Toward Scheduling I/O Request of Mapreduce Tasks Based on Markov Model

Sonia Ikken[1,2(✉)], Éric Renault[1,2], M. Tahar Kechadi[3], and Abdelkamel Tari[4]

[1] Institut Mines-Télécom – Télécom SudParis, Évry, France
[2] Laboratoire Samovar UMR CNRS 5157, Évry, France
{sonia.ikken,eric.renault}@telecom-sudparis.eu
[3] UCD School of Computer Science and Informatics, Dublin, Irlande
[4] University of Abdarahmane Mira, Bejaia, Algeria
tahar.kechadi@ucd.ie, tarikamal59@gmail.com

Abstract. In Cloud storage of multiple CPU cores, many Mapreduce applications may run in parallel on each compute node and collocate with local Disks storage. These Disks storage are shared by multiple applications that use full CPU power of the node. Each application tends to issue contiguous I/O requests in parallel to the same Disk; however if large number of Mapreduce tasks enters the I/O phase at the same time, the requests from the same task may be interrupted by the requests of other tasks. Then, the I/O nodes receive these requests as non-contiguous way under I/O contention. This interleaved access pattern causes performance degradation for Mapreduce application, this is particularly important when writing intermediate files by multiple tasks in parallel to the shared Disk storage. In order to overcome this problem, we have proposed approach for optimizing write access for Mapreduce application. The contributions of this paper are: (1) analyze the open issues on scheduling access request of Mapreduce workload; (2) propose framework for scheduling and predicting I/O request of Mapreduce application; (3) describe each role of component that intervenes in the scheduling theses I/O request on Block-level of storage server to provide contiguous access.

Keywords: Mapreduce · Cloud storage · Disk I/O · Markov model · Scheduling algorithm

1 Introduction

Cloud computing has become a viable, mainstream solution for data processing, storage and distribution. A cloud environment allows sharing of distributed resources, like CPU, Network and Disk based on virtual machine, these resources must be used efficiently for performing data intensive application. Cloud storage systems use modern server nodes based on Disk storage that contains multiple Disk drives and buffers memory. These typically have many CPU cores; it is

S. Boumerdassi et al. (Eds.): MSPN 2015, LNCS 9395, pp. 78–89, 2015.
DOI: 10.1007/978-3-319-25744-0_7

desirable to use increased level of parallelism on each node to be able to use the full CPU power of the node. In distributed computing such as those running in Mapreduce [1] and using its open source implementation Hadoop [2] or similar model are often dominated by I/O bound, particularly reading and writing operations conducted on Disk storage that are limited than the CPU resources, theses Disk storage are shared by multiple processes or number of parallel tasks (or job) and alternate between computation and I/O phases.

A Mapreduce application consists of many maps and reduces tasks that read and write data on distributed file system and local Disks. Each map task has a memory buffer that it writes the output to. The buffer is 100 MB by default, it is a size which can be tuned by changing the Hadoop parameters. Each time the memory buffer reaches a certain threshold, then a new intermediate file is created.

In order to use multi-core systems, Hadoop schedules multiple tasks to run simultaneously on each node. In this case, the I/O resources of node are divided between these tasks and are exposed to I/O contention. Particularly, this situation affect write request of parallel tasks which remains on each Buffer befor reaching the threshold to the shared Disk storage. For example, parallel tasks slot often write intermediate data from each buffer to the Disk, each task tends to issue contiguous write I/O requests, but if multiple tasks issue many I/O requests managed by buffers simultaneously in one node, the requests are handled by local Disk in a non-contiguous way. This interleaved access pattern increases read I/O cost and I/O bandwidth falls, and can cause intermediate data fragmentation in many local file system. This means that Hadoop does not have a control over the allocation of physical blocks for intermediate files, and it can not effectively manage all buffers of multiple tasks, but is dependent on the kernel I/O scheduler and allocation strategy used by file system driver.

Particularly those that do not support optimization used at the OS kernel (Buffer memory policy, scheduling queue Disk) [3,4] cannot take advantage of the properties and behaviors of each particular application and are therefore not able to address the overall efficiency of the system. As the size of the system continues to increase, planners must have a global view of I/O operation needs of all applications in order to make appropriate allocation decision.

In order to solve this problem, we have proposed a framework for scheduling I/O request of parallel Mapreduce task, and Markov model [5] for predicting non contiguous write access to improve I/O access of Mapreduce task on shared Disks.

The remainder of the paper is organized as follows. In Sect. 2, we outline the related work and open issues that directs our research. In Sect. 3, we describe the proposed approach for scheduling write access of Mapreduce application, and the role of each component in this framework. Finally, in Sect. 4 we conclude and summarize our plans for future work.

2 Motivation Example

In this section we show an illustrative example for the restraints of the problem of Disk I/O contention during Mapreduce workload processing. The effect of I/O

contention is dependent on the workflow and I/O patterns used by the Mapreduce application under I/O phases. During Mapreduce workload execution, map and reduce tasks go through a number of phases of execution. If multiple tasks are running at the same time, it is not guaranteed that all those tasks are executing the same phase at the same time as long as each task execute his own I/O phase and form a life cycle throughout the process Mapreduce. However, the tasks are not always executing the same phase is that tasks are typically not identical in duration, but even for tasks that are normally very similar in execution time this situation can occur because of the variance in task execution time caused by I/O contention that are observed in Fig. 2.

During the shuffle phase, reduce task input is divided into segments, with one segment being read from the output files of each map task spread over the entire cluster. While each segment itself is read sequentially, the set of segments for a particular reduce task is not stored sequentially and not guaranteed to be read in any particular order. This situation occurs when writing intermediate data to the shared Disk. For the spill phase if large number of tasks enters the I/O phase at a same time, the requests from the same task may be interrupted by the requests of other tasks. Then I/O nodes receive these requests as non-contiguous way under I/O contention.

3 Background

3.1 Mapreduce Programming Model

Mapreduce [1] uses a divided-and-conquer approach in which input data are divided into fixed size units processed independently and in parallel by tasks, which are executed distributedly across the nodes in the cluster.

Mapreduce applications consists of a map function and a reduce function. As shown in Fig. 1, the input to an application is organized in records, each of which is a < k1, v1 > pair. The map function processes all records one by one,

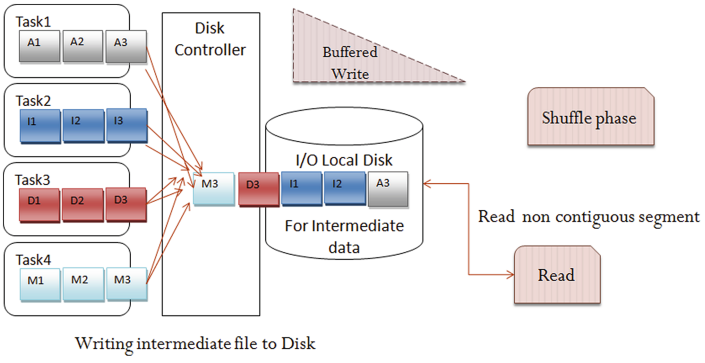

Fig. 1. Disk sharing issue

and for each record outputs a list of zero or more < k2, v2 > records called intermediate data. The map output is stored in an in-memory buffer; when this buffer is almost full then it start (in parallel) the spilling phase in order to remove data from it to the Disk. Then all intermediate data with < k2, v2 > records are collected and reorganized so that records with the same keys (k2) are put together into and shuffled a < k2, list (v2) > record. These < k2, list (v2) > records are then processed by the reduce function one by one, and for each record the reduce function outputs a < k2, v3 > pair. All < k2, v3 > pairs together coalesce into the final result. Map and reduce functions can be summarized in the following equations:

$$\text{map}(< \text{k1, v1} >) \rightarrow \text{list}(< \text{k2, v2} >) \quad (1)$$
$$\text{reduce}(< \text{k2, list(v2)} >) \rightarrow < \text{k2, v3} > \quad (2)$$

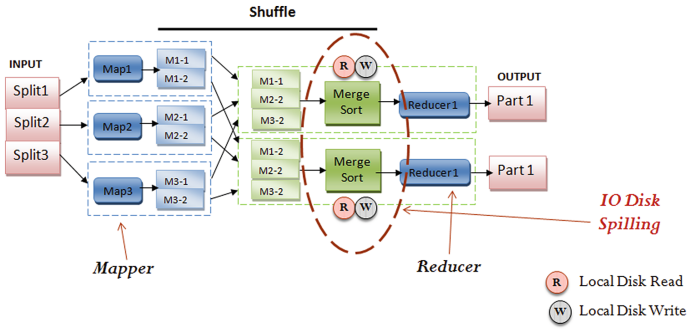

Fig. 2. Structure of map and reduce task in Hadoop

3.2 Linux Disk Scheduler

Disk scheduler are typically work-conserving, since they select a request for services as soon as (or before) the previous request has completed. Since 2.6 version Linux provides four non-conserving Disk I/O schedulers: deadline, anticipatory, noop, and completely fair queuing (CFQ), along with an option to select one of these four at boot time or runtime. The selection is based on a priori knowledge of the workload, file system, and I/O system hardware configuration, among other factors. The anticipatory scheduler (AS) is the default for 2.6 version. Now in our context, we consider process issuing Disk read request synchronously which each process issues a new request shortly after its previous request has finished, and thus maintains at most one outstanding request at any time. "Deceptive idleness" is a situation where a process appears to be finished reading from the Disk when it is actually processing data in preparation of the next read operation. This will cause a normal work-conserving I/O scheduler to switch to servicing I/O from an unrelated process. This situation is detrimental to the throughput of synchronous reads, as it degenerates into a seeking workload. In this

situation the Anticipatory scheduler [8] performs well, and it is based on two assumptions: (1) synchronous Disk requests are issued by individual processes and, thus, anticipation occurs only with respect to the process that issued the last request; and (2) for anticipation to work properly, the anticipated process must be alive; if the anticipated process dies, there is no further anticipation for requests to nearby sectors. Instead, any request that arrives at the scheduler is scheduled for dispatch, irrespective of the requested head position and the current head position.

4 Related Work and Open Issues

There are lots of work on Modeling access behavior and scheduling Mapreduce applications in Cloud environment. We classify the current trends and exiting approaches with respect to traditional scheduling algorithm, and presents open issues that direct our research.

4.1 Modeling I/O Behavior of Mapreduce Workload Without I/O Contention

Although in recent years, there has been an increasing amount of work in modeling the I/O behavior of Mapreduce workload. In [9–14] the authors focusing to reduce the total amount of I/O performed by the application. In [14], the authors model the read I/O behavior of map tasks. In [12], the authors propose a statistical model to evaluate the effect of various configuration parameters of Hadoop-Mapreduce job.

4.2 Scheduling Task, Job and VM Under I/O Contention

The authors propose techniques to address the I/O stream contention in Mapreduce tasks. In [15], they propose to limit the number of concurrent I/O streams, and alleviate the I/O contention by orders the I/O streams in accordance to job priority. In [16], they propose competing applications' I/O by interposing HDFS I/O and use an SFQ-based proportional-sharing algorithm. In [17–19] the authors characterize and predict I/O performance under I/O contention, which focuses primarily on the hardware environment. In [20], they propose a scheduling system that takes I/O contention into account, but it applies to VM scheduling rather than I/O access scheduler.

4.3 Scheduling I/O Access Disk Without I/O Contention

Each I/O scheduler algorithm is performed at the top level, which the I/O request reordering and merging is performed by the filesystem driver, and at the lower level, which the I/O requests are reordered by the I/O device. Disk scheduler plays main role in the service of I/O operation, in [7,21] they trie to optimize the movement of the Disk head to specific goals under limit condition, but each has

difficulties of predicting these movements. Recently, non-conservation works of Disk schedulers, such as CFQ scheduler [7] and Anticipatory scheduler [8] were designed to record the spatial location with concurrent services of interleaved requests from multiple processes. This strategy takes the Disk head slowed after serving a request from a process until the next request from the same process happens or waiting threshold expires.

4.4 Open Issues

Our researches are focuses to analyze the problem of scheduling intermediate data for parallel Mapreduce tasks under I/O contention. Our work differ from others existing approaches, especially in modeling access behavior and scheduling Mapreduce application on block level of shared Disks nodes.

(1) The works we've cited above, make effort on modeling the I/O performance of Mapreduce, the impact of I/O contention on access request is not considered.

(2) The works presented above about mitigating I/O contention focuse to coordinate Mapreduce application at high level, rather than at block level.

(3) Unlike traditional Disk scheduler problems, scheduling I/O workload for Mapreduce application is even more complicated, and it should meet the algorithms of access requests to the Disk, and they take into account information related to the type of workload behavior and traditional Disk scheduler policy.

Table 1, depicts the summarize of the analysis and comparison of the related work based on the criteria that characterizes our orientation.

Table 1. Feature comparison of related work

Related work	Mapreduce workload	Modeling access behavior	I/O contention	Block level scheduler	High level scheduler
[9–14]	X	X			
[15,16,19,20]	X	X			X
[7,8]				X	
Our focus research	X	X	X	X	

5 Proposed Approach

We have proposed an approach that defines a framework to anticipate non-contiguous write request of Mapreduce workload on shared Disk storage. Mapreduce phases are sequential write-only and read-only subtask, and write access is done in parallel using a separate thread from in memory buffer to local Disk in round-robin fashion. We develop a methodology to characterize the interleaved

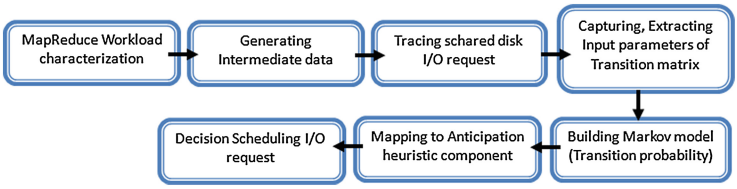

Fig. 3. A Framework for Scheduling write access of parallel MapReduce Tasks

write access submitted from parallel Mapreduce task to the shared Disk, and uses anticipation support to decided when these I/O request will be allocated on the Disk to avoid non-contiguous access, Fig. 1 show the proposed approach. On our study, we need to modeling write access at block-level to make out the non-contiguous write I/O for guiding future scheduler decision to improve access of Mapreduce tasks.

5.1 Markov Model Prediction

To capture non-contiguous write I/O requests, we have model parallel write access streams for Mapreduce tasks under IO contention using Markov model; the choice of model description is critical to its predictive ability. One must determine what application behavior corresponds to a state s, the total number of states N that can be fully described by its transition probability matrix P, and the allowed observations. Given the present state and all past states, if the future state of the system depends only on the present state, the system is said to have the Markov property. In the context of Disk access request for Mapreduce task slot, one can build a transition probability matrix by sampling the state of the Disk system at regular intervals during Mapreduce job processing. We generate I/O trace from Mapreduce workload processed on same Disk storage. We construct a Markov model where each state corresponds to write I/O request at time t. Each state can have one or multiple part of non-contiguous I/O request related to one task; that means the state can be contiguous or non-contiguous write I/O request of a single task. This is because; request can be interrupted by the request from other task in Mapreduce sub phases. The non-contiguous write I/O requests are sequential access on block regions of variable size. Ideally, write request size is chosen to correspond to a split size from HDFS (Hadoop

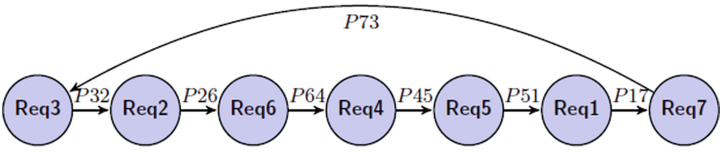

Fig. 4. Markov chain of order m describing contiguous Write I/O request

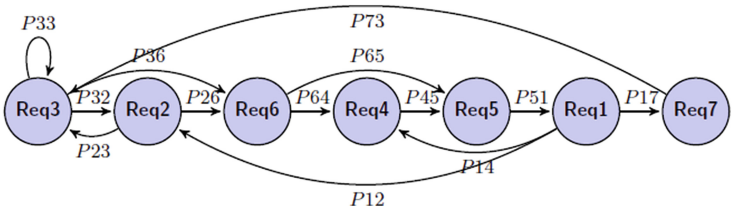

Fig. 5. Markov chain of order m describing Interleaved Write request under I/O contention

Distributed FileSystem). In the context of this paper, we assume that there is only one phase cycle of Mapreduce, then a request size may be more than or equal to 64 Mbyte, and the Hadoop parameters are fixed for each training set of action-sequences. Therefore, the state number is the number of write I/O request of parallel task slot, and If one tasks slot has 64 Mbyte intermediate data size, access to this data region could be modeled by a 64 states Markov model with different sizes of each access.

Observations are write request under I/O contention that changes the current state with some probability to a new state that reflects a new current non-contiguous write request; this is the interleaved write access of each task slot. We process a trace of I/O request so those contiguous write requests that access sequentially the current block region on Disk does incur reflexive transition. Figure 4 illustrates the write access pattern which is deterministic; each task slot tends to issue contiguous write I/O access, but Fig. 5 illustrates a pattern where there is an interleaved write request of two or more task slot since multiple task issue many write I/O request simultaneously on Datanode. This interleaved access pattern can cause intermediate data fragmentation, increases read IO overhead (Fig. 5).

5.2 Scheduling Non-contiguous Write Access Based on Markov Model

Many Disk scheduler algorithm have been proposed that achieved performance applications by taking into account information about characteristic of each individual I/O request and the current state of Disk subsystem [6,7]. In this paper, we use the Anticipatory Disk scheduler [8]. The idea behind this algorithm is to anticipate which streams are most likely to make their deadlines for synchronous read and which are not, based on the estimated supply and demand of time slots in the future. In some situation, the scheduler can cause undesirable delays to IO requests in other environments, particularly when there is update or delayed write like Mapreduce phases. Therefore, fast phase read response is disrupted by interfering writes request in Mapreduce phases. For this situation our idea is having control of non-contiguous write request to improve read access from Mapreduce phases, and use Markov model for predicting these non-contiguous write request.

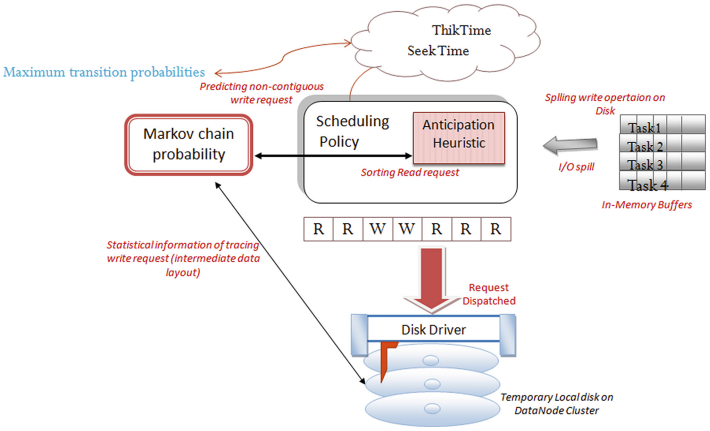

Fig. 6. Adaptive anticipatory scheduler based on Markov model

Then, we propose an adaptive Anticipatory scheduler based on Markov model. The challenge of our heuristic is to know that already serviced requests will have other sub-requests and how to estimate the think time for each read and write requests of all task. Given a write I/O access Markov model, it still does not want to delay write requests indefinitely, however, the buffer ensure that data is eventually written to Disk to prevent in-memory buffers from growing too large or reached a threshold. Also, there is I/O access which is contiguous or proximate requests, and it is not necessary to wait for these requests to serve immediately. This optimizes the overall think time for the scheduler and decrease the overlapping of parallel write request from buffers when allocating on shared Disk storage. Markov model parameters which are the transition probability matrix used in a simulated anticipatory algorithm to find a permutation of the buffered write that substantially does not penalize the future Read operation and reduces expected seek distance of IO request (Fig. 6).

The transition probability matrix by itself is need for predicting non-contiguous write request on Disk device after intermediate data buffered in Round robin fashion. Then, the scheduler calculate seek distance for each available request and it predicts the next state for keeps Disk idle with an estimate think time by using the information maintained by means of the transition probability matrix. For this purpose, the scheduler chooses a sequence of request by repeatedly finding the most likely transition from the current state s to have non-contiguous request from the same task. This approach builds on N-step transition models from Markov theory. The sequence of predicted non-contiguous request stop when a specified number of requests is reached, i.e. when the total probability of the sequence drops below a given threshold. Then, Anticipatory scheduler meets all the non-contiguous sequences found to calculate think time for read and write request giving priority for read operation. It interacts with the transition probability matrix based Anticipatory heuristic to decide if and how long to wait for each task slot. Figure 4 show the interaction between the various component.

5.3 Training Markov Chain

Each application/job has one Markov chain per workload and hardware parameters. These models are trained online by running a Mapreduce workload, and then generating trace of block spill file based on trace tool, which has been linked with a training module. The Markov chain segment size is selected in advance based on the block segment size, and the training module maintains counts of all record block write transition. Most spill file (intermediate data) involves only a single access pattern; in this case, a degenerate Markov chain with one state per block of spill file suffices to model them and training is a trivial calculation. The probabilities for each transition are calculated by dividing the number of occurrences of the transition by the total number of transitions from each block, and only a single training execution is required. The examples in this paper utilize this training algorithm. We implemented Markov chain with the C ++ language and used representative benchmarks hosted on a single node cluster: Terasort and Wordcount benchmarks, we generated 40 GB and 20 GB of intermediate data respectively on Hadoop 1.0.3 version on Ubuntu Linux 14.04 machine. The setting parameters of Hadoop are 4 CPU, 8G RAM and 4 Disks of 1TB) for each training set of sequences with 4 tasks Map and 3 Tasks Reduce in parallel. We stressed a Linux system with blktrace to have Disk I/O of read/write operation during the execution of Mapreduce workload.

6 Conclusion

In this paper, we have proposed framework for scheduling write access of parallel Mapreduce application under I/O contention on block-level. We focused on the issues that occur when multiple applications are running in parallel on a shared node in order to take advantage of multiple CPU cores. To characterize I/O request, we modeled the write access of Mapreduce task based on Markov model to predict the non-contiguous write operation. The model uses knowledge about Mapreduce application by tracing I/O access on lower level from shared Disk, and it is used for making decision to scheduling these accesses on Disk queue. For this purpose, we used Anticipatory scheduler to sort non-contiguous write request from buffers to the Disk using transition probability matrix. This proposed framework is an approach that describe different components that play a key role to scheduling I/O access of intermediate data on block-level, and to decide if using the hinted sequence of transition probability matrix for future scheduler is likely to provide benefit. For future work, we will estimate the prediction accuracy of Markov model, and describe our implementation for possible evaluation.

Acknowledgments. Work funded by the European Commission under the Erasmus Mundus GreenIT project (GreenIT for the benefit of civil society. 3772227-1-2012-ES-ERAMUNDUS-EMA21; Grant Agreement n 2012-2625/001-001-EMA2).

References

1. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
2. Apache Hadoop Core. http://hadoop.apache.org/core
3. Zhang, X., Davis, K., Jiang, S.: Opportunistic data-driven execution of parallel programs for efficient I/O services. In: Proceedings of IPDPS12, pp. 330–341. IEEE (2012)
4. Lofstead, J., Zheng, F., Liu, Q., Klasky, S., Oldfield, R., Kordenbrock, T., Schwan, K., Wolf, M.: Managing variability in the IO performance of petascale storage systems. In: Proceedings of SC10. IEEE Computer Society (2010)
5. Ching, W.-K., Ng, M.K.: Markov Chains: Models Algorithms and Applications. Springer, US (2006)
6. Filip, B., Cyril, G., Qingbo, W., Timothy, T.: Priority IO scheduling in the cloud. In: Proceeding of HotCloud 2013, the 5th USENIX Workshop on Hot Topics in Cloud Computing (2013)
7. Prashant, T., Sushma, S.: A development approach towards self learning schedulers in Linux. Proc. Int. J. Recent Innov. Trends Comput. Commun. **2**(4), 814–819 (2014)
8. Iyer, S., Druschel, P.: Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O. In: ACM Symposium on Operating Systems Principles (SOSP 2001) (2001)
9. Kambatla, K., Pathak, A., Pucha, H.: Towards optimizing hadoop provisioning in the cloud. In: Proceeding of HotCloud. USENIX, Berkeley (2009)
10. Huai, Y., Lee, R., Zhang, S., Xia, C.H., Zhang, X.: DOT: a matrix model for analyzing, optimizing and deploying software for big data analytics in distributed systems. In: Proceeding of SOCC, pp. 4:1–4:14. ACM, New York (2011)
11. Jahani, E., Cafarella, M.J., Ré, C.: Automatic optimization for MapReduce programs. Proc. VLDB Endow **4**(6), 385–396 (2011)
12. Yang, H., Luan, Z., Li, W., Qian, D.: MapReduce workload modeling with statistical approach. J. Grid Comput. **10**, 279–310 (2012). doi:10.1007/s10723-011-9201-4
13. Herodotou, H.: Hadoop performance models, Technical report, Duke University (2010). http://www.cs.duke.edu/starfish/files/hadoop-models.pdf
14. Jindal, A., Quiané-Ruiz, J.-A., Dittrich, J.: Trojan data layouts: right shoes for a running elephant. In: Proceeding of SOCC, pp. 21:121:14. ACM, New York (2011)
15. Siyuan, M., Xian-He, S., Ioan, R.: I/O Throttling and Coordination for MapReduce. Technical Report, Illinois Institute of Technology (2012)
16. Yiqi, X., Adrian, S., Ming, Z.: IBIS: interposed big-data I/O scheduler. In: Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing, pp. 109–110. ACM (2013)
17. Pu, X., Liu, L., Mei, Y., Sivathanu, S., Koh, Y., Pu, C.: Understanding performance interference of I/O workload in virtualized cloud environments. In: Proceeding of CLOUD, pp. 51–58 (2010)
18. Mesnier, M.P., Wachs, M., Sambasivan, R.R., Zheng, A.X., Ganger, G.R.: Modeling the relativetness of storage. In: Proceeding of SIGMETRICS, pp. 37–48. ACM, New York
19. Gulati, A., Shanmuganathan, G., Ahmad, I., Waldspurger, C., Uysal, M.: Pesto: online storage performance management in virtualized datacenters. In: Proceeding of SOCC, pp. 19:1–19:14. ACM, New York (2011)

20. Chiang, R., Huang, H.: TRACON: interference-aware scheduling for data-intensive applications in virtualized environments. In: Proceedings of SC, pp. 1–12 (2011)
21. Celis, J.R., Gonzales, D., Lagda, E., Rutaquio Jr., L.: A comprehensive review for disk scheduling algorithms. Int. J. Comput. Sci. Issues (IJCSI) **11**(1), 74 (2014)