UNIVERSITY OF THESSALY
DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

Μεταπτυχιακό Πρόγραμμα Σπουδών

Επιστήμη και Τεχνολογία των Υπολογιστών Τηλεπικοινωνιών και Δικτύων

«Hadoop-MapReduce performance on SSD for Complex Network analysis»

«Επίδοση του Hadoop MapReduce σε Δίσκους Στερεάς Κατάστασης για Ανάλυση Σύνθετων Δικτύων»

Μεταπτυχιακή εργασία

Μάριος Μπακρατσάς

Επιβλέποντες Καθηγητές:

Κατσαρός Δημήτριος
Μποζάνης Παναγιώτης
Τσομπανοπούλου Παναγιώτα

Βόλος, Σεπτέμβριος 2015

Ευχαριστίες

Με την περάτωση αυτής της μεταπτυχιακής εργασίας θα ήθελα να ευχαριστήσω τους επιβλέποντες καθηγητές της εργασίας για την υποστήριξη που μου προσφέρανε κατά τη διάρκεια της φοίτησής μου, αλλά και κατά την εκπόνηση της διπλωματικής μου εργασίας. Και ειδικά τον κ. Κατσαρό, πρωτίστως για την ευκαιρία που μου έδωσε να εντρυφήσω στο συγκεκριμένο θέμα, καθώς επίσης και για τις χρήσιμες συμβουλές και υποδείξεις του.

Τέλος, ευχαριστώ θερμά την οικογένειά μου για την αμέριστη συμπαράσταση που μου παρείχε όλα αυτά τα χρόνια για την ολοκλήρωση των σπουδών μου.

Περίληψη

Η έλευση των Δίσκων Στερεάς Κατάστασης (SSDs) προκάλεσε μεγάλο ενδιαφέρον για έρευνα και αξιοποίηση στο μέγιστο δυνατό βαθμό των δυνατοτήτων του νέου δίσκου. Η παρούσα εργασία επικεντρώνεται στη μελέτη της σχετικής επίδοσης και των οφελών των SSDs έναντι των μαγνητικών δίσκων (HDDs) όταν χρησιμοποιούνται ως αποθηκευτικά μέσα για το Hadoop MapReduce. Ειδικότερα, με αφετηρία τις προηγούμενες σχετικές εργασίες, εξετάζουμε τις MapReduce διεργασίες και δεδομένα κατάλληλα για εκτέλεση ανάλυσης σε σύνθετα δίκτυα, που παρουσιάζουν διαφορετικά πρότυπα εκτέλεσης. Παρά την πληθώρα των αλγορίθμων και εφαρμογών για ανάλυση σύνθετων δικτύων, εμείς επιλέξαμε προσεκτικά τις μεθόδους αξιολόγησης επιδόσεών μας, έτσι ώστε να περιλαμβάνουν μεθόδους που εκτελούν ταυτόχρονα και τοπικές και σε επίπεδο δικτύου εργασίες σε ένα σύνθετο δίκτυο, και επίσης είναι αρκετά γενικές, με την έννοια ότι μπορούν να χρησιμοποιηθούν ως αρχέτυπα για πιο εξελιγμένες εφαρμογές επεξεργασίας του δικτύου. Αξιολογήσαμε την απόδοση των SSDs και HDDs εκτελώντας αυτούς τους αλγορίθμους σε πραγματικά δεδομένα κοινωνικών δικτύων εξαιρουμένων των επιπτώσεων του εύρους ζώνης του δικτύου που μπορεί δραματικά να αλλοιώσει τα αποτελέσματα. Τα ληφθέντα αποτελέσματα επιβεβαίωσαν εν μέρει παλαιότερες μελέτες που έδειξαν ότι οι SSD είναι επωφελείς για Hadoop. Ωστόσο, δόθηκαν σοβαρές αποδείξεις για τον σημαντικό ρόλο του μοντέλου επεξεργασίας στην εφαρμογή που εκτελείται, και ως εκ τούτου μελλοντικές μελέτες δεν πρέπει να προσθέσουν τυφλά SSDs στο Hadoop, αλλά θα πρέπει να δημιουργήσουν συστατικά για να εκτιμηθεί το είδος του μοτίβου επεξεργασίας της εφαρμογής και στη συνέχεια να κατευθύνουν τα δεδομένα στο κατάλληλο μέσο αποθήκευσης.

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.

Contents

Figure List

List of Algorithms

List of Tables

Chapter 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 [18]. 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 [2], a big data analytics platform.

The Hadoop [5][23] 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/exploit the potential of the new technologically advanced drive [10][11][17]. The lack 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 [15] to assess the benefits of using SSDs in various points of the database architecture, but the Hadoop world has only recently

[11][12][17][24] 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 [12], or because their setup was affected by the underlying network [17], or because their benchmark algorithms and data were mostly read-oriented [11] [17], thus biasing the results in such a way that no clear answer and universally holding conclusions could be drawn.

1.1 Contributions

Our work 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 work 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 [16] where random writes in SSDs are the "killing" application pattern for SSDs (with respect to reads and sequential writes).

The rest of this work is organized as follows: In chapter 2, we present the related work and in chapter 3 we briefly describe Hadoop's structure. In chapter 4 we provide information about the three algorithms that will be evaluated in the storage media and in chapter 5 we describe the performance evaluation. Chapter 6 contains the evaluation results, and finally, chapter 7 concludes the thesis.

Chapter 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 [11][12][17][21][24]. The first effort [12] 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 [17] 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 [11], 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 [21] using Intel's HiBench benchmark [3][4] 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 [24] 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][22] 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.

Finally, some works propose extensions to Hadoop with SSDs. For instance, [10] proposes extensions to enable clusters of reconfigurable active SSDs to process streaming data from SSDs using FPGAs. VENU [14] 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., [15].

Chapter 3

Hadoop Structure

Hadoop is a free framework, written in the Java programming language which allows the processing of large data sets in a distributed computing environment.

HDFS and MapReduce (MR) are the two core components of Apache Hadoop. With latest versions of Hadoop (version 2.x) YARN (Yet Another Resource Negotiator) has been added.

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 128MB) making it suitable for applications that have large 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. Submitting a MapReduce job usually splits the input file to several chunks (block sized) that are processed by map 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 data.

The key functions to be implemented are Map and Reduce. The MR framework functions exclusively on <key,value> pairs. Each map 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.

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.

Output of reduce task is of format <key,value> as well.

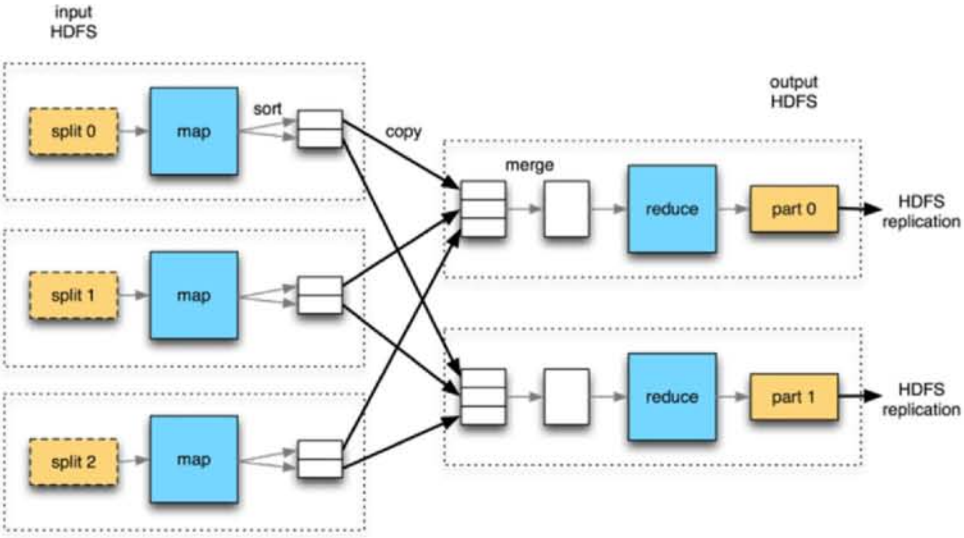


Figure 1 Overview of Map/Reduce and Hadoop

Chapter 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 [13], k-shell computation [20], k-means clustering [26], neural networks [26], etc.

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[1], 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 [19]. 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.

Table 1 Characterization of problems/algorithms examined

Primitive task	Type of analysis	Type of analysis
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

We need to emphasize that it is not the purpose of our work 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 [3][4]) 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 and computer patterns 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 [1], 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 [11][12][17], and based on these features we draw some conclusions on the relative benefits of SSDs. For purposes of our work's self-completeness, we present in the following three sections the selected algorithms and a brief explanation of their operation.

[1] www.facebook.com

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 Algorithm 1.

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:

Algorithm 1 MapReduce pseudo-code for finding mutual friends.

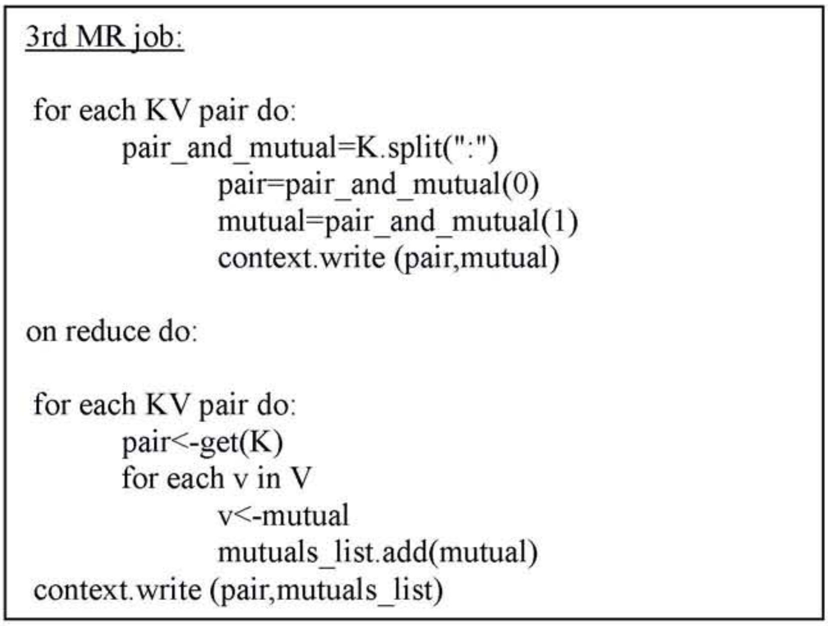
```
1st MR job - CalculateAdjacencyList:

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:

 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)
```

- 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 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 [7] of an iterative algorithm based on message passing technique is used (see Algorithm 2).

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.

Algorithm 2 MapReduce pseudo-code for finding connected components

```
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)
end map

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)
end reduce
```

```
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)
end map

on reduce do:
    for each K[V] pair do:
        if v in Vis 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)
end reduce
```

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. We used the implementation by Walkauskas [6] (pseudo-code in Figure 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 MR Job maps previous input line. And 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 MR Job aggregates the number of triangles that was found from previous job for all chunks.

Algorithm 3 MapReduce pseudo-code for triangle counting.

```
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)
```

Chapter 5

Performance evaluation

For the evaluation of the two disk types *a sample of real data* was required. Recall that earlier efforts e.g., [17] used dummy data files that were read and some primitive statistics were written out. Social networks are a representative sub-genre of complex networks. Thus up to ten real social network graphs were used (Table 2). 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).

Table 2 Social Networks used for evaluation

#	Social Network Name	Num of Nodes	Num of 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	317,08	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

The evaluation will take place along two dimensions. The first one is similar to that in [17] 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 2. 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. 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 [17]. 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.

5.1 The Testing environment

A commodity computer (Table 3) was used for the experiments. Three storage media were used (Table 3) with capacities similar to that used in [17]. CPU and disks were similar to the ones used in [12]. On each of the three drives (one HDD and two SSDs) a separate and identical installation of the latest version of 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 5). In all these setups, speculative execution was disabled and no early shuffling was permitted.

Table 3 System specifications

Hardware		**Software**	
CPU:	Intel i5 4670 3.4Ghz (non HT)	OS:	Ubuntu 14.04 LTS 64bit
RAM:	8gb 1600mhz DDR3 (1333mhz with disabled XMP)	Java SDK:	Oracle Java 1.8.0_25 (8u25)
Disk 1 (HDD):	Western Digital Blue WD10EZEX 1TB	Hadoop Installation:	Hadoop 2.5.2 (pre-built 32-bit i386-Linux native Hadoop library)
Disk 2 (SSD1):	Samsung 840 Evo 120GB	Monitoring tools:	Collectl V3.6.9-1
Disk 3 (SSD2):	Crucial MX100 512GB		

Power saving options and boosting technologies like Turbo-boost and IEST were disabled through BIOS to minimize unexpected fluctuations among executions (Table 4).

Table 4 Disabled BIOS settings

Disable Settings	Description
Turbo boost	Increases core's frequency at high workloads taking in consideration parameters like temperature
Enhanced Halt (C1E)	Power saving feature when system is idle
C3 State	Power saving state enabled during sleep mode
C 6/7 State	They Offer reduced idle power consumption
EIST (Enhanced Intel Speed Step Technology)	Allows the system to dynamically adjust processor voltage and core frequency
Intel Extreme Memory Profile (XMP)	Allows over-clocking compatible DDR3/DDR4 memory to perform beyond standard specifications

Hadoop was installed in pseudo-distributed mode where each Hadoop daemon runs as a separate Java process. Default settings were preserved.

Table 5 Custom Hadoop 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	*4kb->64kb*

Chapter 6

Experimental results

In this section we provide the obtained results for each one of the three algorithms presented earlier, starting with standard TestDFSIO results.

6.1 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. For writing sequential files, with the increase of file size, SSD1"s performance is decreasing, falling behind the HDD. Contrariwise, the SSD2 appears much faster (Figure 2) with stable throughput.

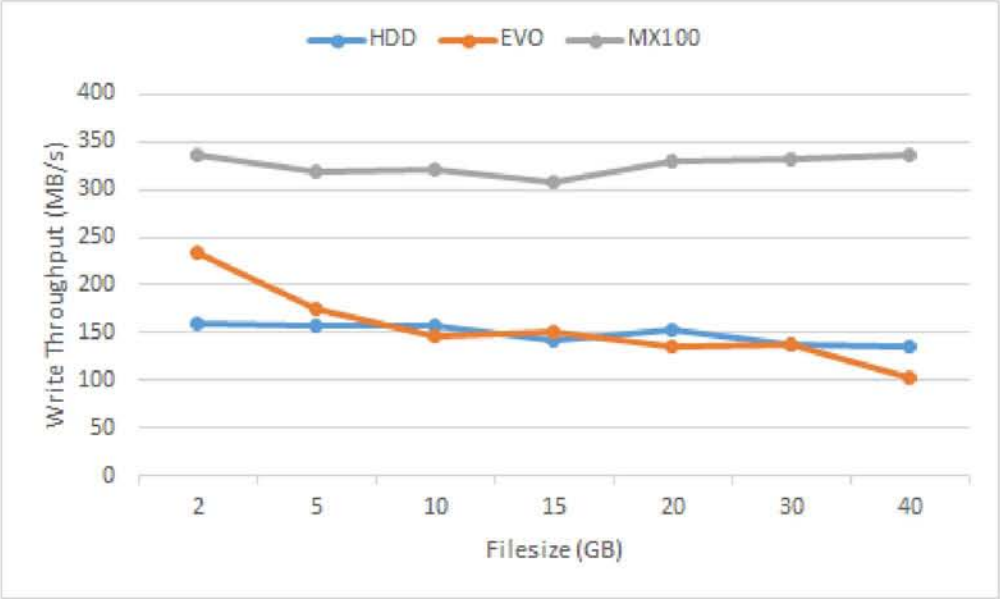


Figure 2 Comparing TestDFSIO write throughput for 3 disks.

As expected, both SSDs' sequential read throughput is outstanding. The magnetic disk again demonstrates stable performance, although noticeably slower than that of the SSDs (Figure 3).

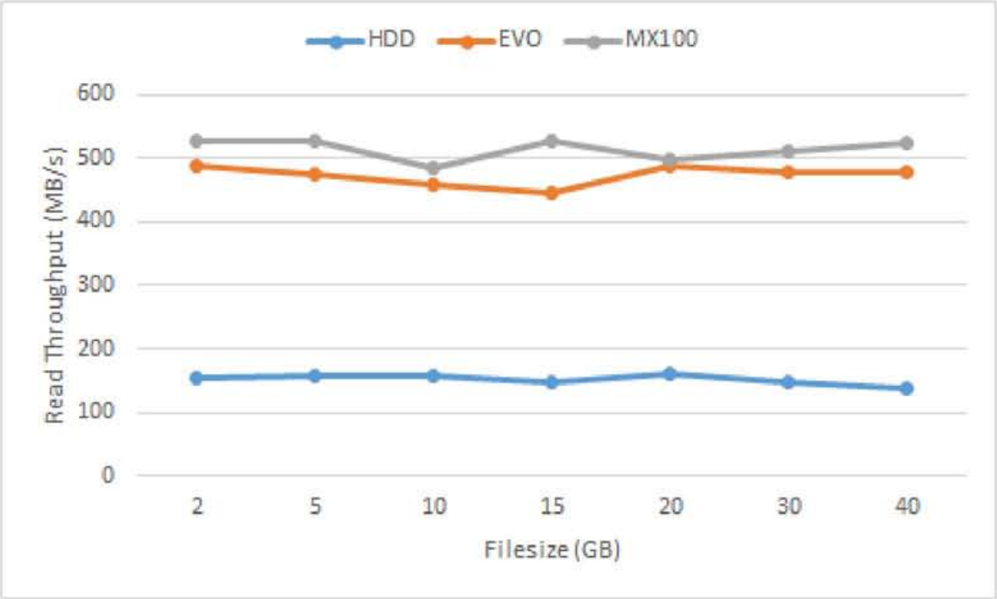
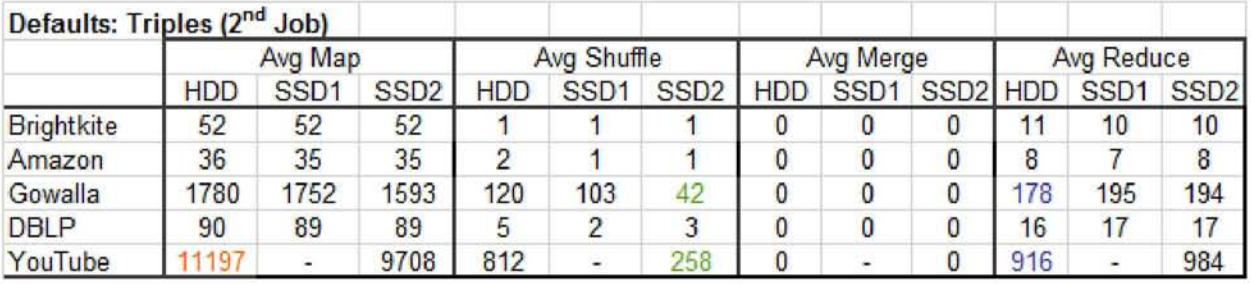


Figure 3 Comparing TestDFSIO read throughput for 3 disks

6.2 Mutual Friends

The complexity of this algorithm is exponential due to the mapper of the 2nd MapReduce job ("creating triples" - as described at 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 execution times for reduce phase, for bigger datasets, where HDD performs lower for map phase. The SSD2 displays superior performance at shuffling.
Times for the 1st M-R job "Creating the adjacency" and the 3rd M-R job "Calculating the mutual friends" were quite low and almost identical between SSDs and HDD.

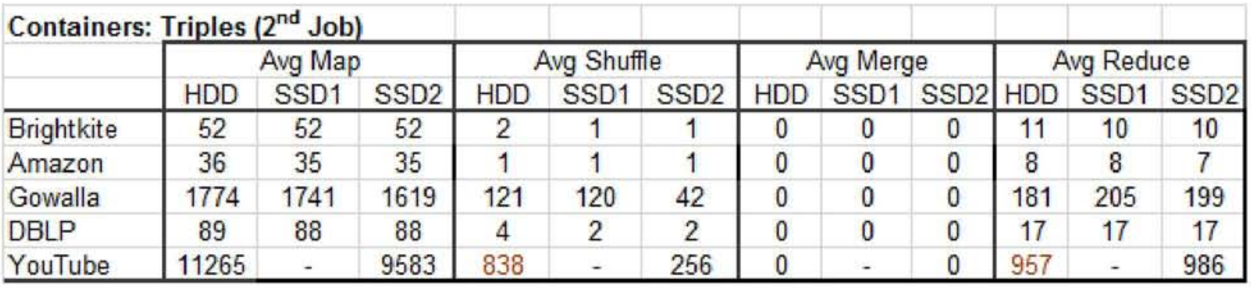
Table 6 Average times for each phase for 2nd job (creating triples) of "mutual friends" algorithm using default settings



Defaults: Triples (2nd Job)												
	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

Modifying container's size doesn't affect the performance in general, comparing to defaults, for the mutual friends' algorithm.

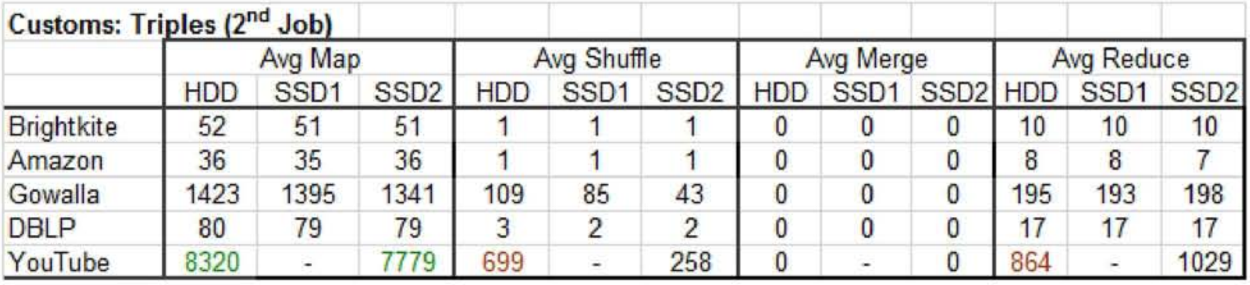
Table 7 Average times for each phase for 2nd job (creating triples) of "mutual friends" algorithm using modified containers settings



Containers: Triples (2nd Job)												
	Avg Map			Avg Shuffle			Avg Merge			Avg Reduce		
	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2
Brightkite	52	52	52	2	1	1	0	0	0	11	10	10
Amazon	36	35	35	1	1	1	0	0	0	8	8	7
Gowalla	1774	1741	1619	121	120	42	0	0	0	181	205	199
DBLP	89	88	88	4	2	2	0	0	0	17	17	17
YouTube	11265	-	9583	838	-	256	0	-	0	957	-	986

Using custom settings decreases map times for big datasets. HDD for YouTube gets increased performance for shuffle and reduce, whereas SSD2 gets same performance or even worst at reduce phase.

Table 8 Average times for each phase for 2nd job (creating triples) of “mutual friends” algorithm using custom settings

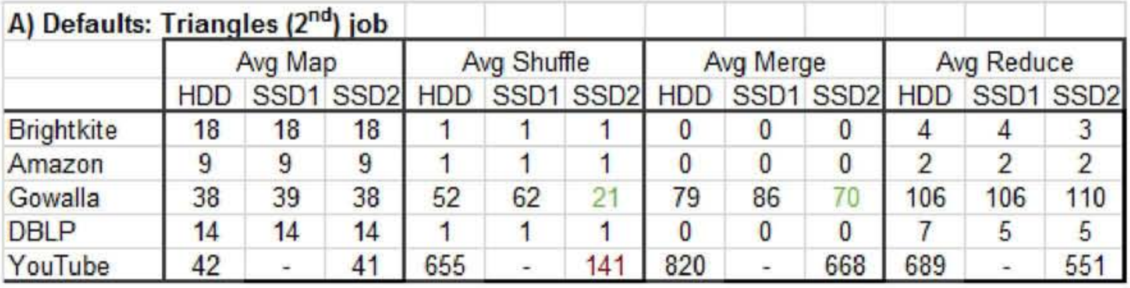


Customs: Triples (2^{nd} Job)												
	Avg Map			Avg Shuffle			Avg Merge			Avg Reduce		
	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2
Brightkite	52	51	51	1	1	1	0	0	0	10	10	10
Amazon	36	35	36	1	1	1	0	0	0	8	8	7
Gowalla	1423	1395	1341	109	85	43	0	0	0	195	193	198
DBLP	80	79	79	3	2	2	0	0	0	17	17	17
YouTube	8320	-	7779	699	-	258	0	-	0	864	-	1029

6.3 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 reduce phase (Table 10). 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 9). But with larger ones (like YouTube dataset), SSDs capabilities become evident for shuffle and merge (sort) phases.

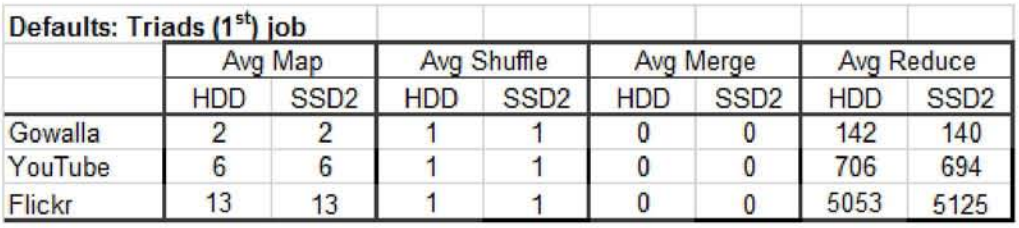
Table 9 Average times for each phase for 2nd job (calculate triangles) of “counting triangles” algorithm, using default settings



A) Defaults: Triangles (2^{nd}) job												
	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

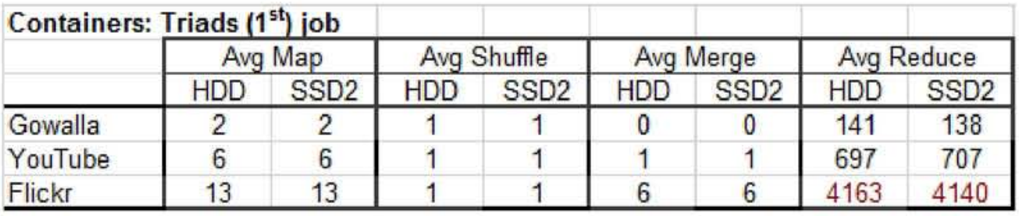
For the 1st MR job (creating triads), map, shuffle and merge phases finished quite fast and with almost zero differentiations among disks. Reduce phase lasted significantly longer with both disks performing equally (Table 9). With containers settings, the biggest dataset of Flickr gets significant improvement for both disk types (Table 10). No further improvement achieved with custom settings.

Table 10 Average times for each phase for 1st job (create triads) of "counting triangles" algorithm, using default settings.



Defaults: Triads (1st) job								
	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

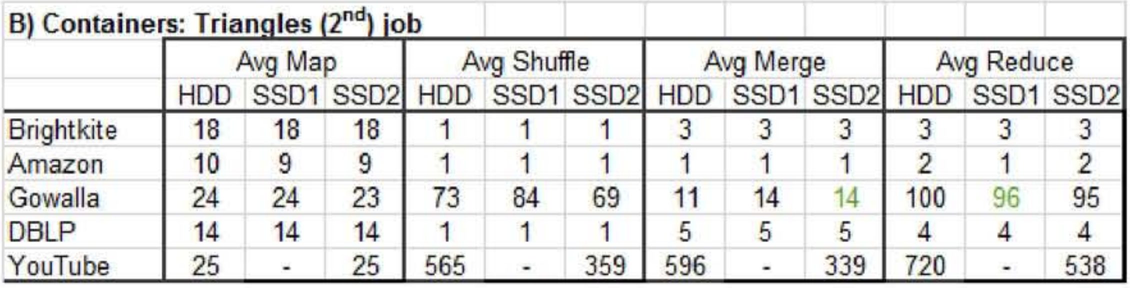
Table 11 Average times for each phase for 1st job (create triads) of "counting triangles" algorithm, using modified containers.



Containers: Triads (1st) job								
	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

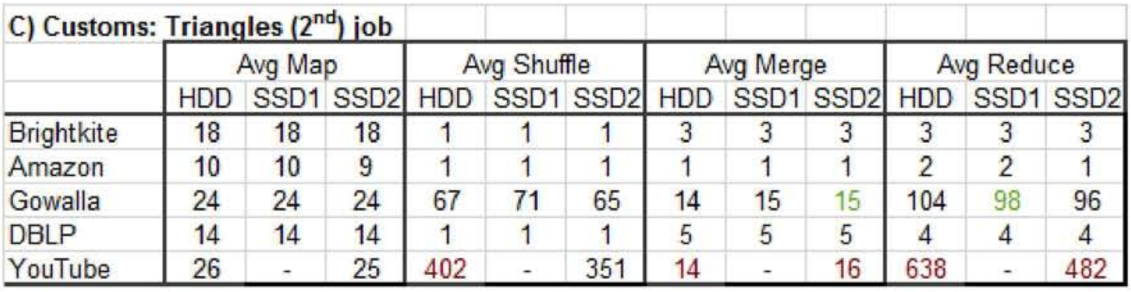
Comparing containers settings (Table 11) to customs (Table 12) for the 2nd job increased performance can be noticed, especially for the magnetic disk. For YouTube at shuffling, only the HDD gets improved times. Both disks get improvement at reduce times. Especially at merge, using custom settings minimizes the sorting times for both disks, favoring mostly the HDD, whose times are much higher than SSD2 with just modified containers settings.

Table 12 Average times for each phase for 2nd job (calculate triangles) of "counting triangles" algorithm, using modified containers settings



B) Containers: Triangles (2nd) job												
	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	3	3	3	3	3	3
Amazon	10	9	9	1	1	1	1	1	1	2	1	2
Gowalla	24	24	23	73	84	69	11	14	14	100	96	95
DBLP	14	14	14	1	1	1	5	5	5	4	4	4
YouTube	25	-	25	565	-	359	596	-	339	720	-	538

Table 13 Average times for each phase for 2nd job (calculate triangles) of "counting triangles" algorithm, using custom settings.



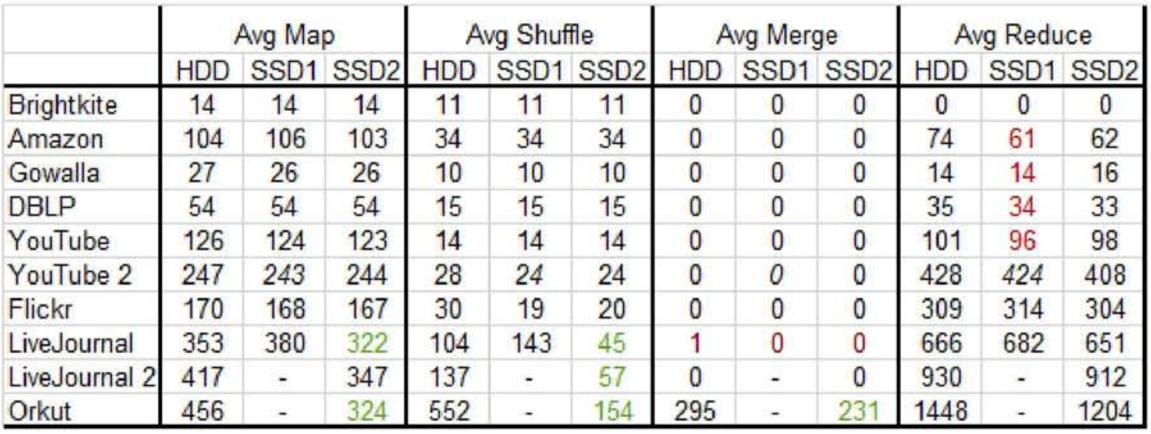
C) Customs: Triangles (2nd) job

	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	3	3	3	3	3	3
Amazon	10	10	9	1	1	1	1	1	1	2	2	1
Gowalla	24	24	24	67	71	65	14	15	15	104	98	96
DBLP	14	14	14	1	1	1	5	5	5	4	4	4
YouTube	26	-	25	402	-	351	14	-	16	638	-	482

6.4 Connected Components

Comparing SSD1 to the HDD, the Connected Components algorithm seems to slightly favor the SSD1 for small datasets (first five ones), at reduce phase. Map, shuffle and phase times are close for both disk types. For the datasets of Flickr and LiveJournal the magnetic disk takes the lead at reduce phase which is mostly characterized as "write" procedure for the Hadoop framework. Surprisingly, SSD1 performs quite slowly at shuffle phase for the LiveJournal dataset. 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.

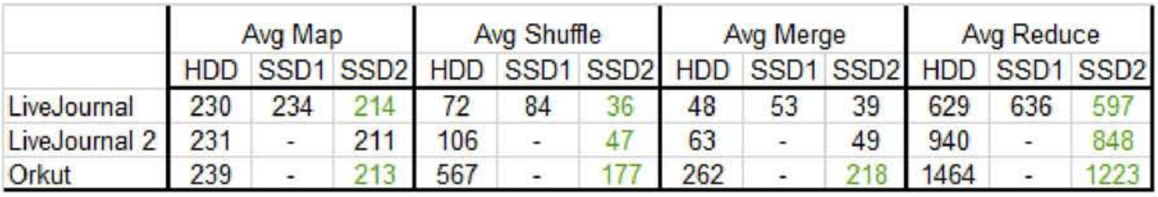
Table 14 Sum of average times for each phase for the iterative Jobs of "Connected Components"



	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
LiveJournal	353	380	322	104	143	45	1	0	0	666	682	651
LiveJournal 2	417	-	347	137	-	57	0	-	0	930	-	912
Orkut	456	-	324	552	-	154	295	-	231	1448	-	1204

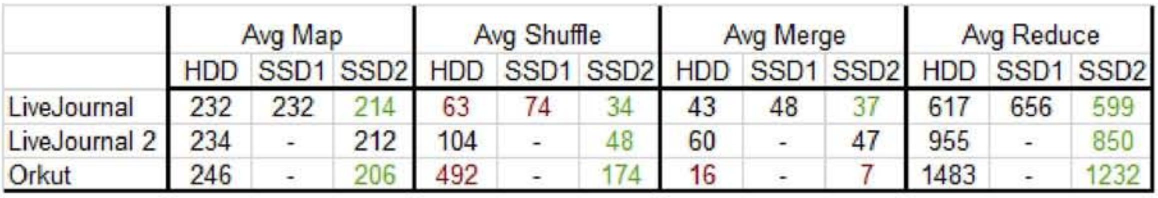
Similarly, to the "Counting Triangles" algorithm, using custom settings minimizes merge time for both disks with big datasets, improving at the same time the shuffling of the magnetic disk. The SSD2 already performs great and gets no further improvement.

Table 15 Sum of average times for each phase for the iterative Jobs of "Connected Components", containers settings



	Avg Map			Avg Shuffle			Avg Merge			Avg Reduce		
	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2
LiveJournal	230	234	214	72	84	36	48	53	39	629	636	597
LiveJournal 2	231	-	211	106	-	47	63	-	49	940	-	848
Orkut	239	-	213	567	-	177	262	-	218	1464	-	1223

Table 16 Sum of average times for each phase for the iterative Jobs of "Connected Components", custom settings



	Avg Map			Avg Shuffle			Avg Merge			Avg Reduce		
	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2	HDD	SSD1	SSD2
LiveJournal	232	232	214	63	74	34	43	48	37	617	656	599
LiveJournal 2	234	-	212	104	-	48	60	-	47	955	-	850
Orkut	246	-	206	492	-	174	16	-	7	1483	-	1232

Hadoop's default settings allowed the 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 (Figure 4), 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 (Figure 5-Figure 7). At reduce, SSD2 performs slightly better.

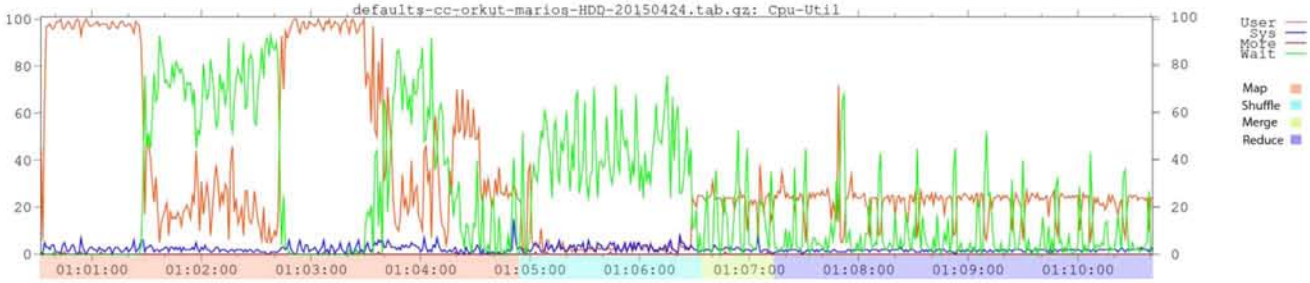


Figure 4 CPU utilization for Connected Components algorithm with Orkut, using HDD, 1st 1teration, default settings

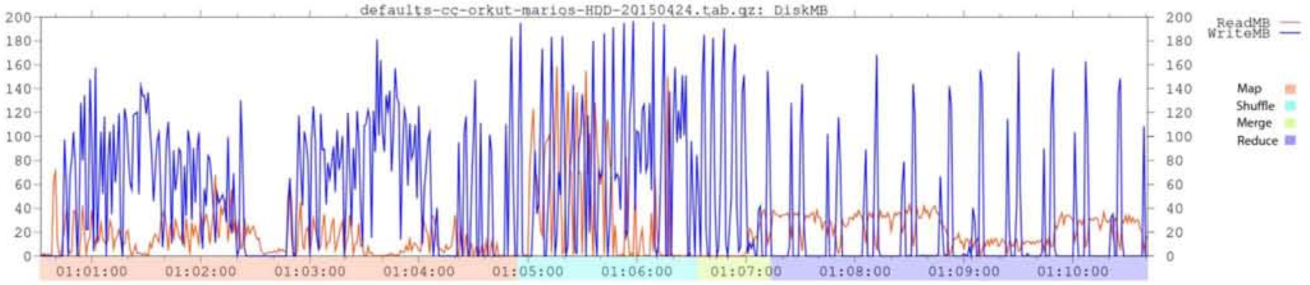


Figure 5 Disk Usage for Connected Components algorithm with Orkut, using HDD, 1st 1teration, default settings

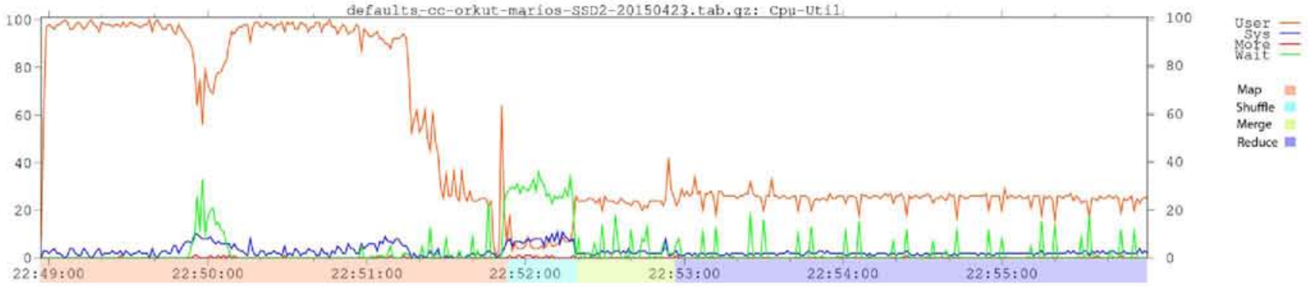


Figure 6 CPU utilization for Connected Components algorithm with Orkut, using SSD2, 1st 1teration, default settings

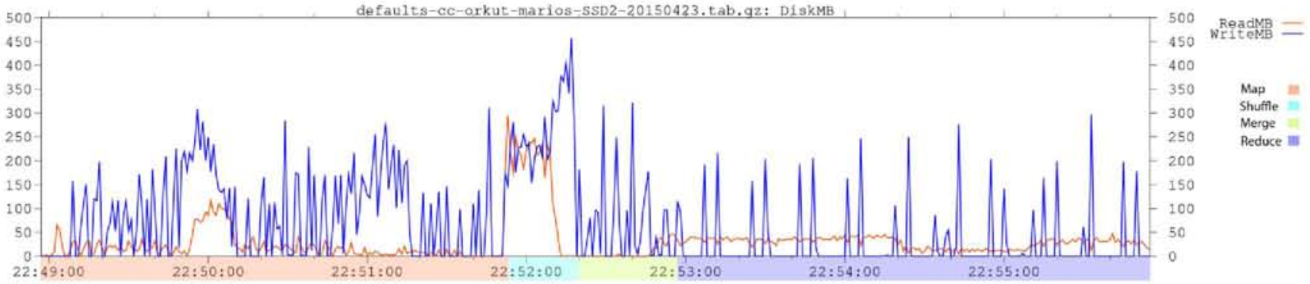


Figure 7 Disk Usage for Connected Components algorithm with Orkut, using SSD2, 1st 1teration, default settings

Using custom settings decreased times for both disk types (Table 17), especially for the magnetic disk, with improved CPU wait times (fig. 8).

Table 17 Execution times for 1st iteration of Connected components algorithm, at Orkut, using default and custom settings.

	Defaults					**Customs**				
	Elapsed	Avg Map	Shuffle	Merge	Reduce		Avg Map	Shuffle	Merge	Reduce
HDD	10mins, 15sec	95	89	36	220	8mins, 38sec (-1min, 37sec)	43	75	1	212
SSD2	7mins, 2sec	60	27	34	184	6mins, 56sec (-6 sec)	37	30	1	181

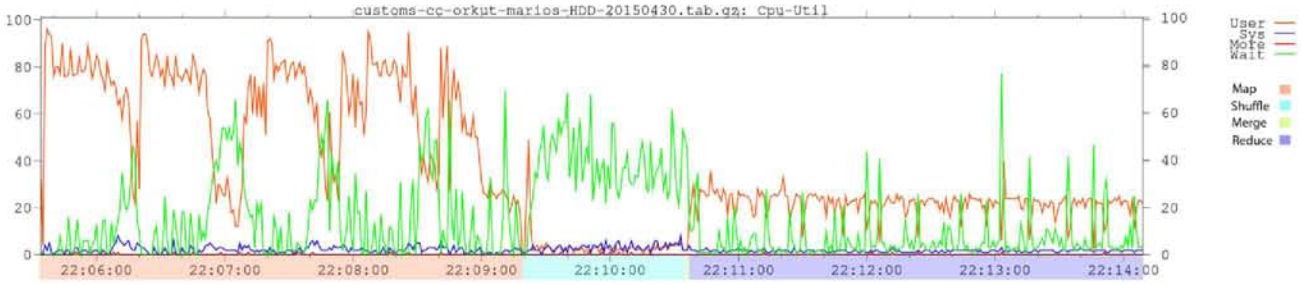


Figure 8 CPU utilization for Connected Components algorithm with Orkut, using HDD, 1st 1teration, custom settings

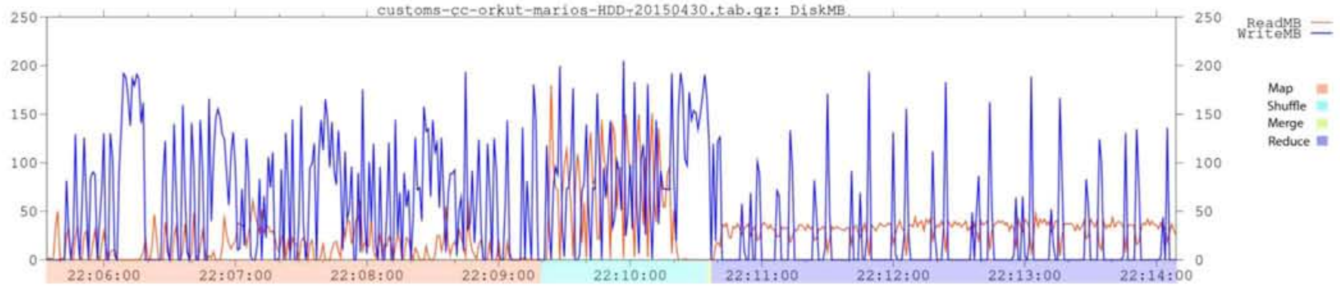


Figure 9 Disk Usage for Connected Components algorithm with Orkut, using HDD, 1st 1teration, custom settings

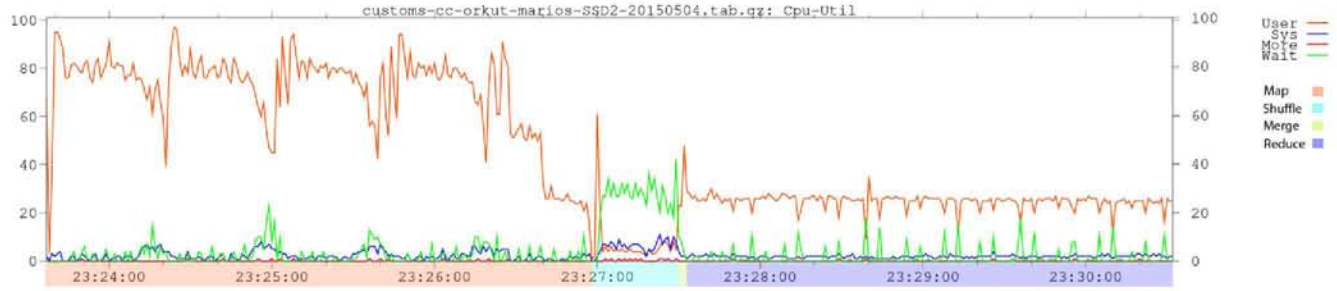


Figure 10 CPU utilization for Connected Components algorithm with Orkut, using SSD2, 1st 1teration, custom settings

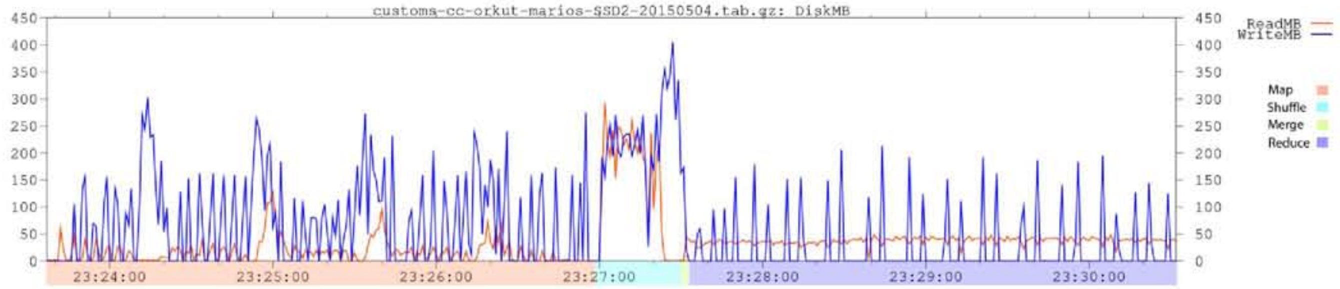


Figure 11 Disk Usage for Connected Components algorithm with Orkut, using SSD2, 1st 1teration, custom settings

6.5 Optimization settings

Hadoop contains a large number of configuration settings [27] that affect system's performance.

We tested a variety of optimizations with the magnetic disk's performance getting significant improvement.

Not all the settings triggered noticeable improvement.

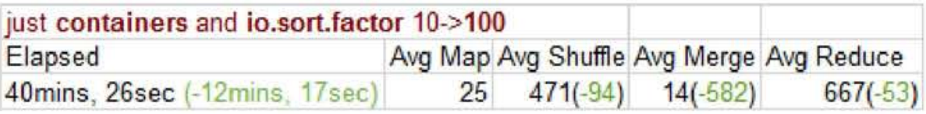
Generally, increasing the number of streams to merge at once while sorting files - *mapreduce.task.io.sort.factor* – from 10 to 100 minimizes the merge (sort) time, favoring mostly the magnetic disk.

Increasing the size of buffer for use in sequence files - *io.file.buffer.size* – from 4kb to 128kb gives a general boost at HDD performance, besides map phase. Further increase of the above setting causes worst performance. The SSD's performance remains unaffected by changing the file buffer's size.

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

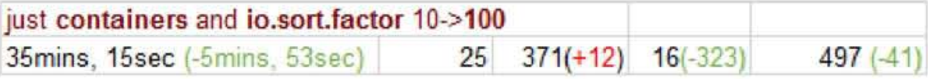
a) The number of streams to merge at once while sorting files. Minimizes merge time for both disk types. Improves HDD shuffling time as well.

Table 18 Performance difference for YouTube dataset at "Counting Triangles", increasing sort factor, for HDD



just **containers** and **io.sort.factor** 10->**100**				
Elapsed	Avg Map	Avg Shuffle	Avg Merge	Avg Reduce
40mins, 26sec (-12mins, 17sec)	25	471(-94)	14(-582)	667(-53)

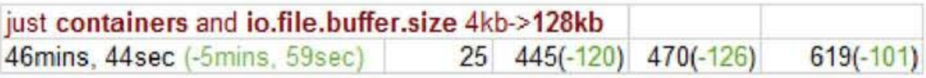
Table 19 Performance difference for YouTube dataset at "Counting Triangles", increasing sort factor, for SSD2



just **containers** and **io.sort.factor** 10->**100**				
35mins, 15sec (-5mins, 53sec)	25	371(+12)	16(-323)	497 (-41)

b) The buffer size for I/O (read/write) operations.

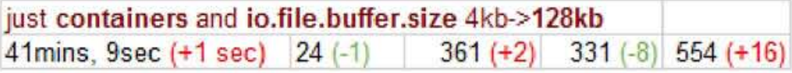
Table 20 Performance difference for YouTube dataset at "Counting Triangles", increasing file buffer size, for HDD



just **containers** and **io.file.buffer.size** 4kb->**128kb**				
46mins, 44sec (-5mins, 59sec)	25	445(-120)	470(-126)	619(-101)

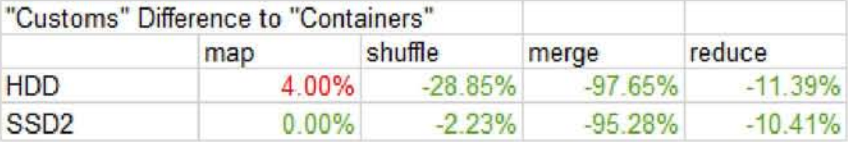
On the other hand, increasing the buffer size for I/O operations had minimal effect on SSD2 performance.

Table 21 Performance difference for YouTube dataset at "Counting Triangles", increasing file buffer size, for SSD2



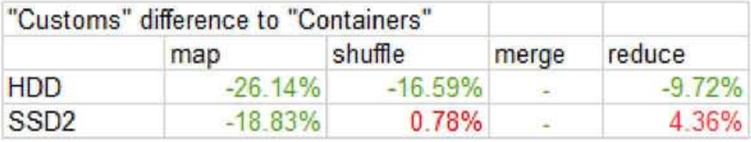
just **containers** and **io.file.buffer.size** 4kb->**128kb**				
41mins, 9sec (+1 sec)	24 (-1)	361 (+2)	331 (-8)	554 (+16)

Table 22 Percentage difference between "customs" and "containers settings for YouTube dataset, at "Counting Triangles" algorithm



"Customs" Difference to "Containers"				
	map	shuffle	merge	reduce
HDD	4.00%	-28.85%	-97.65%	-11.39%
SSD2	0.00%	-2.23%	-95.28%	-10.41%

Table 23 Percentage difference between "customs" and "containers settings for YouTube dataset, at "Mutual Friends" algorithm



"Customs" difference to "Containers"				
	map	shuffle	merge	reduce
HDD	-26.14%	-16.59%	-	-9.72%
SSD2	-18.83%	0.78%	-	4.36%

Chapter 7

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.

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 comparing to the faster SSD2, the magnetic disk provided competitive times for reduce phase, especially with the "mutual friends" algorithm, where it performed marginally better.

Customizing Hadoop settings proves crucial. Magnetic disk's shuffle times can be 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" 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] Y. Chen, A. Ganapathi, R. Griffith, R. Katz, "*The case for evaluating MapReduce performance using workload suites*", Proceedings of the IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 390-399, 2011.

[2] J. Dean, S. Ghemawat, "*MapReduce: Simplified data processing on large clusters*", Proceedings of the USENIX/ACM Symposium on Operating Systems Design and Implementation (OSDI), pp. 137-150, 2004.

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

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

[5] https://hadoop.apache.org/

[6] http://www.vertica.com/2011/09/21/counting-triangles/

[7] http://codingwiththomas.blogspot.de/2011/04/graph-exploration-with-hadoop-mapreduce.html

[8] N. Islam, M. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. Panda, "*High performance RDMA-design of HDFS over InfiniBand*" 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*", Communications of the ACM, vol. 57, no. 7, pp. 86-94, 2014.

[10] A. Kaitoua, H. Hajj, MA. R. Saghir, H. Artail, H. Akkary, M. Awad, M. Sharafeddine, K. Mershad, "*Hadoop extensions for distributed computing on reconfigurable active SSD clusters*", ACM Transactions on Architecture and Code Optimization, vol. 11, no. 2, article 22, 2014.

[11] K. Kambatla, Y. Chen, "*The truth about MapReduce performance on SSDs*", Proceedings of the Large Installation System Administration Conference (LISA), pp. 109-117, 2014.

[12] S.-H. Kang, D.-H. Koo, W.-H. Kang, S.-W. Lee, "*A case for flash memory SSD in Hadoop applications*", International Journal of Control and Automation, vol. 6, no. 1, pp. 201-201, 2013.

[13] T.G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, C. Task, "*Counting triangles in massive graphs with MapReduce*", SIAM Journal on Scientific Computing, vol. 36, no. 5, pp. 48-77, 2014.

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

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

[16] C. Min, K. Kim, H. Cho, S.-W. Lee, Y.I. Eom, "*SFS: Random Write Considered Harmful in Solid State Drives*", Proceedings of the USENIX Conference on File and Storage Technologies (FAST), 2012.

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

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

[19] G. Palla, I. Derenyi, I. Farkas, T. Vicsek, "*Uncovering the overlapping community structure of complex networks in nature and society*", Nature, vol. 435, pp. 814-818, 2005.

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

[21] 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*", International Journal of Modern Communication Technologies & Research, vol. 2, no. 5, 2014.

[22] S. Sur, H. Wang, J. Huang, X. Ouyang, and D. Panda, "*Can high-performance interconnects benefit Hadoop Distributed File System*", Proceedings of the Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC), 2010.

[23] T. White, "*Hadoop: The Definitive Guide*", O'Reilly Media, 2012.

[24] D. Wu, W. Xie, X. Ji, W. Luo, J. He, D. Wu, "*Understanding the impacts of Solid-State Storage on the Hadoop performance*", Proceedings of the International Conference on Advanced Coud and Big Data, pp. 125-130, 2013.

[25] K. Zhang, X.-W. Chen, "*Large-scale Deep Belief Nets with MapReduce*", IEEE Access, vol. 2, pp. 395-403, 2014.

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

[27] http://hadoop.apache.org/docs/r2.5.2/hadoop-project-dist/hadoop-common/ClusterSetup.html