Big Data Research 11 (2018) 1–10

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Hadoop MapReduce Performance on SSDs for Analyzing Social Networks ☆

M. Bakratsas [a], P. Basaras [a], D. Katsaros [b,*], L. Tassiulas [b]

[a] University of Thessaly, Department of Electrical & Computer Engineering, Volos, Greece
[b] Yale University, Department of Electrical Engineering & Yale Institute for Network Science, New Haven, CT, USA

ARTICLE INFO

Article history:
Received 16 January 2017
Received in revised form 5 June 2017
Accepted 23 June 2017
Available online 13 July 2017

Keywords:
MapReduce
Hadoop
Solid state disks
Magnetic disks
Social networks
Big data

ABSTRACT

The advent of Solid State Drives (SSDs) stimulated a lot of research to investigate and exploit to the extent possible the potentials of the new drive. The focus of this work is on the investigation of the relative performance and benefits of SSDs versus hard disk drives (HDDs) when they are used as underlying storage for Hadoop's MapReduce. In particular, we depart from all earlier relevant works in that we do not use their workloads, but examine MapReduce tasks and data suitable for performing analysis of complex networks which present different execution patterns. Despite the plethora of algorithms and implementations for complex network analysis, we carefully selected our "benchmarking methods" so that they include methods that perform both local and network-wide operations in a complex network, and also they are generic enough in the sense that they can be used as primitives for more sophisticated network processing applications. We evaluated the performance of SSDs and HDDs by executing these algorithms on real social network data and excluding the effects of network bandwidth which can severely bias the results. The obtained results confirmed in part earlier studies which showed that SSDs are beneficial to Hadoop. However, we also provided solid evidence that the processing pattern of the running application has a significant role, and thus future studies must not blindly add SSDs to Hadoop, but they should build components for assessing the type of processing pattern of the application and then direct the data to the appropriate storage medium.

1. Introduction

A complex network is a graph with topological features such as scale-free properties, existence of communities, hubs, and so on that is used to model real systems, for example, technological (Web, Internet, power grid, online social networks) networks, biological networks (gene, protein), social networks [23]. The analysis of online social networks (OSNs) such as Facebook, Twitter, Instagram has received significant attention because all these networks store and process colossal volumes of data, mainly in the form of pair-wise interactions, thus giving birth to networks, i.e., graphs which record persons' interactions whose analysis and mining offers both operational and business advantages to the OSN owner.

Modern OSNs are comprised by millions of nodes and even billions of edges; therefore any algorithm for their analysis that relies on a single machine (centralized) – exploiting solely the machine's main memory and/or its disk – is eventually doomed to fail due to lack of resources. Thus, the digitization of the aforementioned relationships produces a vast amount of collected data, i.e., big data [9] requiring extreme processing power that only distributed computing can offer. However, developing a distributed solution is a challenging task because it must deal sometimes with sequential processes. Some analysis algorithms based on distributed solutions that can run only on a small cluster of machines are still insufficient, since modern OSNs are maintained by Internet giants such as Google, LinkedIn and Facebook who own huge datacenters and operate clusters of several thousand machines. These clusters are usually programmed by data-parallel frameworks of the MapReduce type [4], a big data analytics platform.

The Hadoop [29] middleware was designed to solve problems where the "same, repeated processing" had to be applied to peta-scale volumes of data. Hadoop's initial design was based on magnetic disk's characteristics, enforcing sequential read and write operations introducing its own distributed file system (HDFS – Hadoop Distributed File System) with blocks of large size.

Recently with the advent of faster Solid State Drives (SSDs) research is emerging to test and possibly to exploit the potential of the new technologically advanced drive [11,12,21,33]. The lack

☆ This article belongs to Big Data & Neural Network.

* Corresponding author.
E-mail addresses: mmpakrat@gmail.com (M. Bakratsas), pbasaras@gmail.com (P. Basaras), d.katsaros@yale.edu (D. Katsaros), leandros.tassiulas@yale.edu (L. Tassiulas).

http://dx.doi.org/10.1016/j.bdr.2017.06.001
2214-5796/

of seek overhead gives them a significant advantage with respect to Hard Disk Drives (HDDs) for workloads whose processing requires random access instead of sequential access. Even though the cost-per-capacity of SSDs is still high, their adoption could be widespread if their performance was solidly proved to be superior to that of HDDs. The world of databases has long time ago started [18] to assess the benefits of using SSDs in various points of the database architecture, but the Hadoop world has only recently [12,13,21,30] started a similar investigation.

Providing a clear answer to the question of whether SSDs significantly outperform or offer increased performance in same cases compared to HDDs in the Hadoop environment is not straightforward, because the results of a system-analysis-based investigation are affected by the network speed and topology, by the cluster (size, architecture,...), and by the nature of the benchmarks used (MapReduce algorithms, input data). The efforts done so far to provide light to this question suffer either because the experimentation was executed on a virtualized cluster [13], or because their setup was affected by the underlying network [21], or because their benchmark algorithms and data were mostly read-oriented [12,21], thus biasing the results in such a way that no clear answer and universally holding conclusions could be drawn.

This article attempts to start the investigation from a new basis and to provide a clear answer to the following basic question: Ignoring any network biases and storage media cost considerations, do SSDs provide improved performance over HDDs for real workloads that are not dominated by either reads or writes? In this context, our article makes the following contributions:

- It uses a different set of MapReduce jobs, i.e., complex network analysis tasks, which have radically different characteristics from the earlier used benchmarks.
- It isolates "external" dependencies, i.e., network, cost considerations.
- It shows that there exists at least one case where HDDs can deliver superior performance to SSDs, which has not been documented in any earlier study.
- It provides solid evidence that the MapReduce job's read/write behavior will eventually provide the answer of whether SSDs are preferable over HDDs, which is consistent with the conclusions reported in [20] where random writes in SSDs are the "killing" application pattern for SSDs (with respect to reads and sequential writes).

The rest of the article is organized as follows: In section 2 we present the related work, and in section 3 we briefly describe Hadoop's structure. In section 4, we provide information about the three algorithms that will be evaluated in the storage media. Section 5 contains the evaluation results, and finally, section 6 concludes the article.

This paper is based on an earlier look on this topic [2]. In particular, the main augmentation parts in the current paper are the following ones: section 2 has been expanded significantly including more related works; section 3 which gives a brief overview of Hadoop architecture; the whole section 4 which presents in details the examined algorithms is practically new material (only Table 1 appears in the conference version of the article); section 5.3.1 which evaluates the competing disks against an industry standard is new material; and finally, performance results presented in Fig. 7 and Fig. 8 along with the associated explanations are also new material.

2. Related work

Introducing and investigating the usage of SSDs in Hadoop clusters has been a hot issue of discussion very recently. The most relevant work to ours is included in the following articles [12,13,21,26,30]. The first effort [13] to study the impact of SSDs on Hadoop was on a virtualized cluster (multiple Hadoop nodes on a single physical machine) and showed up to three times improved performance of SSDs versus HDDs. However, it remains unclear whether the conclusions still hold in non-virtualized environments. The work in [21] compared Hadoop performance on SSDs and HDDs on hardware with non-uniform bandwidth and cost using the Terasort benchmark. The major finding is that SSDs can accelerate the shuffle phase of MapReduce. However, this work is confined by the very limited type of application/workload used to make the investigation and the intervention of data transfers across the network. Cloudera's employees in [12], using a set of same-rack-mounted machines (not reporting how many of them), focus on measuring the relative performance of SSDs and HDDs for equal-bandwidth storage media. The MapReduce jobs they used are either read-heavy (Teravalidate, Teraread, WordCount) or network-heavy (Teragen, HDFS data write), and the Terasort which is read/write/shuffle "neutral". Thus, neither the processing pattern is mixed nor the network effects are neutral. Their findings showed SSD has higher performance compared to HDD, but the benefits vary depending on the MapReduce job involved, which is exactly where the present study aims at.

The analysis performed in [26] using Intel's HiBench benchmark [6,7] concluded that "...the performance of SSD and HDD is nearly the same", which contradicts all previously mentioned works. A study of both pure (only with HDDs or only with SSDs) and hybrid systems (combined SSDs and HDDs) is reported in [30] using a five node cluster and the HiBench benchmark. Differently from the present work, in that work, the authors investigated the impact of HDFS's block size, memory buffers, and input data volume on execution time showing that when the input data set size and/or the block size increases, then the performance gap between a pure SSD system with a pure HDD system widens in favor of the SSD system. Moreover, for hybrid systems, the work showed that more SSDs result in better performance. These conclusions are again expected since voluminous data imply increased network usage among nodes.

Earlier work [8,27] studied the impact of interconnection on Hadoop performance in SSDs identifying bandwidth as a potential bottleneck. The increase of bandwidth by using high-performance interconnects benefits HDFS performance on both disk types, but especially SSDs. Both conclusions are expected since a lot of data transfer takes place among nodes in map-shuffle-reduce operations. Less related to our study, [1] proposes a performance model using queuing network to simulate the execution time of MapReduce and thus come up with a cost-performance model for SSDs and HDDs in Hadoop, and [5,22] explore how to optimize a Hadoop MapReduce framework with SSDs in terms of performance, and/or cost/energy.

Finally, some works propose extensions to Hadoop with SSDs. For instance, [11] proposes extensions to enable clusters of reconfigurable active SSDs to process streaming data from SSDs using FPGAs. VENU [15] is a proposal for an extension to Hadoop that will use SSDs as a cache for the slower HDDs not for all data, but only for those that are expected to benefit from the use of SSDs. This work still leaves open the question about how to tell which applications are going to benefit from the performance characteristics of SSDs. Remotely related to our work is the discussion about the introduction of SSDs in database systems, e.g., [18].

3. Hadoop structure

Hadoop is an open source framework, written in the Java programming language which allows for processing large data sets in

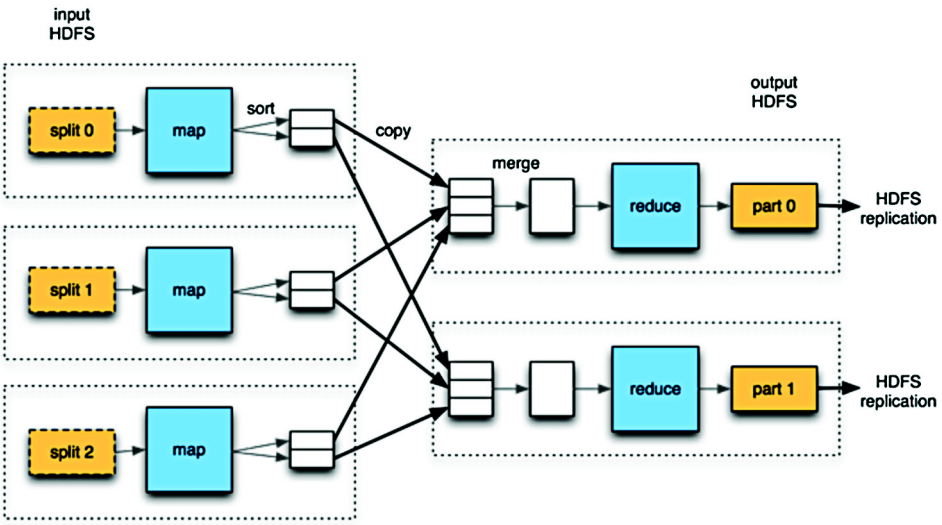

Fig. 1. Overview of Map/Reduce and Hadoop (from [29]).

a parallel/distributed computing environment. HDFS and MapReduce (MR) are the two core components of Apache Hadoop.

HDFS is Hadoop's distributed file system that provides high-throughput access to data, high-availability and fault tolerance. Data are saved as large blocks (default size 128 MB) making it suitable for applications that have huge data sets. It creates replicas of each block and distributes them among the nodes of the cluster.

MapReduce is a software framework that allows to write applications and execute them upon a cluster comprised by a few machines to several thousand commodity machines. It takes care of all cluster maintenance tasks and job scheduling operations and allows the programmer to focus on programming the logic of the application. Submitting a MapReduce job to the master node, results in splitting the input "file" to several chunks (block sized) that are processed by Map and Reduce tasks at parallel. Due to block replication of HDFS, tasks are scheduled to run on nodes where the required chunks of data already exist, minimizing unnecessary transfer of these data.

The key functions to be implemented are Map and Reduce. The MapReduce framework operates on (key,value) pairs. Each Map task processes an input split (block) generating intermediate data of (key,value) format. Then, they are sorted and partitioned by key, so later at Reduce phase, pairs of the same key will be aggregated to the same reducer for further processing. The flow of data is depicted in Fig. 1. Here lays Hadoop's main advantage. Partitions from different nodes with the same key are transferred (shuffle phase) to a single node and then merged (sort phase) and get ready to be fed to the reduce task. The output of Reduce tasks is of format (key, value) as well.

4. Investigated algorithms

Complex network analysis comprises a large set of diverse tasks (algorithms for finding communities, centralities, network growth models, resilience to attacks, epidemics, etc.) that cannot be enumerated here, and whose particular form depends on the field of study (technology, biology, sociometry, medicine) and also on the particular application that the "human miner" is interested in. Apparently, not all these tasks accept distributed solutions (at least, efficient ones) in the form of MapReduce algorithms, but there is already a significant body of works that developed MapReduce algorithms for solving problems such as triangle enumeration [14], *k*-shell computation [25], *k*-means clustering [32], neural networks [31], etc.

Table 1
Characterization of problems/algorithms examined.

Primitive	Type of analysis	Extent
Mutual friends	Neighbor-based	Local network (neighborhood) properties Recommendation queries
Connected components	Path-based	Large-scale network properties, Reachability queries, Resilience queries
Triangle counting	Mixed (extended neighborhood & paths)	Large-scale network properties, Clustering/communities finding queries

Therefore, among all these problems and their associated MapReduce solutions, we had to select some of them based on a) their usefulness in complex network analysis tasks, b) in their suitability to the MapReduce programming paradigm, c) the availability of their implementations (free/open code) for purposes of reproducibility of measurements, and d) complexity in terms of multiple rounds of map-reduce operations. Based on these criteria, we selected three problems/algorithms for running our experimentations. The first algorithm deals with a very simple problem which is at the same time a fundamental operation in Facebook, that of finding mutual friends. The second algorithm deals with a network-wide path-based analysis for finding connected components which finds applications in reachability queries, techniques for testing network robustness and resilience to attacks, epidemics, etc. The third algorithm is about counting triangles which is a fundamental operation for higher level tasks such as calculating the clustering coefficient, or executing community finding algorithms based on clique percolation concepts [24]. We wanted to have problems that deal with both the local and global structure of the network. Table 1 summarizes the "identity" of the examined tasks.

We need to emphasize that it is not the purpose of this article to develop a benchmark suite of algorithms and input data for MapReduce, even though we clearly recognize this need and call for the development of a really generic and representative benchmark; current efforts in this topic (like the Hibench [6,7]) are in a rather infantile age and their tasks (wordCount, k-means clustering, Bayesian classification, PageRank, etc.) are mostly appropriate for information retrieval or basic, traditional data mining tasks. So, our benchmark includes representative (in the notion described above) MapReduce jobs to cover common IO patterns

```
%1st MR job - CalculateAdjacencyList:
ON MAP DO:
 for each KV pair do:
    K<-source_node
    V<-destination_node
    context.write (K,V)
    context.write (V,K)
ON REDUCE DO:
 for each K[V] pair do:
    ego_user<-get(k)
    for each v in V
        add v to nodes_list
    sort the nodes_list
    for each node_id in nodes_list
        append node_id to friendlist
    context.write (ego_user,friendlist)

%2nd MR job - Creating triples:
ON MAP DO:
 for each KV pair do:
    K<-ego_user
    V<-friendlist
    for each friend in friendlist
        for each other_friend in friendlist
            if ego_user<friend then
                context.write (ego_user-friend:other_friend , NULL)
            else
                context.write (friend-ego_user:other_friend , NULL)
ON REDUCE DO:
 for each KV pair do:
    if |V|==2 then
        context.write (triple,NULL)

%3rd MR job
ON MAP DO:
 for each KV pair do:
    pair_and_mutual=K.split(":")
        pair=pair_and_mutual(0)
        mutual=pair_and_mutual(1)
        context.write (pair,mutual)

ON REDUCE DO:
 for each KV pair do:
    pair<-get(K)
    for each v in V
        v<-mutual
        mutuals_list.add(mutual)
 context.write (pair,mutuals_list)
```

Fig. 2. MapReduce pseudo-code for finding mutual friends.

expected to be seen in complex network analysis. We deferred a more advanced method for measuring the performance for multi-job workload such as the one described in [3], because the standalone, one-job-at-the-time method allows for the examination of interaction between MapReduce and storage media without the interventions of job scheduling and task placement algorithms.

We aim at showing that the conclusions about the relative performance of SSDs versus HDDs are strongly depended on the features of the algorithms examined, which has largely been neglected in earlier relative studies [12,13,21], and based on these features we draw some conclusions on the relative benefits of SSDs. For purposes of the article's self-completeness, we present in the following three sections the selected algorithms and a brief explanation of their operation.

4.1. Mutual friends

A common feature of various social networks is providing information of the existence of mutual friends once visiting some other user's profile page. A simple algorithm was implemented for the calculation of mutual friends. The necessary condition is that this pair of users are already friends (connected) with each other. Pseudocode for the MapReduce algorithm is given in Fig. 2.

The basic idea behind the algorithm is for every user (i.e., node) and his friend-list (i.e., adjacency list) to create all possible triples consisting of:

- The owner of the friend-list,
- A user of the friend-list who will make a pair with the owner, and
- Another user of the friend-list who will be the candidate mutual friend.

The same work is performed for each and every user and his friend-list. Eventually, if two exact triples are spotted, then the candidate is classified as a mutual friend for the specified pair. For the implementation three MR jobs are required:

1. Calculation of the adjacency list (friend-list). The input file is a graph containing all the ties among the nodes. Each node is a number unique for each user. All used social network datasets, were un-weighted, undirected graphs. Each line consists of a source node and destination node. Duplicate relationships aren't present in the original files. On the contrary, such supplementary information is necessary for the creation of adjacency lists, thus created by the Map function. Reduce function produces lines of every node and its adjacency list.
2. Creation of all available triples according to the basic concept that was mentioned previously. The Mapper output creates all available triples as key. Value is set to NULL. At Reducer, for a specific Key aggregating two NULL values, confirms the existence of a mutual friend.
3. Creation of the lists of mutual friends. At the Mapper, from each triple the pair is extracted as Key and their mutual as Value. The Reducer completes the creation of mutual friends list for every pair.

4.2. Connected components

Another very useful and primitive process of complex network analysis is the detection of connected components i.e., clusters of nodes where every node of the cluster can be eventually be accessed by any other node of the cluster following a path of arbitrary number of hops. This task finds applications in reachability analysis, in epidemics, i.e., once isolated users or groups are found, the spread of a contagion can be stopped, etc.

For this task, the implementation by Thomas Jungblut [10] of an iterative algorithm based on message passing technique is used (see Fig. 3).

At the first iteration, the algorithm maps every first element as key and its adjacency list in vertex form as a pointsTo tree. Also, it maps each edge of the tree in vertex form. At reduce, the algorithm marks all vertexes having a pointsTo tree as activated. It sets the smallest element of this list (comparing to the key as well), as vertex's minimal. Then, it writes key and vertex in context. At next iterations, map writes each key and vertex as it is. Also for every activated vertex, it loops through the pointsTo tree and writes a message (vertex with empty tree) with the (for this vertex) minimal vertex to every edge of the tree. At reduce, it merges messages with the related vertex and if a new minimum is found then activates the vertex. The updated counter gets incremented. Otherwise deactivates the vertex. Iterations continue till no vertex gets updated.

4.3. Counting triangles

Counting the number of triangles in a graph is a fundamental problem with various applications especially in social network analysis. For example, the clustering coefficient is frequently quoted as an important index for measuring the concentration of clusters in graphs respectively its tendency to decompose into communities.

```
%1st MR job
ON MAP DO:
 for each line (adjacency list)
    realkey<-first edge of adjacency list
    vertex<-all other edges sorted, plus minimal
    context.write (realkey, vertex)
 for all edges in vertex
    context.write (edge, new empty vertex with edge as minimal)

ON REDUCE DO:
 for each KV pair do:
    if V is not message then
       realVertex<-edges of V
       activate realVertex
       increment UPDATED counter
       context.write(key,realVertex)

%2nd MR job
ON MAP DO:
 for each KV pair do:
    context.write (K,V)
    if V is activated then
        for all edges in V
           if edge != minimal of V
               newVertex<-null edges
               newVertex<-minimal of V
               context.write (edge, newVertex)

ON REDUCE DO:
 for each K[V] pair do:
    for every v in V
        if v is not message then
            realVertex<-v
        else
            track newMinimal among messages v in V
    if realVertex.minimal > newMinimal then
        update realVertex with the lower newMinimal
        activate the realVertex
        increment UPDATED counter
    else
        deactivate the realVertex
    context.write(key, realVertex)
```

Fig. 3. MapReduce pseudo-code for finding connected components.

We used the implementation by Walkauskas [28] (pseudo-code in Fig. 4) which includes three MapReduce jobs:

- A triangle exists when a vertex has two adjacent vertexes that are also adjacent to each other. The first job constructs all of the triads in the graph. A triad is formed by a pair of edges sharing a vertex, called its apex. Original edges are written, as well. The above are written as keys with the value of 1 or 0 respectively to distinguish triads from original edges.
- The second MapReduce job maps previous input line, and the Reducer aggregates the triads with the edges for a specific triple. In order for a triangle to exist, there should be at least one candidate triad and the edge connecting the apex. The reducer eventually writes sum to context as "0, sum".
- The third MapReduce job aggregates the number of triangles that was found from previous job for all chunks.

We see that all three algorithms are executed in two or more pairs of 'maps' and 'reduces' which is a desired complexity for our measurements in terms of read and write operations.

5. Experimental environment and results

In this section we describe the system's setup and then we provide the obtained results for each one of the three algorithms presented earlier.

5.1. System setup

A commodity computer (Table 2) was used for the experiments. Three storage media were used (Table II) with capacities similar to that used in [21]. On each of the three drives (one HDD and two SSDs) a separate and identical installation of the required software (Table 3) was used. We emphasize at this point that since we need to factor out the network effects, we used single machine installations. Three different incremental setting setups were used: a) with default settings, allowing 6 parallel maps, b) with modified containers allowing 3 parallel maps, and c) with custom settings (Table 4). In all these setups, speculative execution was disabled and no early shuffling was permitted. We admit the a shortcoming of our study is the fact that we do not have a clear view of the types of storage devices used in the datacenters of the Internet giants (Google, Facebook), but still we are confident that the relative performance of the devices used will support our arguments. Power saving options and boosting technologies like Turbo-boost

```
%1st MR job - TriadConstruction:
ON MAP DO:
 for each KV pair do:
    if K < V write to context

ON REDUCE DO:
 for each K[V] pair do:
    for each v in V
        save v in Array
        context.write (Kv, "zero")
    sort the Array
    for each v of sorted Array
        for each v' following v in the Array
            context.write (vv', "one")

%2nd MR job - TriadConstruction:
ON MAP DO:
 for each KV pair do:
    K<-source_node
    V<-destination_node
    context.write (K,V)

ON REDUCE DO:
 for each K[V] pair do:
    sum all v values in V
    compare the sum to the #v in V
    if not equal
        increase #triangles found by sum
    context.write(zero, count)

%3rd MR job - AggregateTriangles:
ON MAP DO:
 for each KV pair do:
    K<-source_node
    V<-destination_node
    context.write (K,V)

ON REDUCE DO:
 for each K[V] pair (only one pair with "zero" key) do:
    sum all v in V
 context.write (sum, null)
```

Fig. 4. MapReduce pseudo-code for triangle counting.

Table 2
Computer specifications.

CPU	Intel i5 4670 3.4 GHz (non HT)
RAM	8 GB 1600 MHz DDR3 (1333 MHz with disabled XMP)
Disk 1 (HDD)	Western Digital Blue WD10EZEX 1TB
Disk 2 (SSD1)	Samsung 840 EVO 120 GB
Disk 3 (SSD2)	Crucial MX100 512 GB

Table 3
Installed software.

OS	Ubuntu 14.04 LTS 64 bit
Java SDK	Oracle Java 1.8.0_25 (8u25)
Hadoop version	Hadoop 2.5.2 (pre-built 32-bit i386-Linux native Hadoop library)
Monitoring tools	Collectl V3.6.9-1

Table 4
Custom settings.

mapreduce.reduce.shuffle.parallel.copies	5–50
mapreduce.task.io.sort.factor	10–100
mapreduce.map.sort.spill.percent	0.80–0.90
io.file.buffer.size	4 kB–64 kB

and IEST were disabled through BIOS to minimize unexpected fluctuations among executions.

5.2. Input data and performance measures

For the evaluation of the two disk types a sample of real data was required. Recall that earlier efforts e.g., [21] used dummy data files that were read and some primitive statistics were written out. Social networks is a representative sub-genre of complex networks. Thus up to ten real social network graphs were used (Table 5). They were retrieved from https://snap.stanford.edu/ and http://konect.uni-koblenz.de/. The number of nodes and edges vary from a few thousands to a few millions. Thus, we used networks that vary up to two orders of magnitude in their size (number of nodes and/or edges).

The evaluation will take place along two dimensions. The first one is similar to that in [21] using TestDFSIO and the second one is the complex network analysis-oriented that is the focus of this article. We have performed up to five experiments for each of the "Mutual Friends" and "Counting Triangles" algorithms and up to ten experiments for the "Connected Components", one for each dataset shown at Table 5. The latter algorithm acquired less disk space during execution allowing us to evaluate it with larger datasets. The two SSDs were of different size disallowing the execution of some datasets. The most important measures we captured were the Map and Reduce execution times, as also Sort (merge) and Shuffle phase All measured times are in seconds, unless otherwise stated. The aforementioned measures would indicate practical performance differentiations between the two disk types. One common side effect is "cache hits" from previous executions that was also experienced in [21]. In order to give each experiment an equal environment to eliminate any possible interaction effects from previous executions, Hadoop was halted and page cache was flushed, after each experiment. Before each test HDFS was re-formatted.

5.3. Results

5.3.1. TestDFSIO

We begin with the HDFS throughput measurement. Test Distributed File System (TestDFSIO) is an industry-standard benchmark which distributes map tasks that read/write complete dummy files on nodes; each map task reads the complete file and writes some statistics. Reduce tasks simply gather these statistics for output.

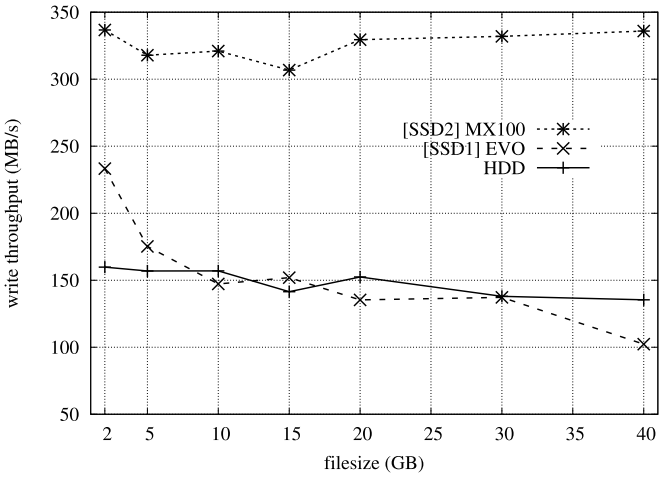

Fig. 5. Comparing TestDFSIO write throughput for 3 disks.

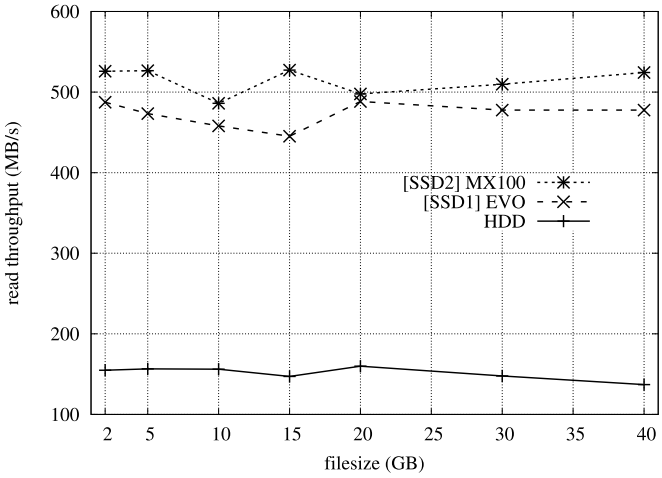

Fig. 6. Comparing TestDFSIO read throughput for 3 disks.

The write throughput performance is presented in Fig. 5. We observe that for writing sequential files, with the increase of filesize, SSD1's performance is decreasing, falling behind the HDD. Contrariwise, the SSD2 appears much faster with stable throughput. The 120 GB Evo, features a second level TurboWrite Cache (TWC). This 3 GB block of high speed SLC memory allows the EVO to write data (nominally) at 370 MB/s, nearly double its normal rate. However, when the TWC is full or can not be used effectively, write speeds drop by around 50%, and this is the pattern that we observe in the plot.

The sequential read performance of the competitors is presented in Fig. 6. As expected, both SSDs's sequential read throughput is outstanding. Moreover, both SSDs attain a read performance

Table 5
Social networks used for evaluation.

	Social network	# nodes	# edges
1	Brightkite location based online social network	58,228	214,078
2	Gowalla location based online social network	196,591	950,327
3	Amazon product co-purchasing network	334,863	925,872
4	DBLP collaboration network	317,080	1,049,866
5	YouTube online social network	1,134,890	2,987,624
6	YouTube (ver. 2) online social network	3,223,589	9,375,374
7	Flickr	1,715,255	15,550,782
8	LiveJournal online social network	3,997,962	34,681,189
9	LiveJournal (ver. 2) online social network	5,204,176	49,174,620
10	Orkut online social network	3,072,441	117,185,083

Table 6
Average times for each phase for 2nd job (creating triples) of "mutual friends" algorithm.

	Avg Map			Avg Shuffle			Avg Merge			Avg Reduce		
	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2
Brightkite	52	52	52	1	1	1	0	0	0	11	10	10
Amazon	36	35	35	2	1	1	0	0	0	8	7	8
Gowalla	1780	1752	1593	120	103	42	0	0	0	178	195	194
DBLP	90	89	89	5	2	3	0	0	0	16	17	17
YouTube	11197	–	9708	812	–	258	0	–	0	916	–	984

Table 7
Average times for each phase for 1st job (forming triads) of "counting triangles" algorithm.

	Avg Map		Avg Shuffle		Avg Merge		Avg Reduce	
	HDD	SSD2	HDD	SSD2	HDD	SSD2	HDD	SSD2
Gowalla	2	2	1	1	0	0	142	140
YouTube	6	6	1	1	0	0	706	694
Flickr	13	13	1	1	0	0	5053	5125

Table 8
Average times for each phase for 2nd job (counting triangles) of "counting triangles" algorithm.

	Avg Map			Avg Shuffle			Avg Merge			Avg Reduce		
	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2
Brightkite	18	18	18	1	1	1	0	0	0	4	4	3
Amazon	9	9	9	1	1	1	0	0	0	2	2	2
Gowalla	38	39	38	52	62	21	79	86	70	106	106	110
DBLP	14	14	14	1	1	1	0	0	0	7	5	5
YouTube	42	–	41	655	–	141	820	–	668	689	–	551

Table 9
Average times for each phase for 1st job (create triads) of "counting triangles" algorithm, with changed container's settings.

	Avg Map		Avg Shuffle		Avg Merge		Avg Reduce	
	HDD	SSD2	HDD	SSD2	HDD	SSD2	HDD	SSD2
Gowalla	2	2	1	1	0	0	141	138
YouTube	6	6	1	1	1	1	697	707
Flickr	13	13	1	1	6	6	4163	4140

close to that given by their specifications, namely 540 MB/s for SSD1 and 550 MB/s for SSD2, and it is practically stable and independent on file size. On the other hand, the magnetic disk again demonstrates stable performance, although noticeably slower than that of the SSDs.

5.3.2. Results on finding mutual friends

The complexity of this algorithm is exponential due to the mapper of the 2nd MapReduce job ("creating triple" – as described at section 4.1) where for each user and his friend-list every possible triple is formed (double "for" used). Thus, the 2nd MapReduce job is the most resource-intensive of the three jobs, rendering it a good inspection point for our measures (see Table 6), whereas the 1st and 3rd MapReduce jobs were fast-executed and almost identical for all disks. For Amazon, Brightkite and DBLP, the three disks performed almost equally. Remarkably, in comparison with both SSD drives, the magnetic disk gives *competitive (and slightly better) execution times for reduce phase for bigger datasets*, whereas HDD performs lower for map phase. The SSD2 displays superior performance at shuffling.

5.3.3. Results on counting triangles

Here, the SSDs outperform the HDD for all the datasets that were tested. At "forming the triads" job, HDD appeared competitive behavior at map and reduce phases (Table 7). The "counting the triangles" job demonstrated greater variations in execution times. With small datasets the performance differentiations between the two disk types are small (Table 8). But with larger ones (like YouTube dataset), SSDs capabilities become evident for shuffle and merge (sort) phases.

For the 1st MR job (creating triads), map, shuffle and merge phases finished quite fast and with almost zero differentiations among disks. Reduce phase lasted significantly longer with both disks performing equally (Table 6). With containers settings, the biggest dataset of Flickr gets significant improvement for both disk types (Table 9). No further improvement achieved with custom settings.

To optimize performance, increasing the following settings provided best results for the magnetic disk, compared to "containers" settings:

a) The number of streams to merge at once while sorting files. We see (Table 10 and Table 11) that it minimizes merge time for both disk types, but it improves the shuffling time of the HDD only. Even though both disks are able to reap benefits from this settings, HDD gains the most.
b) The buffer size for I/O (read/write) operations. Examining the impact of this change (Table 12 and Table 13), we observe that only the HDD is able to exploit efficiently, whereas its impact on SSD2 is mixed and insignificant.

To have a generic idea of the impact of "customs" and the "containers" settings, we present in Tables 14 and 15, the relative performance of HDD and SSD2 for a large network, namely YouTube, which shows that HDD is a better beneficiary.

Table 10
Performance difference for YouTube dataset at "Counting Triangles", increasing sort factor, for HDD.

[HDD] just containers and varying io.sort.factor					
	Elapsed	Avg Map	Avg Shuffle	Avg Merge	Avg Reduce
io.sort.factor:10	52 mins, 43 sec	25	565	596	720
io.sort.factor:100	40 mins, 26 sec	25	471	14	667

Table 11
Performance difference for YouTube dataset at "Counting Triangles", increasing sort factor, for SSD2.

[SSD2] just containers and varying io.sort.factor					
	Elapsed	Avg Map	Avg Shuffle	Avg Merge	Avg Reduce
io.sort.factor:10	41 mins, 08 sec	25	359	339	535
io.sort.factor:100	35 mins, 15 sec	25	371	16	497

Table 12
Performance difference for YouTube dataset at "Counting Triangles", increasing file buffer size, for HDD.

[HDD] just containers and io.file.buffer.size					
	Elapsed	Avg Map	Avg Shuffle	Avg Merge	Avg Reduce
io.file.buffer.size: 4 kB	52 mins, 43 sec	25	565	596	720
io.file.buffer.size: 128 kB	46 mins, 44 sec	25	445	470	619

Table 13
Performance difference for YouTube dataset at "Counting Triangles", increasing file buffer size, for SSD2.

[SSD2] just containers and io.file.buffer.size					
	Elapsed	Avg Map	Avg Shuffle	Avg Merge	Avg Reduce
io.file.buffer.size: 4 kB	41 mins, 8 sec	25	359	339	538
io.file.buffer.size: 128 kB	41 mins, 9 sec	24	361	331	554

Table 14
Percentage difference between "customs" and "containers" settings for YouTube dataset, at "Counting Triangles" algorithm.

"Customs" difference to "Containers"				
	Avg Map	Avg Shuffle	Avg Merge	Avg Reduce
HDD	4.00%	−28.85%	−97.65%	−11.39%
SSD2	0.00%	−2.23%	−95.28%	−10.41%

Table 15
Percentage difference between "customs" and "containers" settings for YouTube dataset, at "Mutual Friends" algorithm.

"Customs" difference to "Containers"				
	Avg Map	Avg Shuffle	Avg Merge	Avg Reduce
HDD	−26.14%	−16.59%	–	−9.72%
SSD2	−18.83%	0.78%	–	4.36%

5.3.4. Results on calculating connected components

Comparing SSD1 to HDD and SSD2, the Connected Components algorithm (Table 16) seems to slightly favor the SSD1 for small datasets (first five ones), at reduce phase which is surprising and somewhat hard to explain, because SSD1 has theoretically inferior performance to SSD2. However, we argue that the function of SSD1's TWC is quite successful. The generic pattern is that map, shuffle and reduce times are close for both disk types for these small datasets, contrary to what the current studies suggest.

When the size of data increases, e.g., for the datasets of Flickr and LiveJournal the magnetic disk takes the lead at reduce phase over SSD1, which is mostly characterized as "write" procedure for the Hadoop framework. SSD1 performs quite slowly at shuffle phase for the LiveJournal dataset, which again is attributed to the TWC delivering inferior performance. The SSD2 generally delivers great performance especially at map and shuffle phase, noticeably as the datasets' size increase. For the reduce phase, HDD falls behind SSD2, but not with a great margin.

To have a better understanding of the reasons behind the above performance behavior between HDD and SSD2, we examined the details of CPU and disk utilization during the execution of the 1st iteration of the connected components algorithm on the largest of our networks, namely Orkut. Hadoop's default settings allowed the

Table 16
Sum of average times for each phase for the iterative jobs of "Connected Components".

	Avg Map			Avg Shuffle			Avg Merge			Avg Reduce		
	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2
Brightkite	14	14	14	11	11	11	0	0	0	0	0	0
Amazon	104	106	103	34	34	34	0	0	0	74	61	62
Gowalla	27	26	26	10	10	10	0	0	0	14	14	16
DBLP	54	54	54	15	15	15	0	0	0	35	34	33
YouTube	126	124	123	14	14	14	0	0	0	101	96	98
YouTube 2	247	243	244	28	24	24	0	0	0	428	424	408
Flickr	170	168	167	30	19	20	0	0	0	309	314	304
LiveJourna l	353	380	322	104	143	45	1	0	0	665	682	651
LiveJournal 2	417	–	347	137	–	57	0	–	0	930	–	912
Orkut	456	–	324	552	–	154	295	–	231	1448	–	1204

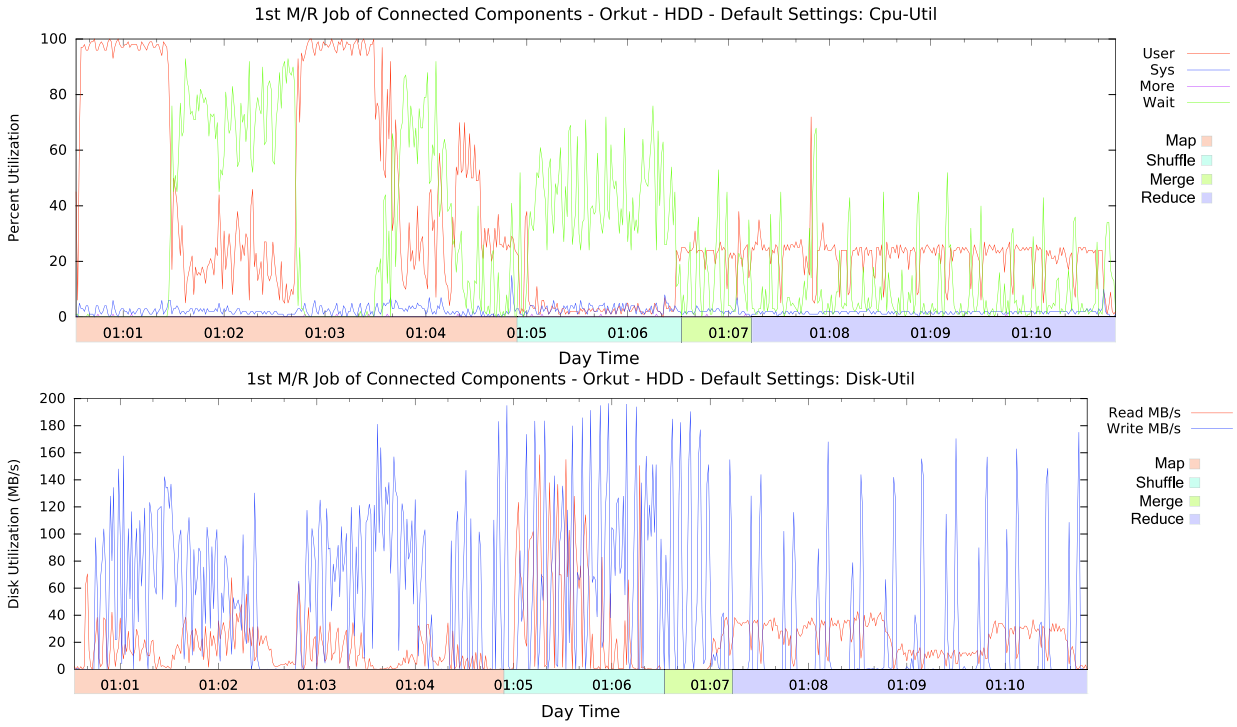

Fig. 7. (Top) CPU utilization for Connected Components with Orkut, using HDD, 1st iteration isolated. (Bottom) Disk usage for Connected Components algorithm with Orkut, using HDD, 1st iteration isolated.

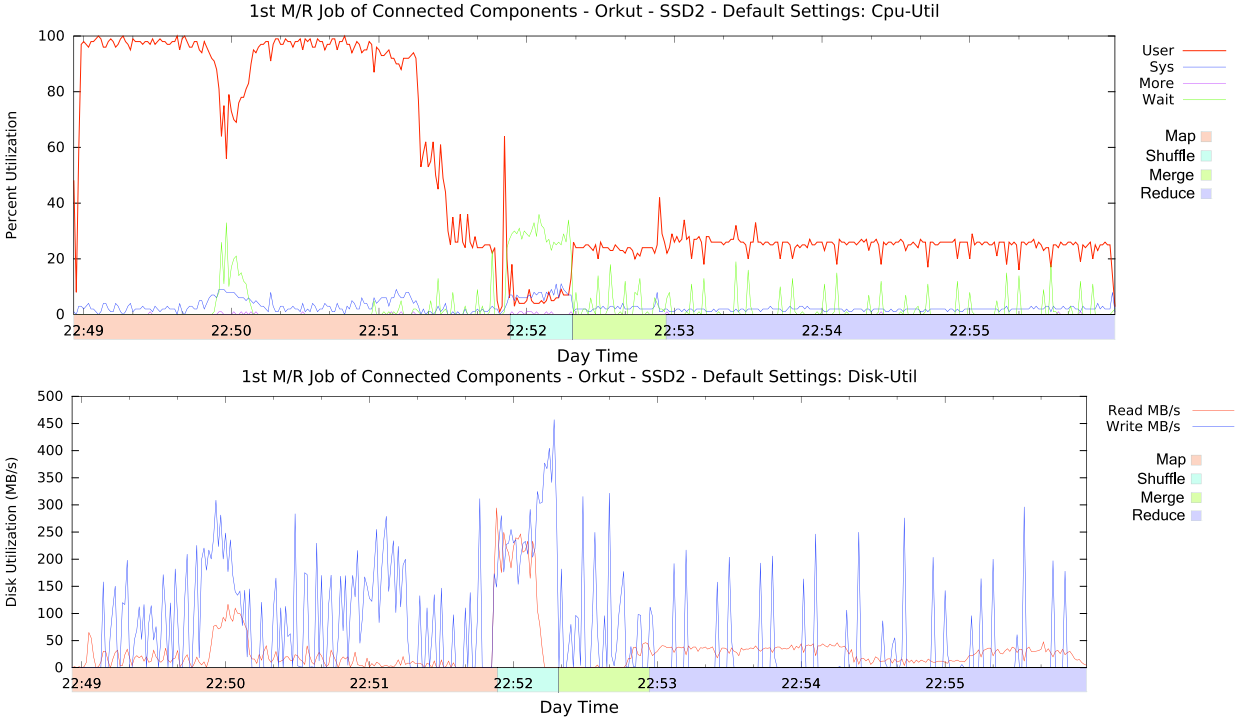

Fig. 8. (Top) CPU utilization for Connected Components with Orkut, using SSD2, 1st iteration isolated. (Bottom) Disk usage for Connected Components algorithm with Orkut, using SSD2, 1st iteration isolated.

execution of up to 6 maps simultaneously. Thus the execution of Orkut dataset (input file of 14 blocks at HDFS) was executed in three waves of maps. The map phase is CPU intensive, hitting 100% utilization. High disk throughput is required as well, with the disk constituting system's bottleneck causing high CPU wait times, especially for HDD, see Fig. 7(top), where during map phase CPU utilization falls between map waves. Consequently using SSD2 provides better CPU utilization. Excessive disk usage appears at shuffle phase demonstrating each disk's capabilities; see Fig. 7(bottom) and Fig. 8. At reduce phase, SSD2 performs slightly better.

The experiments established that default Hadoop settings are not optimized for hard disks, and that the technology of SSDs might have dramatic impact upon their (expected) performance. Most significantly, we provided solid evidence that hard disks can

be competitive to solid state disks for some I/O patterns, at least for the application field that we have investigated.

6. Conclusions

Hadoop platform is used for the processing of big data, especially to run analytics that is computationally intensive, such as social network analysis. Some tasks can be solved with a single or more consecutive and distinct jobs whereas others require iterative ones. Due to the SSD's provided substantial benefits over traditional hard disk drives, Hadoop administrators have started considering the addition or even replacement of the existing HDDs with SSDs. Yet, Hadoop's internal design – especially HDFS – doesn't appear to fully harness the potential of solid state drives.

In this empirical study, we compared the performance of solid state drives and hard disk drives for social network analysis. Three casual complex network analysis algorithms were used leaving space for the implementation and testing of many others, for even larger data sets.

A potential upgrade should be considered based on the tested applications' performance. In our tests SSDs didn't come out as the undisputed winner. There were noticed great performance fluctuations between the two SSDs. The second SSD performed significantly better. Otherwise, in many cases SSD1 and the magnetic disk came into a draw. Although SSD1 was slightly faster in many tests, *in some cases the magnetic disk outperformed the SSD1*. Even *compared to the faster SSD2, the magnetic disk provided competitive or faster times for reduce phase*, especially with the "mutual friends" algorithm.

Customizing Hadoop settings proves crucial. Magnetic disk's shuffle times can be significantly reduced. SSD's performance doesn't present further improvement. Nevertheless, HDD can't catch up with SSD's superior performance at shuffling. With tweaking merge-sort can be performed in less steps minimizing merge's phase times for both disk types, slightly favoring magnetic disk that would perform slower otherwise. For map phase both disk types can get similar performance improvement.

Overall, having no clear storage media winner, the paper suggests that the development of "application profilers" e.g., [16,17,19] that will try to predict the applications' read/write pattern (random/sequential) and then incorporation of them into the Hadoop architecture will help reap the performance benefits of any current or new storage media.

References

[1] S. Ahn, S. Park, An analytical approach to evaluation of SSD effects under MapReduce workloads, J. Semicond. Technol. Sci. 15 (5) (2015) 511–518.

[2] M. Bakratsas, P. Basaras, D. Katsaros, L. Tassiulas, Hadoop MapReduce performance on SSDs: the case of complex network analysis tasks, in: P. Angelov, Y. Manolopoulos, L. Iliadis, A. Roy, M. Vellasco (Eds.), INNS Conference on Big Data, in: Advances in Intelligent Systems and Computing, vol. 529, Springer, 2017, pp. 111–119.

[3] Y. Chen, A. Ganapathi, R. Griffith, R. Katz, The case for evaluating Mapreduce performance using workload suites, in: Proceedings of the IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2011, pp. 390–399.

[4] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Proceedings of the USENIX/ACM Symposium on Operating Systems Design and Implementation (OSDI), 2004, pp. 137–150.

[5] J. Hong, L. Li, C. Han, B. Jin, Q. Yang, Z. Yang, Optimizing Hadoop framework for solid state drives, in: Proceedings of the IEEE International Congress on Big Data, 2016.

[6] S. Huang, J. Huang, J. Dai, T. Xie, B. Huang, The HiBench benchmark suite: characterization of the MapReduce-based data analysis, in: Proceedings of the IEEE International Conference on Data Engineering Workshops (ICDEW), 2010, pp. 41–51.

[7] S. Huang, J. Huang, J. Dai, T. Xie, B. Huang, The HiBench benchmark suite: characterization of the MapReduce-based data analysis, in: Frontiers in Information and Software as Services, in: Lecture Notes in Business Information Processing, vol. 74, Springer, 2011, pp. 209–228.

[8] N. Islam, M. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, D. Panda, High performance RDMA-design of HDFS over InfiniBand, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2012.

[9] H.V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J.M. Patel, R. Ramakrishnan, C. Shahabi, Big data and its technical challenges, Commun. ACM 57 (7) (2014) 86–94.

[10] T. Jungblut, Retrieved on June 4th, 2017, Available at http://codingwiththomas.blogspot.de/2011/04/graph-exploration-with-hadoop-mapreduce.html.

[11] A. Kaitoua, H. Hajj, M.A.R. Saghir, H. Artail, H. Akkary, M. Awad, M. Sharafeddine, K. Mershad, Hadoop extensions for distributed computing on reconfigurable active SSD clusters, ACM Trans. Archit. Code Optim. 11 (2) (2014).

[12] K. Kambatla, Y. Chen, The truth about MapReduce performance on SSDs, in: Proceedings of the USENIX Large Installation System Administration Conference (LISA), 2014, pp. 109–117.

[13] S.-H. Kang, D.-H. Koo, W.-H. Kang, S.-W. Lee, A case for flash memory SSD in Hadoop applications, Int. J. Control. Autom. Syst. 6 (1) (2013) 201–210.

[14] T.G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, C. Task, Counting triangles in massive graphs with MapReduce, SIAM J. Sci. Comput. 36 (5) (2014) 48–77.

[15] K.R. Krish, M.S. Iqbal, A.R. Butt, VENU: orchestrating SSDs in Hadoop storage, in: Proceedings of the IEEE International Conference on Big Data (BigData), 2014, pp. 207–212.

[16] K.R. Krish, B. Wadhwa, M.S. Iqbal, M.M. Rafique, A.A. Butt, On efficient hierarchical storage for big data processing, in: Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2016, pp. 403–408.

[17] S. Lee, H. Min, S. Yoon, Will solid-state drives accelerate your bioinformatics? In-depth profiling, performance analysis and beyond, Brief. Bioinform. 17 (4) (2016) 713–727.

[18] S. Lee, B. Moon, C. Park, S. Kim, A case for flash memory SSD in enterprise database applications, in: Proceedings of the ACM Conference on the Management of Data (SIGMOD), 2008, pp. 1075–1086.

[19] Y.-S. Lee, L.C. Quero, S.-H. Kim, J.-S. Kim, S. Maeng, ActiveSort: efficient external sorting using active SSDs in the MapReduce framework, Future Gener. Comput. Syst. 65(C) (2016) 76–89.

[20] C. Min, K. Kim, H. Cho, S.-W. Lee, Y.I. Eom, SFS: random write considered harmful in solid state drives, in: Proceedings of the USENIX Conference on File and Storage Technologies (FAST), 2012.

[21] S. Moon, J. Lee, Y.S. Kee, Introducing SSDs to the Hadoop MapReduce framework, in: Proceeding of the IEEE International Conference on Cloud Computing (CLOUD), 2014, pp. 272–279.

[22] S. Moon, J. Lee, X. Sun, Y.-S. Kee, Optimizing the Hadoop MapReduce framework with high-performance storage devices, J. Supercomput. 71 (9) (2015) 3525–3548.

[23] M.E.J. Newman, Networks: An Introduction, Oxford University Press, 2013.

[24] G. Palla, I. Derenyi, I. Farkas, T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, Nature 435 (2005) 814–818.

[25] K. Pechlivanidou, D. Katsaros, L. Tassiulas, MapReduce-based distributed k-shell decomposition for online social networks, in: Proceedings of the International Workshop on Personalized Web Tasking (PWT), 2014, pp. 30–37.

[26] P. Saxena, Dr. Jerry Chou, How much solid state drive can improve the performance of Hadoop cluster? Performance evaluation of Hadoop on SSD and HDD, Int. J. Modern Commun. Technol. Res. 2 (5) (2014).

[27] S. Sur, H. Wang, J. Huang, X. Ouyang, D. Panda, Can high-performance interconnects benefit Hadoop distributed file system, in: Proceedings of the Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC), 2010.

[28] Vertica, http://www.vertica.com/2011/09/21/counting-triangles/.

[29] T. White, Hadoop: The Definitive Guide, O'Reilly Media, 2015.

[30] D. Wu, W. Xie, X. Ji, W. Luo, J. He, D. Wu, Understanding the impacts of solid-state storage on the Hadoop performance, in: Proceedings of the International Conference on Advanced Coud and Big Data, 2013, pp. 125–130.

[31] K. Zhang, X.-W. Chen, Large-scale deep belief nets with MapReduce, IEEE Access 2 (2014) 395–403.

[32] W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on MapReduce, in: Proceedings of the International Conference on Cloud Computing (CloudCom), 2009, pp. 674–679.

[33] Z. Zong, R. Ge, Q. Gu, Marcher: a heterogeneous system supporting energy-aware high performance computing and big data analytics, Big Data Res. 8 (2017) 27–38.