource Allocation Algorithm

Container Allocator (CA) is a responsible module for applications, and the allocation of resources. In Yarn, resource requirements of the job can be described as a 5-tuple. By resetting the priority and choosing a best fit node, we can schedule the tasks more effectively and avoid data skew.

Algorithm 2. CSRA algorithm

```
Input:
    $N$: the tasks collection;
    $P$: the processors collection;
    $P < L, T >$: the priority of tasks.
Output:
    Using CSRA to get a proper resource allocation method.
 1: while $P < L, T >$ is not NULL do
 2:    if $R\_data[i] > \frac{data}{num\_r \times RUE}$ then
 3:       $R\_data[i] = \frac{data}{num\_r \times RUE}$; //split the big cluster;
 4:       $R\_data[i]^{'} = R\_data[i] - \frac{data}{num\_r \times RUE}$;//make the left data a new task;
 5:       $n{+}{+}$;
 6:       $R\_data[n] = R\_data[i]'$;
 7:       $s = n$;
 8:    end if
 9: end while
10: Get a new task list $L[i]$
11: for each $i \in [1, s]$ //i stands for task i do
12:    for each $j \in [1, n]$ //j stands for processor j do
13:       Calculate the $E\_RUE$ of each node ;
14:       Get a list ranks based on $E\_RUE$ in descending $P[i]$;
15:       Chose the first processor P[t] with the highest $E\_RUE$ ;
16:       if $R\_apply[j] < R\_left[j]$ then
17:          Divide L[i] to P[t];
18:       else
19:          t++;
20:       end if
21:    end for
22: end for
23: return A proper resource allocation method.
```

CA divides all the tasks into three kinds. They are failed map task, map task and reduce task. The priorities that CA gives to them are 5, 10 and 20. That is to say, if these three kind of tasks apply resource simultaneously, CA will first assign resource to the failed map task. Similar with this resource allocation mechanism, we fine-tune the first element of this 5-tuples: priority by setting it variable instead of constant.

We know that the resource in a node is constant once a job is submitted. Here we can calculate the resource left in a node like Eq. (10):

$$R\_left(t) = R\_total - \sum_{i=0}^{n} R\_used(t_i) \times [u(t_i) - u(t_i - t_{i,0})], n = 0, 1, 2, \cdots \tag{10}$$

As we all know that the resource left at time t is equal to the total resource minus the used resource. In Eq. (10), $R\_left(t)$ stands for the resource left in time t, R_total means the total resource on the node and $R\_used(t_i)$ means the resource being divided to task i, $u(t_i) - u(t_i - t_{i,0})$ is a phase signal and $R\_used(t_i) \times [u(t_i) - u(t_i - t_{i,0})]$ means the resource being occupied by task i in $t_{i,0} < t_i < t$ and released on another time. n stands for the number of tasks who apply resources on the node.

According to all the analysis demonstrate above, we design CSRA to allocate resources. The whole specific processes are shown in algorithm 2.

5 Experiments and Result

In this section, we evaluate and compare the performance of CSRA with traditional scheduling algorithms, Hadoop hash and Hadoop range, following those three applications: Sort, Join and Grep. As Zipf distribution typically expresses the idea of data skew and it is quite common in the data coming from the real world, the data we used in the experiments follow its feature.

The comparisons of the experiments are based on the following two performance metrics:

Average Execution Time. The execution time of an algorithm is its running time for obtaining the output schedule of a given task graph. Among all the three algorithms, the one who gets the minimization average execution time is the one most practical implementation.

Coefficient of Variation. We compute the coefficient of variation in data size across reduce tasks to measure the effectiveness of skew mitigation. We compute the coefficient of variation in data size processed by different reducers. The smaller the coefficient is, the litter will data skew on reducer be.

5.1 Experimental Settings

The hardware configuration in our experiment is 15 servers and each server contains dual-Processors (2.4GHz Xeon E5620), 24GB of RAM and two 150GB disks. We set up our Hadoop 2.4.0 (Yarn) cluster on those severs which are connected by 1Gbps Ethernet (the nodes within a rack are connected through a single switch) and managed by CloudStack which is an open source cloud operating system. We use a virtualization software KVM to construct medium sized VMs with 2 virtual cores, 4GB RAM and 30GB of disk space.

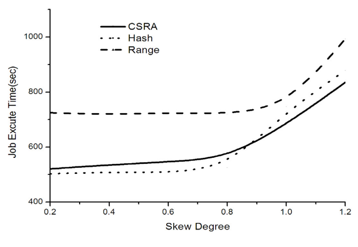
We compare the performance of the three algorithms in various data skew degree, different sizes of data and different output percentage in Grep. The every following experiments conducted a detailed analysis of each group.

5.2 Estimate in Various Data Skew Degree

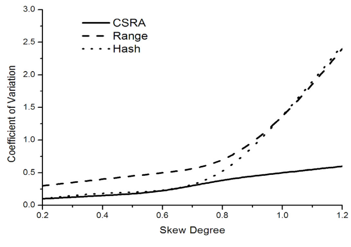
In the first set of experiments, to compare the performance of CSRA with the original hash and range algorithms in Hadoop, we randomly generated 10GB synthetic data following Zipf distribution. Without loss of generality, we take Join as the application. As Join is a reduce-input heavy application, choosing it can obviously highlight the superiority of splitting phase in CSRA. The results have been given in Fig. 1.

Fig. 1(a) depicts the average execution time in different data skew. When σ is relatively low, the execution time of CSRA and Hash is almost the same. But once σ further increased, their gap become obviously. As we can see, the execution time increased substantially when the degree of the skew reaches to a certain threshold 0.8. This phenomenon indicates that when the value of σ is small, it does not have many differences between Hash and CSRA. But when σ increased from 0.8 to 1.0, the growth rate for Hash is 38.6% and 22.0% for CSRA. While Range doesn't increase obviously.

Fig. 1(b) shows the impact does σ have on coefficient of variation. As shown in the picture, the increase of σ has the least impact on CSRA compared with the other two algorithms. Coefficient of variation increases substantially once the degree of skew reaches to a certain threshold. The reason that the two strategies in Hadoop performs worse than CSRA is that they do not detect or split large intermediate clusters.



(a) Job execution time.



(b) Coefficient of variation.

Fig. 1. Performance in various data skew degree.

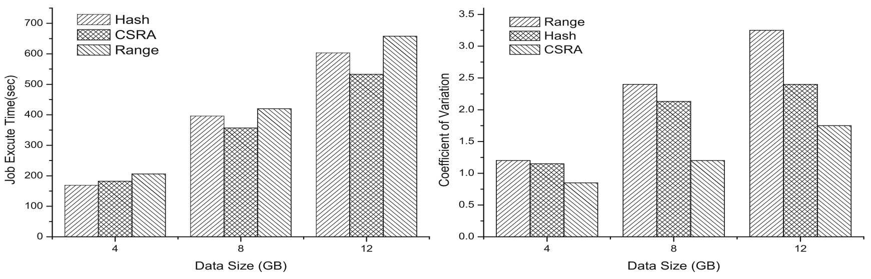
From this experiment, we can conclude that the overhead of CSRA is negligible even in the absence of skew ($\sigma = 0.2$).

5.3 Estimate in Different Data Size

For an efficient comparison, we estimate the effect of data scale. The size of data used in our experiments are varied from 4GB to 12GB, the value of data skew degree is $\sigma = 1.2$.

Fig. 2(a) shows the great superiority of CSRA considering execution time compared with the other two algorithms. This result becomes prominent when the data size is 12GB. The time CSRA saved is almost 23.5% compared with Range. When the data set is in a small size, the perform of CSRA shows litter superiority than Hash and Rang as the function of cluster splitting phase does not play an effective function, but task priority ranking method still play a light role.

As shown in Fig. 2(b), with the increasing of data size, the coefficient of variation here shows considerable gap among all of these three algorithms. The reason is that both Hash and Range allocate resources evenly without considering the actual feature of data. When data skew situation appeared non-negligible, both the execution time and the coefficient of variation performance are not good. For CSRA, cluster splitting phase makes the impact of data skew to the lowest level.



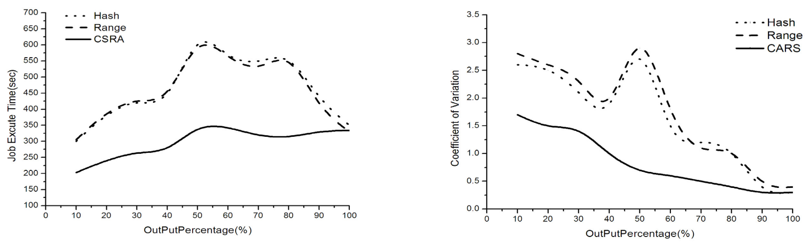
(a) Job execution time. (b) Coefficient of variation.

Fig. 2. Estimate in different data size.

5.4 Estimate in Different Output Percentage in Grep

As we all know that Grep is a command-line utility for searching plain-text data sets for lines matching a regular expression. In order to make the output percentage varies from 10% to 100%, we change the search expression deliberately.

Fig. 3 shows the change of the job execution time and the coefficient of variation when the output percentage increase. To run this application, we randomly generated 10GB synthetic data following Zipf distribution ($\sigma = 1.0$) as usual.



(a) Job execution time. (b) Coefficient of variation.

Fig. 3. Evaluate in different output percentage.

Fig. 3(a) shows the big difference of those algorithms considering the execution time with increasing output percentage. We can see the disparity among those three algorithms. When the output percentage is 50%, they meet their biggest gap. CSRA saves about 78% execution time compared with Hash. It is own to the good performance of cluster splitting and a efficient task priority ranking method. The execution time shows less difference when the output percentage become bigger than 90%, as the application at this moment has very small data skew. The data divided to all the reducers almost evenly.

As we can see from the Fig. 3(b), CSRA performs significantly better when the output percentage is low. When searching unpopular words in the input files like generate results with heavy data skew, CSRA has a considerable advantage over the other two algorithms. When the output percentage is high, the resulting data become more evenly distributed and the performance difference becomes smaller.

6 Conclusion

Data skew alleviation is important in improving the performance of Yarn. This paper has presented CSRA, an algorithm that implements an innovative skew mitigation strategy to improve the performance of Yarn system. The unique features of CSRA are its task priority ranking and supports for cluster splitting. CSRA can handle data skew, but the system reliability should be considered in the future. Performance evaluation demonstrates that the performance improvement of CSRA is significant, and it can adapts to using in various parallel applications on heterogeneous environment.

Acknowledgments. This work is supported by the Key Program of National Natural Science Foundation of China (Grant No. 61133005, 61432005), National Natural Science Foundation of China (Grant Nos. 61370095, 61572176), and Open Foundation of State Key Laboratory of Software Engineering (SKLSE2012-09-18).

References

1. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: Communications of the ACM - 50th anniversary issue, **51** (1), pp. 107–113. ACM, New York (2008)
2. Introduction for Yarn. http://en.wikipedia.org/wiki/Yarn
3. Kwon, Y., Balazinska, M., Howe, B., Rolia, J.: Skewtune: mitigating skew in mapreduce applications. In: SIGMOD 2012 Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 25–36. ACM, New York (2012)
4. Dhawalia, P., Kailasam, S., Janakiram, D.: Chisel: A resource savvy approach for handling skew in mapreduce applications. In: IEEE Sixth International Conference Cloud Computing (CLOUD), pp. 652–660. IEEE Press, Santa Clara (2013)
5. Kc, K., Anyanwu, K.: Scheduling hadoop jobs to meet deadlines. Cloud Computing Technology and Science (CloudCom). In: IEEE Second International Conference, pp. 388–392. IEEE Press, Indianapolis (2010)
6. Polo, J., Carrera, D., Becerra, Y., Torres, J., Ayguad, E., Steinder, M., and Whalley, I.: Performance-driven task co-scheduling for mapreduce environments. In: Network Operations and Management Symposium (NOMS), pp. 373–380. IEEE Press, Osaka (2010)
7. Gates, N., Chopra, S.: Building a high-level dataflow system on top of map-reduce: the pig experience. Proceedings of the VLDB Endowment, vol. 2, no. 2. (2009)
8. Schatz, M.: Cloudburst: highly sensitive read mapping with mapreduce. In: Proceedings of the VLDB Endowment on Bioinformatics, vol. 25, no. 11. pp. 1363–1369. ACM New York (2009)
9. Okcan, A., Riedewald, M.: Processing theta-joins using mapreduce. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 949–960. ACM. New York (2011)
10. Chen, Q., Yao, J., Xiao, Z.: Libra: Lightweight data skew mitigation in mapreduce. In: IEEE Transactions on Parallel and Distributed Systems, pp. 1–14 (2014)
11. Guo, Z., Fox, G.: Improving mapreduce performance in heterogeneous network environments and resource utilization. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster. Cloud and Grid Computing (Ccgrid 2012), pp. 714–716. IEEE Press, Ottawa (2012)
12. Xu, Y., Kostamaa, P.: Efficient outer join data skew handling in parallel dbms. Proceedings of the VLDB Endowment **2**(2), 1390–1396 (2009)
13. Kwon, Y., Balazinska, M., Howe, B., Rolia, J.: Skew-resistant parallel processing of feature-extracting scientific user-defined functions. In: Proceedings of the 1st ACM symposium on Cloud computing, pp. 75–86. ACM. New York (2010)
14. Guo, Z., Pierce, M., Fox, G., Zhou, M.: Automatic task reorganization in mapreduce. In: 2011 IEEE International Conference Cluster Computing (CLUSTER), pp. 335–343. IEEE Press, Austin (2011)
15. Domangue, R., Patch, S.: Some omnibus exponentially weightedmoving average statistical process monitoring schemes. Technometrics **33**(3), 299–313 (1991)
16. Bardet, F., Chateau, T.: Mcmc particle filter for real-time visual tracking of vehicles. In: 11th International IEEE Conference Intelligent Transportation Systems (ITSC), pp. 539–544. IEEE Press, Beijing (2008)