An approach for choosing the right distributed file system

Microsoft DFS vs. Hadoop DFS

Mihai Musatoiu

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

Contact Information:
Author:
Mihai Musatoiu
E-mail: mimu12@student.bth.se

University advisor:
Conny Johansson
Senior lecturer at the Department of Software Engineering, DIPT

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context. An important goal of most IT groups is to manage server resources in such a way that their users are provided with fast, reliable and secure access to files. The modern needs of organizations imply that resources are often distributed geographically, asking for new design solutions for the file systems to remain highly available and efficient. This is where distributed file systems (DFSs) come into the picture. A distributed file system (DFS), as opposed to a "classical", local, file system, is accessible across some kind of network and allows clients to access files remotely as if they were stored locally.

Objectives. This paper has the goal of comparatively analyzing two distributed file systems, Microsoft DFS (MSDFS) and Hadoop DFS (HDFS). The two systems come from different "worlds" (proprietary - Microsoft DFS - vs. open-source - Hadoop DFS); the abundance of solutions and the variety of choices that exist today make such a comparison more relevant.

Methods. The comparative analysis is done on a cluster of 4 computers running dual-installations of Microsoft Windows Server 2012 R2 (the MSDFS environment) and Linux Ubuntu 14.04 (the HDFS environment). The comparison is done on read and write operations on files and sets of files of increasing sizes, as well as on a set of key usage scenarios.

Results. Comparative results are produced for reading and writing operations of files of increasing size - 1 MB, 2 MB, 4 MB and so on up to 4096 MB - and of sets of small files (64 KB each) amounting to totals of 128 MB, 256 MB and so on up to 4096 MB. The results expose the behavior of the two DFSs on different types of stressful activities (when the size of the transferred file increases, as well as when the quantity of data is divided into (tens of) thousands of many small files). The behavior in the case of key usage scenarios is observed and analyzed.

Conclusions. HDFS performs better at writing large files, while MSDFS is better at writing many small files. At read operations, the two show similar performance, with a slight advantage for MSDFS. In the key usage scenarios, HDFS shows more flexibility, but MSDFS could be the better choice depending on the needs of the users (for example, most of the common functions can be configured through the graphical user interface).

Keywords: DFS, MSDFS, HDFS, Microsoft, Hadoop

TABLE OF CONTENTS

1 INTRODUCTION

This paper follows a research pattern according to which two or more systems are comparatively analyzed in order to expose strengths and weaknesses and, wherever possible, make usage recommendations. This paper makes a comparative analysis of two distributed file systems, Hadoop DFS (HDFS) and Microsoft DFS (MSDFS).

Distributed file systems are receiving increasing attention in recent years, mainly due to the rise of cloud computing solutions, which focus on relocation of resources and optimization of their usage [Wu et al, 2014, pp. 156].

Cloud computing has become a technology that affects everyone's daily lives. With more and more of our personal and business data saved "in the cloud", reliable, flexible storage is today a key requirement for any large-scale business. Among the Top 10 players who are investing heavily in cloud solutions are Amazon, IBM, Microsoft, Google, VMware [Business Insider, 2013].

The development of cloud computing has brought with it the diversification of storage solutions. In this area, file systems play an important role. From the classic Local File System, file systems have today evolved into many complementary forms. Currently, there are several types of file systems – Local FS, Shared FS, SAN FS, Clustered FS, Network FS, Distributed FS, Parallel FS [Bandulet, 2009, pp. 1].

Distributed File Systems (DFSs) play an important role in this ecosystem. Being a type of network file system, it allows clients to read and write files from what appears to be a single file server. In fact, for availability and reliability purposes, the files are physically stored on several servers. The files can be stored either as complete, mirrored copies, or they can be first divided into basic units of storage, called blocks, and then spread across the file servers [Bandulet, 2009, pp. 8-9]. The two distributed file systems analyzed in this paper are examples of these two categories (MSDFS distributes complete copies of the files, while HDFS distributes smaller storage units called blocks).

1.1 Research questions and focus

The research is focused on finding out how two well-known distributed file systems, Microsoft DFS (MSDFS) and Hadoop DFS (HDFS) perform on a set of strictly-defined workloads, focusing on the two most important functions of any file system (reading and writing files), and on a number of key usage scenarios.

The **main research questions** are:

- **RQ1**: In which areas (for which attributes analyzed) is HDFS superior to MSDFS and why?
- **RQ2**: In which areas, on the contrary, is MSDFS superior to HDFS and why?
- **RQ3**: Which of the two systems is superior in the key usage scenarios observed, and why? (see sub-questions RQ 3.1 – 3.4 below)

The questions will be answered through running the experiments and analyzing the results. The goal is to produce measurements and observations that expose how the two DFSs perform and which one is to be preferred depending on the type of work that the user needs to do. The goal of the experimental part of the paper is to define workload scenarios that can lead to conclusions on the performance of the two DFSs when working in increasingly stressful conditions.

For this goal to be achieved, three **methods** are used:

- the size of the file being transferred is gradually increased
- the quantity of data being transferred is fragmented into (tens of) thousands of small files
- the systems are observed in a number of key usage scenarios:
    - "One datanode goes offline":
        - **RQ3.1**: Is all data still available when an external client tries to access it?
        - **RQ3.2**: What is the replication factor after the disconnection? (original replication factor is 3)
    - "Two users open the same file, modify it and save":

        - **RQ3.3**: Have any of the newly added lines been lost?
    - "An administrator wants to configure the file replication schema"
        - **RQ3.4**: What main options are available for the file replication schema?

A motivation for choosing these key scenarios is given in section *1.5 Method of investigation*.

The methods expose how the distributed file systems handle the actual I/O operations and how efficiently they handle the metadata (the data that describe the location and the naming of the files that exist in the file system), both functionalities being of crucial importance when defining the effectiveness of a file system. They also allow for observing how the systems handle some usual working tasks or events that can occur in everyday operations (the key usage scenarios detailed above).

Since the experiment is run on a cluster of small dimensions (4 computers), the research applies to small clusters of commodity hardware.

1.2 Rationale

From a network performance point of view, as well as from a software development view, the performance of the file system that applications run on is of utmost importance, considering that I/O operations and network access can often act as high-latency components.

For this reason, a software architect nowadays should make a choice, whenever possible, regarding the file system that the software will run on.

Since the alternative "proprietary vs. open-source" software is a recurrent problem that implies many long-term consequences, a comparison between Microsoft DFS and Hadoop DFS can make it easier to select the appropriate solution.

1.3 Problem definition

The problem analyzed in this paper is finding out which of the two distributed file systems (HDFS and MSDFS) is to be preferred under different working conditions. The main lines taken when defining the working conditions are

- the size of the files (smaller files versus larger files)
- the number of the files (one bigger file versus many smaller files, that together amount to the same size as the bigger file).
- a set of key usage scenarios is defined and observed.

1.4 Aims

The conclusions of the paper should allow for a system administrator to make a more informed choice when selecting one of the two distributed file systems, depending on the type of work that is predominant for the respective situation.

For example, different working scenarios like the ones below could favor one DFS over the other:

- the file system stores small files, that change often (for example, in the publishing business)
- the file system stores big files, that change very rarely (for example, used in scientific data analysis)

1.5 Method of investigation

The method used in the experiment is benchmark measurements that will allow for conclusions on how the two analyzed systems perform under increasingly stressful conditions (increasing the size of files being transferred and fragmenting the transferred files into many smaller files). The measurements are done using an ad-hoc, micro-benchmark (see section *2.4.1 Benchmarks* for details and a classification of available benchmarks in the field of file systems).

The speed of read and write operations for files of increasing sizes (1 MB, 2 MB, 4 MB, 8 MB, and so on up to 4096 MB) is measured, as well as the change in performance when replacing one big file with many small files (for example, replacing one 1024 MB file with 16,384 files of 64 KB each, amounting to the same total data quantity of 1024 MB). The measurements are interpreted both for each of the two DFSs individually (to show how a certain DFS scales up) and for the two DFSs at the same time, as a face-to-face comparison.

Moreover, the systems will be observed in a number of key usage scenarios:

- one node goes offline
- two users open the same file, modify it and save it
- an administrator wants to configure the file replication schema – how flexible are the options

These key scenarios were determined based on some of the most important attributes observed at file systems in the production environment. According to [Bancroft et al, 2000, pp. 104], a SAN file system (a file system designed to work across a network, much like DFSs) is expected to maintain a very high level of performance, interoperability, maintainability and availability. A list of important attributes is mentioned by the cited source, among which the following can be mentioned:

- Shared concurrent reading and writing of a single file (equivalent of the "two users open the same file, modify it and save it" scenario)
- Scaling in terms of number of clients (equivalent of the "one node goes offline" scenario)
- Comprehensive set of administrative tools for configuration, monitoring and troubleshooting (equivalent of the "an administrator wants to schedule the file replication schema")

Moreover, the chosen scenarios are also backed up by two of the design issues related to distributed file systems presented by [Chow, 1997]: **concurrency control** (equivalent of the "two users are trying to write to the same file at approximately the same time" scenario), and **data replication** (equivalent of the "an administrator wants to schedule the file replication schema" scenario), and an issue presented by [Weil et al, 2006, pp. 312-314]: **scalability** (equivalent of the "one node goes offline" scenario).

1.6 Expected outcome

By analyzing the result of the measurements, conclusions are drawn that allow for defining a clear hierarchy of the two distributed file systems, depending on the various workload scenarios in which they are used (few files, many files, various sizes of the files).

Usage recommendations are made, like examples of situations in which one distributed file system should be preferred to the other.

1.7 Chapters Overview

The paper is divided into following chapters:

- **Introduction**
    - This chapter contains the problem definition and motivation, the research questions, the aims, a brief description of the investigation method and the expected outcome.
- **Theory about (distributed) file systems**
    - This chapter contains background information necessary to understand the problem (file systems, distributed file systems, generalities on MSDFS and HDFS). The chapter also contains an overview of how others have treated the topic, as well as motivation for how this problem is treated in an unique way in this paper)
- **Method**
    - This chapter contains a description of how the literature survey was designed and a description of the experiment design. The experiment description is detailed into Goal, Setup and Methodology.
- **Results**
    - The results of the experiment are presented in this chapter. Tables and charts are used to illustrate the results wherever appropriate.
- **Analysis and Answers to Research Questions**
    - This is the chapter that presents the results in a broader sense, with interpretation and comments that pinpoint the answers to the research questions.

- **Conclusions**
    - In this chapter, the work is summarized and put in a bigger perspective. This chapter is meant to act as a summary for the whole work, so that anyone reading only this chapter would be able to get a quick grasp of what the work is about and what the outcome was.
- **Future Work**
    - This section presents a proposal for how the problem can be further investigated. New methods and new attributes that can be investigated are proposed.

2 THEORY ABOUT (DISTRIBUTED) FILE SYSTEMS

This chapter presents the concepts related to the file systems and distributed file systems in general, and to MSDFS and HDFS in particular, that are being used in this paper and that are required in order to define the problem and to understand the results.

There are many examples of previous comparative analysis papers in this field, out of which only a few are:

- "File Systems in Linux and FreeBSD: A comparative study", by Kuo-pao Yang and Katie Wallace (2011) – which compares file systems in Ubuntu Linux and FreeBSD and then analyzes the best utilization
- "Comparative Analysis of Common Approaches in Distributed File Systems (Based on WebNFS, AFS and CIFS), by Yuri A. Izrailevsky (1997) – which analyzes the commercial implementations of Sun's WebNFS, TransArc's AFS and Microsoft's CIFS distributed file systems
- "A Comparative Experimental Study of Parallel File Systems for Large-Scale Data Processing", by Zoe Sebepu, Kostas Magoutis, Manolis Marazakis and Angelos Bilas, which compares two parallel file systems, PVS2 and Lustre (2008).

2.1 File Systems

File systems impose structure on the address space of one or more physical or virtual devices. They have evolved significantly over time. Starting from the initial local file system, during the years many more types of file systems have appeared, focusing on special areas or requirements, like remote file access, sharing of data, distributed file access, archiving, security etc. It is believed that there are about a thousand different file systems on the market [Bandulet, 2009, pp. 1-2].

According to a taxonomy of the file systems mentioned by the cited source, there are three different major families of file systems: **local**, **shared**, and **network** file systems.

➢ The **local** file system is co-located with applications on a single server. The storage is either directly attached to the server or accessed via a Storage Area Network (SAN).

- The **shared** file systems introduce the additional complexity of allowing several clients to access the same data concurrently. Shared file systems can be further categorized as *SAN file systems* (a master/slave architecture, where the metadata server plays the role of the master) and *cluster file systems* (the metadata server is no longer a dedicated machine, but the metadata service is distributed across all participants).

- The **network** file systems are similar to the fore-mentioned SAN file systems, the major difference being that in a SAN system clients use a block-based protocol (like ATA or SCSI), while in network file systems the communication between the server and client is done using a protocol layered on a local area network, like NFS, CIFS, HTTP or FTP. The **Distributed File Systems** are a type of network file system [Bandulet, 2009, pp. 2-9].

2.2 Distributed File Systems (DFSs)

The underlying principles of distributed file systems (DFSs) have been defined as early as 1990 in a paper that has been widely quoted in research studies afterwards. Eliezer Levy and Abraham Silberschatz, two researchers at the Department of Computer Sciences of the University of Texas at Austin, USA, wrote at a time when the Internet was still in its very early days that distributed file systems were mainly a solution for sharing space and data, by using a "collection of loosely coupled machines interconnected by communication network" [Levy & Silberschatz 1990].

This definition still stands today, despite the significant additional complexity that the phrase "communication network" nowadays implies. However, the details related to the design of the DFSs are today much more complex, due to the availability of new hardware and software solutions.

The many advantages brought by DFSs (easy to scale up, increased reliability) come hand in hand with a series of problems that the designers of such systems need to be well aware of. A few examples of such problems are how to solve the synchronization between the computers, how to ensure the consistency of the data (same data on all replicated locations), or how to grant concurrent access to the same file for several users at the same time, as the classical solutions available in the client/server paradigm usually stop working in the distributed world.

The reason for which these problems occur is mostly a matter of where the data is stored. In the traditional client/server architecture, the data is stored on a server, while in the distributed file system, the data is usually stored simultaneously on several nodes for replication purposes, with a node being defined as a computer operating in DFS [Bzoch&Safarik 2011, pp. 764]

Newer papers have repeated and established a few key attributes that need to be considered when designing or choosing a DFS for usage.

2.2.1 Key attributes of the DFSs

In the ISO 25010 standard, which replaces the older ISO 9126, eight key categories of computer software quality attributes are identified: Functional suitability, Reliability, Usability, Performance efficiency, Maintainability, Portability, Compatibility and Security. These have additional sub-attributes. The software quality attributes "do not necessarily lend themselves to direct measurement", where direct measurement means that there is a single countable value that provides a direct indication of the attribute being examined [Pressman 2010, pp. 404]. However, the quality attributes constitute a good basis for indirect measures and make for an excellent checklist for assessing the quality of a system, according to the cited source.

Keeping the eight ISO-defined quality attributes in mind, the literature survey for this paper revealed some derived (quality) attributes that any designer of a DFS should keep in mind in order to ensure an increased level of quality of the system.

Even if these quality attributes cannot be directly measured, they constitute a solid base for more applied work on defining and improving the quality of a software system: "a software team should be able to develop a set of quality characteristics and associated questions that would probe the degree to which each factor has been satisfied" [Pressman 2010, pp. 404].

One of the first published literature sources in the field of DFSs affirms that DFS design should be discussed "mainly in the context of **transparency**, **fault tolerance** and **scalability**" [Levy & Silberschatz 1990]. To these three basic concepts, at least a few others are recurrently occurring in the field literature: the **single-point-of-failure** problem, **load-balancing** and **file replication**. Details about these, as reflected in the cited literature references, are provided below.

- **Location transparency**

Definition: It is the attribute that allows clients to access remote files as if they were local, and it is up to the DFS to locate the files and arrange for the transport of data [Levy & Silberschatz 1990, pp 3]. This attribute can be related to the quality attribute *operability* defined by [ISO/IEC 25010:2011, pp. 12] as *the degree to which a product or system has attributes that make it easy to operate and control* (a sub-attribute of Usability).

If in a local file system the user is always aware on which volume of which hard-drive any file is located, in the distributed system he does not know, only by looking at the name of a file, what the physical location of the file is (which computer in the network actually hosts it), but he can nevertheless retrieve the file from the file system and open it on his local machine. A related concept is location independence, which means that the name of a file does not need to be changed if its physical location in the system is changed.

- **Fault-tolerance**

Definition: The degree to which a system, product or component operates as intended despite the presence of hardware or software faults [ISO/IEC 25010:2011, pp. 13]. In the cited standard, this is a sub-attribute of Reliability.

In the case of distributed file systems, the faults can be of varied nature. Network failures, operating system crash, disk drive failures in one of the computers (a special vulnerability are the computers with central roles, like for example namespace servers), are examples that have a rather high probability of occurring in a real-life distributed system. That is why a modern, efficient file system that is distributed across a network should implement at least to some extent measures that would allow it to continue to function even in the event of such incidents.

- **Scalability**

Definition: Scalability is the capability of a system to adapt to increased service load [Levy & Silberschatz 1990, pp 3]. This attribute can be related to the quality attribute *adaptability* defined by [ISO/IEC 25010:2011, pp. 15] as *the degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments (*a sub-attribute of Portability).

Scalability is another key feature when discussing distributed file systems, as this type of systems are, by nature, dispersed in space, and can always expect an increasing number of users that request an increasing quantity of resources.

Scalability is an attribute of utmost importance when designing a distributed file system. It can and should be measured with precision and clear goals should be set when designing the non-functional requirements of the system.

However, scalability is a concept more complicated than simply saying that the system "grows as needs are growing". An important question is also how the growth is designed and implemented. When compared with a non-scalable system, a scalable system should, first, tolerate more gracefully the stress of increasing load (the performance should decrease at a slower pace), and second, it should resist more time to stress (resources should be depleted at a later point in time) [Levy & Silberschatz 1990, pp. 3].

- **The single-point-of- failure (SPOF) problem**

Definition: A single point of failure (SPOF) is a potential risk posed by a flaw in the design, implementation or configuration of a circuit or system in which one fault or malfunction causes an entire system to stop operating. [TechTarget, February 2012]. This attribute can be related to the quality attribute *Reliability* defined by [ISO/IEC 25010:2011, pp. 13] as *the degree to which a system, product or component performs specified functions under specified conditions for a specified period of time*.

Having a single computer that takes a central role in the distributed file system, whether it is controlling the other machines in some way, or it is a unique data storage container, introduces the existence of a single point of failure (SPOF)

Some of the DFSs that exist on the market are taking various approaches to mitigate this risk. However, considering this attribute is of high importance when evaluating a DFS.

In MSDFS, the SPOF problem is easily solved by the introduction of additional servers that store the metadata required to create and maintain the logical namespace (namespace servers). In this way, the namespace information is made redundant and if one of the namespace servers goes offline, the other(s) are still able to serve the needed information to clients.

In HDFS, however, the SPOF problem was more acute up to version Hadoop 2.0.0. The Namenode machine (namespace server) was the only single machine that stored metadata of file system and, thus, was the single point of failure for HDFS. After that version, a new feature called HDFS High Availability was introduced, which provides the option of running two redundant Namenodes with a hot standby [Apache Software Foundation, 2014, section Background]. In other words, if the Namenode crashes, a fast failover to a new Namenode is possible. In [Khan et al., 2012], another solution to the SPOF problem in HDFS is proposed. The solution consists in initially replicating the NameNode on two other machines and then working normally with the Datanodes (the computers that hold the actual data). The main Namenode will be called Coordinator, while the ones that the main Namenode data was replicated to will be called Participant Namenodes. The Coordinator and the Participants will regularly perform updates to their metadata using a 2 phase-commit protocol (2PC). If the Coordinator goes offline, any of the Participants can be promoted to become the new Coordinator [Khan et al., 2012, pp. 1]. Since HDFS is an open-source project, this or other solutions to the SPOF problem may be officially added to the project in the future.

- **Load-balancing**

Definition: Load balancing is dividing the amount of work that a computer has to do between two or more computers so that more work gets done in the same amount of time and, in general, all users get served faster [TechTarget, September 2005]. This attribute can be related to the quality attribute *resource utilization* defined by [ISO/IEC 25010:2011, pp. 11] as *the degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements* (a sub-attribute of Performance efficiency).

To load-balance the DFS means to move data between different nodes in order to achieve a comparable amount of load on the hard disks of the nodes, so as to avoid situations when some nodes are over-loaded (with their performance decreasing because of that), while others are only storing small quantities of data.

The load-balancing can be done either manually, by a system administrator, or automatically (for example, HDFS has such a command that is also designed to run only during periods of low-overhead in the network, to minimize the risk for creating a bottleneck).

- **File replication**

Definition: In the distributed file system, important or frequently used data can be stored on several nodes. This is called replication [Bzoch&Safarik 2011, pp. 764]. This attribute can be related to the quality attribute *availability* defined by [ISO/IEC 25010:2011, pp. 13] as *the degree to which a system, product or component is operational and accessible when required for use* (a sub-attribute of Reliability).

Replication means that the same file is placed on several nodes that are part of the DFS. When a node crashes, the data that was hosted on it is still available on the other nodes.

Replication is an attribute that is essential for improving the availability of files in distributed systems, and it is preferably done by hiding the details of the replication from the users.

Replication is usually done for increasing performance and/or for improving reliability. Depending on the primary goal, data is copied to the nodes that are physically closer to the client (performance) or to the nodes that are less probable to malfunction (reliability) [Bzoch&Safarik 2011, pp. 764].

The implementation of the replication feature has, however, a number of possible problems associated. The replication update activity, if it is done too often or involves too many computers in the network, can easily become a bottleneck in the file system. That is why the designers of distributed file systems nowadays put in place customizable replication schemes, that allow for system administrators to schedule the heavy-load movement of files at night or at times when the usage in the system is at a minimum. Modern algorithms like the ones implemented in Microsoft's DFS system, allow for the replication (copying across the network) of only those portions in the file that have been changed, avoiding thus the transfer of the entire file and greatly reducing the risk for congestion in the network.

- **Security**

Definition: Clients of the DFSs must prove their identity to an authentication entity in the system. The data, which flow between a client and a node, must be also resistant against attackers. This property is known as security [Bzoch&Safarik 2011, pp. 764]. In [ISO/IEC 25010:2011, pp. 13], *Security* is defined as *the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization*.

In DFSs, security is usually achieved by using network authentication protocols (example: Kerberos) and file permissions through Access Control Lists. Moreover, the content stored on nodes can be stored on a secured (encrypted) file system, such as TransCrypt or Microsoft EFS [Bzoch&Safarik 2011, pp. 765-767].

2.2.2 Main structural concepts in DFSs (definitions)

The definitions in this section are referenced by [Microsoft Technet, 2005] and [Apache Software Foundation, 2013].

- **Cluster** – A network of computers that make up a distributed file system. Typical components are the Namespace server and the Datanodes (see below).
- **Namespace server / Namenode** – A server component of the DFS that stores the metadata required to create the logical namespace, or the virtual space of the file system that is physically distributed on all member computers but appears as a continuous, local file space to the user.
- **Datanode** – All computers in the DFS that hold the actual data being shared by the users of the DFS.
- **Client** – The external user of the DFS that requires files and file services from the DFS.
- **File** – The logical unit of information used to stored data in the DFS.
- **Block** – The smallest sub-unit that makes up a File. A block is always in its entirety stored on the same physical machine (it cannot be further divided for distribution purposes).

2.2.3 Read and Write, core operations in DFSs

Probably the two most important operations in a DFS are reading and writing files. A high-level description of the data flow for each of these two operations could make up for a Behavioral Modelling of the DFS systems that would allow for the development of a measurement model [Wu et al. 2014, pp. 158]. A brief summary of how the two main operations are presented in the cited source is included below. Variations from this general model apply of course to specific DFSs, and the variations essential to the two file systems analyzed by this paper are presented in the respective sections, on MSDFS and HDFS.

Read – In order to read a file, a Client first contacts the Namespace server, which in turn translates the name of the file into a list of blocks and returns it to the Client. The Client now knows what blocks it needs to read and the physical location of each block. The Client connects directly to the Datanodes that hold the respective blocks and reads the data, either sequentially or in parallel.

Write (with Replication) – In order to write to a DFS, the Client first contacts the Namespace server to request the creation of the file. When the Client receives a confirmation that the file now exists in the namespace (as metadata only, no actual data was yet written), the Client contacts the Namespace server again and requests to add a new block. The Namespace server sends the Client the permission along with the address of a Datanode, which the Client then sends the Write request to. The Datanode starts writing to its own disk and simultaneously sends another Datanode a Write request, thus replicating the data coming from the Client. This process repeats for as many Datanodes that are required to accommodate the block (if the replication factor is 3, three Datanodes are involved in the chain). When the write is complete, each Datanode notifies the previous Datanode in the chain, and the Datanode that was contacted by the Client notifies the Client that the write succeeded. Whenever the Client needs to write a new block, it asks the Namespace server for the allocation of a new block and sends the Write request to the first Datanode, and the process repeats.

2.2.4 The problem of storing many small files in DFSs

With the rise of social networks and e-commerce giants, the necessity of storing many files of small dimensions (thumbnails, user short messages and so on) has been dramatically increasing. Traditional DFSs are designed to handle big files, but the new challenges of the modern digitalized world require change. The huge popularity of social networking services like Facebook, as well as of e-commerce websites like Amazon, has led to the necessity of storing and accessing huge numbers of small files, like photos uploaded by users or photos of commercial items. An example is Taobao, the number 1 e-commerce website in China, which stores about 28.6 billion photos, with the average size of each photo being only 17.45 KB. This shift in usage requires new research in finding ways for the DFSs to be better at handling many small files too [Fu et al. 2014, pp. 1].

The problem arises from the way DFSs are built. In a DFS, files are physically stored on data nodes, while a metadata server (or namenode, or namespace server) is keeping records of where each file in the file system is located. The server provides this information to clients wishing to read data from the DFS, who in turn contact the datanodes directly and request the data.

Because of this architecture, when the DFS is storing a huge number of small files, keeping the metadata in order becomes a burden for the metadata server, causing delays in the whole system. Not to mention that the waiting time for a client may be longer when determining the location of the file than the actual transfer of the file, a situation that is not desirable.

Designers of DFSs have taken a few approaches to optimize the support of small files in these systems. Among them, to reduce the data block size (the size of the smallest unit that a file can be broken into), and to combine many small files into bigger files or in groups of files [Fu et al. 2014, pp. 2].

HDFS is a file system from the Hadoop world, natively oriented towards the "big data" paradigm, meaning that the file system is optimized for offering good performance when storing files of big dimensions (up to terrabytes per file). Because of this design, HDFS allows storing millions of files, but billions is beyond the capabilities of current hardware [White, 2012, pp. 46]. The reason is that all files present in the file system are kept track of using the Namenode computer, the one machine that handles all metadata in the cluster, meaning that too many files can make this central computer inoperable. A solution to the problem is proposed by [Mackey et al., 2009]. The solution consists of archiving many small files into single files, thus increasing the possible address space of the Namenode without needing to increase the RAM memory. According to the authors, this technique can reduce the metadata footprint in RAM by a factor of 42 [Mackey et al., 2009, pp. 1-2].

On the other hand, MSDFS is a file system that comes from the Windows world, that of regular users who mostly keep personal files (rarely of big dimensions) on the hard disk. This dichotomy poses some natural questions. How much is HDFS negatively affected when the user stores many small files in the system, instead of the fewer big files that the system is optimized for? How much is MSDFS negatively affected, on the other hand, when the user burdens the system with files of big dimensions (a few gigabytes)? These, and others from this area, are questions that are addressed in the practical section of this paper.

2.3 Microsoft DFS (MSDFS)

Microsoft first launched the client version of Distributed File System as part of the operating system Windows NT 4.0 and added the server version as an add-on to the NT 4.0 system. Later, it became a standard component of all Windows Server versions, starting with Windows 2000 Server.

2.3.1 Key attributes of the MSDFS

Microsoft's Distributed File System (MSDFS) allows administrators to group the folders that need to be shared among users into logical locations, by using a common namespace, so that the folders appear to be in the same place even if they actually are located on different servers. Such common namespace becomes a virtual view of the shared folders in the respective organization [Microsoft TechNet, March 2003].

MSDFS is built on top of the NTFS file system, as an additional namespace layer that organizes the folders already shared in the NTFS and inherits file permissions already existent in the NTFS.

An administrator can decide which shared folders appear in the namespace, the hierarchy under which they are grouped (which does not need to be the one from the physical servers where the folders are stored) and what names the shared folders should have in the namespace (which can be different from the NTFS share name). The users can navigate through the logical namespace without knowing the real names under which the folders as actually stored on the servers, or the addresses of those servers.

The essential DFS technologies are DFS Namespaces (defines a logical namespace across several physical servers) and DFS Replication (an engine that keeps folders synchronized between servers) [Microsoft Technet, July 2009].

The above generalities can be summarized into a number of essential attributes.

- **Location transparency**

The location transparency of the data allows the administrator to move data physically from one server to another (for maintenance purposes, for example), without needing to inform the users. The users do not need to change any path or other attribute of their normal way of accessing the files in the DFS namespace. This feature is referred to by Microsoft as simplified data migration.

- **Increased availability of file server data**

This is an attribute very similar to the fault-tolerance attribute referred to in section *2.1.1 Key attributes of the DFSs*, since the failure of one server does not mean a service interruption as long as the data was replicated on at least one other server.

- **Load-sharing**

Providing a unique logical name for a folder that is heavily used in a network and physically copying that folder to several servers would mean that requests for that folder would be more evenly distributed to more servers, instead of over-loading only one server.

- **Security integration**

The security of the MSDFS is already enforced by the NTFS native shared folder permissions, as MSDFS is an additional layer on top of NTFS.

2.3.2 Core technologies in MSDFS

According to Microsoft, the two main technologies in MSDFS are DFS Namespaces and DFS Replication [Microsoft TechNet, July 2009].

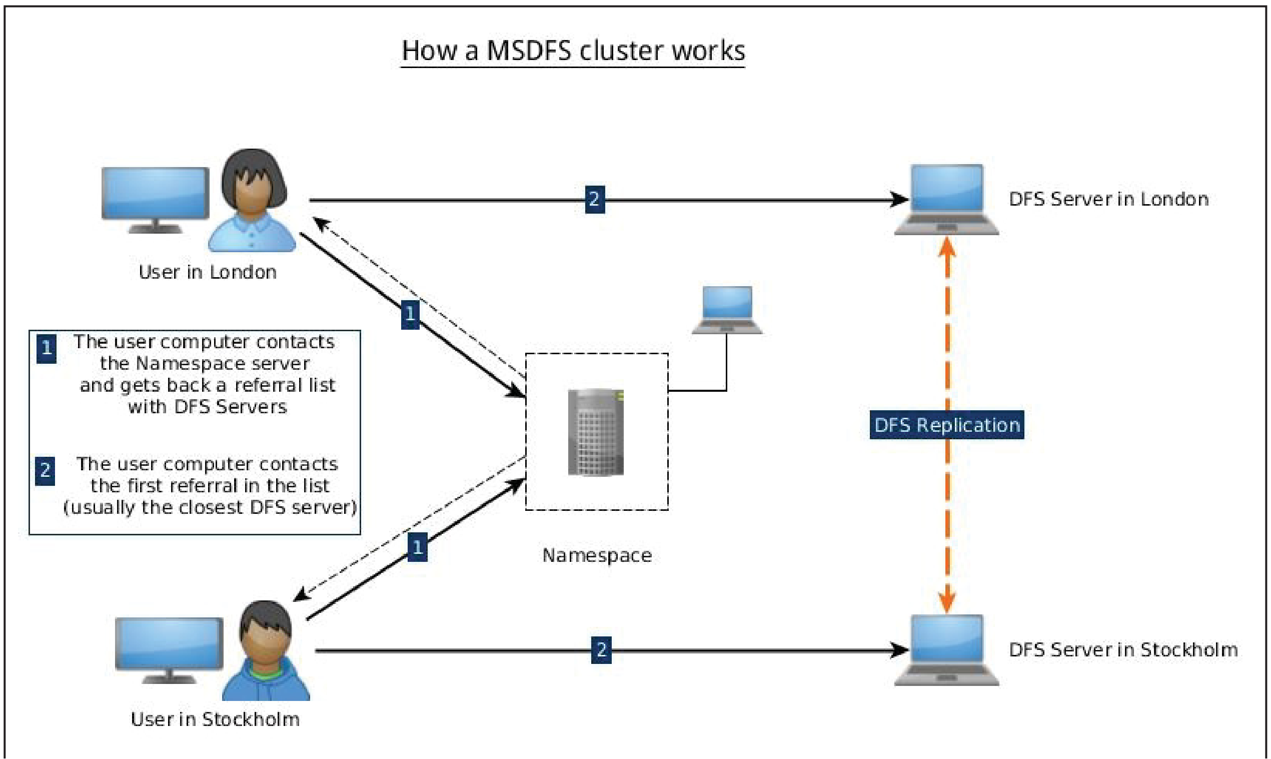
DFS Namespaces is the technology that enables the organization of shared folders into a new, logical, hierarchy, available across the network independently on the physical location of the folders.

This technology has a locality feature implemented, which automatically connects users to shared folders that are physically placed on a server that is in the same Active Directory Domain Services site, when possible. Thus, the rerouting of the user over WAN connections is avoided, as it would imply increased latency.

DFS Replication is a replication engine that improves folder availability by keeping folders synchronized across servers. If a server goes down, the data is still available from the other server(s) where it is replicated. The engine supports replication scheduling (allows the administrator to schedule the replication during periods when the network is less used) and bandwidth throttling (limit the bandwidth used by the replication service to ensure good functionality of other communication in the network).

Figure 1 shows how DFS Namespace and DFS Replication work together in a MSDFS cluster.

Figure 1. How a MSDFS cluster works [Microsoft Technet, August 2005]


How a MSDFS cluster works
User in London
DFS Server in London
1 The user computer contacts the Namespace server and gets back a referral list with DFS Servers
2 The user computer contacts the first referral in the list (usually the closest DFS server)
1
2
DFS Replication
Namespace
1
2
User in Stockholm
DFS Server in Stockholm

To keep the load in the network low, the Namespace server does not store the identity of the clients that are using the cluster. Instead, the clients are keeping in cache the list with DFS servers [Marinkovic et al. 2012]. Thus, when a client in the future needs to contact the DFS server again, it refers to the cache and contacts the DFS server directly, without involving the Namespace server. This operation can be repeated over and over again until the cache needs to be refreshed, usually for 15 minutes.

However, the flexibility given by the independence of the clients from the Namespace server can also be a risk. Because the client can use its cache for about 15 minutes without asking the Namespace server for updates, this means also that updates in the structure of the namespace will not be propagated at once to clients. This is a risk for data corruption and, for the MSDFS to become a viable cluster solution, configuration consistency must be maintained at all times [Marinkovic et al. 2012].

The DFS Replication uses a compression protocol called **Remote Differential Compression** (RDC), which is able to detect only the changes occurred in the files (insertions, removals, re-arrangement of data in the files) and only replicate the changed parts in the files, thus avoiding to replicate whole files and greatly saving on the amount of data moved through the network.

The RDC is especially of use in a scenario that implies small changes made to big files. A test run by Microsoft on a mix of 780 Office files (Word documents, Powerpoint presentations and Excel spreadsheets), showed that the savings in bytes transferred with help of RDC was of 50 percent in average, and the percentage increases with the size of the files that are being changed. Another example is a 2 MB Powerpoint presentation that gets slightly changed (only 60 KB of data are changed). Transferring only the 60 KB of new data means a 97 percent saving in bytes transferred.

2.3.3 MSDFS Use Case Scenarios

MSDFS can be useful especially in a number of key scenarios; among them, “data collection” and “data distribution” [Microsoft TechNet, August 2005].

The **data collection** scenario refers to the fact that DFS Replication allows for the data to be replicated to a backup server, thus making the backup operations easier and less error-prone. An important feature here is RDC, which minimizes the data traffic by transferring to the backup server only the changed parts in the files and not the whole files.

A second type of situation when MSDFS is of use is **data distribution**, which refers to the fact that data can be easily shared between the users of an organization. For this purpose, using DFS Replication would be enough, but adding the DFS Namespace as well provides for a way to store important shared folders on different servers, thus improving the availability of heavily used data.

An example of this second scenario can be a magazine that collects article drafts from writers around the world. This could be also called as an "implementing business processes" scenario [Lee & Vellore, 2006]. For example, a writer located in Asia may save their draft articles in the DFS namespace at the address \\magazine.com\drafts_in_asia, reachable from Windows Explorer. This address, in fact, is a shortcut to a regional DFS server physically located at \\asia.magazine.com, and the article is actually saved in the folder \\asia.magazine.com\drafts_in_asia. At the same time, a reviewer located at the headquarters in North America, would connect to the same DFS address \\magazine.com\drafts_in_asia, which would translate this time to the physical address \\headquarters.magazine.com\drafts_in_asia, he would open the draft article, review it and move it into a new DFS location, \\magazine.com\artwork_queue. A member of the artwork team, located in Europe, would connect to this DFS location and open the reviewed article, while the actual physical server would be \\europe.magazine.com\artwork_queue, that was updated through DFS Replication to get the desired reviewed article. Like this, neither the writer nor the reviewer or the artwork team member do need to remember the names of all physical servers that they connect to.

2.4 Hadoop DFS (HDFS)

Hadoop Distributed File System (HDFS) is a core element of the Apache Hadoop platform, launched in 2005 by the Apache Foundation as an open-source framework written in Java that focuses on distributed storage and processing of big amounts of data.

HDFS is the architecture element that handles the distributed storage of data. The key concept is that HDFS splits files into blocks (usual size is 64 or 128 MB) and distributes the blocks to all computers that make up the clusters.

The computers that hold data are called Datanodes, while the central computer that holds the metadata (a catalogue that maps blocks to the Datanodes where they are stored) is called the Namenode.

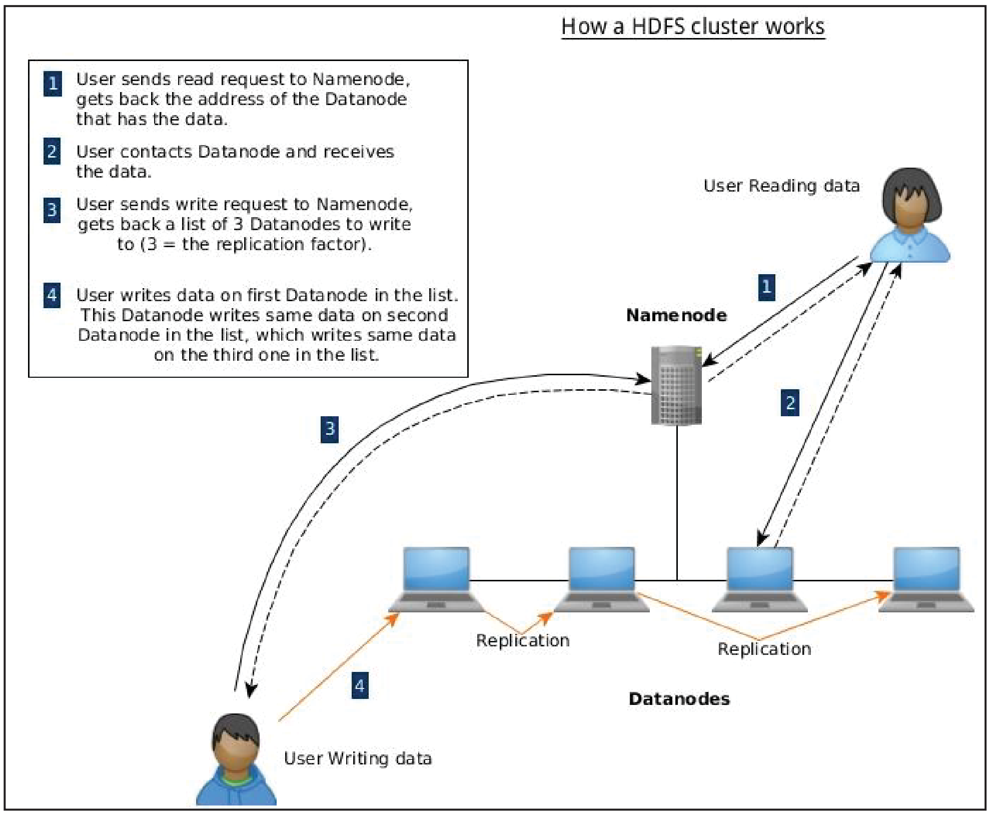
HDFS is built according to the master/slave paradigm. Any request made by an external client to the cluster goes first to the Namenode, which replies with a referral to the Datanode that is supposed to do the requested operation (create, read, write, delete). Then, the client contacts the respective Datanode directly. See how this works in the Figure "How a HDFS cluster works" below.

An application reads data by using a standard Java file input stream. This means that the application does not know that the file is located beyond the native file system. The stream is, however, manipulated to actually retrieve data from HDFS instead [Shafer et al., 2010, pp. 123]. First, the client contacts the Namenode to request permission to read. If granted, the client receives a list of the HDFS block IDs that make up the requested file, with each block mapped to a list of Datanodes that store the respective block. Then, the client opens a connection, hopefully to the closest node, and receives the block.

An application writes data to HDFS by creating a file and writing to it. When a client opens a file for writing, it is granted a lease and no other client can write to that file. However, other clients can read from the file while the lease is held by the writer [Shvachko et al., 2010, pp. 4-5].

To write data to HDFS, an application treats the output file as a standard output stream. However, the data is internally fragmented into HDFS-sized blocks (for example, 64 MB) and then smaller packets (for example, 64 KB) by the client thread. Packets are sent to a FIFO queue that can hold up to 5 MB, thus decoupling the application from the possible latencies of the network transfer [Shafer et al., 2010, pp. 123].

Figure 2. How a HDFS cluster works



How a HDFS cluster works
1 User sends read request to Namenode, gets back the address of the Datanode that has the data.
2 User contacts Datanode and receives the data.
3 User sends write request to Namenode, gets back a list of 3 Datanodes to write to (3 = the replication factor).
4 User writes data on first Datanode in the list. This Datanode writes same data on second Datanode in the list, which writes same data on the third one in the list.
User Reading data
Namenode
1
2
3
4
Replication
Replication
Datanodes
User Writing data

2.4.1 Key attributes of the HDFS

Some key assumptions and goals of HDFS are described in [Borthakur, 2007]. A brief summary of the most relevant features follow.

- **Fault-tolerance**

HDFS is built to be highly fault-tolerant. Hardware failure is not only a possibility, but a certainty, when one runs clusters with hundreds or thousands of Datanodes, each storing some part of the data in the file system. One may count on the fact that some component of HDFS is always non-functional, which makes fault-detection and quick recovery key design goals.

- **Optimization for "big data"**

HDFS is designed more for batch processing rather than interactive use by users [Borthakur, 2007, pp. 3]. The file system is optimized for storing and processing large data sets (according to the "big data" paradigm), with a typical file in HDFS being hundreds of megabytes, to gigabytes, to terabytes in size.

- **Write-once-read-many**

The access model write-once-read-many means that files are created and written to, but once they are stored, they cannot be opened and edited like in a classical file system. Rather, when they need to be changed, they are deleted and recreated with the new content.

- **Location transparency**

The Namenode offers location transparency to the HDFS, as the user is not aware of the actual physical location of the files it accesses. Much like MSDFS, the HDFS builds a virtual namespace (of the form hdfs://path/to/file) that makes the actual location irrelevant to the end user.

- **Portability**

HDFS is entirely written in Java, which makes it well-known for its portability. All communication protocols are layered on top of the ubiquitous TCP/IP protocol [Borthakur, 2007, pp. 9].

- **Scalability**

This is also a key attribute of the system. A HDFS cluster scales computation capacity, storage and bandwidth by simply adding new commodity servers [Shvachko et al., 2010, pp. 1].

2.4.2 Core technologies in HDFS

Load-sharing is achieved through a technology called **cluster rebalancing**. According to it, data is moved from one Datanode to another to make sure that the quantity of data is distributed among the Datanodes as evenly as possible. The balancer operates with a threshold value between 0 and 1. The cluster is considered to be balanced if, for each Datanode, the utilization of the node (ratio of used space to total capacity of the node) differs from the utilization of the cluster (ratio of used space to total capacity of the cluster) by no more than the threshold [Shvachko et al., 2010, pp. 7]. The balancer is a tool that can be run by an administrator.

Because transferring huge files through the network is always error-prone, HDFS implements a **data-integrity** technology based on checksum checking on the contents of the files. The checksums of each file are stored in special, hidden areas of the namespace. When a client accesses a file in the system, it verifies the checksum to find out if the transfer was made successfully. If an error occurred, the respective block can be retrieved from another Datanode, that has a copy of the block.

Data replication is another key feature of HDFS. Files are divided into blocks of equal size, which are replicated across the cluster. Most commonly, the replication factor is set at 2 or 3, meaning that each block of each file is present on 2 or 3, respectively, different Datanodes, for fault-tolerance. The Namenode periodically receives so-called heartbeats from each Datanode, to know that they are still alive. When a Datanode goes offline, the Namenode orders the re-replication of the blocks that were stored on that Datanode to other Datanodes in the cluster, to keep the replication schema intact.

Another key concept of the system is its ability to **move computation rather than move data**. In other words, the code necessary to compute the data is distributed through jar files to the Datanodes that store the data which needs to be computed. In this way, network congestion is minimized and the overall throughput of the system is increased [Borthakur, 2007, pp. 3-4].

The namespace of the HDFS is kept entirely in RAM. The metadata comprised of the list of nodes and the corresponding blocks stored on each node is stored in a so-called *image* (a file named fsimage). Live updates to the metadata (blocks that are added or removed, for example) are not written to the image file, but instead are appended to a log file (the *journal*), to avoid random file writes that take time. The image is reloaded next time the system starts, and all logged changes are applied to bring the system up to date. Because this is a process that takes time, a **secondary namenode** is used. The job of the secondary namenode is not to do the same kind of work as the namenode (storing and serving metadata), but to periodically read the changes in the log and apply them to the image file. In this way, the system will start faster the next time.

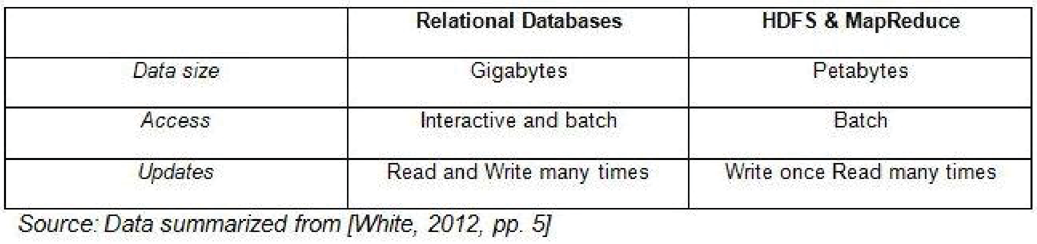
2.4.3 HDFS Use Case Scenarios

HDFS is intimately connected to the Hadoop palette of technologies, as it is the default storage platform for the Hadoop services, which are mainly used to analyze very big amounts of data in a fast and efficient manner.

One of the most recurrent use scenario of Hadoop and HDFS is **data-intensive fast processing**. The additional technology used in this case is MapReduce, a programming paradigm that uses a parallel, distributed algorithm to process data locally on the Datanodes and combine the partial results into a final result back at the client application. Real-life examples where this is used are data flows coming from stock exchanges (for example, the New York Stock Exchange generates about one terabyte of data every day), social networks, data-intensive computing models like weather data simulators or scientific projects that analyze past data to make predictions. More recently, even the Formula 1 competition has aligned itself to the "big data" trend, as the racing cars rushing on track with up to 300 km/h are packed up with live sensors whose data are analyzed on-the-spot and delivered back to the team members in the form of reports that help them take quick decisions.

Why not use a regular database to store data and then retrieve it in an application? When should HDFS be preferred? The advantage of HDFS comes from a trend in disk drives – seek time is improving slower than transfer rates [White, 2012, pp. 5]. That is why HDFS, through the MapReduce paradigm, takes the approach of **fast streaming data** to the application. In some way, MapReduce can be considered a complement to the traditional relational database approach. HDFS and MapReduce are preferred when the problem is to analyze whole datasets, in a batch fashion, while the relational database model is better when the dataset has already been indexed and the application is mainly interested in making point queries or updates. In other words, HDFS is best on problems where data is written once and read many times, while relational databases perform better on problems that use datasets that are continuously updated (see Table "HDFS vs. Relational Databases" below).

Table1. HDFS vs. Relational Databases



	Relational Databases	**HDFS & MapReduce**
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and Write many times	Write once Read many times

Source: Data summarized from [White, 2012, pp. 5]

Another scenario when HDFS is widely used is **data storage and analysis**. Due to HDFS's highly fault-tolerant profile, data is stored in clusters made of commodity hardware. Failure of some machines is no longer seen as a risk that may or may not occur, but more as a certainty. Replication solves this problem, as each block is distributed on several different machines, usually on at least two different racks, to guarantee availability of data in case of hardware failures. Real-life situations where HDFS is used in this way are in health-care (storing and processing medical records), telecom (operators need to store and analyze billions of call records and service-related parameters), logistics (transport companies are storing sensor data from vehicles to optimize operations), retail (billions of products with their related information) [Kerzner & Maniyam, 2014].

2.5 How to Evaluate the Performance of Distributed File Systems

Distributed file systems have been around for many years, with the first detailed papers on the topic being published in early 1990s. However, when researchers wanted to analyze the performance of the different DFSs on the market, they could rarely use benchmarks or analysis tools that fitted in the case of local file systems.

Even if widely-known evaluation benchmarks do exist for a large variety of computing paradigms, similar systems for DFSs are rarely seen (for example, no benchmark is available for reviewing the performance of the web services running on DFSs). Thus, the main path that researchers take is making own experiments by running the distributed file systems and analyze their results, or compare the results with those from other existing distributed file systems [Wu et al. 2014, pp 156].

For distributed file systems (like HDFS, analyzed further in the practical part of this paper, GFS, the file system developed by Google for its own use, MooseFS or PVFS), there are no well-recognized benchmarks for evaluation or analysis.

Instead, researchers have used in-house, micro-benchmarks to measure specific aspects of the file systems and show the differences between file systems [Wu et al. 2014, pp. 165].

Among the reasons for which benchmarking DFSs is difficult, experts in the industry reunited at a conference on this topic (Annual File and Storage Systems Benchmarking Workshop, held in May 2008 at the University of California Santa Cruz) have stated a few key aspects: storage variety (not only regular HDDs, but also RAID, NAS – network-attached storage -, SANs – storage area networks –, flash, object storage or virtualization may be used), file system variety (mainly differences in cache effects and network latencies were mentioned), operating system variety, workload variety, asynchronous activity (many processes and threads running on the operating system, difficult to keep track of) and operating system caches, including disk caches that can affect the storage behavior [Traeger et al. Oct. 2008].

However, benchmarks oriented to file systems in general do exist and can be used at least to some extent to review a number of key aspects of the distributed file systems as well.

A classification of benchmarks for file systems, as well as discussion about the extent that they could be used for evaluating distributed file systems, follows in the next section, along with the motivation of choice of some of the models presented that will be used in the practical section of this paper.

2.5.1 Benchmarks

In order to evaluate software components, benchmarking is one of the most widely used methods, as it provides objective and verified ways of measuring the performance of the analyzed component.

The usage of the benchmark allows scientists, software developers or potential buyers to understand how different systems work under real-life conditions or in specific scenarios, as well as the differences between those systems, and to make decisions accordingly. Choosing the most relevant benchmark is a difficult thing and may require extensive research.

However, benchmarking file systems is especially difficult due to complexity issues introduced by input/output devices, the presence of caches or other components of the operating systems according to a study that surveyed not less than 415 file systems and storage benchmarks described in 106 papers [Traeger et al. May 2008].

One way to solve this problem is that the user interested in doing the measurements acts in his own settings, trying to emulate the work scenario that is closest to the real-life need that the system is intended to serve. In other words, the user is building his own benchmark. This scenario can work well if 1) the problem being investigated is of small dimensions (i.e. a specific characteristic of the system) and 2) the ad-hoc benchmark is carefully described, so that persons interested in continuing the work can recreate the environment reasonably well.

One can distinguish between three main types of file system benchmarks: macro-benchmarks, trace-based and micro-benchmarks [Traeger et al. May 2008, pp. 5].

The **macro-benchmarks** would typically cover scenarios when the user is interested in measuring multiple attributes of the system simultaneously and getting an overview of the system's performance.

On the other hand, the **trace-based benchmarking** would mimic a real-life set of tasks that the system typically does, in order to show the performance achieved during this repetitive set of activities.

The **micro-benchmark** is of specific interest in the context of this work, as it is used to analyze *a small set of attributes of the system in order to highlight some individual aspect* of it, and not the system as a whole, which would be an ambition beyond the scope of this paper.

Micro-benchmarks can be further categorized in ***publicly-known micro-benchmarks***, ***ad-hoc benchmarks*** and ***system utilities*** [Traeger et al. May 2008, pp. 32]. The authors of the cited paper are analyzing two ***publicly-known micro-benchmarks*** ("Bonnie and Bonnie++" and "Sprite LFS") and are generally describing the ad-hoc benchmarks and the system utilities alternatives.

The "Sprite LFS" benchmark can be implemented in two variants, the "large file"- and the "small file"-benchmark. The "large file" variant follows a number of steps to create a 100 MB file, read it, write another 100 MB to the file, read 100 MB randomly from the new file, and then read the whole file sequentially. The "small file" variant creates a number of 10,000 files each with the size of 1 KB, reads the files and then deletes them.

The ***ad-hoc benchmarks***, defined as benchmarks that are produced for in-house use, have some drawbacks and some advantages. One drawback of the ad-hoc micro-benchmarks is the difficulty of reproducing them because their details are usually not made available to other researchers, and even when they are made available, a new implementation has a high chance of doing some things differently and thus influence the final result of the measurement. Another drawback is the difficulty of testing such benchmarks, or trusting that they were properly tested, for the same main reason - they do not circulate as-such between a large number of researchers.

However, the ad-hoc micro-benchmarks have their strengths, as well. Because of their high flexibility in addressing a specific problem, they are widely used in practice. For example, out of 106 research papers surveyed in [Traeger et. al. May 2008], 62 are using ad-hoc micro-benchmarks for at least one of their experiments, with a total of 191 ad-hoc micro-benchmarks used.

The ***system utilities*** are standard OS tools like "grep", "cp", "diff" or "gzip", which have the advantage of applying the same operations on the chosen set of data, but present the drawback of not providing some standard input file that is part of the benchmarking activity. Thus, if the input file differs from user to user, the results of the measurements will not be comparable, even if the same tool was used.

A reasonable way to use micro-benchmarks is to isolate a certain attribute of a system to show its behavior in a specific scenario that can be found in the real world. Moreover, ad-hoc micro-benchmarks are a good tool to explore the behavior of a system in the initial phase of benchmarking, before selecting a macro-benchmark or a trace-based benchmark.

In this paper, the method of **micro-benchmarking** is used to show specific aspects of the two analyzed file systems and how they differ, in order to draw conclusions that pertain only to those specific aspects and not the file systems as a whole. That would be the subject of a wider-scoped analysis, which would require the use of one or several benchmarks from the macro or trace-based categories.

2.6 The MSDFS vs. HDFS comparison – review and conclusions following the literature study

2.6.1 Literature study review

A number of 30 references were included in the research section. They are covering three different categories of topics. First, there are the papers that cover distributed file systems in general. Second, there are the papers that cover the Hadoop Distributed File System. Third, there are the papers that cover the Microsoft Distributed File System (see MSDFS section below).

The references consist mainly of scientific papers. The best covered section, from this point of view, is the one on distributed file systems.

On the other hand, the sections that cover the specifics of HDFS and MSDFS, contain, besides scientific papers, other types of sources as well, like official technical articles (published or approved by the official owner or open-source foundation that is responsible for MSDFS and HDFS, respectively), web articles that provide definitions of DFS quality attributes, one book (on HDFS), magazine article (for example, one published in a Dell corporate magazine on the topic of MSDFS) and even a U.S. Patent (that covers, in the background section, some specifics of the MSDFS).

The variety of references can partly be explained through the scarcity of official detailed documentation due to proprietary rights over the code (the case of Microsoft DFS) and partly through the specificity of the open-source world, that bases itself on a whole ecosystem of resources published openly, often in the form of technical articles (the case of HDFS). The cited technical articles provide details of functionality or architecture that were not found in scientific papers but were relevant to this thesis work.

The sources used to retrieve the articles were BTH Summon (http://bth.summon.serialssolutions.com), Google Scholar (http://scholar.google.com), IEEE Xplore (http://ieeexplore.ieee.org.miman.bib.bth.se), EngineeringVillage (https://www.engineeringvillage.com), Microsoft Technet (https://social.technet.microsoft.com).

The resources used in the literature survey for this paper are covering the theoretical aspects of file systems, distributed file systems, specifics of MSDFS, specifics of HDFS, and aspects of benchmarking for file systems, being thus a complete review of the themes that are present in the paper.

A complete list of the references used in this paper can be found in the **References** chapter.

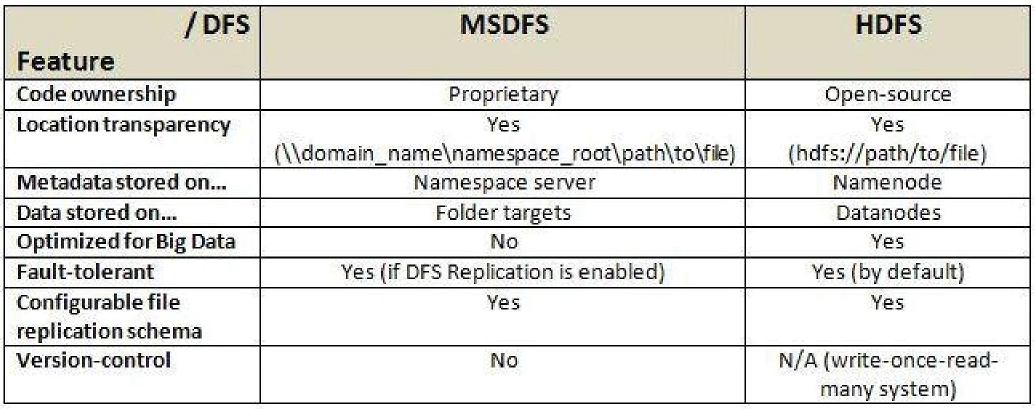
2.6.2 Conclusions after the literature study

Both DFSs provide the general functionality of distributed file systems, out of which location transparency, file replication and scalability are the most important. The differences occur on several areas, for example on how the data is spread to the different nodes. If HDFS divides each file into blocks, which then are spread on the nodes in the cluster, the MSDFS replicates whole files. This second technique can be seen as less flexible, but MSDFS has an innovative technology that offsets this problem: a remote differential compression (RDC) algorithm that allows for the replication engine to only transfer the changed parts in a file, and not the whole file.

Another area where the two systems are different is in the way security is treated. If HDFS requires special configurations (by Kerberos), MSDFS inherits the NTFS native shared file and folder permissions which work “out-of-the box”.

When it comes to the usage scenarios, the two systems differ in that MSDFS is usually in place in sites where Windows is a tradition, as it comes as a solution on top of Windows to provide DFS functionality. HDFS, on the other hand, was designed to handle “big data”, files of big dimensions which rarely change, and its main purpose is to provide the data to storage and analysis services that run on top of it.

Table2. MSDFS vs. HDFS – Quick Overview



/ DFS Feature	MSDFS	HDFS
Code ownership	Proprietary	Open-source
Location transparency	Yes (\\domain_name\namespace_root\path\to\file)	Yes (hdfs://path/to/file)
Metadata stored on...	Namespace server	Namenode
Data stored on...	Folder targets	Datanodes
Optimized for Big Data	No	Yes
Fault-tolerant	Yes (if DFS Replication is enabled)	Yes (by default)
Configurable file replication schema	Yes	Yes
Version-control	No	N/A (write-once-read-many system)

2.7 Novelty of this paper

A direct comparison of this type between HDFS and MSDFS has not been done yet, according to the published literature. This can be explained partly through the novelty of the HDFS system, which has only started to come into mainstream in the last few years. However, it is mainly used by very big companies (Yahoo, Facebook, Amazon) and it is promising when it comes to the necessity of solving the problem of increasing quantities of data to be stored.

The comparative analysis HDFS vs. MSDFS is relevant as the two distributed file systems come from different "worlds" (proprietary - Microsoft DFS - vs. open-source - Hadoop DFS). At the time when the research activity for this paper was ended (March 2015), there was virtually no scientific work out there to thoroughly compare the two, at least from a theoretical point of view, if not from a practical point of view as well (performance comparison). According to the literature study, only articles explaining the characteristics of each of the file systems separately were previously published, as well as articles putting Hadoop DFS in the context of other open-source alternatives.

3 METHOD

This chapter describes the design experiment to be followed in order to answer the research questions (defined in section *1.1 Research questions and focus*). Moreover, the design of the literature survey is presented.

3.1 Design of literature survey

3.1.1 Criteria for identifying relevant literature

Scientific papers on the general field of file systems were searched to determine which criteria are most relevant to analyze in order to determine the performance of a file system. Some of the articles found here were older (one is from 1990), but this is not a problem since the principles of storing and accessing files on file systems have not changed that much over time.

Another field that was researched is the general area of distributed file systems. Questions like “what are the main attributes that differ when comparing them to regular, local file systems” were considered important. For this section, only newer articles (2-3 years old) were considered because the distributed file systems, being a newer field, have undergone significant changes since their introduction.

Other scientific articles were targeted to explain the working principles of HDFS and MSDFS, respectively, while others were supposed to put each of the two file systems in context or compare them with other file systems.

3.1.2 Search strategy

Scientific papers were searched on Blekinge Institute of Technology’s specialized search engine (that searches on several databases that focus on academic content), on Google Scholar and other academic research sources (for example Arxiv.org, DBLP, IEEE). For each relevant article the keywords searched for were written down, as well as details about the location where the papers were found - universities, academic think-tanks, organizations, associations etc.

Example on search strings that were used: “What to consider when evaluating file systems”, “DFS architecture”, “distributed file systems working principles”.

The articles found were filtered and those that use a higher-level approach when explaining the working principles of the file systems were preferred. The articles that go deep into the architectural details of the file systems were excluded, as this kind of technical approach is not in the scope of this paper.

3.2 Design of experiment

In this section, the goal of the experiment is presented, as well as the setup and the methodology.

3.2.1 Goal

The goal of the empirical study is to produce a comparative analysis of the two distributed file systems (MSDFS and HDFS) from a number of different perspectives, with the central perspective being the speed of the writing and reading operations. The file replication time is included and measured as an integral part of the writing operation, since the replication is a key element of any distributed file system and often an essential reason when choosing such a file system in the first place.

3.2.2 Setup

The working environment is made up of two clusters, each running one of the two distributed file systems, MSDFS and HDFS, respectively.

Both clusters are made up of four computers, one acting as namenode (that stores metadata about the location of the real data), and the rest of three computers acting as datanodes (that store the real data). In practice, both clusters run on the same hardware, meaning that the same four computers are used to run both clusters. However, the clusters do not run simultaneously, but one at a time. The fact that both clusters run on exactly the same hardware guarantees that the measurements will expose the differences in software architecture only, excluding any deviations that may be due to hardware differences.

Each of the four computers that compose the cluster is running the operating systems Windows Server 2012 (that hosts MSDFS) and Ubuntu Linux 14.04 (that hosts HDFS). Both operating systems are running the 64-bit architecture. One computer is running the two operating systems as native dual-boot installations, while the other three computers run the two operating systems as Oracle VirtualBox virtual machines - two guest operating systems installed onto the same host operating system. Each switch from one operating system to the other is preceded by a reboot of the host computer.

Configuration of the four computers

Namenode

Asus A55A , Intel core i7, 8 GB RAM
Host OS: Ubuntu 14.04 64 bit
Guest OS1: Windows 2012 Server (runs MSDFS)
Guest OS2: Ubuntu 14.04 64 bit (runs HDFS)

Datanode1

Dell Latitude e4310, Intel core i5, 4 GB RAM
Host OS: Windows 7
Guest OS1: Windows 2012 Server (runs MSDFS)
Guest OS2: Ubuntu 14.04 64 bit (runs HDFS)

Datanode2

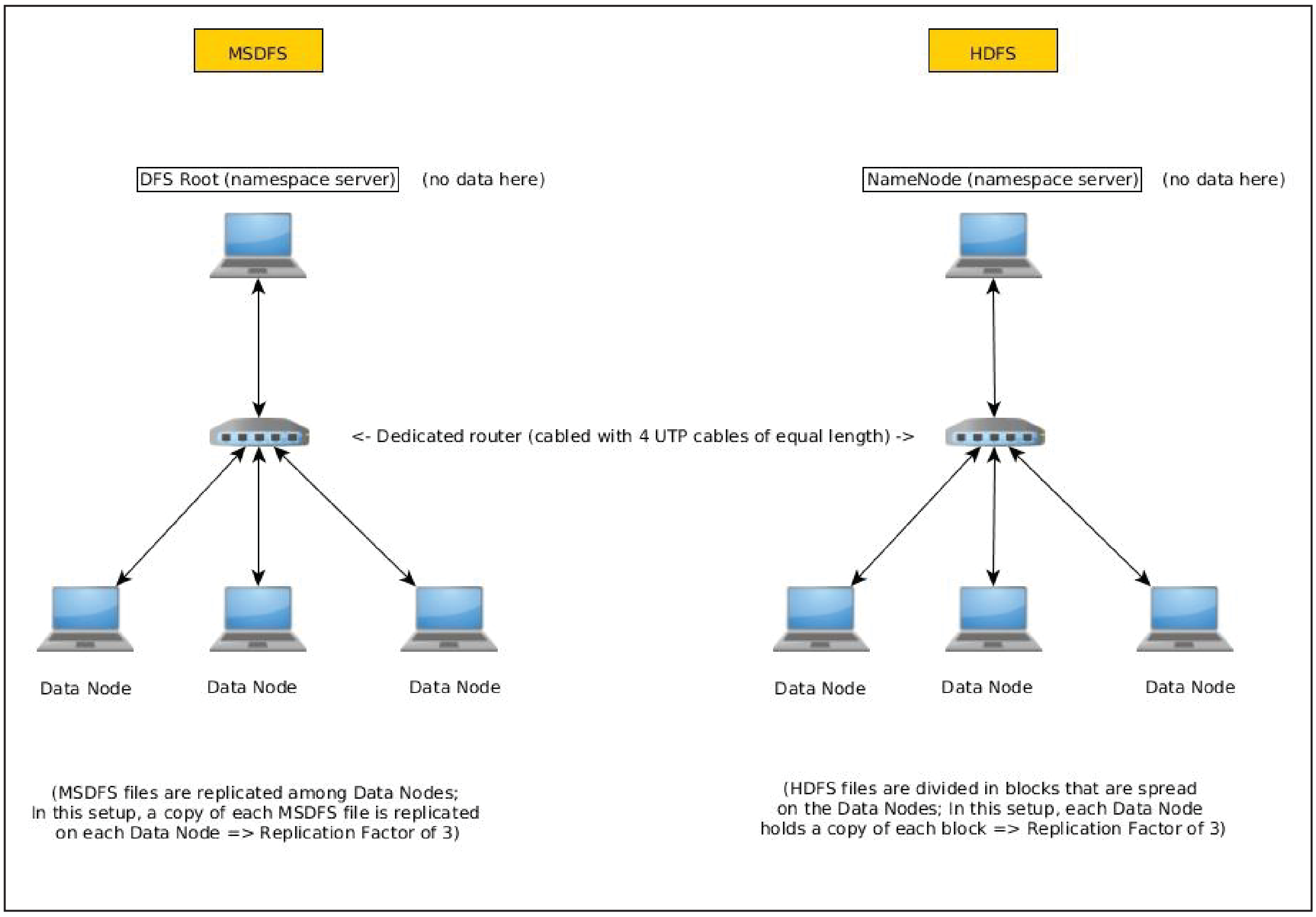
HP Probook 4520s, Intel core i3, 4 GB RAM
Host OS: Windows 7
Guest OS1: Windows 2012 Server (runs MSDFS)
Guest OS2: Ubuntu 14.04 64 bit (runs HDFS)

Datanode3

Fujitsu Siemens Esprimo Mobile, Intel Celeron, 4 GB RAM
Dual-installation:
Native OS1: Windows 2012 Server (runs MSDFS)
Native OS2: Ubuntu 14.04 64 bit (runs HDFS)

The computers making up the cluster are connected to each other through a dedicated router (that does not do any other work), and the connection between the computers and the router is made through UTP cables of equal length (2 meters) in order to exclude the variations brought into the measurement activity by the use of a wireless network. Static IP addresses are provided to all computers to avoid variations given by the DHCP protocol activity and no other software is running when measurements are made, to keep the results accurate (a diagram of the setup is visible in Figure 3).

Figure 3. Experiment setup


MSDFS
HDFS
DFS Root (namespace server)
(no data here)
NameNode (namespace server)
(no data here)
<- Dedicated router (cabled with 4 UTP cables of equal length) ->
Data Node
Data Node
Data Node
Data Node
Data Node
Data Node
(MSDFS files are replicated among Data Nodes; In this setup, a copy of each MSDFS file is replicated on each Data Node => Replication Factor of 3)
(HDFS files are divided in blocks that are spread on the Data Nodes; In this setup, each Data Node holds a copy of each block => Replication Factor of 3)

3.2.3 Methodology

Reading and writing files are two most important DFS operations [Wu et al, pp. 158]. As a consequence, any software architect that chooses a distributed system for his system needs to first consider the efficiency of these two basic operations. For this reasons, these two operations make up the central point of the empirical study. Other operations like deleting files could have been of interest, but were kept out of scope for this paper due to time limitations and could make the object of future work.

3.2.3.1 Generalities

The test files are produced by using the IOzone benchmark (http://www.iozone.org). IOZone is an open-source solution for file system benchmarking that is widely used in the industry. It has the advantage of being available for nearly all main platforms, like Linux, MacOS, Solaris, BSD and Windows (through the Cygwin project). In 2007 it has been rewarded the Infoworld BOSSIE award of the best file I/O tool [InfoWorld, September

2007]. The results produced by IOZone and used in the experiment chapter of this paper are binary files of controlled sizes.

The files are read/written from/to the DFS by using the hard disk of the namenode computer as source, respective destination for the test data. Since this computer hosts no data belonging to the DFS, it acts as an external client whenever writing data from its hard disk to the DFS or reading data from the DFS and storing it onto its hard disk.

The timing of the read/write operations is done by using system tools on the namenode computer (for example, the "time" command in the Linux environment shows the actual running time of the read/write operation, including the file replication time in the case of writing). For a subset of the operations (the write operation on MSDFS), a digital stopwatch was used, since the MSDFS ecosystem does not offer any automated tool for measuring the replication time, other than visually checking and comparing the times recorded in log files. More details on this are included in section *3.2.4 Validity threats*.

The following two sections (3.2.3.2 and 3.2.3.3) will present the methodology used to find the answers of research questions **RQ1** - *In which areas (for which attributes analyzed) is HDFS superior to MSDFS and why?* and **RQ2** - *In which areas, on the contrary, is MSDFS superior to HDFS and why?*, as defined in section *1.1 Research questions and focus*.

3.2.3.2 Part 1. Write&Read files of increasing size

In this part, the writing/reading times are measured for a file of increasing size. The smallest file has the size 1 MB, and the following ones are successively doubling in size (2 MB, 4 MB, and so on), with the biggest file being 4 GB (4096 MB) in size.

The results will be summarized in a table like the one below.

Table 3. Measurement layout (files of increasing size)

File Size (MB)	Read time (seconds)	Write time (seconds)
1		
2		
4		
8		
16		
32		
64		
128		
256		
512		
1024		
2048		
4096		

The data collected at this step is analyzed in order to show how the two DFSs scale when dealing with files of varying size. An expected outcome is to show which DFS is better when dealing with small files and big files, respectively. Moreover, a threshold may be found, that indicates after which file size one DFS is to be preferred to the other one.

3.2.3.3 Part 2. Write&Read one big file vs. many small files

In this part, each DFS will be used to show the read/write time for the same quantity of data, but in two different scenarios. The first scenario is "1 big file" (for example, one file with the size of 512 MB), while the second scenario is "many small files" (but amounting to the same total size; for example 8192 files, each with the size of 64 kbytes, amounting to the same total of 512 MB).

This comparison will be made for different total sizes, from 128 MB and successively doubling up, until the maximum size of 4096 MB. The size of the small files will be constant, at 64 kbytes. This can be summarized in the following table. The measured times are an average of a different number of measurements (20 for read operations, 10 for write operations)

Table 4. Measurement layout (one big file vs. many small files)

One big file	**Read time (s)**	**Write time (s)**	**Many small files**	**Read time (s)**	**Write time (s)**
128 MB			2048 files x 64 KB		
256 MB			4096 files x 64 KB		
512 MB			8192 files x 64 KB		
1024 MB			16384 files x 64 KB		
2048 MB			32768 files x 64 KB		
4096 MB			65536 files x 64 KB		

The purpose of this part is to show how the fragmentation of data affects performance in the two DFSs. For example, HDFS is known, by design, to be better at handling fewer big files than many smaller files. This experiment part will try to prove this, but the most interesting conclusion is the performance comparison between HDFS and MSDFS. For example, it is interesting to find out which of the two DFSs sees a quicker performance degradation when dealing with an increasing number of small files.

3.2.3.4 Part3. Key usage scenarios

This section presents the methodology used to find the answers of **RQ3** – *Which of the two systems is superior in the key usage scenarios observed, and why?* and of sub-questions **RQ3.1 – 3.4**, as defined in section *1.1 Research questions and focus*, and detailed below.

Besides the reading and writing operations, the empirical study will comparatively analyze a few key usage scenarios, identified according to the motivation presented in section *1.5 Method of investigation*, that define the robustness and flexibility of a distributed file system:

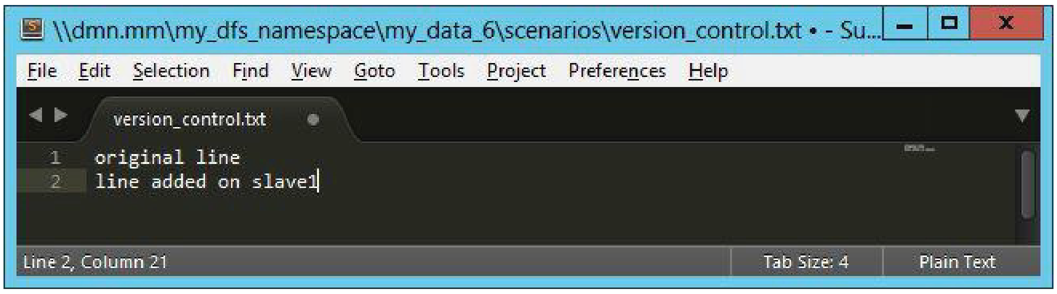
- "One datanode goes offline – how does the cluster react?" (fault-tolerance)

  One datanode is disconnected from the cluster by taking out the network cable. The following will be observed:

  - Is all data still available when an external client tries to access it? **(RQ3.1)**
  - What is the replication factor after the disconnection? (originally it is 3, meaning that each file is present in three places in the cluster for redundancy and fault-tolerance purposes) **(RQ3.2)**

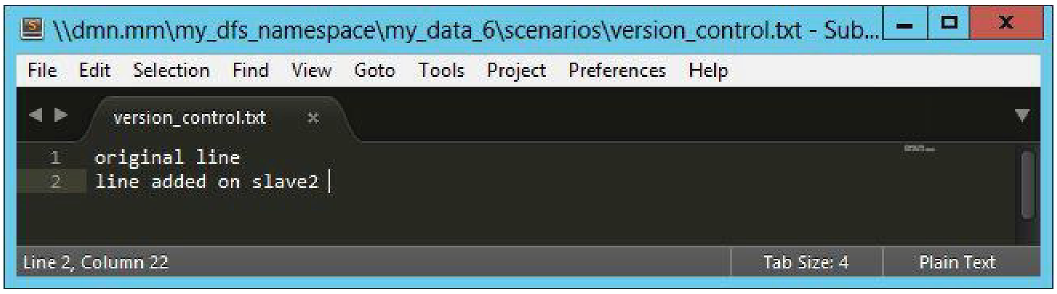
- "Two users open the same file, modify it and save it" (version-control)

  One file called "version_control.txt" containing one line of text ("original line") is opened from two data nodes (slave1 and slave2) at the same time. On slave1 a new line is added below the original line ("line added from slave1"), and exactly the same is done on slave2, where a new line is added to the original line ("line added from slave2"). Now the contents of the file (unsaved) on slave1 are, first, "original line" and, below, "line added on slave1", and on slave2 (also unsaved) are, first, "original line", and below, "line added on slave2" (see screenshots below).


\\dmn.mm\my_dfs_namespace\my_data_6\scenarios\version_control.txt • - Su...
File Edit Selection Find View Goto Tools Project Preferences Help
version_control.txt

```
original line
line added on slave1
```

Line 2, Column 21
Tab Size: 4
Plain Text


\\dmn.mm\my_dfs_namespace\my_data_6\scenarios\version_control.txt - Sub...
File Edit Selection Find View Goto Tools Project Preferences Help
version_control.txt

```
original line
line added on slave2
```

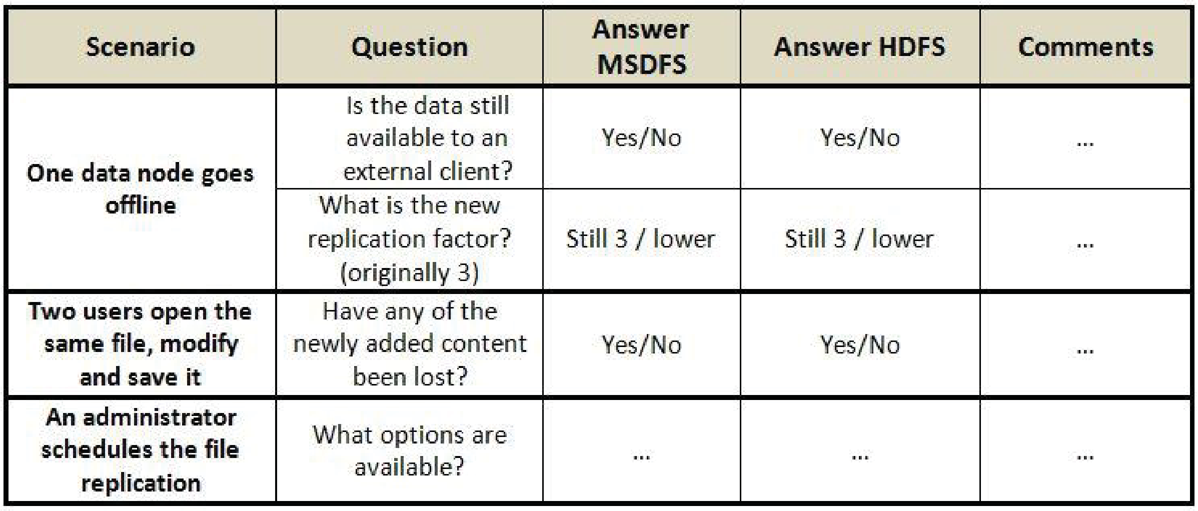
Line 2, Column 22
Tab Size: 4
Plain Text

Now, the file on slave1 is saved first, and after that, the file on slave2 is also saved. Both files are closed. An external client is used to access the MSDFS cluster and open the file.

- Have any of the newly added lines been lost? **(RQ3.3)**

- "An administrator wants to configure the file replication schema" (replication settings flexibility)
    - What main options are available for the file replication schema? **(RQ3.4)**

Table 5. Expected results of key usage scenarios experiment part



Scenario	Question	Answer MSDFS	Answer HDFS	Comments
One data node goes offline	Is the data still available to an external client?	Yes/No	Yes/No	...
	What is the new replication factor? (originally 3)	Still 3 / lower	Still 3 / lower	...
Two users open the same file, modify and save it	Have any of the newly added content been lost?	Yes/No	Yes/No	...
An administrator schedules the file replication	What options are available?	...	...	...

3.2.4 Validity threats

3.2.4.1 Setup-related threats

- Because the setup involves the presence of virtual machines, the question of the delays due to the communication between the host OS and the guest OS (the virtual machine) appears. The amplitude of these delays is difficult to predict and could vary between various parts of the experiment. The method chosen to minimize this threat was to provide the same hardware platform to both HDFS and MSDFS, be it native installation (the case of one computer) or virtual machine installations (the case of the other three computers). Since the goal of the empirical study is to just expose the differences between the two distributed file systems, and not to record absolute values of the measurements, this method minimizes the impact of communication delays between the host and the guest operating systems, as long as the analyzed file systems run, in turn, on exactly the same hardware (be it native or virtual).

- Because of the unique setup chosen for this thesis, the absolute times measured are not very relevant, as the exact setup cannot be easily reproduced. However, the relative times (comparisons and differences between the two DFSs) should be relevant enough and reproducible on other setups.

3.2.4.2 Methodology-related threats

- In the experiment for this paper, the large files are limited to 4096 MB (4 GB) in size, for practical reasons like available storage space and the time available for doing the measurements. For a more clear perspective on how the two DFSs handle large files, however, bigger sizes would have given better results.
- The small files are set at a size of 64 KB. This was decided after several “trial & error” attempts in order to ensure a reasonably low number of files and a reasonable running time of the read and write operations. Having smaller files (for example, with the size of 1 KB) may have been a better choice in order to expose how the analyzed DFSs react when dealing with small files, but the additional overhead would have led to impractically long running times for the time span allocated to this thesis.
- The measured times are an average of a different number of measurements (20 for read operations, 10 for write operations). The difference (10 vs. 20) is due to the fact that write operations take on average 4-5 times longer time than read operations and that the write operations on MSDFS cannot be automated with scripts (see the bullet below for details). Taking into account the time budget allocated to this experiment, increasing the number of measurements to more than 10 for each write operation would have been impractical.
- The measurements done on MSDFS, on write operations, have a bigger error margin because they were done using a digital stopwatch, as the MSDFS ecosystem does not provide any automated tools to measure the replication time of a file between datanodes. Due to some limitations in the monitoring tools available for MSDFS, the only ways available to time the replication operation that were identified during the research phase were by manually checking logs or by manually timing using a stopwatch. The latter method was used, since the first one would have been too time-consuming.
- The last scenario (“An administrator schedules the file replication”) proposes an open question (“What options are available?”). Due to the possibly large variety of options present in the two DFSs, a direct comparison between the two may be difficult to achieve. The method to mitigate this risk is to select the most important options that can be compared and provide additional comments to further clarify the validity of the comparison.

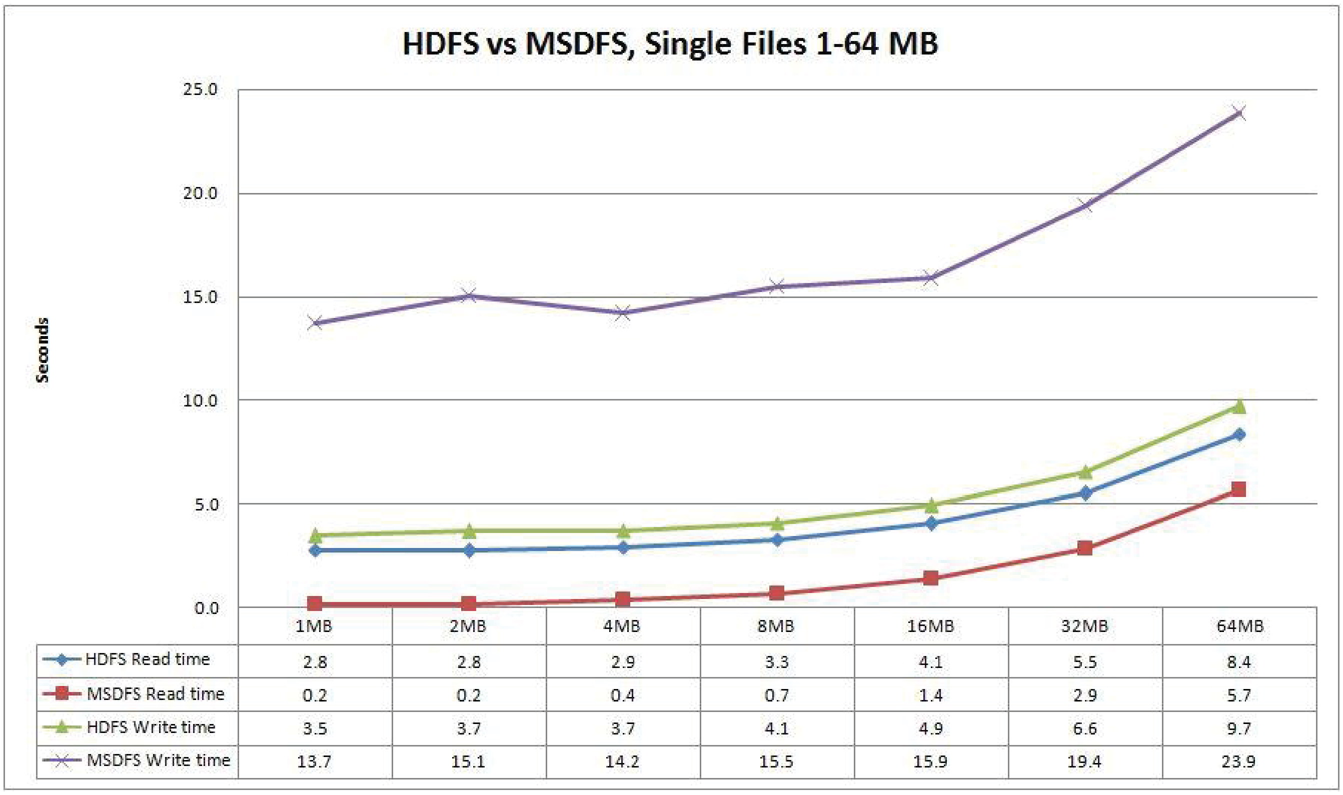
4 RESULTS

In this chapter, the results of the measurements are presented for the three parts of the experiment.

4.1 Part1. Write&Read single files of increasing size

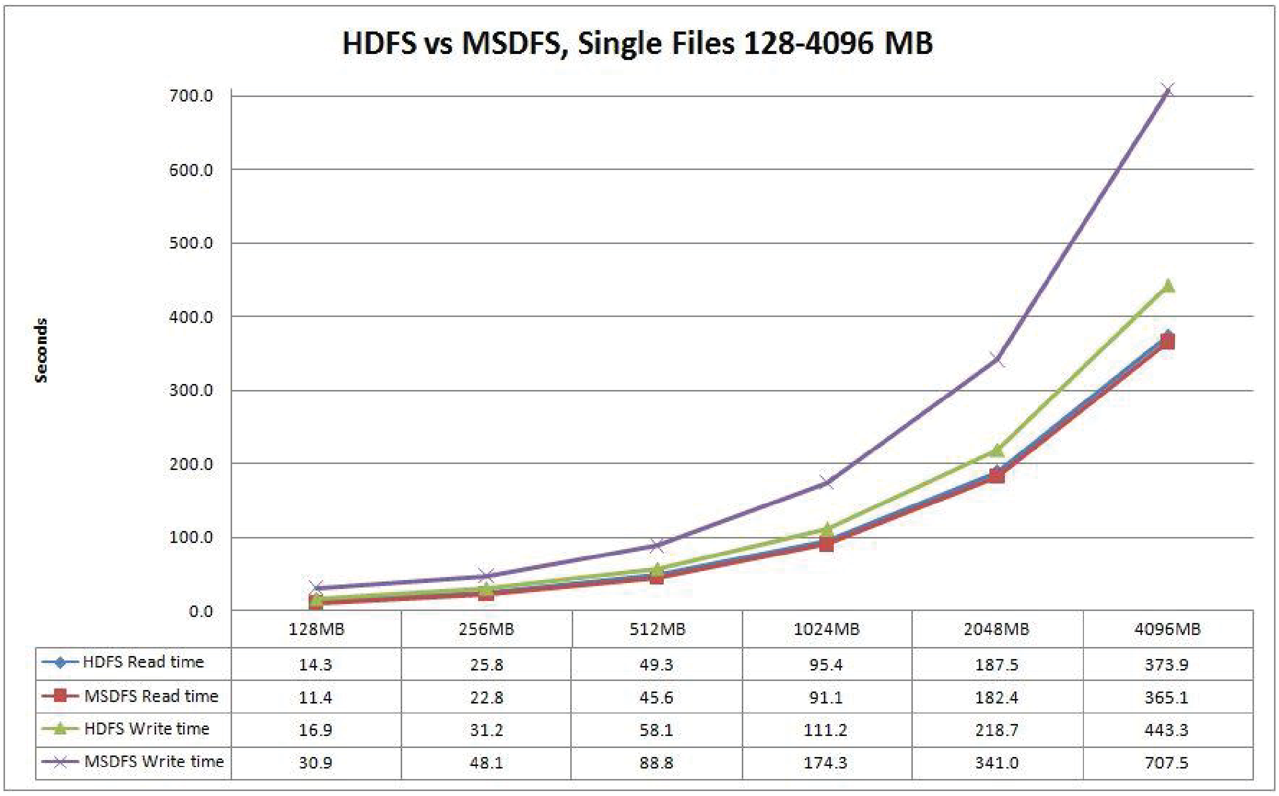
Figures 4 and 5 present a quick summary of the results for this part of the experiment. More details and interpretation of the results follow.

Figure 4. Measurements result overview (files of increasing size, 1-64 MB)



	1MB	2MB	4MB	8MB	16MB	32MB	64MB
HDFS Read time	2.8	2.8	2.9	3.3	4.1	5.5	8.4
MSDFS Read time	0.2	0.2	0.4	0.7	1.4	2.9	5.7
HDFS Write time	3.5	3.7	3.7	4.1	4.9	6.6	9.7
MSDFS Write time	13.7	15.1	14.2	15.5	15.9	19.4	23.9

Figure 5. Measurements result overview (files of increasing size, 128 – 4096 MB)



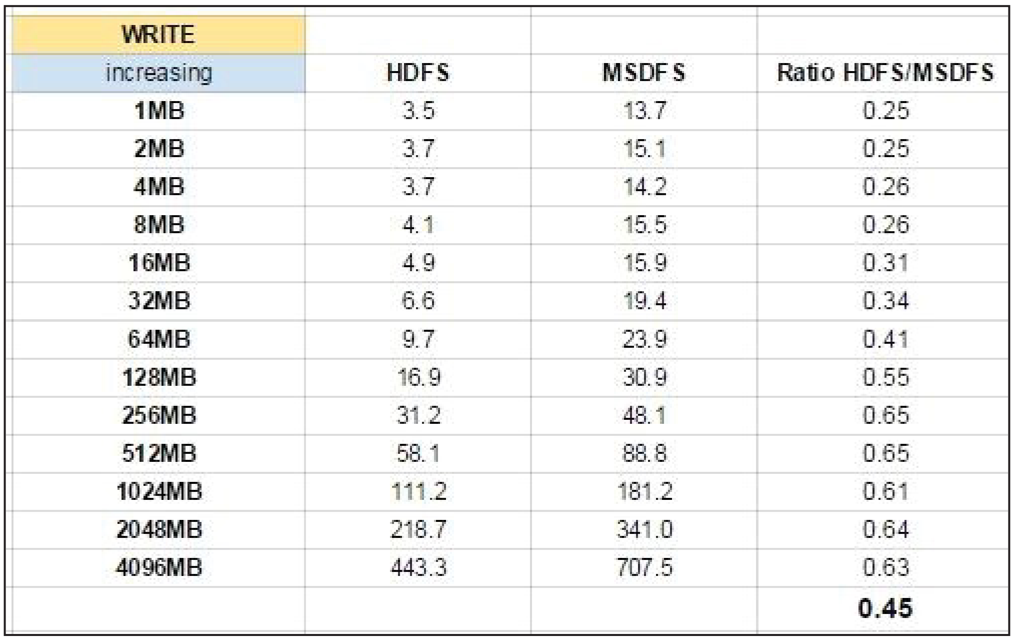
	128MB	256MB	512MB	1024MB	2048MB	4096MB
◆ HDFS Read time	14.3	25.8	49.3	95.4	187.5	373.9
■ MSDFS Read time	11.4	22.8	45.6	91.1	182.4	365.1
▲ HDFS Write time	16.9	31.2	58.1	111.2	218.7	443.3
× MSDFS Write time	30.9	48.1	88.8	174.3	341.0	707.5

Subsequently, the results for read and write operations, respectively, are detailed.

- Write

HDFS has performed better in this part of the experiment. Even if some of the files are below the 128 MB threshold (the size of the basic block of storage in HDFS), writing data from the local file system to the distributed file system is still on average two times faster in the case of HDFS than in the case of MSDFS (average ratio of 0.45, see Table 6).

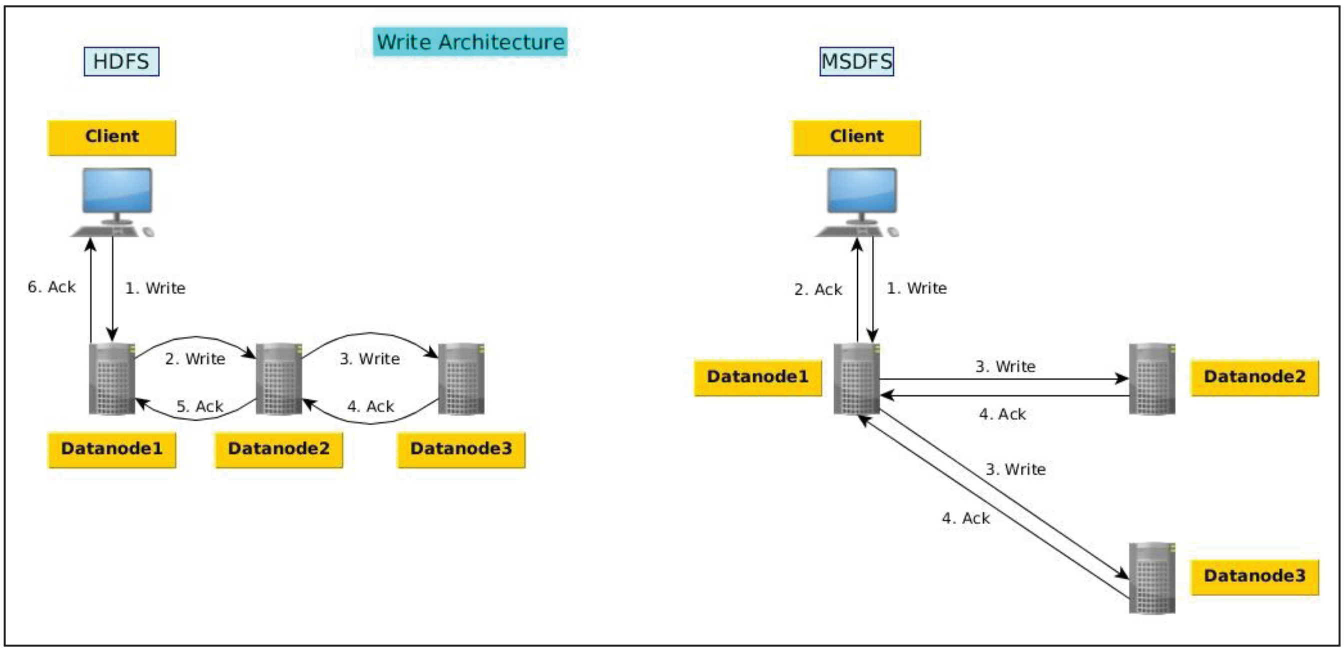
Table 6. Measurements results HDFS vs. MSDFS (files of increasing size – WRITE time in seconds)



WRITE			
increasing	HDFS	MSDFS	Ratio HDFS/MSDFS
1MB	3.5	13.7	0.25
2MB	3.7	15.1	0.25
4MB	3.7	14.2	0.26
8MB	4.1	15.5	0.26
16MB	4.9	15.9	0.31
32MB	6.6	19.4	0.34
64MB	9.7	23.9	0.41
128MB	16.9	30.9	0.55
256MB	31.2	48.1	0.65
512MB	58.1	88.8	0.65
1024MB	111.2	181.2	0.61
2048MB	218.7	341.0	0.64
4096MB	443.3	707.5	0.63
			0.45

During the experiment, the following write architecture was observed in the two distributed file systems - see Figure 6. The main difference is that HDFS first replicates each block of data on the three data nodes and then moves on to the next block (with acknowledge messages being sent successively from the last datanode back to the previous node and so on until the client gets the acknowledgement that the block was successfully copied), while MSDFS first copies the whole quantity of data to the first datanode, the acknowledge message is sent back to the client, and then the first datanode starts the replication process by copying the data to the other datanodes.

Figure 6. Write Architecture (comparison HDFS vs. MSDFS)



According to the measurements done during the experiment, the HDFS write architecture is more efficient, since the write operation takes only 45 percent of the time required by the MSDFS (see Table 6).

- Read

The situation turns around, however, when it comes to reading data from the distributed file system to the local file system. In this case, MSDFS was four times faster (ratio 4.2, see Table 7 below), but the number may be a little deceiving, because it is mainly based on results from transferring files with the size under 128 MB.

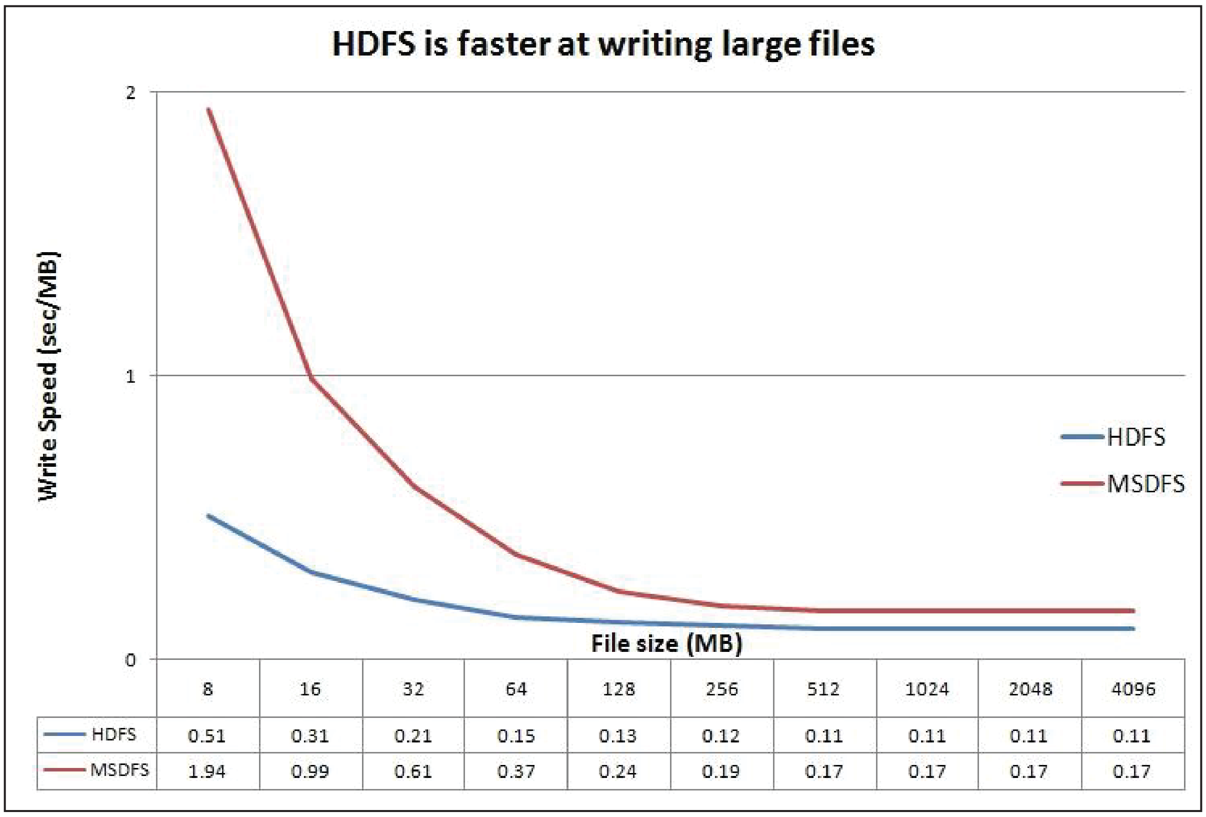
If one would look closer, and divide the files into two categories, below and above the 128 MB threshold (as the basic block size of storage in HDFS is 128 MB), the results would be very different in the two categories (see Table 7 below). For the smaller files (under 128 MB), MSDFS is nearly 7 times faster than HDFS (ratio 6.9), having the advantage of the fact that the distributed file system is built on the regular NTFS file system that Windows runs, and the reading operation pretty much involves only the regular steps required when copying from another computer in the LAN. For the larger files (above 128 MB), the situation changes, and the two DFSs are similar in performance, with a minor advantage of the MSDFS (ratio 1.1). The improved performance of HDFS in the case of files over the 128 MB threshold can be explained by the fact that the overhead required to decide from which data node the data will be transferred is leveraged by the architectural design optimized for large files.

Table 7. Measurements results HDFS vs. MSDFS (files of increasing size – READ time in seconds)

READ					
increasing	**HDFS**	**MSDFS**	Ratio HDFS/MSDFS		just small files
1MB	2.8	0.2	16.32		16.32
2MB	2.8	0.2	13.78		13.78
4MB	2.9	0.4	7.16		7.16
8MB	3.3	0.7	4.71		4.71
16MB	4.1	1.4	2.88		2.88
32MB	5.5	2.9	1.93		1.93
64MB	8.4	5.7	1.47	just big files	1.47
128MB	14.3	11.4	1.26	1.26	**6.9**
256MB	25.8	22.8	1.13	1.13	
512MB	49.3	45.6	1.08	1.08	
1024MB	95.4	91.1	1.05	1.05	
2048MB	187.5	182.4	1.03	1.03	
4096MB	373.9	365.1	1.02	1.02	
			4.2	**1.1**	

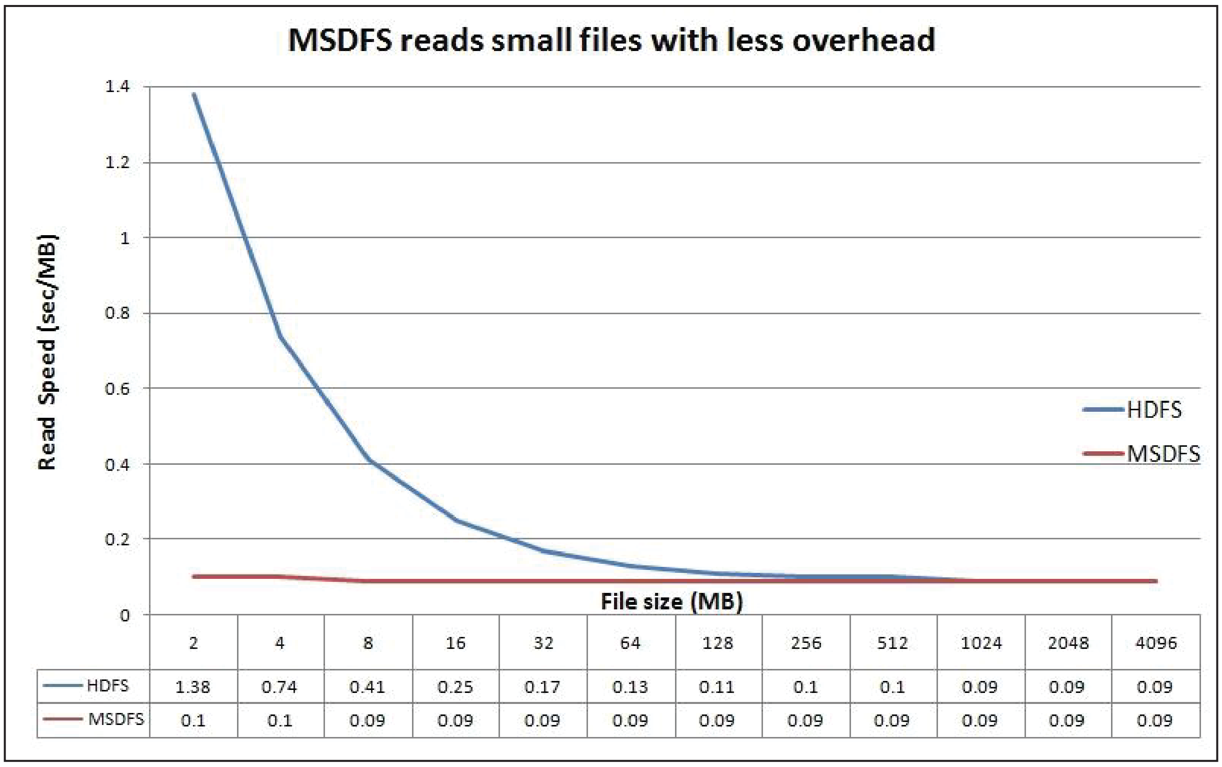
When analyzing how effectively each DFS handles files that are getting bigger and bigger, it is useful to see how the write and the read speed (time / MB) are changing (see Figure 7 and Figure 8). The conclusions that arise from the measurements are that, first, HDFS is faster at writing large files, and, second, that while MSDFS has less overhead when reading small files (and thus beats HDFS at this type of operation), it reads large files with roughly the same speed as HDFS. These conclusions are illustrated in the charts below.

Figure 7. HDFS vs. MSDFS (write speed, files of increasing size)



File size (MB)	8	16	32	64	128	256	512	1024	2048	4096
HDFS	0.51	0.31	0.21	0.15	0.13	0.12	0.11	0.11	0.11	0.11
MSDFS	1.94	0.99	0.61	0.37	0.24	0.19	0.17	0.17	0.17	0.17

Figure 8. HDFS vs. MSDFS (read speed, files of increasing size)

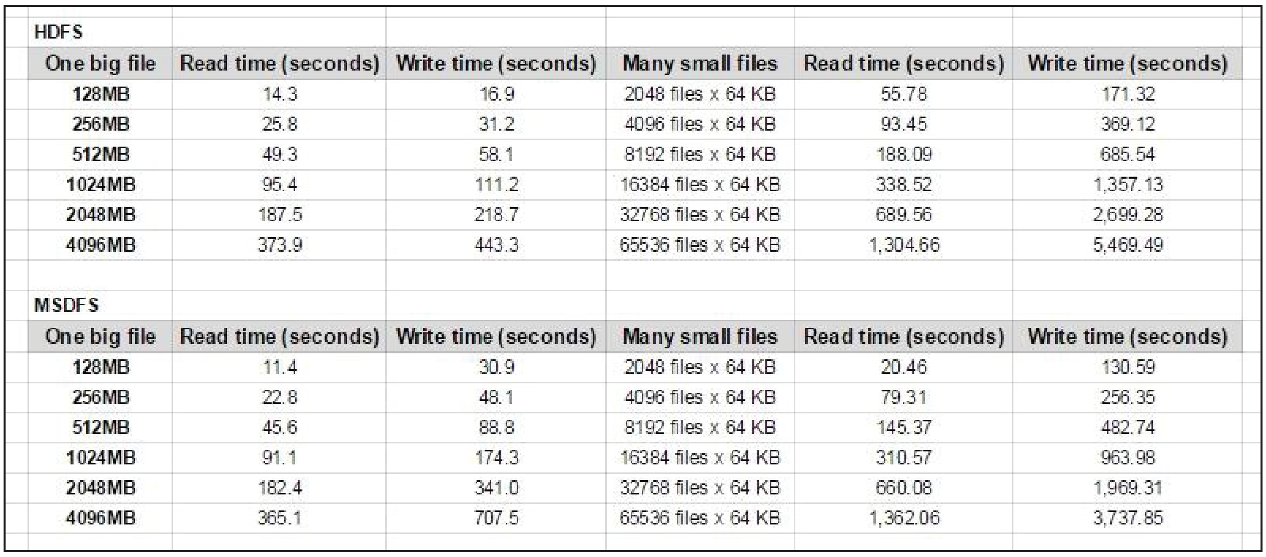


File size (MB)	2	4	8	16	32	64	128	256	512	1024	2048	4096
HDFS	1.38	0.74	0.41	0.25	0.17	0.13	0.11	0.1	0.1	0.09	0.09	0.09
MSDFS	0.1	0.1	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09

4.2 Part2. Write&Read one big file versus many small files

The following table presents a quick summary of the results of this part of the experiment. Details and interpretation of the results follow.

Table 8. Measurements result overview (one big file vs. many small files)



HDFS

One big file	Read time (seconds)	Write time (seconds)	Many small files	Read time (seconds)	Write time (seconds)
128MB	14.3	16.9	2048 files x 64 KB	55.78	171.32
256MB	25.8	31.2	4096 files x 64 KB	93.45	369.12
512MB	49.3	58.1	8192 files x 64 KB	188.09	685.54
1024MB	95.4	111.2	16384 files x 64 KB	338.52	1,357.13
2048MB	187.5	218.7	32768 files x 64 KB	689.56	2,699.28
4096MB	373.9	443.3	65536 files x 64 KB	1,304.66	5,469.49

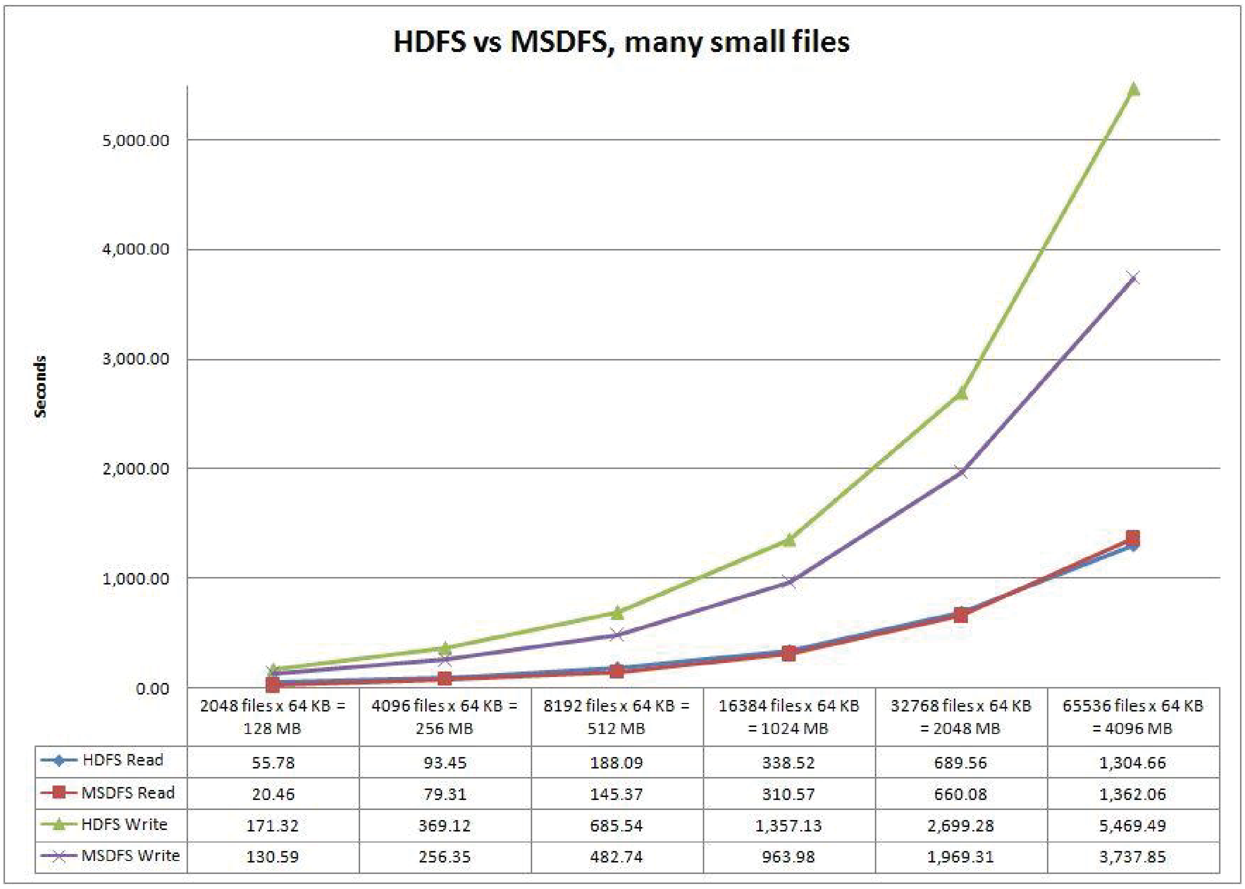
MSDFS

One big file	Read time (seconds)	Write time (seconds)	Many small files	Read time (seconds)	Write time (seconds)
128MB	11.4	30.9	2048 files x 64 KB	20.46	130.59
256MB	22.8	48.1	4096 files x 64 KB	79.31	256.35
512MB	45.6	88.8	8192 files x 64 KB	145.37	482.74
1024MB	91.1	174.3	16384 files x 64 KB	310.57	963.98
2048MB	182.4	341.0	32768 files x 64 KB	660.08	1,969.31
4096MB	365.1	707.5	65536 files x 64 KB	1,362.06	3,737.85

4.2.1 Write&Read many small files

When it comes to transferring many small files, the conclusion of the experiment is that HDFS is slower than MSDFS, for both writing and reading operations (see Figure 9).

Figure 9. HDFS vs. MSDFS (many small files)

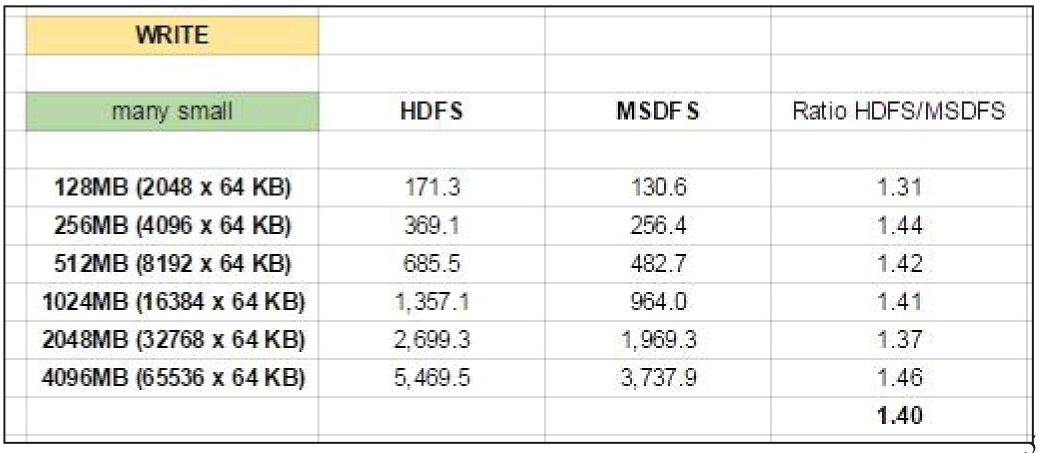


	2048 files x 64 KB = 128 MB	4096 files x 64 KB = 256 MB	8192 files x 64 KB = 512 MB	16384 files x 64 KB = 1024 MB	32768 files x 64 KB = 2048 MB	65536 files x 64 KB = 4096 MB
HDFS Read	55.78	93.45	188.09	338.52	689.56	1,304.66
MSDFS Read	20.46	79.31	145.37	310.57	660.08	1,362.06
HDFS Write	171.32	369.12	685.54	1,357.13	2,699.28	5,469.49
MSDFS Write	130.59	256.35	482.74	963.98	1,969.31	3,737.85

- Write

In the case of data being uploaded from the local file system to the distributed file system, the ratio between the average time needed by HDFS and the time needed by MSDFS was 1.40, meaning that HDFS was by 40 percent slower than MSDFS.

Table 9. HDFS vs. MSDFS (many small files, WRITE time in seconds)

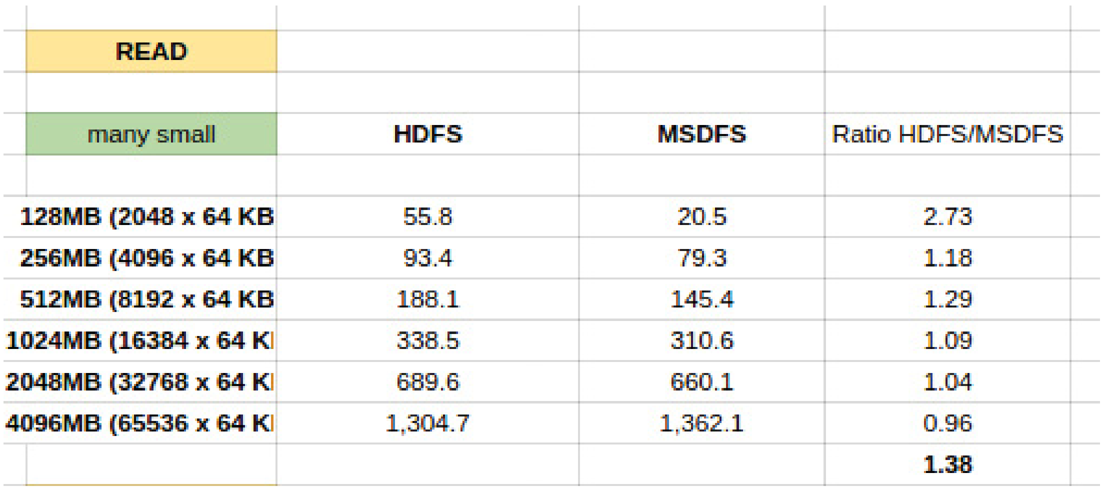


WRITE			
many small	HDFS	MSDFS	Ratio HDFS/MSDFS
128MB (2048 x 64 KB)	171.3	130.6	1.31
256MB (4096 x 64 KB)	369.1	256.4	1.44
512MB (8192 x 64 KB)	685.5	482.7	1.42
1024MB (16384 x 64 KB)	1,357.1	964.0	1.41
2048MB (32768 x 64 KB)	2,699.3	1,969.3	1.37
4096MB (65536 x 64 KB)	5,469.5	3,737.9	1.46
			1.40

- Read

Similarly, in the case of data being downloaded from the DFS to the local file system, HDFS is also slower than MSDFS. The ratio in this case was not very different (1.38), meaning that HDFS was by 38 percent slower than Microsoft's distributed file system.

Table 10. HDFS vs. MSDFS (many small files, READ time in seconds)



READ			
many small	HDFS	MSDFS	Ratio HDFS/MSDFS
128MB (2048 x 64 KB	55.8	20.5	2.73
256MB (4096 x 64 KB	93.4	79.3	1.18
512MB (8192 x 64 KB	188.1	145.4	1.29
1024MB (16384 x 64 K	338.5	310.6	1.09
2048MB (32768 x 64 K	689.6	660.1	1.04
4096MB (65536 x 64 K	1,304.7	1,362.1	0.96
			1.38

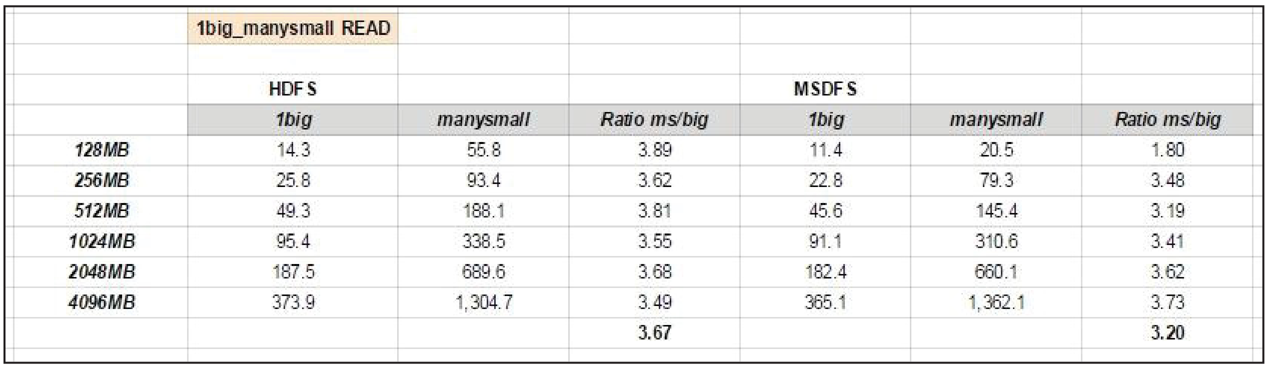
An explanation for this behavior is the fact that HDFS is optimized for transferring and storing files of at least 128 MB (which is the size of a basic block of storage), while the files used in this part of the experiment had the size of 64 KB each. On the other hand, Microsoft's MSDFS is built to behave more like a natural desktop file system (with the advantage of it being distributed), where small files are used with a much higher frequency.

4.2.2 Performance degradation when switching from one big file to many small files - HDFS vs. MSDFS

It is natural that performance should decrease when transferring a constant quantity of data, but fragmented into many small files. Pretty much any file system that has ever been invented would perform worse when transferring, for example, 65,000 files with the size of 64 KB each (4 GB in total) compared to when transferring one single file with the same size of 4 GB. The overhead of keeping track of so many file names can be tremendous. However, the interesting thing to observe when comparing HDFS and MSDFS is how fast the performance is eroding when the quantity of data that is divided in many small files is increased and which of the two is in better shape when dealing with a large quantity of small files.

The conclusion of this part of the experiment is that MSDFS performs better, overall, when burdened with many small files. The difference between the two DFSs is insignificant for the reading operation - MSDFS reads many small files 3.2 times slower than the same quantity of data grouped in a single file, while HDFS sees a performance decrease of 3.67 times in the same conditions – see Table 11.

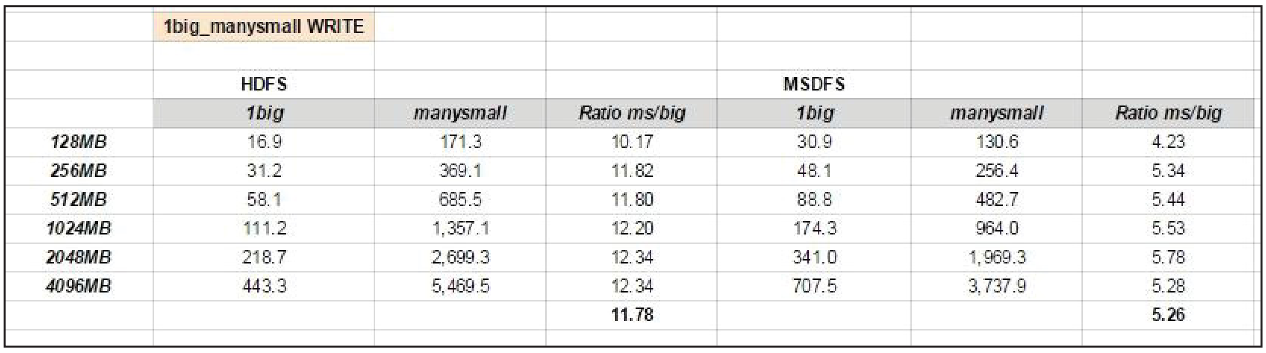
Table 1. HDFS vs. MSDFS (1 big file vs. many small files, READ time in seconds, RATIO manysmall/1big)



	1big_manysmall READ					
	HDFS			MSDFS		
	1big	*manysmall*	*Ratio ms/big*	*1big*	*manysmall*	*Ratio ms/big*
128MB	14.3	55.8	3.89	11.4	20.5	1.80
256MB	25.8	93.4	3.62	22.8	79.3	3.48
512MB	49.3	188.1	3.81	45.6	145.4	3.19
1024MB	95.4	338.5	3.55	91.1	310.6	3.41
2048MB	187.5	689.6	3.68	182.4	660.1	3.62
4096MB	373.9	1,304.7	3.49	365.1	1,362.1	3.73
			3.67			**3.20**

However, the difference is clear on the write operation. MSDFS is two times more effective than HDFS when burdened with writing many small files compared to writing one big file. More exactly, MSDFS has a performance worsening ratio of just 5.26 when writing many small files compared to writing the same quantity of data grouped in a single file, while HDFS experiences a 11.78 ratio in the same scenario, meaning that it takes nearly 12 times longer time on average for HDFS to deal with many small files than with one big file.

Table12. HDFS vs. MSDFS (1 big file vs. many small files, WRITE time in seconds, RATIO manysmall/1big)



	1big_manysmall WRITE					
	HDFS			MSDFS		
	1big	*manysmall*	*Ratio ms/big*	*1big*	*manysmall*	*Ratio ms/big*
128MB	16.9	171.3	10.17	30.9	130.6	4.23
256MB	31.2	369.1	11.82	48.1	256.4	5.34
512MB	58.1	685.5	11.80	88.8	482.7	5.44
1024MB	111.2	1,357.1	12.20	174.3	964.0	5.53
2048MB	218.7	2,699.3	12.34	341.0	1,969.3	5.78
4096MB	443.3	5,469.5	12.34	707.5	3,737.9	5.28
			11.78			**5.26**

4.2.3 Part3. Key usage scenarios

The results are summarized in Table 13 and detailed in the remainder of the section.

Table13. Key usage scenarios – results summary

Scenario	Question	Answer MSDFS	Answer HDFS	Comments
One data node goes offline *(fault tolerance)*	Is the data still available to an external client?	Yes	Yes	Both DFSs continue to operate normally after the disconnection, as copies of the data still exist on other nodes.
	What is the new replication factor? (originally 3)	2 (no re-replication attempted)	2 (re-replication is attempted)	Only HDFS takes measures to recover and bring the replication factor back to the original value.
Two users open the same file, modify and save it *(version control)*	Have any of the newly added content been lost?	Yes	No	MSDFS allows random edits of files but offers no version control (data can be lost when two users modify and save the same file), HDFS only allows appends on files, but does not lose data in the described scenario.
An administrator configures the file replication *(replication settings flexibility)*	What options are available?	Replication groups, connections (directions of replication), replication topologies can be configured.	Block size and replication factor can be configured at file level, the NameNode automatically handles the rest.	HDFS shows less flexibility (none of the mentioned MSDFS options are present), but is more robust in case of disaster due to default rack-aware replication.

4.2.3.1 One datanode goes offline – how does the cluster react? (fault-tolerance)

Research questions:

- *RQ3.1: Is all data still available when an external client tries to access it?*

MSDFS. Yes, all data is still completely available, with each of the two remaining nodes holding a copy of each file.

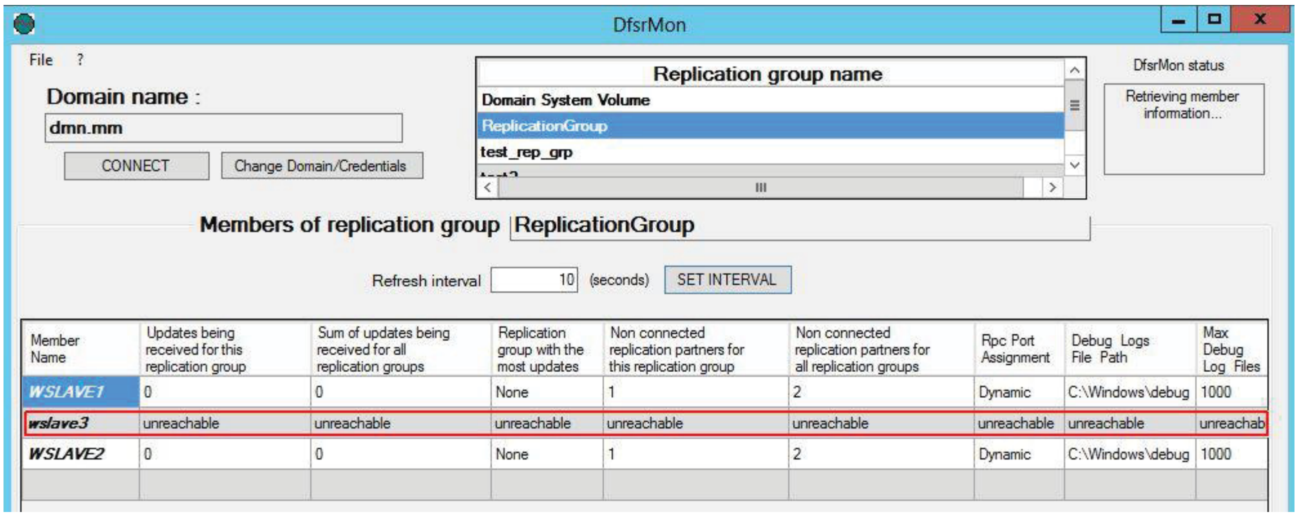
HDFS. Yes, because the remaining two nodes each contain two copies of each block in the cluster (each file is divided into blocks).

Conclusion: Both DFSs continue to operate normally if one node goes offline.

- *RQ3.2: What is the replication factor after the disconnection? (originally it is 3, meaning that each file is present in three places in the cluster for redundancy and fault-tolerance purposes)*

MSDFS. The new replication factor is 2. The disconnected node has disappeared from the cluster with all its data (see Figure 10). However, the data is still available on the other two nodes (and that is why the replication factor is now 2).

Figure 10. DfsrMon monitoring tool shows that the disconnected node is unreachable.

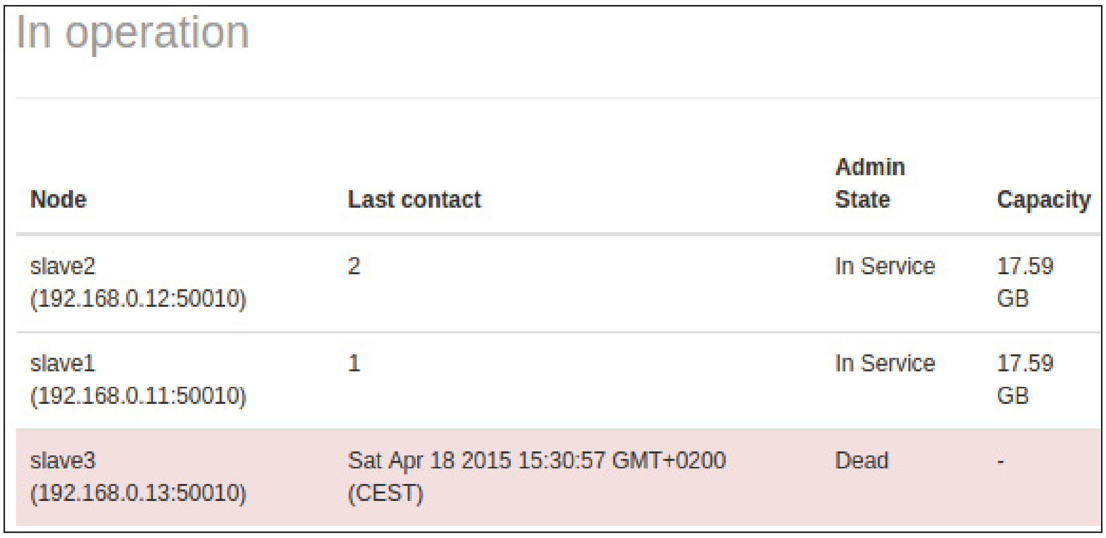


Member Name	Updates being received for this replication group	Sum of updates being received for all replication groups	Replication group with the most updates	Non connected replication partners for this replication group	Non connected replication partners for all replication groups	Rpc Port Assignment	Debug Logs File Path	Max Debug Log Files
WSLAVE1	0	0	None	1	2	Dynamic	C:\Windows\debug	1000
wslave3	unreachable	unreachable	unreachable	unreachable	unreachable	unreachable	unreachable	unreachab
WSLAVE2	0	0	None	1	2	Dynamic	C:\Windows\debug	1000

HDFS. The new replication factor is 2 immediately after the disconnection, but the namenode notices that the value is below the configured value of 3 and asks for the re-replication of the missing blocks. In the cluster scenario used in this experiment, only three datanodes make up the cluster. This means that if one node goes offline, the replication factor of 3 cannot be achieved. However, in scenarios with more than three nodes in the cluster, copies of the missing blocks are identified on the remaining nodes and re-replicated to other nodes (where the blocks are not already present). In this way, the replication factor is again brought up to 3, even if one node was unexpectedly disconnected.

Conclusion: HDFS has a better replication architecture which allows it to keep the replication factor constant even if the number of live nodes decreases. This is achieved thanks to the block-based architecture (files are divided into blocks, and several copies of the blocks are spread across nodes). MSDFS, on the other hand, has always a replication factor equal to the number of live nodes.

Figure 11. The HDFS web interface shows slave3 node as dead after disconnection.



In operation

Node	Last contact	Admin State	Capacity
slave2 (192.168.0.12:50010)	2	In Service	17.59 GB
slave1 (192.168.0.11:50010)	1	In Service	17.59 GB
slave3 (192.168.0.13:50010)	Sat Apr 18 2015 15:30:57 GMT+0200 (CEST)	Dead	-

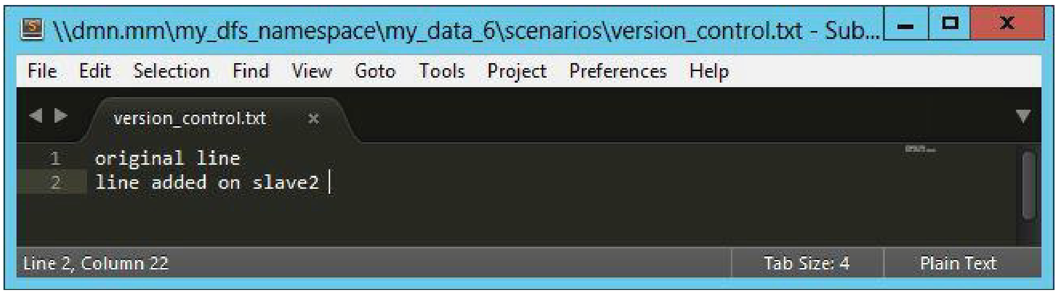
4.2.3.2 Two users open the same file, modify it and save it. Both files are closed – does any of the two users lose his changes? (version-control)

Answer to research questions:

- *RQ3.3: Have any of the newly added lines been lost?*

<u>MSDFS</u>. Yes, the latest save (on slave2) has overwritten the changes done by the first save (on slave1). The contents of the file are now the ones in Figure 12.

Figure 12. Changes made by slave1 were overwritten by those made by slave2 (MSDFS).



\\dmn.mm\my_dfs_namespace\my_data_6\scenarios\version_control.txt - Sub...

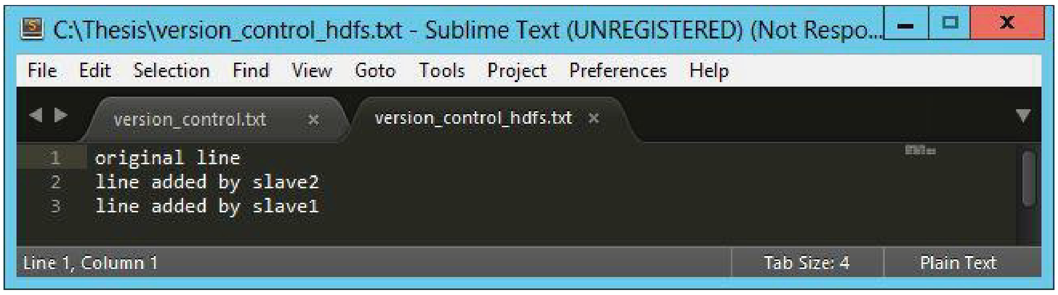
```
original line
line added on slave2
```

File Edit Selection Find View Goto Tools Project Preferences Help
version_control.txt
Line 2, Column 22
Tab Size: 4
Plain Text

This happens because all new data is replicated to the other nodes, so the first save (on slave1) is propagated to the other nodes, and then the second save (on slave2) is also propagated, meaning that the latest version of the file becomes the one on slave2.

HDFS. No content was lost. HDFS allows append operations to files that already exist in the cluster. In the experiment, two clients attempted to append new data to a file in the cluster. Both operations succeeded. However, the order cannot be guaranteed. In the experiment, the original contents of the file was a line of text ("original line"), and the two clients tried each to append a new line ("line added by slave1" and "line added by slave2"). Even if the first append command was run on slave1, followed immediately by the second append command run on slave2, it was the line on slave2 that first got written to file. Thus, the final content of the file was the one depicted in Figure 13.

Figure13. Both changes were kept, but the order is not guaranteed when two clients concurrently try to append data (HDFS).


C:\Thesis\version_control_hdfs.txt - Sublime Text (UNREGISTERED) (Not Respo...
File Edit Selection Find View Goto Tools Project Preferences Help
version_control.txt
version_control_hdfs.txt
Line 1, Column 1
Tab Size: 4
Plain Text

```
original line
line added by slave2
line added by slave1
```

Conclusion: MSDFS allows editing of files, but does not offer version control (contents can be lost when two users are editing the same file). HDFS only offers the possibility to append new content at the end of existing files, but guarantees that two users accessing the same file will not lose contents after saving the changes.

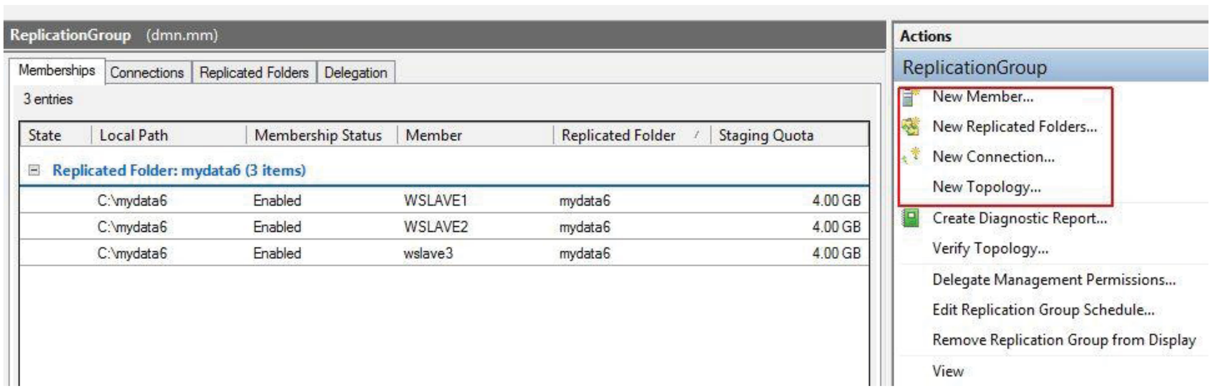
4.2.3.3 An administrator wants to configure the file replication schema – what options are available and how flexible are they? (replication settings flexibility)

Answer to research questions:

- *RQ3.4: What main options are available when configuring the file replication?*

MSDFS. The main options can be accessed as actions inside a replication group (a logical entity that allows an administrator to quickly configure one or several replicated folders) – see Figure 14.

Figure14. Options available in MSDFS to configure the file replication.


ReplicationGroup (dmn.mm)
Memberships
Connections
Replicated Folders
Delegation
Actions
ReplicationGroup
New Member...
New Replicated Folders...
New Connection...
New Topology...
Create Diagnostic Report...
Verify Topology...
Delegate Management Permissions...
Edit Replication Group Schedule...
Remove Replication Group from Display
View

3 entries

State	Local Path	Membership Status	Member	Replicated Folder	Staging Quota
Replicated Folder: mydata6 (3 items)					
	C:\mydata6	Enabled	WSLAVE1	mydata6	4.00 GB
	C:\mydata6	Enabled	WSLAVE2	mydata6	4.00 GB
	C:\mydata6	Enabled	wslave3	mydata6	4.00 GB

- Add a new member: a new computer can be added as a datanode
- Add a new replicated folder: an administrator can add new folders to the replication group, by indicating the name of the new folder and the location in the hierarchy of the namespace, as well as the target folders on the datanodes (the physical, local folders on each datanode that point to the common replicated folder in the namespace).
- Add a new connection: a connection is a direction in which the data in a folder is replicated (for example, the data on node1 is replicated to node2 but not the other way around). All directions and options are possible, meaning that the administrator has complete control over the directions in which the data is replicating in the replication group.
- Add a new topology: it contains the same functionality as the previous option (add a new connections), but it is quicker to configure. The main options that are present here are: full-mesh topology (all nodes replicate data to all other nodes) and hub-and-spoke (one node, the hub, has the data that needs to be replicated to the other nodes, the spoke members).

HDFS. HDFS does not support replication groups, adding connections or topologies. Only the blocks of a file are replicated (not folders as a whole), and the replication is exclusively taken care of by the NameNode, the metadata server, which periodically receives reports from each of the datanodes in the cluster and can select any datanode to host a certain block (the administrator cannot influence the decisions taken by the NameNode).

The block size and replication factor are configurable per file. This means, for example, that a file that is very important could have an individual replication factor that is higher than the default. This option is present in MSDFS in a slightly different form (a replication group has filters that can include or exclude files in the folder from replication). HDFS has a default policy for how the replication takes place, taking advantage of its replication design that fragments files in blocks which are then spread across the nodes. For a replication factor of 3, one copy of each block is kept on a node on the local rack, while the other two copies are kept on two different nodes, but on the same remote rack (remote rack for fault-tolerance purposes, and the same remote rack in order to minimize I/O overhead in the cluster). This default policy can be changed if the cluster administrator wants to. In the experiment for this paper, all three nodes belong to the same rack, and each block is replicated to each node.

Conclusion: Both DFSs offer a basic set of replication configuration, enough to ensure that all data is properly replicated for fault tolerance purposes. MSDFS offers some configuration options that do not exist in HDFS, for example to set the direction in which the replicated data flows between two servers (from server A to server B but not the other way around). HDFS, on the other hand, automatically spreads the data to different racks, thus being by default better prepared in the eventuality of failure of a whole rack (for example, in case of disaster).

5 ANALYSIS AND ANSWERS TO RESEARCH QUESTIONS

According to the Research Questions formulated in section 1.1, the analysis section reveals the areas where HDFS is superior to MSDFS (answer to **RQ1**), the areas where MSDFS is superior to HDFS (answer to **RQ2**), how the two systems behave in the key usage scenarios selected (answer to **RQ3**) and refers back to the relevant parts of the experiment that support the presented data.

Both products that are analyzed in the paper follow the basic design lines of any successful distributed file system. Both are offering location transparency, fault-tolerance and a configurable file replication schema.

The two most important operations in any file system, read and write, were followed in the practical part of the paper.

For the reading part, MSDFS is generally better, especially when dealing with many small files. In section 4.2.2.1 of the experiment (*Write&Read many small files*), **Table 10** illustrates that HDFS takes on average 38 percent longer time than MSDFS to read the same quantity of many small files. The only area where HDFS is performing similarly well to MSDFS is on reading large, not fragmented files - this is graphically illustrated by **Figure 5** (the *HDFS read time* and *MSDFS read time* curves) and **Table 7,** (the "just big files" part shows a HDFS/MSDFS ratio of 1.1 for files equal to and above 128 MB). When switching from one big file to the same quantity of data divided into many small files, MSDFS is again better (the reading performance worsens 3.2 times, compared to 3.7 times for HDFS) – this is demonstrated in **Table 11**.

For the writing part, HDFS is superior, but only on writing large, not fragmented files – this is very visible in **Figure 5** (see curves *HDFS Write time* and *MSDFS Write time*) and when measuring how the writing performance worsens when switching to many small files: if MSDFS sees a 5 times performance worsening factor, HDFS reports a whooping 12 times performance worsening factor – see **Table 12**.

The conclusion of this part of the experiment is that the relevant differences between MSDFS and HDFS appear especially when the data is fragmented into many small files, where MSDFS is clearly superior (refer to **Table 11** for reading and **Table 12** for writing). The strong point of HDFS, on the other hand, is on large, not fragmented files (it is two times faster than MSDFS on writing – see **Figure 5** and **Table 6**, and is as good as MSDFS on reading – see *"just big files"* in **Table 7**).

Thus, a system administrator should be aware that MSDFS performs generally better in an environment where small files are the rule, while HDFS is a better choice if the environment consists mainly of big files.

So far, based on the above analysis, answers to RQ1 and RQ2 can be provided.

Answer to RQ1 (*In which areas is HDFS superior to MSDFS and why?*) is: HDFS is superior on writing large, not fragmented files, due to its design optimized for handling large files and its write architecture as described in **Figure 6** - *Write Architecture (comparison HDFS vs. MSDFS)*.

Answer to RQ2 (*In which areas, on the contrary, is MSDFS superior to HDFS and why?*) is: MSDFS is superior on handling many small files (both reading and writing), due to its design and intended usage as part of the native Windows environment – see **Tables 10, 11** and **12**.

In the last part of the experiment, key usage scenarios were observed and sub-questions RQ3.1-3.4 were answered (see section *4.2.3 Part 3. Key usage scenarios*) in order to answer **RQ3** (*Which of the two systems is superior in the key usage scenarios observed, and why?)*.

In the **first scenario**, when a node goes unexpectedly offline, HDFS offers an intelligent way of recovering after the disappearing of a node, by re-replicating the missing blocks to the rest of the nodes in order to bring the replication factor back to the original value. MSDFS, on the other hand, is entirely "manual" in this field, meaning that the replication factor consists only of the number of nodes that exist in the cluster. If a node goes down, the replication factor decreases automatically by 1, and can only increase to the original value if the missing node reappears online or if a new node containing the same data is added to the cluster. For this scenario, **HDFS is superior** because it can automatically bring up the replication factor back to the original value, by making new copies of the under-replicated blocks.

In the **second scenario**, that tests the version control (concurrent access on the same file), the two systems behave quite differently. MSDFS offers editing options, but no concurrency support (changes of one user can overwrite those of another user, if the access is done at approximately the same time). HDFS only offers an append function (content can be added at the end of an existing file), but with protection to concurrent access, meaning that if two users try to append content at the same time, both will succeed (the order is not guaranteed) and no content will be lost. For this scenario, **each system has advantages over the other one, respectively**.

In the **third scenario**, that observes the flexibility of the replication schema, both systems have rather detailed possibilities of setting up how the data is replicated. MSDFS is probably easier to use, since it provides a GUI for this scenario, while HDFS requires the administrator to provide command line instructions. In this scenario, **MSDFS is superior** due to the greater variety of options and the GUI.

Answer to RQ3 (*Which of the two systems is superior in the key usage scenarios observed, and why?*): Depending on the scenario chosen, each system has advantages over the other one, respectively. It remains for the system administrator to decide which system better satisfies the needs, depending on the specific requirements of the users involved.

6 CONCLUSIONS

This thesis answered three main research questions in order to pinpoint which of the two distributed file systems (MSDFS and HDFS) is better in a number of scenarios (reading and writing few large files and many small files, “a node goes offline”, “two users open and modify the same file”, “an administrator configures the replication schema”).

A cluster of four computers alternatively running the two DFSs was used in order to create the above mentioned scenarios. After running the experiments and analyzing the results, the following **key findings** were produced.

HDFS is more efficient than MSDFS when handling large files. The experiment proved this conclusion through the measured times for the reading and writing operations for big files, which show that HDFS is two times faster than MSDFS on writing, and is as good as MSDFS on reading (see section *5 Analysis and answer to research questions* for direct references to experiment results).

As it is shown in the literature survey part of the paper, and the experiment confirmed this idea, HDFS is less flexible when handling small files, because it is by definition a file system optimized for “big data”, meaning that it operates with blocks of big dimensions (default is 128 MB) and is more efficient when transferring big files.

Moreover, HDFS is to be preferred in an environment where files do not need to be changed often (an append function is present in HDFS, but no random write function is available).

When it comes to MSDFS, it is better at handling many small files. The experiment showed that for HDFS it takes on average 38 percent longer time to read, and 40 percent longer time to write the same quantity of data as MSDFS, when the data is represented by small files. Moreover, MSDFS is more efficient when switching from not fragmented data to many small files representing the same quantity of data (the writing performance worsens by a factor of 5, which is much better than the factor of 12 reported in the same conditions for HDFS; for the reading performance, the difference between the two DFSs is smaller, but MSDFS is still the better system (3.20 performance worsening factor for MSDFS, 3.67 for HDFS).

An explanation for this result is that MSDFS runs on top of the native Windows platform, which is optimized for regular users, who usually have smaller files, which change more often compared to a “big data”-style platform where the files would tend to be larger and not change often.

In the key usage scenarios observed, both HDFS and MSDFS have advantages over the other system, respectively, depending on the intentions of the user. If the user prefers a GUI, then MSDFS is the better choice. If however the user prefers more flexibility in how the cluster deals with nodes going offline, HDFS is the better choice, as it can keep the replication factor unchanged by re-replicating the data that was present on the missing node.

7 FUTURE WORK

Future work to improve the research done in this paper could imply the usage of a more standardized benchmark, which would provide results easier to reproduce. This is however an ambition difficult to achieve in the near future, since a benchmark for both HDFS and MSDFS does not yet exist. Moreover, the MSDFS ecosystem is not rich in monitoring tools, still relying heavily on logging.

The use of a macro-benchmark that is optimized for the two DFSs could provide results that are more relevant for the file systems as a whole, and not only for specific areas or functionalities.

Other possible areas of work are experimenting with different hardware setups, larger clusters and bigger files, with a higher number of iterations on the measured operations. Adding other distributed file systems to the set of measured results would contribute more to the problem of choosing which file system to use in various scenarios.

Another possible area for future work is trying to define one common benchmark that is appropriate for the two DFSs, and perform experiments on the ease of setting up tests for the distributed file systems.

Until a more specialized benchmark will become available, future work could focus on comparing other aspects of the two distributed file systems, like scalability and security.

8 REFERENCES

8.1 Literature

Apache Software Foundation, 2014, *HDFS High Availability Using the Quorum Journal Manager*, Retrieved on March 7th, 2015, Last Updated on February 11, 2014, http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithQJM.html.

Apache Software Foundation, 2013, *HDFS Architecture Guide*, Last Published August 4th, 2013, Retrieved on May 16, 2015, http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

Bancroft, M., Bear, N., Finlayson, R.H., Isicoff, R., Thompson, H., 2000, *Functionality and Performance Evaluation of File Systems for Storage Area Networks (SAN),* Storage Technologies Knowledge Based Center, Department of Defense, In the Eighth NASA Goddard Conference on Mass Storage Systems and Technologies, 2000, http://www.ptpnow.com/sanresearch/funct_wp.pdf.

Bandulet C., 2009, *The Evolution of File Systems – How the digital world manages its data*, SNIA Education, Storage Networking Industry Association, http://www.snia-europe.org/objects_store/Christian_Bandulet_SNIATutorial%20Basics_EvolutionFileSystems.pdf.

Borthakur, D., 2007, *The Hadoop Distributed File System: Architecture and Design*, in Hadoop Project Website, Vol. 11, 2007, pp. 21, https://svn.eu.apache.org/repos/asf/hadoop/common/tags/release-0.16.3/docs/hdfs_design.pdf

Business Insider, 2013, *The 10 Most Important Companies in Cloud Computing,* Published on April 20, 2013, Retrieved on May 16, 2015, http://www.businessinsider.com/10-most-important-in-cloud-computing-2013-4?op=1&IR=T

Bzoch, P., Safarik, J., 2011, *Security and reliability of distributed file systems*, Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS'2011, vol. 2, pp. 764-769, http://ieeexplore.ieee.org.miman.bib.bth.se/xpl/articleDetails.jsp?tp=&arnumber=6072873&queryText%3DDistributed+File+Systems+.LB.DFS.RB.

Chow, R., 1997, *Chapter 6: Distributed File Systems* in "Distributed Operating Systems and Algorithms", Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, http://www.cise.ufl.edu/~chow/sl6.pdf (chapter outline).

Fu, S.; He, L.; Huang, C.; Liao, X.; Li, K., 2014, *Performance Optimization for Managing Massive Numbers of Small Files in Distributed File Systems*, IEEE Transactions on Parallel and Distributed Systems, Vol. PP, Issue 99, pp. 1, http://ieeexplore.ieee.org.miman.bib.bth.se/xpl/articleDetails.jsp?tp=&arnumber=6977977&queryText%3Ddistributed+file+systems+performance.

InfoWorld, 2007, *Best of open source in storage*, Retrieved on April 27th, 2015, Published on September, 10, 2007, http://www.infoworld.com/article/2649738/infrastructure-storage/best-of-open-source-in-storage.html.

ISO/IEC 25010:2011, *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, Article nr. STD-913126, Released on March 1st, 2011, https://enav-sis-se.miman.bib.bth.se/Standard/?std=STD-913126.

Kerzner, M; Maniyam, S., 2014, *Hadoop Use Cases and Case Studies*, Chapter 10 in Hadoop Illuminated, published and updated continuously for the Apache Software Foundation community, Retrieved on March 7th, 2015, http://hadoopilluminated.com/hadoop_illuminated/index.html.

Khan, M.A.; Memon, Z.A.; Khan, S., 2012, *Highly Available Hadoop NameNode Architecture*, in 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT), pp.167-172, 26-28 Nov. 2012, http://ieeexplore.ieee.org.miman.bib.bth.se/xpls/abs_all.jsp?arnumber=6516346&tag=1.

Lee, M., Vellore, M., 2006, *Exploring the Distributed File System in Microsoft Windows Server 2003 R2*, in Dell Power Solutions, The Magazine for Direct Enterprise Solutions, pp. 87, printed in May 2006 in the USA by Dell Product Group, Dell Inc., www.dell.com/downloads/global/power/ps2q06-20050301-Lee.pdf.

Levy, E. & Silberschatz, A., 1990, *Distributed File Systems: Concepts and Examples*, Journal ACM Computing Surveys (CSUR), Vol. 22, Issue 4, Dec. 1990, pp. 321-374, ACM New York, NY, USA, http://dl.acm.org/citation.cfm?id=98169.

Mackey, G.; Sehrish, S.; Jun Wang*, 2009, Improving metadata management for small files in HDFS*, in CLUSTER '09. IEEE International Conference on Cluster Computing and Workshops, pp.1-4, Aug. 31 2009-Sept. 4 2009, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5289133.

Marinkovic, V. Z., Wong, T. K., Coimbatore, S., Makkar, C., Suresh, S., Marinov, B., Vogel, R., 2012, *Load sharing cluster file systems* (US Patent), https://www.google.com/patents/US8180747.

Pressman, Roger, 2010, *Software Engineering, A Practitioner's Approach*, 7th Edition, McGraw-Hill, http://highered.mheducation.com/sites/0073375977/information_center_view0/index.html.

Shafer, J.; Rixner, S.; Cox, A.L., 2010, *The Hadoop distributed filesystem: Balancing portability and performance*, 2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS), pp.122-133, 28-30 March 2010, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5452045.

Shvachko, K., Kuang, H., Radia, S., Chansler, R., 2010, *The Hadoop Distributed File System*, 26th IEEE Symposium on Mass Storage Systems and technologies, Yahoo!, Sunnyvale, pp. 1-10, May 2010, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5496972.

TechTarget, February 2012, *Single Point of Failure*, Retrieved on April 27th, 2015, Published in February, 2012, http://searchdatacenter.techtarget.com/definition/Single-point-of-failure-SPOF.

TechTarget, September 2005, *Load Balancing*, Retrieved on April 27th, 2015, Published in September, 2005, http://searchnetworking.techtarget.com/definition/load-balancing .

Traeger, A., Zadok, E., Joukov, N., and Wright, C. P. May 2008. *A nine year study of file system and storage benchmarking*.ACM Trans. Storage 4, 2, Article 5 (May 2008), 56 pages. DOI = 10.1145/ 1367829.1367831 http://doi.acm.org/10.1145/1367829.1367831

Traeger, A., Zadok, E., Miller, E., Long D., Oct. 2008, *Findings from the First Annual File and Storage Systems Benchmarking Workshop*, The Usenix Magazine, Oct. 2008, Vol. 33, Number 5, http://fsbench.filesystems.org/papers/fsbench-workshop1.ps.

Yongwei Wu; Feng Ye; Kang Chen; Weimin Zheng, *Modeling of Distributed File Systems for Practical Performance Analysis*, IEEE Transactions on Parallel and Distributed Systems, vol.25, no.1, pp.156-166, Jan. 2014, http://ieeexplore.ieee.org.miman.bib.bth.se/xpl/articleDetails.jsp?tp=&arnumber=6410316&queryText%3Ddistributed+file+systems+performance.

Weil, S.A., Brandt, S.A., Miller, E. L., Long, D.D.E., Maltzahn, C., 2006, *Ceph: A Scalable, High-Performance Distributed File System,* Proceedings of the 7th symposium on Operating systems design and implementation, USENIX Association, http://ceph.com/papers/weil-ceph-osdi06.pdf.

White, T., 2012, *Hadoop: The Definitive Guide*, 3rd Edition, O'Reilly Media / Yahoo Press, http://shop.oreilly.com/product/0636920021773.do.

8.2 Microsoft TechNet Library[1]

Microsoft TechNet, March 2003, *What is DFS?*, Retrieved on March 1st, 2015 on TechNet web platform, Updated on March 28, 2003, https://technet.microsoft.com/en-us/library/cc779627.aspx

Microsoft TechNet, August 2005, *Overview of the Distributed File System Solution in Microsoft Windows Server 2003 R2*, Retrieved on March 6th, 2015 on TechNet web platform, Updated on August 22, 2005, https://technet.microsoft.com/en-us/library/d3afe6ee-3083-4950-a093-8ab748651b76

Microsoft TechNet, July 2009, *Distributed File System*, Retrieved on March 2nd, 2015 on TechNet web platform, Updated on July 15, 2009, https://technet.microsoft.com/en-us/library/cc753479(v=ws.10).aspx

[1] Microsoft's web platform for resources and tools designed to help IT professionals.