Republic of Yemen

Ministry of Higher Education

&Scientific Research

Al-Rayan University

Faculty of Higher Studies

Republic of Yemen

Ministry of High Education

&Scientific Research

Al-Rayan University

Faculty of Higher Studies

ENHANCEMENT OF OPERATING SYSTEM DATA TRANSFER RATE FOR MANY SMALL FILES

Thesis submitted to Al-Rayan University to complete the requirements of obtaining a Master's degree in Information Technology

of

obtaining a Master's degree in Information Technology

By

Magid Abdulla Basmael

Magid Abdulla Basmael

Supervisor

Dr. Saeed Mohammed Baneamoon

Dr. Saeed Mohammed Baneamoon

2020 م / 1441 هـ

2021 م / 1442 هـ

Approval of the Proofreader

I certify that the master's dissertation titled **(ENHANCEMENT OF OPERATING SYSTEM DATA TRANSFER RATE FOR MANY SMALL FILES)** submitted by the student **Magid Abdulla Basmael** has been linguistically reviewed under my supervision and has become in scientific style and clear from linguistic errors and for that I sign.

Proofreader: Abdullah Amer Al-kathiri

Academic Title: Assistant Professor

University: Hadhramout University

Signature : [illegible]

Date: 25/6/2020

Approval of the Scientific Supervisor

I certify that the master's dissertation titled, **(ENHANCEMENT OF OPERATING SYSTEM DATA TRANSFER RATE FOR MANY SMALL FILES)** submitted by the student **Magid Abdulla Basmael** has been completed in all its stages under my supervision and so I nominate it for discussion.

Proofreader: **Associate Professor Dr. Saeed Mohammed Baneamoon**

Signature: [illegible]

Date: **30/6/2020**

The Discussion Committee Decision

Based on the decision of the President of the University No. (   ) in the year _______ regarding the nomination of the committee for discussing the master's thesis entitled **(ENHANCEMENT OF OPERATING SYSTEM DATA TRANSFER RATE FOR MANY SMALL FILES)** for the researcher **Magid Abdulla Basmael** We, the head of the discussion committee and its members, acknowledge that we have seen the aforementioned scientific thesis and we have discussed the student in its contents and what related to it.

Chairman of the Committee

Associate Professor Dr. Saeed Mohammed Baneamoon

Signature: ....[illegible]............

Committee member	**Committee member**
Associate Professor	**Assistant Professor**
Dr. Naziha Mohammed Al-idroos	**Dr.** Mohammed Abdullah Bamatraf
Signature: ......[illegible]......	**Signature:** ......[illegible]......

Acknowledgment

I would like to express my gratitude to my supervisor Dr. Saeed Mohammed Baneamoon for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore, I would like to thank my friends Fahd Bashen and Ali Basunbol for introducing me to the topic as well for the support on the way. Also, I would like to thank all persons, who have willingly shared me their precious time. I will be grateful forever for your love.

Enhancement of Operating System Data transfer Rate for Many Small Files

Abstract

Computer is a great machine, used in various fields, helps process much data and execute several tasks in a short time. One of the main and frequent tasks is the data transfer from a source to a destination. In the past, computer machines were slow and processed very little data. So, the impact of computer operating system's data transfer rate was not a big problem. Later, the demand for data has increased tremendously in all fields, and even the computer itself and its operating system perform many tasks internally. So, this increasing of usages and processes on data has generated some challenges to the operating system. The main problem is the effect of the operating system data transfer rate due to size changes and the number of files that would be transferred, specifically many small files. So, this problem has led to a time wastage and resources overhead.

The aim of this research is to highlight and analyze the problem of affecting data transfer rate of the operating system due to many small files transfer and provide a new proposed technique to improve it.

This research proposes a new technique to transfer data locally for many small files, based on the principle of merge and on-the-fly extraction without any modifications to the operating system architecture or file system hierarchy, in order to reduce very many operations that the operating system performs during the file transfer process. On the other hand, this research approach proposes an improvement in term of the data transfer rate locally for many small files, resulting to better operating system performance and thus an improvement of the user performance.

There are solutions that enhance small file's issues, but many of them are for distributed systems, and if not, no one is for "data transfer rate locally". The research approach depends on experiments, analysis, evaluation, and presents a technique that improves the operating system's data transfer rate locally for many small files from a source to a destination. The result of the new proposed technique proves that it enhances OS's data transfer rate of many small files and its efficiency.

We certify that we have read the present work and that in our opinion it is fully adequate in scope and quality as thesis towards the partial fulfillment of the Master Degree requirements in

Information Technology

From

College of Higher Studies

Date: 8/7/2021

<u>Supervisor:</u>

1. **Dr. Saeed Mohammed Baneamoon**
   Assistant Prof. of Department of Computer Engineering, College of Engineering & Petroleum, College of Computers & Information Technology, Hadhramout University

Signature:

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
B	Byte
b	Bit
CPU	Central Processing Unit
DTR	Data Transfer Rate
FFS	Fast File System
GB	Gigabyte
HDFS	Hadoop File System
HDD	Hard Disk Drive
IRP	Input/output Request Packet
KB	Kilobyte
MB	Megabyte
OS	Operating System
PC	Personal Computer
RAM	Random Access Memory
SRB	Storage Request Block

CHAPTER ONE: INTRODUCTION

1.1 Introduction

In the early digital computing, people were penetrating with a few amounts of data, and there were less problems affecting data transfer. Later, requesting data drastically increased.

Technology moved on, so speeds and volumes spread vertically and horizontally. In modern technology, speeds and volumes are beyond imagination. The average annual personal computer data traffic worldwide from 2015 to 2022 would raise up from 0.4 to 1.3 exabytes [1], and based on Statcounter, the desktop Windows 10's version market share worldwide is 70.98% [2].

A search was done on Windows 10's "C:\" home volume, there are 414142 files up to 256KB. So, it can be said, that even the operating system contains internally many small files to work properly.

On a PC, when transferring big-size files locally from a source to a destination, the data transfer rate is adequately fine. In contrast, if many small-size files are transferred, the operating system data transfer rate is so negatively affected [3].

There are contributions that have improved big data transfer rates on both, personal computer and distributed architecture. Hence, this thesis will focus on the challenge of operating system data transfer rate for small files in order to enhancement the performance of operating system.

1.2 Motivation

Even on current computers, there is a lack in the operating system data transfer rate when many small files are transferred locally from a source to a destination. In such a case, this means a time delay for users. Also, the operating system itself contains many small files to work properly, and all of those may overhead system resources. Also, all previous works focus only on distributed systems, and does not focus on local systems. So, this problem really needs to be dug into the reasons behind, and try to contribute some solution for this interesting thing.

1.3 Problem Statement

Changes of files' sizes affect the operating system data transfer rate. Therefore, when transferring many small files locally from any source to any destination using a PC, the operating system does not function optimally with the data transfer rate. There have been contributions to reducing this problem and improving performance, but they are very few in the area of personal computer operating systems and need more improvements.

In the past, people used to deal with little data and the problem was not apparently negatively affected for the operating system. But later, as a technology performance consequence, the number of personal computers increased very quickly and spread to be used for different purposes, hence the data usage increased, and this resulted in too much data. Among these data, there are many small files, the challenge in which poor handling by the operating system leads to time waste and resources overburden.

The general purpose of this research is to find a way to improve the operating system's data transfer rate for many small files locally without straining system resources. Also, the root of the problem will be tracked to see the possible causes underneath. Statistical and experimental method will be used, by appropriate tools to accomplish the experiments to conduct evaluation and comparison.

1.4 Objectives

In this thesis, a new proposed technique will be used as a development of data transfer rate of PC's operating system, as well as the analysis of the operating system mechanism of transferring files to determine the weakness reasons when transferring many small data locally.

This research addresses some issues in software development. The first issue is analysing the problem of weaknesses of data transfer rate of the operating system for many small data. The second issue is unsuitability of both techniques all the time, neither the proposed technique nor the operating system technique. Therefore, the objectives of the research relate to improving the efficiency of data transfer rate for small-size files in operating system. In more detail, the objectives are:

1) To propose a new technique that will improve Operating Systems data transfer rate of the small size files without system resources overburden.
2) To suggest an algorithm that decides - before transfer process - which technique is suitable to be applied to files being transferred, the proposed data transfer technique or using the traditional Operating System's technique to benefit the appropriate mechanism after decision.

1.5 Scope and Limitations

The research is specific for the Operating Systems' data transfer rate locally. Any other environments - even in many cases, the problem does exist - are out of scope, such as cloud computing, distributed systems or any other networking environments. It aims to the improvement of data transfer rate of small files less than 1MB and applied on Windows 10's OS.

1.6 Research Approach

In order to carry out the new proposed technique, research approach of this thesis will be followed at stages as the flowchart in Figure 1.1.

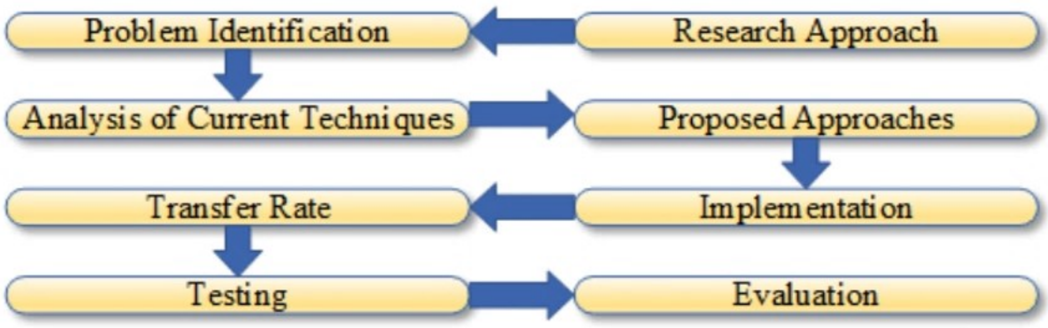


Figure 1.1 Flowchart of Research Approach

1.6.1 Problem Identification

At this stage, the problem of the weaknesses of the operating system data transfer rate locally for small files will be identified clearly, addressing the root cause of the problem and including the problem's effects on businesses.

1.6.2 Analysis of Current Techniques

This stage is to put this research on a right place. So, current techniques that present enhancement of data transfer rate of an operating system locally will be analysed.

1.6.3 Proposed Approaches

In order to introduce a solution for enhancing operating system's software weaknesses when transferring many small data, this stage will introduce a demonstration of a new proposed technique to reduce the reasons impact for this vulnerability and improve the performance of the operating system data transfer rate for many small files.

1.6.4 Implementation

This stage will detail the implementation steps for accomplishment of experiments of the new proposed technique to achieve the objectives of the research. So, quantitative approach is introduced by using various tools for experiments test, analysis and evaluation.

1.6.5 Transfer Rate

Transfer rate is quantity that is transferred repeatedly from a source to a destination in a period of time. In this research, it is the files that are transferred every second. The transfer rate of the operating system for many small files is still weak. Therefore, this stage is to design a model for enhancing the issue of many small files data transfer rate poor efficiency of an operating system.

1.6.6 Testing

At this stage, the new model will be tested on groups of small files to determine its efficiency and the file size range it operates on.

1.6.7 Evaluation

The operating system data transfer mechanism is fine for big data. But it has software weaknesses in case of small files, because it does not have enhanced mechanisms to minimize the many operations of a file transfer processes, neither minimizing the operations themselves for one file nor minimizing the operations for many files. So, at this stage, the new proposed technique will be evaluated and compared to other similar proposed techniques, presenting their strengths and weaknesses.

1.7 Research Contributions

Computer data transfer rate is critical, so enhancing it is a challenge either for local environments or distributed environments. So, the thesis will introduce the following contributions:

1) The thesis is considered as an opportunity to present the weaknesses of the data transfer rate mechanisms of a computer operating system.
2) The optimization for operating system data transfer rate is simple, efficient, reduces time and increases productivity.
3) The use of the suggested algorithm that combine both techniques the proposed technique and the operating system technique based on file sizes before starting the transfer process leads to more flexibility.

1.8 Structure of Thesis

The thesis is organized into five chapters. Chapter 2 introduces a whole idea of impact of sizes change and many small data on the transfer rate of the operating system. Chapter 3 provides an overview of the previous researches' contributions related to the problem stated. Chapter 4 presents the experiments, analysis and evaluation. Chapter 5 introduces the conclusion, limitation and future work.

CHAPTER TWO: BACKGROUND

2.1 Introduction

In regular daily life, and whatever device used, whether it is a computer, a mobile, or other, data is often transferred locally from a place to another. For example, the process of copying files from a folder to another folder, from a volume to another or even from a hard disk to a USB storage device, etc. When transferring files locally from one place to another using a computer, it takes time, which may be very short and may be very long based on files' sizes and number. For example, using a PC, when transferring 20 files from volume D to a USB storage device, if the total size of the files is 2 megabytes, and the time spent in the transfer process is 20 seconds, then the data transfer rate in this case could be expressed in two terms as follows:

1-Data transfer rate = 2MB ÷ 20 seconds = 0.1MB/s.

So, in this case, every second it transfers 0.1 megabytes.

2-File transfer rate = 20 files ÷ 20 seconds = 1file/second.

It means that every second, one file is transferred, and this term will be used because it is appropriate for this thesis.

Because the operating system data transfer rate is affected due to many transferred small files, so, it implies that the "file transfer rate" changes according to the change of their sizes and number, even if the data transfer rate of the PC has the same value. So, the time of transferring 20 files the total size of them is 2000 KB differs from the time of transferring 40 files the total size of them is 2000 KB even if the PC data transfer rate is the same [3].

2.2 Operating System Input/Output Request

To use a computer easily and correctly, it is powered by a so-called an "operating system". The operating system is a software which acts as an interface between the end user and computer hardware as shown in Figure 2.1. So, to access resources of a computer to use them for various tasks easily without errors, such as transferring data among storage devices, printing a paper on printing devices or any other task resource, the operating system carries out those tasks based on the user's operation instructions. Most of the operations are hidden from the user, so the user executes various tasks seamlessly without having to know or learn complicated details of software [4].

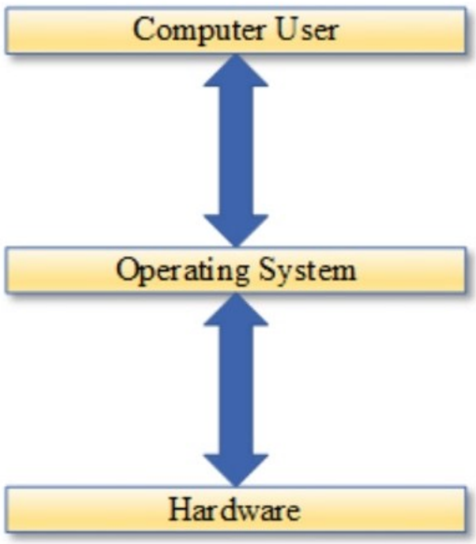


Figure 2.1 Operating System

To use the computer resources seamlessly, the computer operating system is divided into interfaces as shown in Figure 2.2, each of which provides operations from/to upper/lower interface. Application interface exists between the end user and Call interface. It provides simple data and information, and easy graphics shown on the screen to the end user as a mouse pointer, menus and windows, and isolates the user from complicated details of software that helps the resources to operate correctly [5].

Call interface sits between Application interface and Service interface. It provides an interface to the services made available by an operating system. So, it is nothing else than a "call" to execute a corresponding service/services for a task accomplishment coming from Application interface instructed by the end user [5].

Service interface is placed between Call interface and the hardware (resources). It is considered low-level because it is closer to the resources. Think of Service interface as an executive manager that executes the operation instructions coming from Call interface through the resources [5].

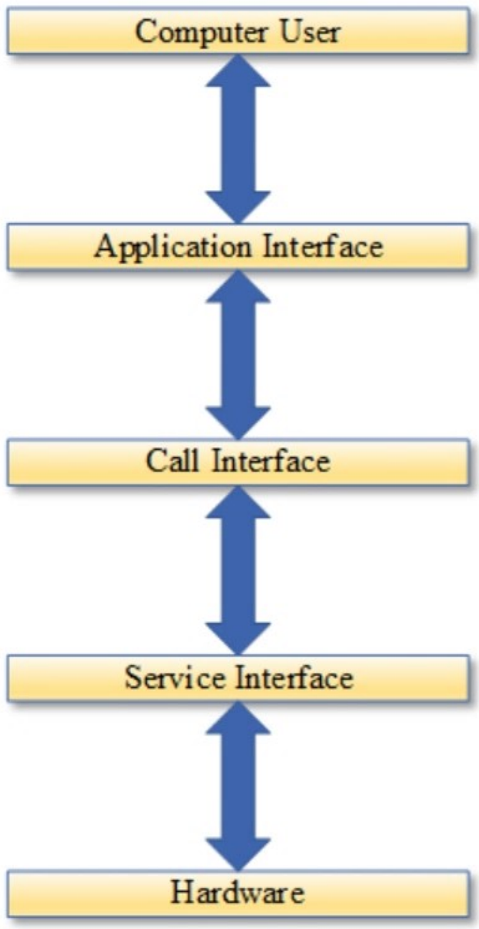


Figure 2.2 Operating System's Interfaces

From above, it can be seen, that when a user executes a task by a computer, the task operations must pass through all of the OS's interfaces. For example, to transfer data, a user must use an Application interface such as a "copy data" interface from a source and a "paste data" interface to a destination, and consequently, the operating system is instructed to execute the data transfer task. So, the Application interface picks up the appropriate "call/calls" from Call interface. In the example, "File Management Call, "Device Management Call" and/or other calls are directed to the Service interface, which in turn executes the data transfer logically by the appropriate service/services to. In the example, "File Management Service" is used to instruct the resources (source and destination) to execute the data transfer physically from one to another until complete.

Input/output processes are essential and complicated operations of an operating system. They are processed by so-called "IRP" requests, which are I/O Manager responsibility in case of Windows OS [6]. So, from Figure 2.3, it can be seen how many and complex the I/O operations. Also, it shows that only an "open a file" task I/O request passes through ten main phases.

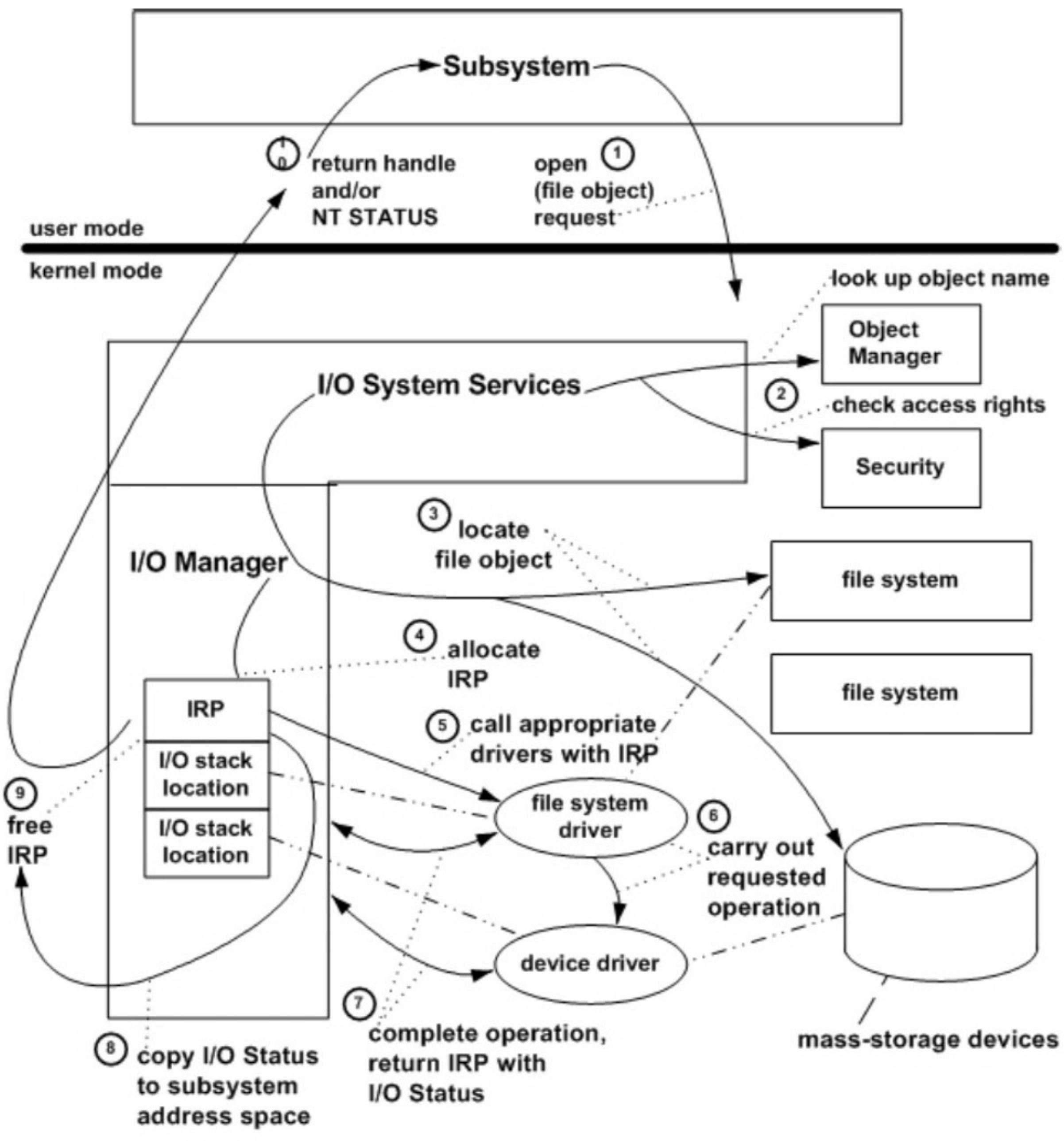


Figure 2.3 I/O's Operations of an "Open a file" Task

From above, it can be seen that any task would be executed using a computer, it must pass through many interfaces forth and back. This is called "layered OS" approach [5].

To see the negative effects of the layered approach of the operating system because of many interfaces and operations, a file transfer from a source to a destination is a sample task, which is now viewed from the perspective of only the system calls which have been demonstrated in this section. From Figure 2.4, it can be seen ten system calls to execute a file transfer task from a source to a destination. These are not all calls, they are for demonstration purposes, and in the real world, there are many calls [5].

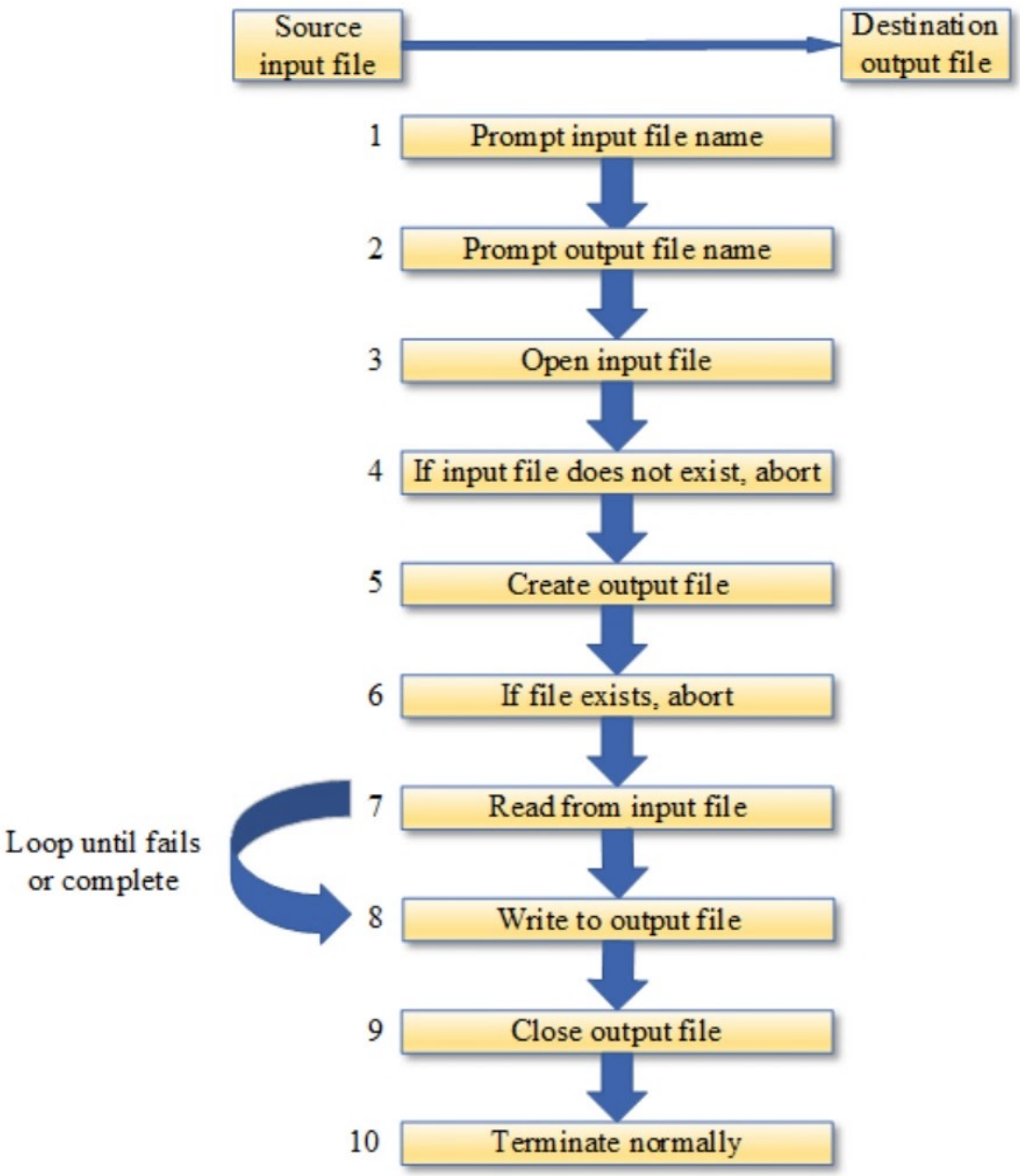


Figure 2.4 System Calls for a Data Transfer Task

2.3 Operating System Data Transfer Execution

The behavior of change of file transfer rate of the operating system locally according to change of file size and number is a challenge, and there must be something behind [3]. This will be researched and the origin of this problem will be investigated more. This will be achieved by tracking how the file is transferred from a source to a destination logically and physically within the operating system.

To successfully perform a file transfer task, it goes through three (often considered two, but in this research, it will be considered three for thesis purposes) main steps by the operating system called "modes" [7], as shown in Figure 2.5:

1) User Mode.
2) Kernel mod.
3) Physical mod.

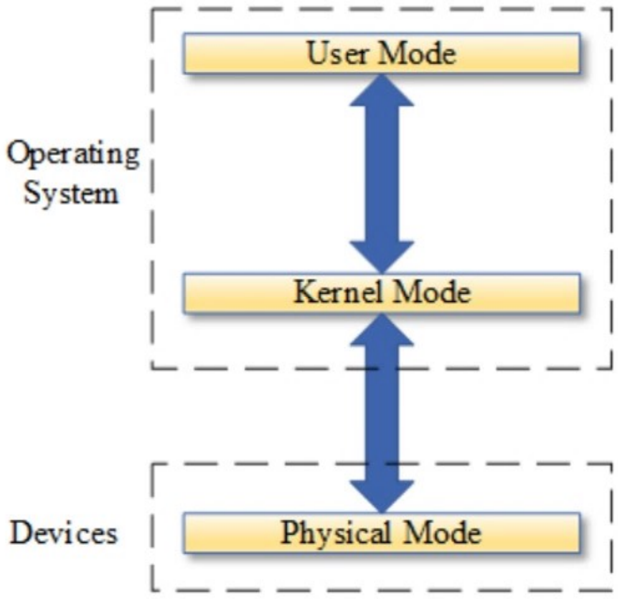


Figure 2.5 Three Modes of a Device I/O

For example, if a user is transferring a file from a source to a destination, the user interacts simply only with the user mode (i.e. file copy/paste process interfaces). The rest of the file transfer task processes is hidden and handled automatically by the kernel mode and the physical mode without user interference. The physical mode performs the processes to store data physically. The kernel mode is intermediate mode between the user mode and the physical mode. In the case of file transfer, the kernel mode transforms the file transfer operation instructions (i.e. file copy/paste operation instructions) from the user with the help of the user mode to the storage device with the help of the physical mode and monitors the file transfer process until successful file transfer completion [7].

Furthermore, each of those modes has more intermediate layers. Part of them is to deal with the data and the rest is for control. So, the more those control operations increase, the more the time delay they waste and the more the resources they consume [3].

Figure 2.6 shows that the operating system communicates to the device via a so-called a “driver”. The driver is closer to the device and is used to instruct the device physically, according to conditions and criteria specified by an operating system. For example, when a file is copied from a hard disk drive to a USB storage device, the user uses the appropriate application interface for that. Then, the operating system communicates with the storage device via the appropriate device driver through appropriate system calls and services. So, the driver is often considered a low-level, closest to the physical layer [5] [8].

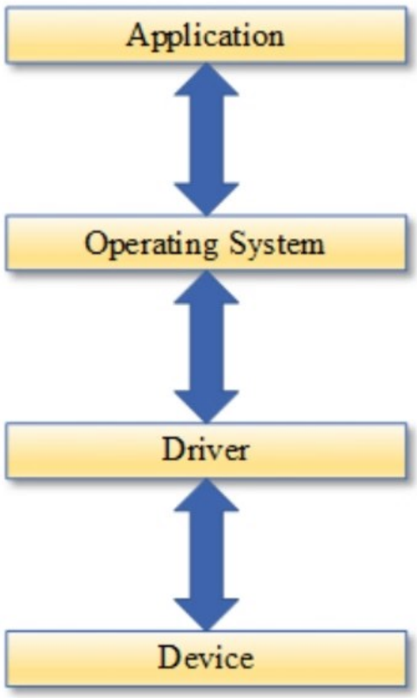


Figure 2.6 Interaction Between Operating System and Device

If Figure 2.6 is more analyzed, the process becomes more complicated as shown in Figure 2.7. It is shown that the driver consists of parts, each of which performs a specific operation [8].

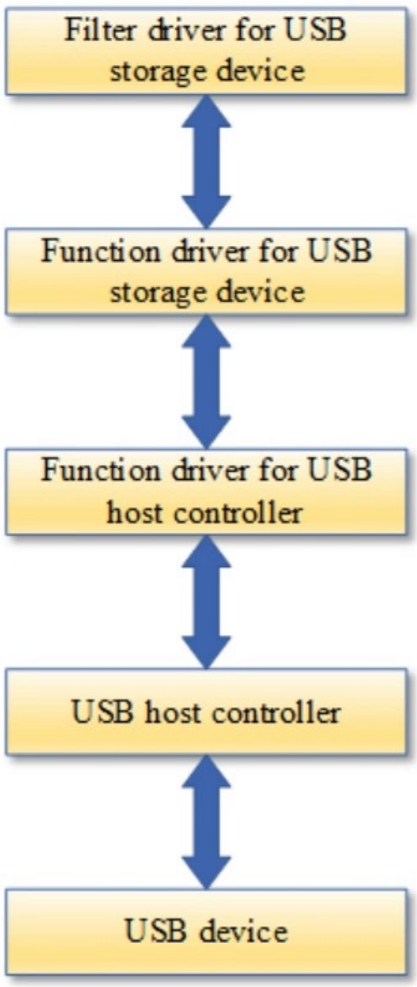

Filter driver for USB storage device
Function driver for USB storage device
Function driver for USB host controller
USB host controller
USB device

Figure 2.7 Low-level Interactions of a Device

So, in this research, the approximate number of file transfer operations locally will be presented to ensure that the more those operations, definitely leads to more time delay and resources consumption.

2.4 Counting OS's Operations of File Transfer

This research will show the effect of change of size and number of transferred files on the operating system's data transfer rate. So, this research will track the operations that the operating system performs to transfer one file locally from a source to a destination to count them. To do that, Windows OS is chosen for its popularity [2]. Unfortunately, Windows OS has a hurdle, it is closed source system and dealing with it is somewhat difficult [5].

The instantaneous input/output operations of data being transferred from a source to a destination are controlled by 6 layers as shown in Figure 2.8. When a file is to be transferred locally from a source to a destination, a computer is instructed by a simple Application interface to select a file which a source contains, and a destination to which the selected file would be transferred. Then, Application interface provides the appropriate operation instructions of transferring the file to the API interface to provide the appropriate file transfer calls to the lower phase. After System Service interface receives the appropriate file transfer calls, it provides the appropriate file transfer service to the lower phase. In this phase, I/O Manager based on the appropriate file transfer system service provides the appropriate file transfer IRP packets to the Device driver, which in turn provides appropriate operation instructions for a device storage to HAL phase which provides the appropriate operation instructions for a specified transfer bus to transfer the file bits physically from the source and store them on the destination until complete successfully [9].

As described previously, it can be seen, any task, even a small task an operating system executes, it passes through many phases. So, in this research, the file transfer task operations are divided into read operations and write operations to try to count the total operations accurately.

So, to count approximately the number of operations of one file transfer, attention is paid to track the operations from the user mode to the physical mode.

To transfer a file from a source to a destination successfully, Windows OS performs many read/write operations from/to storage devices. To do that, It provides Application Programming Interface to invoke the appropriate calls of transferring a file [5].

A "read" operation can be performed by calling ReadFile and a "write" operation can be performed by calling WriteFile [10].

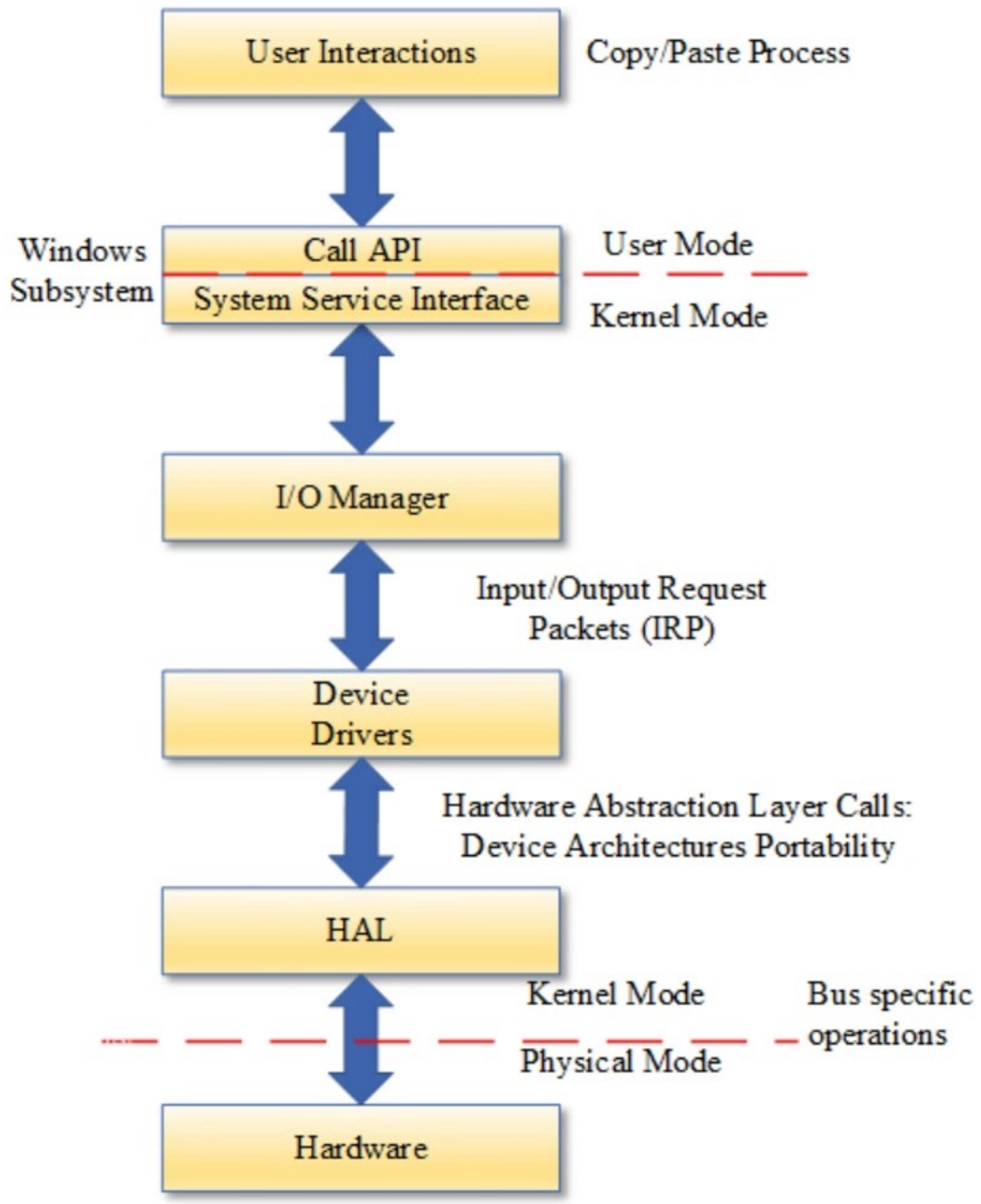


Figure 2.8 Complete Communication Between User and Storage Device

In the user mode, there are 3 calls as follows [10]:

1) WriteFile.
2) ntWriteFile.
3) KiFastSystemCall.

In the kernel mode, there are 4 calls as follows [10]:

1) KiFastSystemCall.

2) ntWriteFile.
3) IopSynchronousServiceTail.
4) IofCallDriver.

In the physical or low-level mode, there are 5 calls as follows [10]:

1) IRP for upper-filter driver.
2) IRP for storage class driver.
3) SRB for lower-filter driver.
4) SRB for storage port driver.
5) SRB for bus-specific command.

In fact, there are more multiple operations that are made between the user mode and the kernel mode before the file is written to a storage device. WriteFile call contains 12 operations. ntWriteFile call contains 9 operations. KiSystemService call contains 5 operations. IopSynchronousServiceTail call contains 6 operations. IofCallDriver call contains 1 operation [10]. Added sysenter and sysexit [11] operations which help enter/exit to/from the user mode from /to the kernel mode.

All operations that concern one successful file transfer task from a source to a destination are calculated. Table 2.1 shows the approximate number of write's operations.

Table 2.1 Number of Operations of One File Transfer Task

Call per mode		Number of operations
user	WriteFile	13
	ntWriteFile	10
	KiFastSystemCall (sysenter)	2
	Sysexit	2

Table 2.1 Number of Operations of One File Transfer Task (Continued)

kernel	Sysexit	6
	KiSystemService	6
	IopSynchronousServiceTail	7
	IofCallDriver	2
low level	IRP for upper-filter driver	1
	IRP for storage class driver	1
	SRB for lower-filter driver	1
	SRB for storage port driver	1
	SRB for bus-specific command	1
Total Write's operations		53

There are read's operations too and must not be omitted. Supposing the read's operations number is equal to the write's operations number, then, the total read/write number is 106 operations.

So, one successful file transfer task locally from a source to a destination needs to approximately 106 operations. For example, if we want to transfer 2000 files locally from a source to a destination, the operating system will have to execute 212000 operations to complete the entire files transfer task successfully.

Figure 2.9 shows an overview of the components of Windows OS. It can be seen, that the operating system contains many components, each of which contains more internal subcomponents. So, In the real world, each task is executed in a computer from any source to any other source, such as transferring data locally from a storage device to another one, passes through many Input/Output (I/O's) operations by the operating system [12].

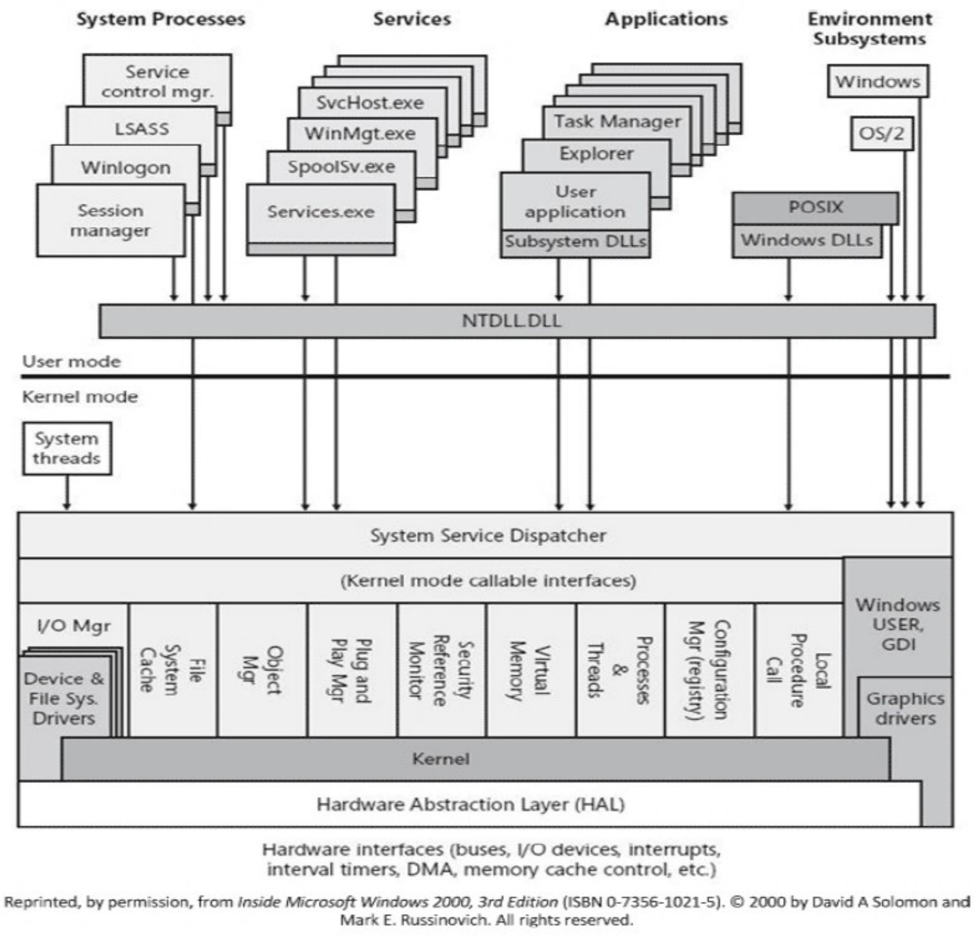

System Processes
Services
Applications
Environment Subsystems
Service control mgr.
LSASS
Winlogon
Session manager
SvcHost.exe
WinMgt.exe
SpoolSv.exe
Services.exe
Task Manager
Explorer
User application
Subsystem DLLs
Windows
OS/2
POSIX
Windows DLLs
NTDLL.DLL
User mode
Kernel mode
System threads
System Service Dispatcher
(Kernel mode callable interfaces)
I/O Mgr
Device & File Sys. Drivers
File System Cache
Object Mgr
Plug and Play Mgr
Security Reference Monitor
Virtual Memory
Processes & Threads
Configuration Mgr (registry)
Local Procedure Call
Windows USER, GDI
Graphics drivers
Kernel
Hardware Abstraction Layer (HAL)
Hardware interfaces (buses, I/O devices, interrupts, interval timers, DMA, memory cache control, etc.)
Reprinted, by permission, from Inside Microsoft Windows 2000, 3rd Edition (ISBN 0-7356-1021-5). © 2000 by David A Solomon and Mark E. Russinovich. All rights reserved.

Figure 2.9 Windows OS Components [12]

This is a so-called "layered OS" approach, in which the operating system is divided into a number of discrete layers [5].

2.5 Chapter Summary

To transfer a file locally from a source to a destination, a computer operating system such as Windows OS executes many read/write operations. So, in case of many small files, the number of read/write operations increases bigger and bigger leading to a wasted time.

On the other hand, this implies that many initial operations per many new small files being transferred in a very short time will lead to not only time delay, but system's resources consumption too. Also, some parts of the operating system itself is made up of many small files that may affect its data transfer rate.

Most operating systems depend on the layered approach. Although a layered approach software has had some success, it is generally not ideal for designing operating systems due to performance problems.

So, the main cause behind the problem of weaknesses of the operating system data transfer rate of many small files, is that a complete file transfer task from a source to a destination passes through many read/write operations.

Unfortunately, due to the requirements that increase according to challenges such as improving protection of the transfer process, etc., the operating system pays less if no attention to the data transfer rate speed.

CHAPTER THREE: LITERATURE REVEIW

3.1 Introduction

Weakness of operating system data transfer rate of small files is a challenge. There are many contributions state-of-the-art to enhance the problem in the literature. Those contributions vary upon perspective view of the problem reasons and environments. Some focused on local environment or distributed environment. Some contributions focused on physical structure or logical structure and some others focused on both. On the other hand, there are contributions enhanced the hard disk drive, others enhanced the main memory (RAM) and some others enhanced the internal build of storing. Also, there are contributions enhanced the problem by adding extra hardware.

3.2 Related Works

The following review summarizes the previous works related to the thesis:

- Ahn et al. (2009) proposed a scheme called "A multiple-file write scheme for improving write performance of small files in Fast File System" [13].

The scheme basic idea is to collect large numbers of modified small files in a buffer cache, then write the data to disk in large disk I/O's to improve the performance of small file writes.

- Lensing et al. (2010) proposed an approach called "hashFS: Applying Hashing to Optimize File Systems for Small File Reads" [14].

The approach depends on "hash" approach to hash pathname approach to increase small file read performance for a workload typical in Web 2.0 scenarios and does not rely on a name- based or temporal locality or large in-memory lookup tables. A single small file read is performed with a single seek nearly independent of the organization and size of the file set or the available cache.

- Dong et al. (2012) contributed an approach named "An optimized approach for storing and accessing small files on cloud storage" [15].

The approach improves the memory available on NameNode limitation of HDFS scalability. Also, it classifies files into three types: structurally-related files, logically-related files, and independent files. File merging strategy and file grouping strategy are adopted. Also, it adds other adaptations.

- Fu et al. (2013) work is “iFlatLFS: Performance Optimization for Accessing Massive Small Files” [16].

The work presents a file system called iFlatLFS. iFlatLFS directly accesses raw disks and adopts a simple metadata scheme and a flat storage architecture to manage many small files and each file access needs only one disk operation except when updating files.

- Zhang et al. (2014) proposed a method named “A Strategy to Deal with Mass Small Files in HDFS” [17].

The method proposes a merging and prefetching mechanism in order to deal with the inefficiency for middle size small file. It introduces different solutions for different file size distributions and file formats.

- Tao et al. (2014) proposed a strategy named “Small File Access Optimization Based on GlusterFS” [18].

The strategy aims at optimizing small file access performance in distributed file system. This redesign structure of file metadata, minimized its size and merged small file into large file.

- Gupta et al. (2015) proposed a framework called “An extended HDFS with an AVATAR NODE to handle both small files and to eliminate single point of failure” [19].

The framework improves the file correlation analysis for prefetching and merging to the existing combined files.

- Bende et al. (2016) proposed a method called "Dealing with Small Files Problem in Hadoop Distributed File System" [20].

The method introduces CombineFileInputFormat that gives performance in terms of overhead as it involves slight overhead by combining multiple files into single split and increases reading efficiency of small files.

- Jing et al. (2016) proposed a method named "An Optimized Approach for Storing Small Files on HDFS-based on Dynamic Queue" [21].

The method contrary to the problem of HDFS which is poor in dealing with many small text files generated by network. It classifies the text files with period classification algorithm, chooses suitable queue for different size of small files, then merges the files in queue, stores merged files to save memory, then generates secondary index and uses files prefetching strategy to improve the efficiency of access files.

- Bok et al. (2017) proposed a scheme called "An Efficient Cache Management Scheme for Accessing Small Files in Distributed File Systems" [22].

The scheme is proposed for the distributed cache management that applied cache metadata synchronization cycle to improve small file access speed and minimize network load with NameNodes in HDFS. In addition, the proposed scheme maintains NameNode block metadata and cache metadata in the client cache, reducing unnecessary file accesses. In short, the scheme uses metadata and cache in both server and, if the file is in the client, there is no need for file access. If the file is not in the client, then the scheme accesses firstly the file in the NameNode cache, if not, the scheme will access the file in the NameNode.

- Cheng et al. (2017) proposed a scheme called "Optimizing Small File Storage Process of the HDFS Which Based on the Indexing Mechanism" [23].

The scheme proposes a small file merging scheme based on indexing mechanism. The proposal improves file access and executes the merging job by filtering the files with two parameters; file-type and the amount of storage space required by the file, and then encrypted before the files are passed onto HDFS.

- Lyu et al. (2017) proposed a strategy called "An Optimized Strategy for Small Files Storing and Accessing in HDFS" [24].

The strategy uses three aspects: 1) an optimized merging algorithm based on the size of small files, which reduces the memory usage on NameNode. 2) A mapping file to quickly locate the target file. 3) Using prefetching and caching for improving the reading speed.

- Ahad et al. (2018) proposed an approach named "Dynamic Merging based Small File Storage (DM-SFS) Architecture for Efficiently Storing Small Size Files in Hadoop" [25].

The approach proposes a two-way approach for effectively storing different files. The first part deals with identifying the size of the files, while the second part deals with classifying the files on their type. using DFM algorithm. Similar types of small files are merged together. Furthermore, the contents of the files are encrypted.

- Peng et al. (2018) proposed a model named "Hadoop Massive Small File Merging Technology Based on Visiting Hot-Spot and Associated File Optimization" [26].

The model adds two modules: merging module and caching module. In merging module, a set of correlated files is combined, as identified by the client, into a single large file. In caching module, design a special memory subsystem in which some data are duplicated for quick access.

- Matri et al. (2018) proposed a model called "TyrFS: Increasing Small Files Access Performance with Dynamic Metadata Replication" [27].

The model enables clients to locate any piece of data in the cluster independently of any metadata server. A dynamic metadata replication module adapts to the workload to efficiently replicate the necessary metadata on nodes from which the associated data is frequently accessed. Clients can then read frequently accessed data directly, without involving additional servers.

- Xiong et al. (2019) proposed a strategy named “A Small File Merging Strategy for Spatiotemporal Data in Smart Health” [28].

The strategy proposed a merging strategy for small spatiotemporal data files in smart health. This method takes advantage of the spatiotemporal locality and related of user access, it improves the efficiency of file reading and reduce user access delay.

- Tao et al. (2019) proposed an approach named “LHF: A New Packet based Approach to Accelerate Massive Small Files Access Performance in HDFS” [29].

In this approach, small files problem of HDFS is improved and the files to be merged do not need to be sorted, which makes appending additional files to existing merged file easier.

3.3 Critical Evaluation of Existing Approaches

The review of current literature of data transfer rate in operating system has led to the identification of certain limitations that require further exploration in the thesis.

- Ahn et al. (2009) [13]. The scheme is based on modifying a file system hierarchy to deal with only the files that their contents are 12 blocks (1 block equals 64B) and less.

- Lensing et al. (2010) [14]. The approach enhanced only file read process, modify the file system and directed to the web environment.

- Dong et al. (2012) [15]. The approach is for a distributed system and has a set of many techniques that are sophisticated and might add time delay.

- Fu et al. (2013) [16]. This scheme is for 1KB to 64KB files, depends on modifying a file system and directed to a distributed architecture.

- Zhang et al. (2014) [17]. This method uses an additional hardware (store) and is distributed system oriented.

- Tao et al. (2014) [18]. The approach uses file merging, by file metadata, ignores the file sizes above 50KB to 1MB and is for distributed systems.

- Gupta et al. (2015) [19]. This framework is a file format specific, adds RAM overheads and is for distributed architecture.

- Bende et al. (2016) [20]. The mechanism adds an additional node layer for virtual IP's, does not address the efficiency and is for distributed system.

- Jing et al. (2016) [21]. The method is only for txt files and server based. It stores the merged files in the server RAM just to speed access. In such a case, this design in addition it depends on many processes to be accomplished, it will fill up RAM as soon as many files are generated from the small merged files.

- Bok et al. (2017) [22]. The scheme is for file access and a distributed system, ignores the files data problem, does not utilize merging technique and uses volatile memory that means data cache loss if memory is down.

- Cheng et al. (2017) [23]. This proposal is distributed environment dependent, improves file access and is surrounded by parameters.

- Lyu et al. (2017) [24]. The proposal improves memory usage, file reading and is for a distributed system.

- Ahad et al. (2018) [25]. The proposal has techniques which are so hard, performs a modification on a file system and is for distributed architecture.

- Peng et al. (2018) [26]. This model is for a distributed system, uses extra hardware and merge the files correlated and identified by user.

- Matri et al. (2018) [27]. This model improves metadata, only file access, does not use merging technique and is for distributed system.

- Xiong et al. (2019) [28]. This strategy is for health spatiotemporal sensors, file reading and access.

- Tao et al. (2019) [29]. This approach is for file access not file transfer and is distributed specific.

Table 3.1 summarizes the strengths and limitations of the previous works related to the thesis.

Table 2.2 Critical Analysis of Relevant Approaches

No.	Author/Year	Method	Strength	Limitation
1	Ahn et al./2009	Collect small files in a buffer, then write them in large disk I/O's	Take advantage of the buffer space well	Involves files that only contain up to 12 blocks for only a single file system.
2	Lensing et al./2010	Depends on hash function for pathname	Reduces file read delay	For only file read, single file system and web.

Table 3.1 Critical Analysis of Relevant Approaches (Continued)

3	Dong et al./2012	Improves HDFS NameNode memory	Classifying files into three types leads to more enhancement	for a distributed system and has set of many sophisticated techniques
4	Fu et al./2013	Directly accesses raw disks	Each file access needs only one disk operation	Only for file access process and distributed system
5	Zhang et al./2014	Merging and prefetching mechanism	Introduces different solutions for different file size and file format	Adds extra hardware and for distributed system
6	Tao et al./2014	Redesigns file metadata structure using merging	Optimizes small file access performance	Ignores file sizes above 50KB and for distributed system
7	Gupta et al./2015	Uses system RAM analysis for prefetching and merging	Using RAM for prefetching and merging is much faster than using disk	The more the files, the larger the RAM size and overhead, it is for distributed system

Table 3.1 Critical Analysis of Relevant Approaches (Continued)

8	Bende et al./2016	Combining multiple files into single split	Increases reading efficiency	For only file read process and distributed system
9	Jing et al./2016	Classifies the text files, uses merging and prefetching	Increases file access efficiency	Only for .txt files and server system
10	Bok et al./2017	Serially Checks meta data for accessed file in client, then NameNode cache, then NameNode	Increases file access efficiency	Does not utilize merging and for distributed system
11	Cheng et al./2017	File merging based on indexing mechanism	Increases file access efficiency	For distributed system and file access process only
12	Lyu et al./2017	Uses three aspects: file merge, file map, and prefetch and cache	Increases file read efficiency	For distributed system and file read process only

Table 3.1 Critical Analysis of Relevant Approaches (Continued)

13	Ahad et al./2018	Uses two approaches: 1) identifying files size and 2) classifying the files on their types	Increases file read/write efficiency	For a single file system and distributed system
14	Peng et al./2018	Adds merging and caching modules with extra hardware and the files are identified by users	Increases file read/write efficiency	For distributed system and merging identified by users might affect the method
15	Matri et al./2018	Locate any piece of data in the cluster independently of any metadata server	Increases file access efficiency	Does not use merging and for distributed system
16	Xiong et al./2019	Merge for small spatiotemporal files in health	Increases file access efficiency	For only health spatiotemporal sensors and read

Table 3.1 Critical Analysis of Relevant Approaches (Continued)

17	Tao et al./2019	Appending additional files to existing merged file	Exploits remaining merged file size for increasing file access efficiency	For file access and distributed system

3.4 Remarks

In this thesis, the issues stated previously will be tackled to solve the problem of weaknesses of operating system data transfer rate of small files. Hence, from the previous works, a number of remarks are noted as follows:

1) Focuses on a specific environment, such as a distributed system or a health field.
2) Modifies the internal structure of a file system.
3) Adding extra hardware which will have to yield to basic modifications and/or increase the cost.
4) Un-addresses the efficiency.
5) Relies on more complex steps which may cause other problems such as resource consume or producing other problems in case of errors raise up.
6) Focuses on a specific aspect only, such as access only, read only or write only.
7) Uses techniques other than those approved in this research.
8) Focuses on a specific file format(s).
9) Improves only metadata, neglecting the data itself.
10) Focuses on a certain file size or very small range of file sizes.

Therefore, in this research, it will be proposed a new technique that will improve the operating system data transfer rate of small-size files. On the other hand, this thesis will suggest an algorithm that decides - before transferring process - which technique is suitable to be applied to files being transferred, the proposed data transfer technique or using the traditional operating system technique to benefit from the two techniques.

3.5 Chapter Summary

This chapter has reviewed the current literature that has led to the identification of certain strengths and limitations of the existing approaches. This chapter has presented a discussion of several of the previous researches.

The vast majority of the previous researches are for distributed systems, a very few are for local systems. They are either sophisticated implemented by many steps and may produce overheating, or adding extra hardware.

This research presents a new proposed technique, that improves the “data transfer rate locally” of the operating system, for many small with non-cumbersome techniques, includes all file formats, does not add hardware, and independent of the file system.

CHAPTER FOUR: DESIGN AND IMPLEMENTATION

4.1 Introduction

As described previously, to transfer a file locally from a source to a destination, the operating system, such as Windows 10's OS, executes many I/O's operations for the task accomplishment. Consequently, this leads to the software weaknesses of the operating system data transfer rate and more system resource consume, especially in a case of many small files transfer.

So, to enhance this problem, a new technique is proposed. Its main idea is minimizing I/O's operations of Windows 10's OS. This is done by merge and on-the-fly extract, as described in details in this chapter.

4.2 Design

The computer specification: CPU: Intel Core i3™3110M 2.4GHz, HDD: SATA3 500GB 6Gb/s 5400r/m, RAM: DDR3 4GB 800MHz.

Firstly, files are divided into 12 groups or folders based on their sizes, to see the size at which the proposed technique produces negative results. In this thesis, these sizes are:

512B, 1KB, 2kB, 4kB, 8kB, 16kB, 32kB, 64kB, 128kB, 256kB, 512kB and 1MB.

The first folder contains the files that each file of them is 512B, the second folder contains the files that each file of them is 1kB, and so forth until the folder that contains the files of 1MB of each.

To create each one of those folders with its related size and number of files, FileTool64 application is used. All of those folders are stored in a source of a clean formatted volume (E:\ partition).

Using the data transfer technique of Windows 10, the files of size 512B are transferred to a destination of a clean formatted volume (F:\ partition).

The complete transfer time of the traditional technique of the operating system is approximately 27.22s, and the data transfer rate is 73.48 file/s.

4.3 Proposed Technique

The proposed technique will be as follows:

First, the files are merged together. For this process, we choose:

- PeaZip application [30], because it is lightweight and for research experiments it is the fastest file merging and extraction tool.

- .wim merging file format [31], because it is the fastest merging and extraction file format for the research experiments.

The 2000 files of 512B size are merged together in the source to build one packet file. The time for this task is 0.28s.

As soon as that packet is created in the source, it is extracted and all the files that it contains are extracted "on-the-fly" to speed up the data transfer rate to the destination. The time for this task is 11.57.

The complete transfer time of the proposed technique for these two tasks is approximately 11.85s and the data transfer rate is 168.78 file/s.

Figure 4.1 depicts the overall principle of the proposed technique.

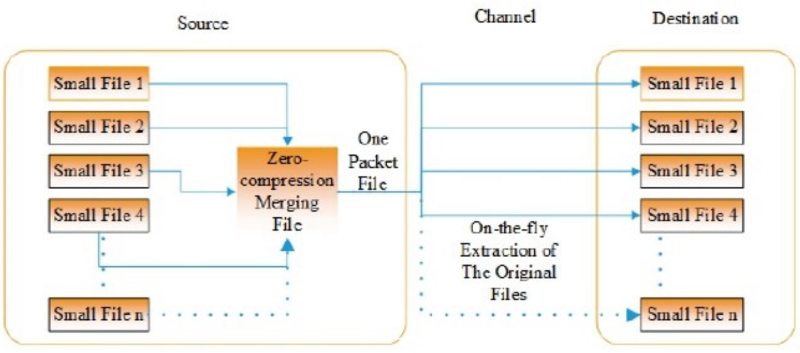


Figure 2.10 Principle of Proposed Technique

For confirmation purposes, attention is paid to observe the file size range of the proposed technique, and the above scenario is repeatedly applied to all other size groups or folders. Table 4.1 summarizes the differences of data transfer times between the proposed technique and OS's technique of the all file groups.

Table 2.3 Data Transfer Time of Proposed and OS's Techniques

Folder (2000 files)	Total folder size (KB)	Proposed technique time (second)			OS time (second)
		Merging	On-the-fly Extraction	Total	
512B	1000	0.28	11.57	11.85	27.22
1KB	2000	0.50	14.22	14.72	31.28
2KB	4000	0.43	12.72	13.15	29.35
4KB	8000	0.50	14.36	14.86	31.43
8KB	16000	0.43	14.64	15.07	28.79
16KB	32000	0.58	17.86	18.44	31.43
32KB	64000	0.92	18.72	19.64	31.35
64KB	128000	1.14	20.57	21.71	37.08
128KB	256000	2.78	20.86	23.64	48.14
256KB	512000	13.78	24.50	38.28	71.58

Table 2.4 Data Transfer Time of Proposed and OS's Techniques (Continued)

512KB	1024000	85.93	76.22	162.15	108.21
1MB	2048000	192.07	140.93	333	289.14
Mix of above	461400	11.42	20.72	32.14	71.29

4.4 Evaluation of The Proposed Technique

The operating system data transfer rate suffers when transferring small files due to many operations per each file transfer session. So, to decrease the effects in such a case, "merging and extracting on-the-fly" technique is proposed. This technique implies that before transferring many small files locally from a source to a destination on a computer, they are merged together to create only one packet file. When the packet file is created, it is extracted on-the-fly to the destination to speed up the extraction as much as possible. The merging and on-the-fly extracting technique improves the data transfer rate of many small files locally.

Validation can be done by analysing the calculations of time difference, data transfer rate and efficiency between the operating system data transfer technique and the proposed data transfer technique.

4.4.1 Evaluation of Enhancement of Data Transfer Time

Table 4.1 summarizes all calculations of the time delay differences of the transfer processes of the files' groups between the proposed data transfer technique and Windows 10 OS's technique. It is shown, that using the proposed data transfer technique, the data transfer time delay for the 2000 files each of which has a size of 512B is 11.85 seconds, whereas using the traditional operating system data transfer technique, the data transfer time delay of them is 27.22 seconds. Refer to Table 4.1 for the remaining time delay difference.

4.4.2 Evaluation of Enhancement of Data Transfer Rate Efficiency

Experiments are applied to show the results from perspective view of data transfer rate (or time) efficiency $(Efficiency = \frac{Proposed\ Technique\ DRT - OS\ DTR}{Proposed\ Technique\ DRT} \times 100)$, which expresses DTR improvement or time saving percentage. Table 4.2 presents the proposed data transfer technique and OS's data transfer technique for all files' groups.

Table 2.5 Efficiency of Data Transfer Rate of Proposed Technique

Folder (2000 files)	Proposed technique DTR (Files/s)	OS DTR (Files/s)	Time saving (second)	Efficiency (%)
512B	168.78	73.48	15.37	56.47
1KB	135.87	63.94	16.56	52.94
2KB	152.09	68.14	16.2	55.20
4KB	134.59	63.63	16.57	52.72
8KB	132.71	69.47	13.72	47.66
16KB	108.46	63.63	12.99	41.33
32KB	101.83	63.80	11.71	37.35
64KB	92.12	53.94	15.37	41.45
128KB	84.60	41.55	24.5	50.89

Table 4.2 Efficiency of Data Transfer Rate of Proposed Technique (Continued)

256KB	52.25	27.94	33.3	46.52
512KB	12.33	18.48	-53.94	-49.85
1MB	6.01	6.92	-43.86	-15.17
Mix of above	62.23	28.05	39.15	54.92

There is an improvement of all file groups except 512KB and 1MB.

Figure 4.2 shows the efficiency of the proposed technique data transfer rate compared with operating system.

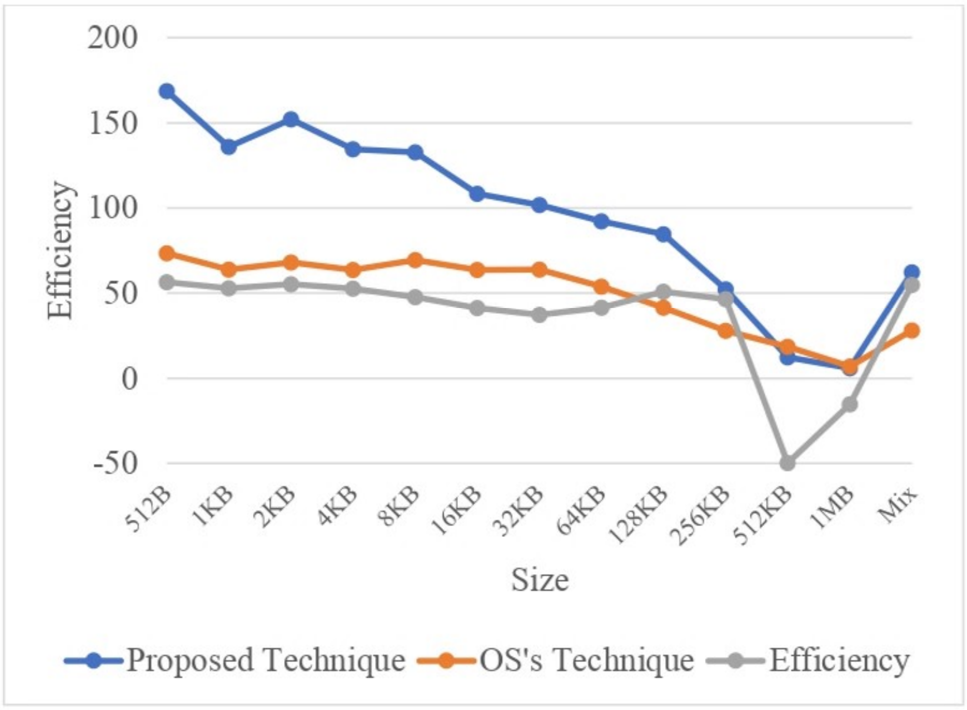
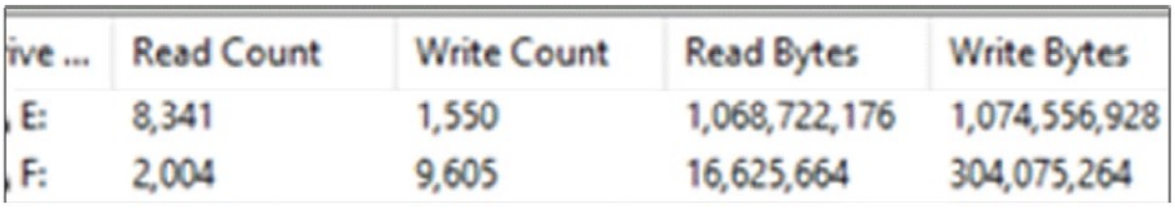


Figure 2.11 Efficiency of Data Transfer Rate According to Size

So, it is validated that the number of the files that are transferred per second is improved using the proposed technique compared to the traditional OS's technique, hence, the proposed technique improves the operating system data transfer rate.

On the other hand, resource consumption is validated. The tests are observed from perspective view of disk read/write operations of the file transfer using DiskCountersView application.

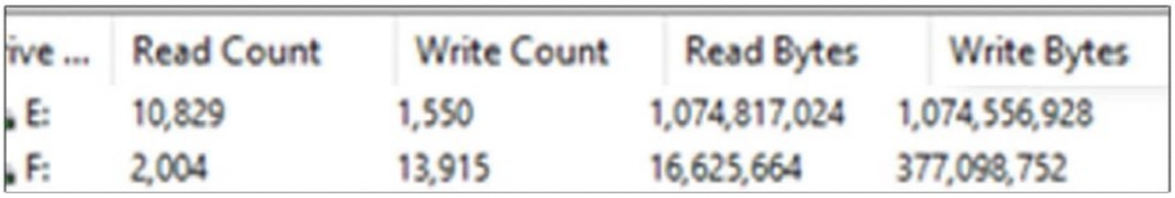
Figure 4.3 shows disk read/write counts per second of the file transfer operations evaluation of Windows 10's data transfer technique. It is shown that source read count is 8341 read/second and the destination write count is 9605 write/second.



ive ...	Read Count	Write Count	Read Bytes	Write Bytes
E:	8,341	1,550	1,068,722,176	1,074,556,928
F:	2,004	9,605	16,625,664	304,075,264

Figure 2.12 Count of Disk Data Transfer Read/Write of Windows OS

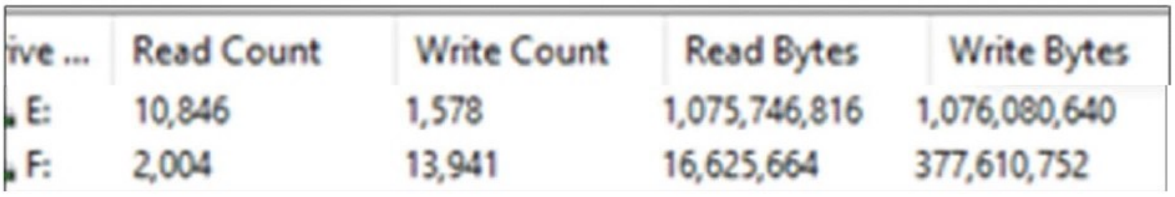
Figure 4.4 shows disk read/write counts per second of the file transfer operations evaluation of the proposed data transfer technique during merging. Here, the source read count is 10829 read/second.



ive ...	Read Count	Write Count	Read Bytes	Write Bytes
E:	10,829	1,550	1,074,817,024	1,074,556,928
F:	2,004	13,915	16,625,664	377,098,752

Figure 2.13 Count of Disk Data Transfer Read/Write During Merging

Figure 4.5 shows disk read/write counts per second of the file transfer operations evaluation of the proposed data transfer technique during on-the-fly extraction. The source read count is 10846 read/second and the destination write count is 13941 write/second.



ive ...	Read Count	Write Count	Read Bytes	Write Bytes
E:	10,846	1,578	1,075,746,816	1,076,080,640
F:	2,004	13,941	16,625,664	377,610,752

Figure 2.14 Count of Disk Data Transfer Read/Write During Extraction

So, the average read count per second of the proposed technique from the source to the destination is approximately 10837 read/second, and the write count per second of the proposed technique from the source to the destination is approximately 13941 write/second.

Hence, the average read/write count from the source to the destination of the data transfer using the proposed data transfer technique is greater than that of the operating system. So, system resources are exploited efficiently.

4.5 Proposed Algorithm for Choosing Suitable Technique

For maximum utilization, the proposed technique algorithm is designed to use both techniques; the proposed data transfer technique and OS's technique. Figure 4.6 is a graph derived from Table 4.2. The graph shows that the proposed data transfer technique is not always efficient. It shows that the proposed data transfer technique improves the data transfer rate of the files' sizes up to 379KB/file because at this point, we get 0% efficiency. In this thesis, 0% efficiency is considered as "THRESHOLD" between the new proposed technique and the operating system technique. For reliability, THRESHOLD is minimized to 341KB/file (10% off).

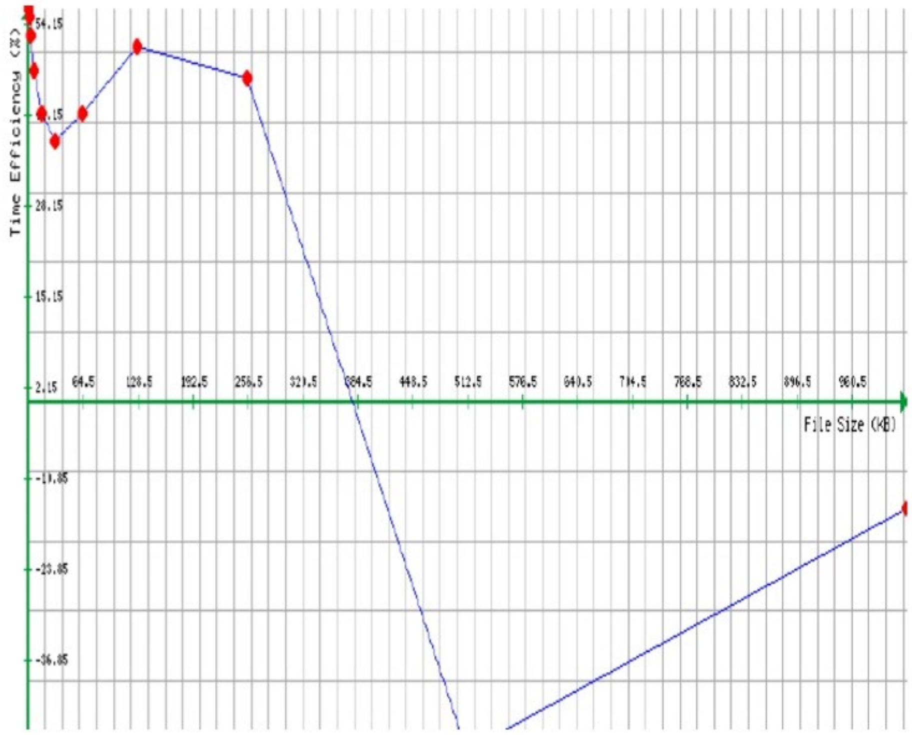
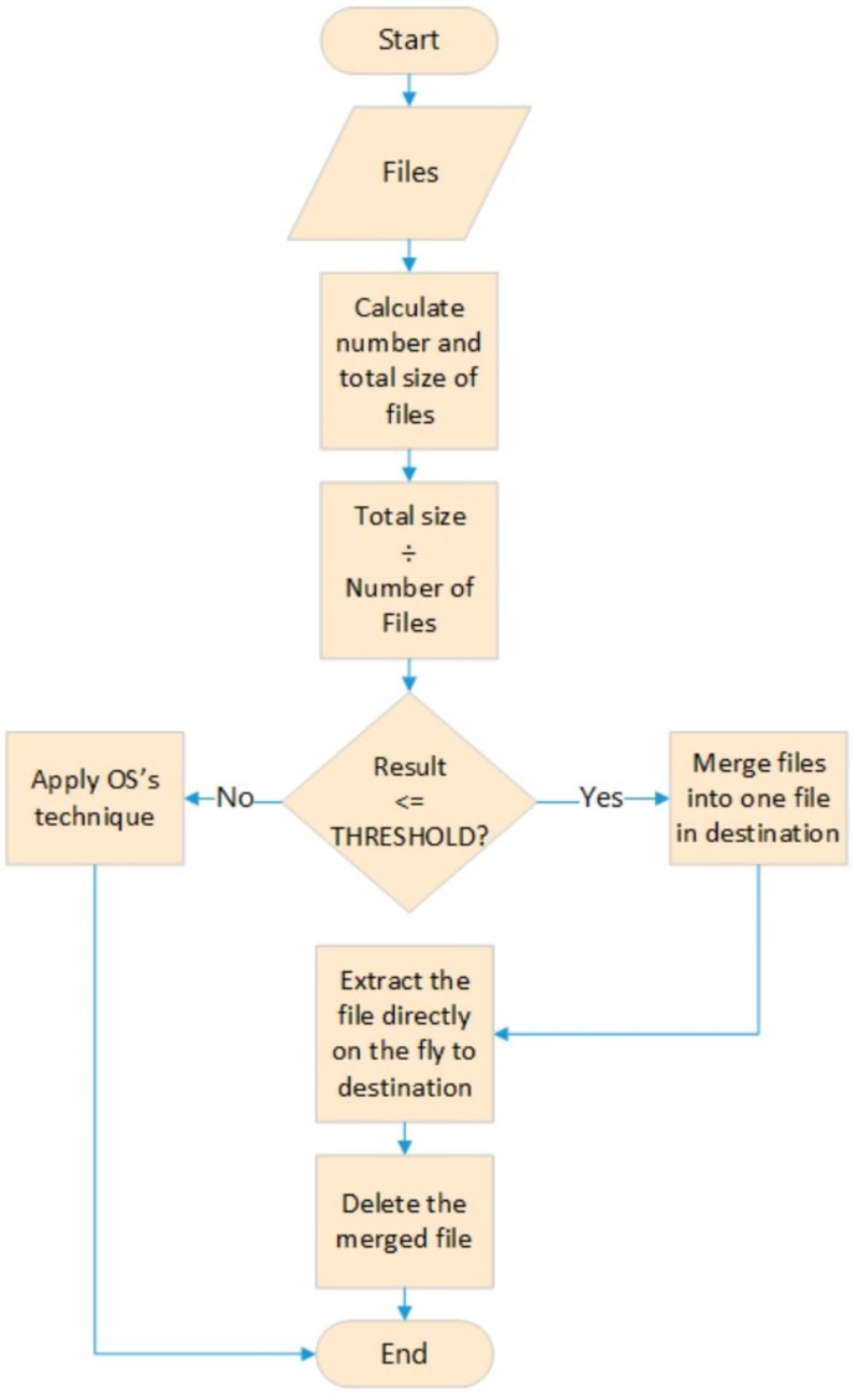


Figure 2.15 Change of Time Efficiency According to File Size

Therefore, the proposed technique is applied if:

THRESHOLD = Total size of files ÷ number of files = 341KB/file

Consequently, to enhance the data transfer rate, a new algorithm for the new technique is proposed, working as presented in the flowchart shown in Figure 4.7.



Start
Files
Calculate number and total size of files
Total size ÷ Number of Files
Result <= THRESHOLD?
No
Apply OS's technique
Yes
Merge files into one file in destination
Extract the file directly on the fly to destination
Delete the merged file
End

Figure 2.16 Algorithm Flowchart for Choosing Appropriate Technique

4.6 Evaluation of Proposed Algorithm for Choosing Suitable Technique

If there are many files to be transferred locally from a source to a destination, the algorithm calculates the result of division of total size of files by their total number. If the result is equal or less than THRESHOLD, the new proposed technique is applied to transfer the files. If the result is more than THRESHOLD, the traditional operating system technique is applied to transfer the files.

For example, if there are 4000 files to be transferred, and their total size is 25MB, then:

25MB ÷ 4000 files = 6.4KB/file ≤ THRESHOLD

Therefore, "size per file" is less than THRESHOLD, so, the proposed data transfer technique is suitable and applied to transfer the files.

If there are 50 files to be transferred, and their total size is 25MB then:

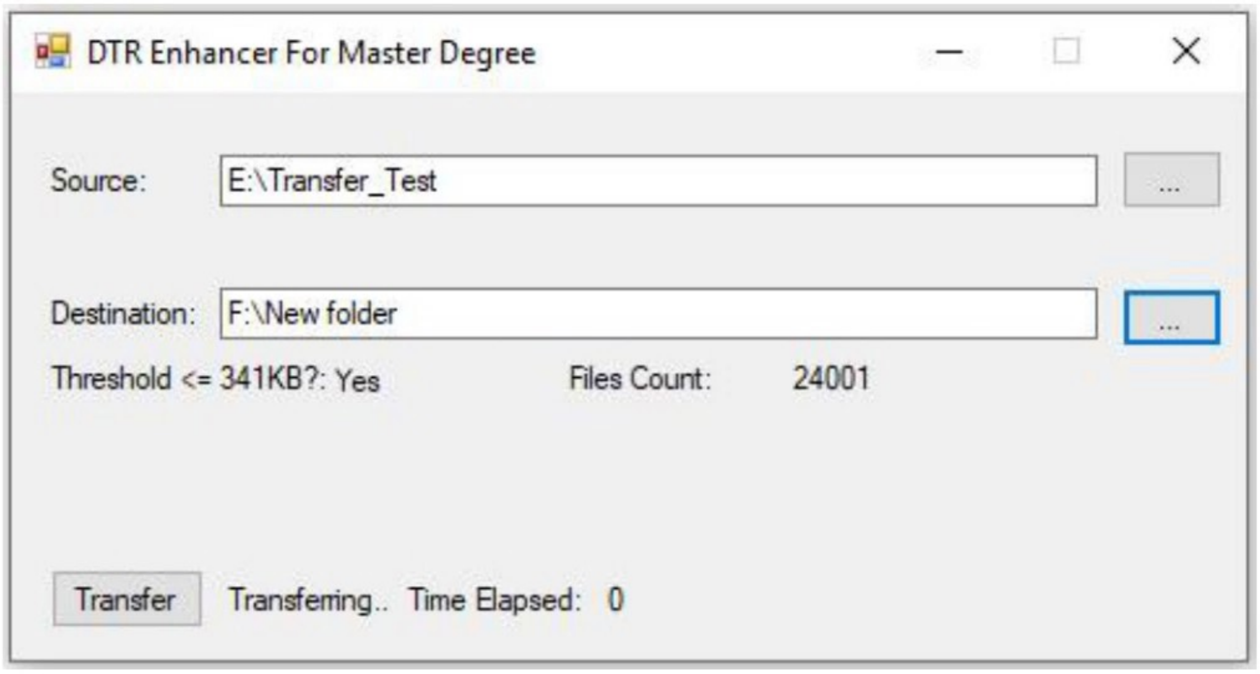
25MB ÷ 50 files = 512KB/file > THRESHOLD

So, "size per file" is more than THRESHOLD, and therefore, Windows 10's operating system traditional data transfer technique is suitable and applied to transfer the files. Consequently, both techniques are used for maximum exploitation.

4.7 Coding Algorithm

To make the new proposed technique work automatically, the algorithm is coded using Visual Basic 2017 programming language as shown in Figure 4.8.

The algorithm code is evaluated as well. Three small size file folders are created. The first folder contains 8,000 files, the second folder contains 16,000 files and the third folder contains 24,000 files. All three folders are tested and the results of the proposed data transfer technique are positive and efficient than that of the traditional operating system.


DTR Enhancer For Master Degree

Source: E:\Transfer_Test

Destination: F:\New folder

Threshold <= 341KB?: Yes   Files Count: 24001

Transfer   Transfering.. Time Elapsed: 0

Figure 2.17 Screen Shot of the Algorithm Code

4.8 Comparison of Results With Previous Works

Previous researches have presented solutions for distributed environment systems. They have introduced improvements to challenge of the problem of poor software of an operating system when transferring many small files. However, current researches have no approaches to improve this problem “locally” in term of “data transfer rate”. Nevertheless, this research is compared to two previous researches as well.

Using merging technique in buffer, Ahn et al. [13] proposed a technique that enhanced the small file write of a server-based environment by modifying Fast File System FFS. Its aim is the file size of 768B or less, and the average efficiency is 33%.

The new proposed technique is introduced for enhancing local based small data transfer rate from a source to a destination without file system modification and it is not for file write only. Also, the new proposed technique enhances the data transfer rate of more file size up to 341KB by mean efficiency of 50.89%.

Tao et al. [28] proposed a new technique for improvement of small files for distribution-based Hadoop File System HDFS. The approach proposed a merging technique with two steps, the first step is merging small files into some part files and the second step is that meta-information is built into the index files to improve file access only. Also, the approach supports appending additional files to a current archive. Again, our new technique is not for a distributed system and not only for improving file access, it is specifically for data transfer rate improvement of an operating system locally from a source to a destination.

Figure 4.9 shows a comparison of the three approaches in term of the time of data transfer. Using the new proposed approach, the data transfer time is less than CW-FFS and LHF approaches.

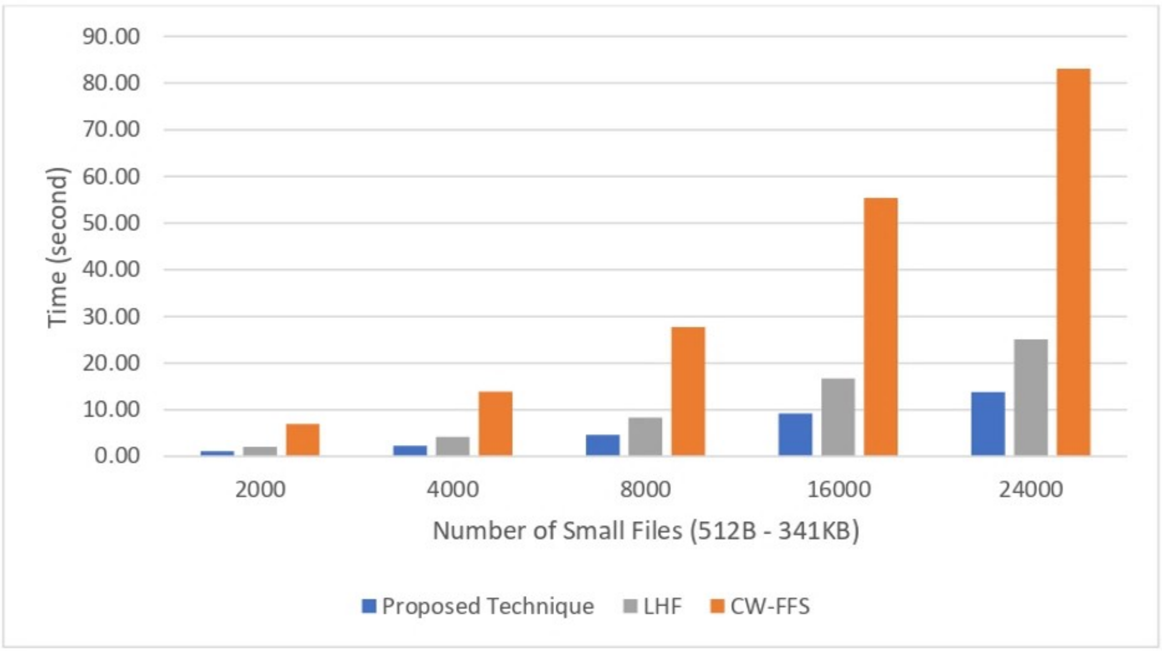


Figure 2.18 Comparison of Data Transfer Time

4.9 Chapter Summary

Locally, data transfer rate of an operating system is affected negatively according to file size changes, specifically when transferring many small files. So, this thesis proposed a new technique approach to enhance the problem. Suitable tools were used to implement the design, experiments and test. Also, coding the technique is implemented. The new proposed technique was applied to Windows 10’s OS environment. The enhancement was proven from two perspective views of the operating system software; the data transfer rate and resources consumption. Also, the technique was designed to use both approaches of the proposed and the operating system. The new proposed technique was compared with other related works, and it was proven that it was more efficient.

CHAPTER FIVE: CONCLUSION AND FUTURE WORK

5.1 Conclusion

Locally, big files data transfer rate of the computer operating system is fine. The real problem is the software weaknesses of the operating system data transfer rate due to changes of the file sizes. The problem raises up when many small files transfer. This is because the operating system executes many I/O’s operations from a source to a destination for each file transfer process at very a small time period. This introduces two negative reactions to the operating system:

1) Very low data transfer rate.
2) High resources consumption.

Therefore, this thesis has focused on three aspects:

1) Analyzing the data transfer task implemented by the operating system to find out the reasons that lead to the occurrence of software weakness of the data transfer rate of many small files.
2) Introducing a new proposed technique to enhance the operating system data transfer rate of the small files with less system resources consumption.
3) Suggesting an algorithm that chooses the suitable mechanism between the proposed data transfer technique and the traditional OS’s technique based on the files size and number.

So, the operating system based on layered approach, such as Windows 10’ OS have some software weaknesses. For some conditions and needs, the operating system developers still add more I/O’s operations, and do not pay attention to negative effects of the operations addition.

There are many solutions, but the vast majority of them, if not all are for distributed systems. They improve data access only, read only or write only. A very few solutions, or may be no one improve the operating systems data transfer locally. In this research, both have been focused on.

The research focuses on Windows 10's operating system data transfer approach. This thesis has proven that the more operations the operating system performs while transferring a file, leads to more time delay and more resources consumption in case of many small files. So, whatever the reasons which enforce the operating system developer to add more operations when transferring files, it is necessary to pay attention to the time delay and resources consumption caused by adding those operations.

So, this research proposed a technique approach depends on three steps:

1) Pure merging in the source to minimize the files' transfer process operations which an operating system executes.
2) On-the-fly extraction from the source to the destination to speed up the extraction process.
3) Deletion of the created merging file.

Applying the approach steps introduce a new technique, which enhances the data transfer rate weaknesses of the operating system when transferring many small files. The new technique has been tested, evaluated and compared with other techniques, specifically in terms of data transfer rate locally from a source to a destination. Simplicity is an advantage of the new proposed technique to keep it at less number of layers. It is efficient in terms of time and resources consume

5.2 Limitation of The Proposed Approaches

The algorithm is efficient and has been validated. Nevertheless, it contains some limitations:

1) At some unknown circumstances, it does not enhance small files data transfer rate.
2) There is no more control over the internal parameters of the merging file format to make up pure zero-compression merging. So, the research supposes zero-compression merging.
3) The algorithm is coded - for simplicity - using Visual Basic 2017 programming language, and this language is considered high-level and slow.

5.3 Future Work

The limitations described previously can be addressed as future work. Further, future works are as follows:

1) Tuning the internal merging file format parameters to create a pure merging file with zero-compression.
2) Coding the algorithm using a fast programming language.
3) Enhancing the data transfer rate of Windows OS for any files' sizes.
4) Applying the technique to enhance the "data transfer rate" on a distributed environment operating system.

References

[1] *Monthly mobile PC data traffic worldwide 2015-2022*. (n.d.). Statista. Retrieved April 5, 2020, from https://www.statista.com/statistics/739014/worldwide-monthly-traffic-mobile-pc/

[2] *Desktop Windows Version Market Share Worldwide*. (n.d.). StatCounter Global Stats. Retrieved April 5, 2020, from https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide

[3] *Why does copying multiple files take longer time than copying a single file of same size? - Quora*. (n.d.). Retrieved June 21, 2020, from https://www.quora.com/Why-does-copying-multiple-files-take-longer-time-than-copying-a-single-file-of-same-size#

[4] *Operating System Tutorial: What is, Introduction, Features & Types*. (n.d.). Retrieved June 8, 2020, from https://www.guru99.com/operating-system-tutorial.html

[5] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. (2018). *Operating System Concepts*. 10th Edition.

[6] tedhudek. (n.d.). *Example I/O Request—An Overview—Windows drivers*. Retrieved June 8, 2020, from https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/example-i-o-request---an-overview

[7] EliotSeattle. (n.d.). *User mode and kernel mode—Windows drivers*. Retrieved April 22, 2020, from https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode

[8] *MicrosoftDocs/windows-driver-docs*. (n.d.). GitHub. Retrieved April 22, 2020, from https://github.com/MicrosoftDocs/windows-driver-docs

[9] Chudzikiewicz, J., & Furtak, J. (n.d.). (2012). *Cryptographic Protection of Removable Media with a USB Interface for Secure Workstation for Special Applications*. Journal of Telecommunication and Information Technology. Military

University of Technology, Warsaw, Poland

[10] van Gorp, R., & van Bockhaven, C. (n.d.). (2018). *Low-level writing to NTFS file systems*. MSc Security and Network Engineering.

[11] *SYSENTER - OSDev Wiki*. (n.d.). Retrieved June 16, 2020, from https://wiki.osdev.org/SYSENTER

[12] tedhudek. (n.d.). *Overview of Windows Components—Windows drivers*. Retrieved June 8, 2020, from https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/overview-of-windows-components

[13] Ahn, W. H., Lee, K., Oh, J., Min, K., & Hong, J. S. (2009). *A multiple-file write scheme for improving write performance of small files in Fast File System*. Information Processing Letters, 109(18), 1021–1026. https://doi.org/10.1016/j.ipl.2009.05.010

[14] Lensing, P., Meister, D., & Brinkmann, A. (2010). *hashFS: Applying Hashing to Optimize File Systems for Small File Reads*. In 2010 International Workshop on Storage Network Architecture and Parallel I/Os (pp. 33–42). Incline Village, NV, USA: IEEE. https://doi.org/10.1109/SNAPI.2010.12

[15] Dong, B., Zheng, Q., Tian, F., Chao, K.-M., Ma, R., & Anane, R. (2012). *An optimized approach for storing and accessing small files on cloud storage*. Journal of Network and Computer Applications, 35(6), 1847–1862. https://doi.org/10.1016/j.jnca.2012.07.009

[16] Fu, S., Huang, C., He, L., Chaudhary, N., Liao, X., Yang, S., … Li, B. (2013). iFlatLFS: Performance optimization for accessing massive small files. In 20th Annual International Conference on High Performance Computing (pp. 10–19). Bengaluru (Bangalore), Karnataka, India: IEEE. https://doi.org/10.1109/HiPC.2013.6799116

[17] Zhang, S., Miao, L., Zhang, D., & Wang, Y. (2014). *A Strategy to Deal with Mass Small Files in HDFS*. In 2014 Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics (pp. 331–334). Hangzhou, China: IEEE. https://doi.org/10.1109/IHMSC.2014.87

[18] Tao, X., & Alei, L. (2014). *Small file access optimization based on GlusterFS*. In Proceedings of 2014 International Conference on Cloud Computing

and Internet of Things (pp. 101–104). Changchun, China: IEEE. https://doi.org/10.1109/CCIOT.2014.7062514

[19] Gupta, T., & Handa, S. S. (2015). *An extended HDFS with an AVATAR NODE to handle both small files and to eliminate single point of failure*. In 2015 International Conference on Soft Computing Techniques and Implementations (ICSCTI) (pp. 67–71). Faridabad, India: IEEE. https://doi.org/10.1109/ICSCTI.2015.7489606

[20] Bende, S., & Shedge, R. (2016). *Dealing with Small Files Problem in Hadoop Distributed File System*. Procedia Computer Science, 79, 1001–1012. https://doi.org/10.1016/j.procs.2016.03.127

[21] Jing, W., & Tong, D. (2016). *An Optimized Approach for Storing Small Files on HDFS-based on Dynamic Queue*. In 2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI) (pp. 173–178). Beijing: IEEE. https://doi.org/10.1109/IIKI.2016.55

[22] Kyoungsoo Bok, Jongtae Lim, Hyunkyo Oh, & Jaesoo Yoo. (2017). *An efficient cache management scheme for accessing small files in Distributed File Systems*. In 2017 IEEE International Conference on Big Data and Smart Computing (BigComp) (pp. 151–155). Jeju Island, South Korea: IEEE. https://doi.org/10.1109/BIGCOMP.2017.7881731

[23] Wenjuan Cheng, Miaomiao Zhou, Bing Tong, & Junhong Zhu. (2017). *Optimizing small file storage process of the HDFS which based on the indexing mechanism*. In 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA) (pp. 44–48). Chengdu, China: IEEE. https://doi.org/10.1109/ICCCBDA.2017.7951882

[24] Yanfeng Lyu, Xunil Fan. (2017). *An optimized strategy for small files storing and Accessing in HDFS*. In 2017 IEEE International Conference on Computational Science and Engineering (CSE) and International Conference on Embedded and Ubiquitous Computing (EUC). Xi'an, China: IEEE. https://doi.org/10.1109/CSE-EUC.2017.112

[25] Ahad, M. A., & Biswas, R. (2018). *Dynamic Merging based Small File Storage (DM-SFS) Architecture for Efficiently Storing Small Size Files in Hadoop.*

Procedia Computer Science, 132, 1626–1635. https://doi.org/10.1016/j.procs.2018.05.128

[26] Peng, J., Wei, W., Zhao, H., Dai, Q., Xie, G., Cai, J., & He, K. (2018). *Hadoop Massive Small File Merging Technology Based on Visiting Hot-Spot and Associated File Optimization*. In J. Ren, A. Hussain, J. Zheng, C.-L. Liu, B. Luo, H. Zhao, & X. Zhao (Eds.), Advances in Brain Inspired Cognitive Systems (Vol. 10989, pp. 517–524). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-00563-4_50

[27] Matri, P., Perez, M. S., Costan, A., & Antoniu, G. (2018). *TýrFS: Increasing Small Files Access Performance with Dynamic Metadata Replication*. In 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) (pp. 452–461). Washington, DC, USA: IEEE. https://doi.org/10.1109/CCGRID.2018.00072

[28] Xiong, L., Zhong, Y., Liu, X., & Yang, L. (2019). *A Small File Merging Strategy for Spatiotemporal Data in Smart Health*. IEEE Access, 7, 14799–14806. https://doi.org/10.1109/ACCESS.2019.2893882

[29] Tao, W., Zhai, Y., & Tchaye-Kondi, J. (2019). *LHF: A New Archive Based Approach to Accelerate Massive Small Files Access Performance in HDFS*. In 2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService) (pp. 40–48). Newark, CA, USA: IEEE. https://doi.org/10.1109/BigDataService.2019.00012

[30] *PeaZip free archiver utility, open extract RAR TAR ZIP files*. (n.d.). Retrieved April 6, 2020, from https://www.peazip.org/

[31] LLC), T. M. (Aquent. (n.d.). *Windows Imaging File Format (WIM)*. Retrieved April 6, 2020, from https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-7/dd799284(v%3dws.10)

تحسين معدل نقل البيانات لنظام التشغيل للملفات الصغيرة الكثيرة

الخلاصة

الكمبيوتر هو جهاز رائع، يستخدم في مختلف المجالات، يساعد على معالجة الكثير من البيانات وتنفيذ العديد من المهام في وقت قصير. واحدة من المهام الرئيسية والمتكررة هي نقل البيانات من المصدر إلى الوجهة. في الماضي، كانت أجهزة الكمبيوتر بطيئة ومعالجة البيانات عليها قليلة جدًا. لذا، لم يكن تأثير معدل نقل البيانات في نظام تشغيل الكمبيوتر مشكلة كبيرة. في وقت لاحق، ازداد الطلب على البيانات بشكل كبير في جميع المجالات، وحتى الكمبيوتر ونظام التشغيل نفسيهما يؤديان العديد من المهام داخليًا. لذا، فإن زيادة البيانات والعمليات عليها قد ولدت بعض التحديات على نظام التشغيل. المشكلة الرئيسية هي تأثير معدل نقل بيانات نظام التشغيل بسبب التغيرات في الحجم وعدد الملفات التي سيتم نقلها، خاصة الملفات الصغيرة الكثيرة. لذلك، أدت هذه المشكلة إلى آثار سلبية على أداء المهام المختلفة، التي يتم إجراؤها بشكل متكرر محليًا من قبل المستخدمين، بسبب ضياع الوقت واستهلاك الموارد.

الهدف من هذا البحث هو إبراز وتحليل مشكلة التأثير على معدل نقل البيانات لنظام التشغيل بسبب نقل العديد من الملفات الصغيرة واقتراح تقنية جديدة لتحسينها.

يقترح هذا البحث تقنية جديدة لنقل البيانات محليًا للعديد من الملفات الصغيرة، استنادًا إلى مبدأ "دمج الضغط الصفري النقي والاستخراج المباشر" دون أي تعديلات على معمارية نظام التشغيل أو بنية نظام الملفات، من أجل تقليل العمليات الكثيرة التي يقوم بها نظام التشغيل أثناء عملية نقل الملفات.

إن اقتراح تقنية البحث هذا يقدم تحسينًا فيما يتعلق بمعدل نقل البيانات محليًا للعديد من الملفات الصغيرة، مما يؤدي إلى تحسين أداء نظام التشغيل وبالتالي تحسين أداء المستخدم.

هناك حلول تعزز مشكلات الملفات الصغيرة، ولكن الكثير منها مخصص للأنظمة الموزعة، وإذا لم يكن كذلك، فإنه لا يوجد بحث منها مخصص لتحسين معدل نقل البيانات محلياً لنظام التشغيل. يعتمد نهج البحث على التجارب والتحليل والتقييم، ويقدم تقنية تعمل على تحسين معدل نقل البيانات في

نظام التشغيل محليًا للعديد من الملفات الصغيرة من المصدر إلى الوجهة. تثبت نتيجة التقنية المقترحة الجديدة أنها تعزز معدل نقل بيانات نظام التشغيل للعديد من الملفات الصغيرة بكفاءة.

الجمهورية اليمنية

وزارة التعليم العالي والبحث العلمي

جامعـــــة الريـــان

كلية الدراسات العليا

AL-RAYAN UNIVERSITY

تحسين معدل نقل البيانات لنظام التشغيل للملفات الصغيرة الكثيرة

رسالة مقدمة إلى جامعة الريان لاستكمال متطلبات نيل درجة الماجستير، تخصص تقنية معلومات

إعـــداد
ماجد عبدالله باسماعيل

إشـــــراف
د/ سعيد محمد بانعيمون

1442هـ / 2021م