Distributed Storage APIs using HBASE and HDFS for Encrypted Personal Health Records

Metha Wangthammang

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Computer Engineering
Prince of Songkla University
2017

Thesis Title	Distributed Storage APIs using HBASE and HDFS for Encrypted Personal Health Records
Author	Mr. Metha Wangthammang
Major Program	Computer Engineering

Major Advisor

....
(Asst. Prof. Dr. Sangsuree Vasupongayya)

Examining Committee:

....Chairperson
(Assoc. Prof. Dr. Sinchai Kamolphiwong)

....Committee
(Asst. Prof. Dr. Sangsuree Vasupongayya)

....Committee
(Prof. Dr. Verapol Chandeying)

The Graduate School, Prince of Songkla University, has approved this thesis as Partial fulfillment of the requirements for the Master of Engineering Degree in Computer Engineering

....
(Assoc. Prof. Dr. Teerapol Srichana)
Dean of Graduate School

This is to certify that the work here submitted is the result of the candidate’s own investigations. Due acknowledgement has been made of any assistance received.

....Signature
(Asst. Prof. Dr. Sangsuree Vasupongayya)
Major Advisor

....Signature
(Mr. Metha Wangthammang)
Candidate

I hereby certify that this work has not been accepted in substance for any degree, and is not being currently submitted in candidature for any degree.

....Signature
(Mr. Metha Wangthammang)
Candidate

Thesis Title	Distributed Storage APIs using HBASE and HDFS for Encrypted Personal Health Records
Author	Mr. Metha Wangthammang
Major Program	Computer Engineering
Academic Year	2016

ABSTRACT

This work proposed a distributed storage (DSePHR) system for storing encrypted personal health record data (PHR) by identifying properties of the encrypted PHR data and using the properties to create an index for the encrypted PHR data. The encrypted PHR data contain health related information of an individual. Both HDFS and HBase are used as a fundamental storage framework. The DSePHR provides a set of APIs for storing and retrieving the encrypted PHR data. The DSePHR eliminates a high memory consumption issue on the cloud storage caused by storing a lot of small files. The experimental results showed that the DSePHR still preserve the scalable feature while consuming less memory. Furthermore, DSePHR can deliver similar upload / download time performance when the mixture of file size and the ratio of read and write activities change.

ชื่อวิทยานิพนธ์	ส่วนต่อประสานโปรแกรมประยุกต์ที่เก็บข้อมูลแบบกระจาย โดย เฮชเบส และเฮชดีเอฟเอส สำหรับข้อมูลสุขภาพส่วนบุคคลซึ่งถูกเข้ารหัส
ผู้เขียน	นายเมธา หวังธรรมมั่ง
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ปีการศึกษา	2559

บทคัดย่อ

งานวิจัยชิ้นนี้ (DSePHR) ออกแบบและพัฒนาระบบการเก็บข้อมูลแบบกระจายสำหรับข้อมูลสุขภาพส่วนบุคคลซึ่งถูกเข้ารหัส (Encrypted PHR data) โดยการระบุคุณลักษณะของข้อมูลซึ่งถูกเข้ารหัสและใช้คุณลักษณะดังกล่าวในการสร้างดัชนีสำหรับข้อมูลสุขภาพที่ถูกเข้ารหัส ข้อมูลสุขภาพส่วนบุคคลที่ถูกเข้ารหัสประกอบไปด้วยข้อมูลที่เกี่ยวข้องกับสุขภาพส่วนบุคคล เฮชเบส (HBase) และเฮชดีเอฟเอส (HDFS) ถูกใช้ในงานวิจัยนี้สำหรับเป็นกรอบงานการเก็บข้อมูลพื้นฐาน (Storage framework) งานวิจัยนี้จัดเตรียมชุดของส่วนต่อประสานโปรแกรมประยุกต์ (APIs) สำหรับจัดเก็บและดึงข้อมูลสุขภาพที่เข้ารหัสที่ถูกจัดเก็บไว้ในพื้นที่เก็บข้อมูลแบบกระจาย งานวิจัยนี้ได้ปรับปรุงประเด็นการใช้หน่วยความจำเยอะ (High memory consumption) อันเนื่องมาจากการจัดเก็บไฟล์ขนาดเล็กจำนวนมากลงบนพื้นที่เก็บข้อมูลแบบกระจาย ผลลัพธ์การทดลองแสดงให้เห็นว่าระบบที่ได้พัฒนาขึ้นยังคงรักษาคุณสมบัติการขยายตัวเอาไว้ได้ในขณะที่ใช้หน่วยความจำที่น้อย นอกจากนี้ ระบบที่ได้พัฒนาขึ้นสามารถให้ประสิทธิภาพเวลาที่ใกล้เคียงในการอัพโหลดและดาวน์โหลด (upload /download) เมื่อมีการผสมกันของขนาดไฟล์ และอัตราส่วนของกิจกรรมการอ่านและการเขียนได้เปลี่ยนไป

ACKNOWLEGDEMENT

I would like to express my deepest gratitude toward my advisor, Assistant Professor Dr. Sangsuree Vasupongayya, for the helpful suggestions. I have studied a lot regarding the research methodology. I am very proud to be her advisee.

I would like to express my warmest gratitude toward Associate Professor Dr. Sinchai Kamolphiwong and Professor Dr. Verapol Chandeying for their useful comments.

I would like to thank my senior and friends in Center for Network Research (CNR) and Graduate student room for their supports and assistances.

I would like to thank Thanathip Limna and Khachain Wangthammang for thesis suggestion on the implementations and experimental settings.

I would like to thank the scholarship from the National Research University Project of Thailand's Office of the Higher Education Commission for the financial support, Funding no. MED540548S.

Metha Wangthammang

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Preventive care such as exercise, enough relaxation, health check can lead people to gain healthy life. Healthcare professional or doctor can use personal health related data to provide a suggestion to the data owner in order to decrease the risk of disease that can be occurred in the future [1], [2]. Therefore, storing personal health data is essential for the preventive care. The health data of person are typically stored at a hospital and controlled by the hospital [3]. The health data owner cannot access his/her data directly because of the hospital security and policy. Most hospitals use a private storage system because the risk of patient's data leaking from an external attacker can be reduced. In a situation of changing a hospital, the health data do not automatically transfer to the new hospital. Requesting the health data to be transferred to a new hospital can be complicated because the private storage system usually does not support exported operation. Therefore, a concept that the health data owner can store his/her health data is promoted. The concept is called personal health records (PHRs).

The PHRs concept allows the PHR owner to fully control his/ her health data. The owner can create, edit, and share his /her health data. The owner can store his/her PHR data to a PHR provider of his/her choice. The PHR data can be grouped into 11 types such as problem list, allergy data, home-monitored data, medication [3]. The PHR data also have a few properties that are different from that of regular data type. For example, the PHR data has only one actual owner and belongs to the owner only; the PHR data consist of various data sizes and types; the PHR data can cover all data of the PHR owner life time [4]. The PHR data can be everything that is related to health information and the amount of the PHR data is increasing every day. Therefore, the volume of the PHR data will be large. Thus, the PHR storage requires a big storage to store all PHR data from many PHR users. Moreover, the security of the health data will be critical because the health data are now outside the hospital and no longer protected by the hospital.

The PHR data are high volume, high variety and high velocity. The PHR contain the life-time data of a person (high volume). The PHR data can be document file, audio file, video file and image file (high variety). The PHR data can slowly occur such as medication data or quickly occur such as monitor device data (high velocity). So the PHR data can be considered as big data [5]. Storing the PHR data into a cloud storage is an appropriate solution to deal with the PHR big data characteristic. The cloud storage can scale its capacity when the PHR system requires more capacity. The PHR data that are on the cloud storage, require a security to be protected from an unauthorized access. Thus, the PHR data must be encrypted. Related works [6]–[10] introduced the methodology to encrypt the PHR data and stored the data into a general cloud storage. However, these works do not address a method to store the PHR data on a general cloud storage. Typically, the general cloud storages are not designed for storing and retrieving the encrypted PHR data because they do not provide any particular features to access the encrypted PHR data in their storage.

Since the storage system cannot read the content of encrypted PHR data in order to classify or create an index of the data for convenient retrieving, the encrypted PHR data need a particular access pattern such as sorted by time, owner that the access pattern is different from general binary files. The general binary files such as image file, audio file or document file in other system have access pattern by using its properties or content. For example, the access pattern of the image file can be using its resolution, type of images to determine specify image. The access pattern of video file can be using its length, resolution to determine specify video. While encrypted PHR data do not provide any properties, the access pattern can be using only time and owner of the data. General cloud storages such as Dropbox and Google drive, storage on software as a service (SaaS), are directory based storage. Amazon S3 and CACSS [11], storages on platform as a service (PaaS), are buckets style storage. Although both storage types can store encrypted PHR data, it is difficult to access the encrypted PHR data with the PHR user requirement. The PHR users mostly access their data with sorted by the owner and time patterns because the encrypted PHR data belongs to only one owner and relates to the life of the owner. Moreover, the encrypted PHR data lacks of information related to its data because the data must be protected in an encrypted form. These general cloud storages do not support creating metadata to indicate the encrypted PHR data for retrieving by owner and time. Metadata is important to create an index of the encrypted PHR data. Therefore, the encrypted PHR data storage should pre-process the data to identify its properties and bring its properties to create an index by following PHR user access pattern for convenient retrieving.

This thesis focused on designing and developing application programming interface (API) in order to design the schema and create the index mechanism for convenient retrieving of the encrypted PHR data on a distributed storage, called DSePHR which stands for " Distributed Storage APIs using HBASE and

HDFS for Encrypted Personal Health Records”. General cloud storage [11] can suffer the high consumption memory problem due to storing a lot of small files. Hence, our proposed distributed storage eliminates such problem. The proposed distributed storage supports both small size file and large size because most of PHR data is a document file (small file) and some PHR data can be video (large file). Moreover, the encrypted PHR data should be sorted by the encrypted PHR data properties: owner and time. The user can conveniently access his/her data by specifying the owner and time. The storage supports storing massive encrypted PHR data. The design of the proposed DSePHR is presented in chapter 3 while the performance of proposed DSePHR is demonstrated in the experimental section. The evaluation and discussion of the experiment are also presented in chapter 4.

1.2 Objectives

1. Design a schema and a structure for indexing and storing the encrypted PHR data.
2. Develop the API for storing and retrieving the encrypted PHR data.
3. Evaluate the developed APIs on synthetic PHR workloads.

1.3 Scopes

1. This work uses python 2.7.10, Flask framework, Flask-RESTful, Hadoop 2.7.1, HBase 1.0 to develop the DSePHR API.
2. Storing and retrieving encrypted PHR data are provided by DSePHR API.
3. A number of incoming simultaneous requests is limited by CPU, memory and LAN interface of DSePHR web service.

4. Synthetic workloads with mixture of read and write requests are used for evaluating the DSePHR.

1.4 Contributions

1. An API set for storing encrypted PHR data to a cloud storage.
2. A set of HBASE configuration tuning for performance.
3. A performance result of the prototype DSePHR developed as a test platform including memory consumption, storing and retrieving time of various file type distribution and mixture of read and write requests.
4. A synthetic PHR workload

CHAPTER 2

LITERATURE REVIEW

Background and related works are presented in this chapter including personal health records, PHR storage requirements, big data, Hadoop Distributed File System (HDFS), Apache HBase, Ganglia which is a cluster monitor tool and CryptDB.

2.1 Personal Health Records

Personal Health Record (PHR) is a system that allows a person to store, manage, and share his/her health related information. PHRs are different from Electronic Health Record (EHR) [3]. The EHR is the health data owned by the healthcare institutes and the data are collected or created by the physician or healthcare institute staffs. Thus, the access to EHR data is controlled by the healthcare institute according to the institute's policy and the legal regulation of that country. The PHRs, on the other hand, is the data that is owned and controlled by its owner. The PHR owner has a full control on who can access his/her data, how the data can be accessed, and when the access can occur.

The Markle foundation defines PHRs as "an Internet-based set of tools that allows people to access and coordinate their lifelong health information and make appropriate parts of it available to those who need it" [4]. According to the definition, the PHR system can contain many kind of health related information such as weight, height, blood type, blood pressure, symptoms, medication usage, information from doctor, allergy data and demographic data.

Currently, there is no standard for PHR data. However, the PHR data can be grouped in to 11 types [3], including problem list, procedures, major illnesses, provider list, allergy data, home-monitored data, family history, social history and lifestyle, immunizations, medications, laboratory tests. The detail of each PHR data type will be explain.

- The problem list is a document that contains an important health problem of the patient or individual such as current disease, injuries from an accident. The problem list can be in a form of the document file type such as word-processing file, pdf, CCD file and CCR file. The source of the problem list can be patient or the PHR owner, EHR or doctor.
- The procedure is a description of a step or an action to achieve a treatment. The procedure can exist in both a document file format and a media file format such as a guide line of medication (word processing file, pdf file) and a step of operation or surgery (video file). The source of the procedure can be the PHR owner, EHR or a medicare claim
- The major illness is the information that describes the illness or surgery of a patient such as congenital disease or current disease. The Major illness can exist in a form of a document file format such as CCD file, CCR file, word-processing file and pdf file. The source of the major illnesses can be the PHR owner, EHR or a medicare claims
- The provider list is a list of health care providers or institutions that the PHR owner gets his/her treatment. The provider list is a document file in a form of CCD file, CCR file, word-processing file or pdf file. The source of the provider list includes the PHR owner and the EHR.
- The allergy data is the document that describes the allergy symptom of patient such as food allergy, skin allergy and respiratory allergy. The allergy

data can be in a form of a document file such as CCD file, CCR file, word-processing file or pdf file. The source of the allergy data includes the PHR owner and the EHR.

- The home-monitored data is the data from any home health care device such as sensor data from heart rate instrument or stethoscope. The home-monitored data can be in a form of text files such as txt file, csv file and spreadsheet file. The source of the home-monitored data includes the PHR owner and the equipment.
- The family history is the document that describes the history of the disease or symptom of member in the PHR owner family in the past. The doctor can use the family history to diagnose a current disease of the patient. The family history can be in a form of document file type such as CCD file, CCR file, word-processing and pdf file. The source of the family history data includes the PHR owner and EHR.
- The social history and lifestyle is the document that describes the personal life, occupation, favorite activity of the PHR owner. The doctor can use the information to improve the treatment. The social history and lifestyle can be in a form of document file such as CCD file, CCR file, word-processing or pdf file. The source of the social history and lifestyle data includes the PHR owner and EHR.
- The immunizations are a history list of immunizations that the PHR owner had received in the past such as polio, rubella and tetanus. The can be in a form of a document file such as a CCD file, CCR file, spreadsheet file or word-

processing file. The source of the immunization data includes the PHR owner, EHR and the immunization registry (if any)

- The medications are the document that contains a list of medication that the patient is currently using, including the amount and schedule to take each medicine. The medication can be in a form of a document file such CCD file, CCR file, pdf file and word-processing file. The source of medication data include the PHR owner, EHR and medicare claim.
- The laboratory tests are the result or discussion of laboratory processes such as medical check-up. The laboratory data can be in a form of text, table, image and exist as both document file format (word-processing file, CCD file, CCR file) and media file format (image file).

Besides the 11 PHR data types, the users can store other information such as nutrition, exercise and sleeping habits in their PHRs. Moreover, the PHR owner can allow the physician to add some health related information to his/her PHRs. Therefore, the PHRs can be viewed as a lifelong health related information storage of all its members. At this point, the PHR system must support various data type and a large amount of data.

Existing PHR systems including My health record [12], Google Fit [13], Microsoft HealthVault [14]. My health record is a national digital health record system of Australia. My Health Record allows the related people such as an authorized person or provider to view, upload, download the clinical information of a person. In 2015-2016, there are 90 million document files with various file types were uploaded to My Health Record system [15]. Google Fit is a fitness activity tracking system installed on a mobile phone to track the daily life or the fitness activity such as walking, running and cycling of its owner. The activity tracking data from the mobile phone is then sent to Google server to be stored. The stored data can be later visualized as an overall activity. There is a lot of data that needs to be stored. Microsoft HealthVault is a web-based PHR system that allows the user to directly input his/her health data or upload various health document such as CCD file, CCR file, image file, word-processing file and pdf file. Microsoft HealthVault also supports a connectivity with the smart health device such as weight scales, blood pressure monitors and heart rate monitors. Thus, there is a various data type and a lot of data to be stored in the system. It can be seem from various existing PHR system that the PHR system must be able to store and manage such large volume of data. Thus, the storage requirements of such system are interesting.

2.2 PHR Storage Requirements

An important component for a PHR system is the PHR storage. Since the entire life-time health related information of a person can be stored on the PHR system. The PHR storage must support everyday life health data, high availability, scalability and security.

The PHR data storage must provide a high write throughput performance to support everyday life health data. The data from each member can be of various varieties daily. The nature of the data input process in the PHRs is that each user will record his/ her personal health related information from various sources. The data will be viewed less often and the data will usually be analyzed during the viewing process. Therefore, a high volume of data from various sources is expected so that the storage must be able to receive all data into the storage. Thus, the PHR data storage must be evaluated when the mixture of requests contains a large number of write requests than that of the read requests.

The high availability and fast response of the storage system are also expected from the PHR system. The PHR data contains the information that will be helpful to the physicians or caregivers in order to save or to treat the PHR owner during life-threatening situations such as the emergency staffs during a car accident. The victim of the accident might require an immediate healthcare at the crime scene. The basic health information such as blood type and a history of allergic reactions must be available to these people. Also, the request for further information of the victim from the system must be provided immediately. Thus, the time it takes to store the data and retrieve the data must be evaluated.

The PHR storage must be able to scale up in order to cope with the amount of data from its members. Furthermore, a good management of the system must be able to satisfy several levels of requirement needs. Some sets of data may not be accessed real-time or some data may be outdated. With the usage patterns of some information such as the old health checkup records or the old x-ray images in the PHRs, such information may be moved or switched to slow-response storages in order to save spaces. Thus, the during-operation adding of more physical storages and a good management of data storage are a challenge for designing the PHR data storage. Thus, the PHR storage must be evaluate when the storage node is added in order to show the response time during such operation.

The last requirement for the PHR data storage is the security and privacy issues. The PHRs may contain some sensitive information, thus the PHR data must be protected and controlled. Typically, the access control of any data is achieved by applying an access policy to either the data and/or the user, while the confidentiality of data is achieved by applying an encryption technique to the data. With big data property and the nature of the system usages, the data may be resided on a cloud-based storage and available to the users. Thus, the security aspect of the system must be considered as the fifth requirement. To accomplice such goals, the PHR system must support some forms of data privacy and security implementations.

2.3 PHR and Big Data

Big data is defined in [16] as

> “ data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or does not fit the structures of your database architectures. To gain value from this data, you must choose an alternative way to

Three components to classify big data are variety, velocity and volume. For the variety component, the data can be of various formats such as unstructured or structured data. With structured data, it is easy to stored and analyzed because the data are tagged and organized. The unstructured data, however, is difficult to stored, analyzed and virtualized. The velocity component refers to the data that can be submitted to the system in various forms such as stream, real-time or batch. The volume can scale from terabytes to zettabytes. Potential applications of big data are identified in five domains including healthcare, public sector, retail, manufacturing and telecommunications. PHRs can be classified in the healthcare domain. In order to classify PHRs as a big data application, three components (i.e., variety, velocity and volume) are analyzed and discussed.

For the variety component of the PHRs, everything concerning health related information of an individual can be collected and stored. Each user utilizes the system differently. The data can be from various sources such as the PHR owner, the physician of the PHR owner, or the caregiver of the PHR owner. For example, the PHR owner might want to store his daily exercise; a physician might want to store x-ray images, laboratory results and medical treatments; the caregiver might want to store the data recorded from the wearable sensors. Thus, the variety of data types must be handled by the PHR system. Some data are structured data such as medical treatments and laboratory results, while some data are un-structured or semi-structured such as the signal recorded from wearable sensors and daily exercise information.

For the velocity component of the PHRs, the data from various sources create various data transfer rates. According to the example given earlier, real-time streaming data can be collected from the PHR owner's wearable sensors. The physician, on the other hand, may upload a batch of medical health checkup results into the PHR database on behalf of the PHR owner. Furthermore, some information in the PHR system may be accessed in real-time fashion such as the personal caregiver may want to monitor the PHR owner's vital sign signal which is stored on the PHR system.

To handle lifelong health related information of its members, the amount of data to be stored in the PHR system is gigantic. To illustrate the point, assuming a big city with one million people use the PHR system and each member generates approximately 300 KB of data daily. Thus, there are 300 GB of data stored in the system daily and 109 TB of data yearly. This calculation is based on one entry of a single record per day of each user. However, the data of each user can be generated from various sources and several times per day. Thus, the stored data of each user can scale up very fast similar to the personal data on the Facebook account of each user.

2.3.1 Cloud Storage

The PHR data requires a scalable storage to store its data because the volume of the data always increases. Traditional storage may face a problem that cannot scale up to support high volume of data. A cloud storage comes to be a solution for a big data storage because the cloud storage is a scalable storage. The cloud storage can be software as a service (SaaS) such as Dropbox, Google Drive, iCloud or platform as a service (PaaS) such as Hadoop (HDFS), OpenStack (Swift). However, the SaaS is not appropriate to use as fundamental framework because the mechanism of the storage cannot be easily adjustable in order to support any specific requirement of the data. For instance, the encrypted PHR data require a special metadata for searching, identifying and accessing the data. Therefore, PaaS is more appropriate than SaaS because it is adjustable. OpenStack is a cloud computing framework containing swift as an object storage. However, OpenStack does not include the database. Hadoop is a big data computing framework containing HDFS as a distributed storage. HBase is a non-relational database with column family database type. HBase can use HDFS for storing the actual data files. HBase also inherits features of HDFS such as distributed storage, replication and fault-tolerance. Hadoop and HBase are selected as a fundamental framework in this thesis.

2.3.2 Cloud Storage for Health Care

Because the volume of health care data is huge, many works apply the cloud storage or big data framework due to its scalable feature. The related works of a cloud storage for health care data is explained next.

MedCloud [17] is a health care system which designs to follow Health Insurance Portability and Accountability Act (HIPAA) policy. The objective of MedCloud is to exchange health related information between providers. Cloud computing and storage are adopted in the work for sharing health related information. Using cloud solution, the user can store their EMR or PHR data on the system. Hadoop and HBase are selected for basic framework of MedCloud.

The improvement of CACSS [11] has developed to be a generic cloud storage for demonstration purposes in order to show any organization or institution who wants to establish a private cloud storage. Hadoop and HBase are used as a fundamental framework for the development. The actual data are stored on HDFS while the metadata are stored on HBase. The main advantage is to store unstructured data on a cloud storage. However, the design for a metadata retrieval especially by the owner of the data had not been addressed. Retrieving the data by its owner may take long time because the system must search the whole database to filter the data and display the result.

CHISTAR [18] has developed to handle scalability issue facing by traditional EHR in hospital in United States. Traditional EHR system (VistA) is designed in term of client- server connection. There is a scalable limitation in traditional EHR system. Thus, CHISTAR transforms the traditional EHR system to cloud computing to solve scalability issue. HDFS is adopted as a part of the storage and MapReduce is applied for data processing before the data is sent to HBase.

Wiki health [19] is a cloud storage system which is designed to store health care sensor data. Wiki health was developed on top of [11] and used Hadoop and HBase as its fundamental framework. Health sensor data such as Electroencephalography or Electrocardiography can be stored on the Wiki health directly. Health sensor data are stored on HBase. Wiki health allows the user to attach unstructured data to the sensor data. However, to retrieve any unstructured data file is complicated because the user does not have a direct access to the data.

2.3.3 Other PHR Systems

Patient-centric and fine-grained data access control to the personal health records system with multiple PHR owner on a cloud storage environment was provided in [6]. The multi-authority attribute based encryption (MA-ABE) was adopted in the work to provide security and multiple owner feature. The work mainly focuses on how the data is encrypted while preserving fine-grained access control on the data. The explanations on how to store the encrypted data in a storage are not mentioned.

Cipher-policy attribute based encryption (CP-ABE) was adopts in [7] to traditional existing PHR system named Indio X, and moved to cloud computing. The objective of the work was to preserve the privacy sharing model and fine-grained access control on cloud computing. However, the design and approach to store the encrypted PHR data to a cloud storage are not mention.

The work in [8] was extended from [6] and also adopted MA-ABE. The scalability of storage and communication cost are calculated and explained in this work. However, the approach to store the encrypted PHR data in the cloud storage is not mentioned.

The work in [9] designs the EHR system on cloud computing. The main aim of the work is that the users can select some parts of their PHR data to share to physician while the PHR data is still encrypted. The three features including searchability, physician revocation and local decryption are provided in the work. Although the designed EHR system work on the cloud computing environment. The design and explanation on how to store the encrypted PHR data in cloud storage is not mentioned.

The work in [10] extends the original secure PHR system to handle an emergency situation when dealing with untrustworthy players. The work also applies cipher text policy attribute based encryption and access control on the PHR data. An emergency stuff can access the patient PHR data from cloud storage. However, the cloud storage is used for storing the encrypted PHR data without any modification.

2.4 Hadoop Distributed File System (HDFS)

Hadoop distribution file system (HDFS) [20] is a storage part of Hadoop [21] which is developed by apache to support big data for storing and processing. Hadoop is designed for parallel processing, high availability, fault tolerance and scalability. Hadoop has 2 main parts including distributed storage (HDFS) and parallel processing (MapReduce). In this work, the distributed storage is considered and parallel processing part is ignored because this work mainly focuses on how to store and retrieve the encrypted data on a cloud storage. Thus, HDFS is applied as the fundamental storage framework.

HDFS is the part of a distributed file system of Hadoop. There are two HDFS node types: Namenode and Datanode. Namenode maintains metadata of all files in HDFS such as a location of blocks, a number of replication. The metadata is stored in the memory of Namenode for fast retrieval. Another Namenode is Secondary Namenode which is a backup node of Namenode. Datanode is responsible for storing the actual data in HDFS. With HDFS, a client can directly contact to the Datanode which stores the actual data. For a write request, the client sends a request to ask the Namenode in order to get a list of available nodes. After that, the client can upload the data directly to any available Datanode. When the data uploading is complete, the Datanode will send its' data to another Datanode until the number of data replication is reached. For a read request, the client sends a request to inquire Namenode for the requested file information. The Namenode will reply with a list of Datanode containing the requested data. After that, the client will download the data from the Datanode directly. The HDFS architecture is shown in Figure 2.1.

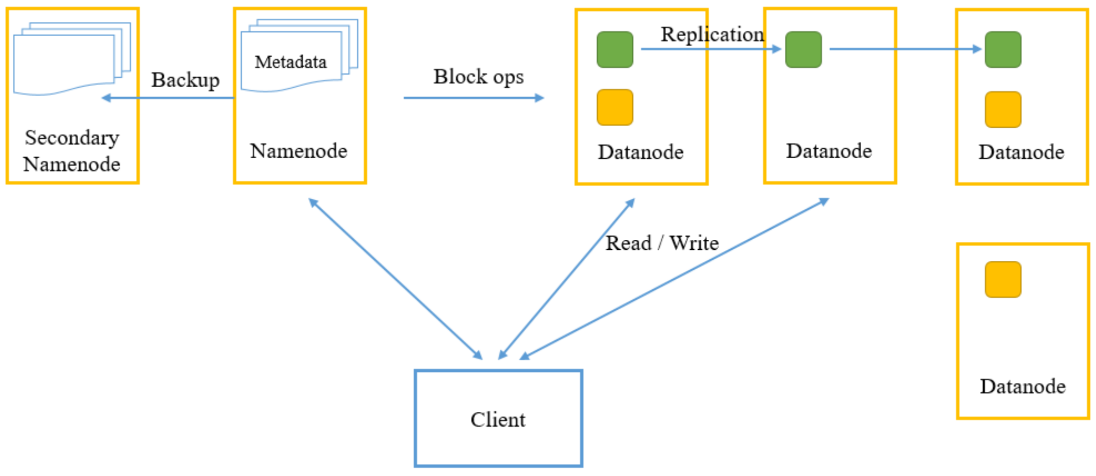


Figure 2.1 HDFS Architecture

When the data arrives at the HDFS, the data will be split into blocks. Each block is stored in the Datanode (default block size is 128 MB). The Metadata of all blocks are stored in the Namenode. The number of default replication blocks is three and these blocks are sent to be stored on a different Datanode. Three replication blocks are the key of the high availability and the fault tolerance. If some Datanode fail, HDFS can use the replicated data from other Datanode. Moreover, more capacity can be added to HDFS without the need to shutdown the system.

The major issue of HDFS is the memory consumption of the small file size. Basically, the HDFS can store any file size. Although the default HDFS block size is 128 MB and the data that is smaller than the block size will use 1 block, the actual size of each block depends on the actual size of the data. Storing a lot of small size files can cause a high memory consumption issue. For a fast access, HDFS stores the metadata of the whole data set in the memory using the Namenode. A volume of memory consumption directly depends on a number of files in the HDFS. A file in HDFS takes approximately 1354 bytes of the Namenode memory. The Metadata of a single file takes 250 bytes and three replicated blocks take 368*3 bytes. Storing a lot of small files consumes the Namenode memory more than storing large files. Although both situations use the same storage capacity. To store a lot of small file, HDFS requires an approach to handle the memory consumption issue.

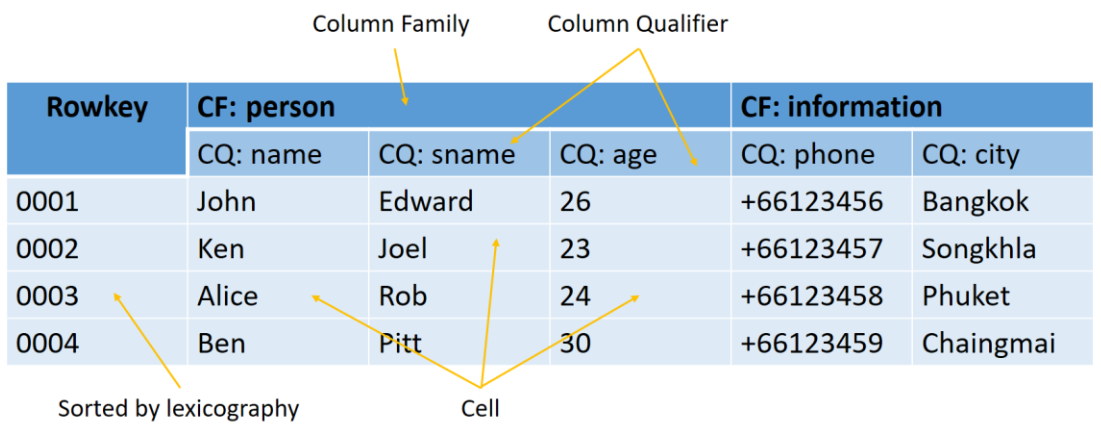
The discussion in [22] provided a method to store image files in HBase by an engineer from ImageShack [23]. The HBase cluster is applied to store image files for 2 years without the memory issue. The image file is considered as BLOB (large binary object) and stored in HBase. However, the HBase tuning configuration is necessary. The advantage of the approach is that HBase will pack these small files to a large file and send the packed file to HDFS automatically.

Medoop [24] is a health care platform which supports health information exchange (HIE) in China because of the health data growing. Medoop selects Hadoop and HBase as its developing framework. Medoop solves storing small size file issue by merging the small file to large file and creating index metadata file for the large file. Both large files and index metadata files are stored into HDFS. HBase is used to store the user information.

The optimize approach for storing small file [25] analyzes and finds the cutoff point between small file and large file when both files are stored to HDFS. A result of cutoff point file is 4.35MB. Moreover, the work also introduces the methodology to handle a small file by merging many small files to a single large file and creating an index file. Both large file and an index file are stored into HDFS. The work solves only a lot of small files storing problem and acts a like general cloud storage. The work does not provide any mechanism for easy retrieving the encrypted PHR data.

2.5 Apache HBase

Apache HBase is a non-relational database which designs to support a large data set. HBase can use HDFS as a storage and inherit the advantage of HDFS such as replication, scalability and fault tolerance. HBase is a column oriented database type but it is different from a traditional database like MySQL. There is no relation between the column in each table. HBase cannot use a join operation to query the summarized data. HBase schema component includes table, column-family, column-qualifier, row-key and cell. Figure 2.2 shows the HBase schema. Both “person” and “information” are column family (CF). The “person” CF includes “name”, “sname” and “age” column qualify (CQ). Accessing the data stored in HBase can perform by specifying some components. Table is a group of row-key and contain column-families. Column-family is a group of column-qualifiers. Cell is a place that can store the actual data. Call can store multi-version of the data. All row-keys in HBase are always sorted by lexicography.


Column Family
Column Qualifier
Sorted by lexicography
Cell

Rowkey	CF: person			CF: information	
	CQ: name	CQ: sname	CQ: age	CQ: phone	CQ: city
0001	John	Edward	26	+66123456	Bangkok
0002	Ken	Joel	23	+66123457	Songkhla
0003	Alice	Rob	24	+66123458	Phuket
0004	Ben	Pitt	30	+66123459	Chaingmai

Figure 2.2 HBase schema

HBase is able to use HDFS or a local file system for storing its database files. To achieve a fully distributed mode, HBase must use HDFS as its file system. HBase has three types of services including master, region server and zookeeper. Master is responsible for administrator operations such as creating a table and deleting a table. Region server is responsible for storing the data which arrives to HBase. Zookeeper stores global metadata of every table in HBase. Figure 2.3 shows HBase architecture. One RegionServer can handle more than one Region. A number of Regions depends on RegionServer memory. Each Region can contain more than one store. Each store has one MemStore. When the data arrives to HBase, the data will be saved to Memstore first. After the size of Memstore reaches the maximum threshold, HBase will flush the data in Memstore to Hfile. When HBase has many

Hfiles, HBase will merge them together into StoreFile. The StoreFile is the actual data file of HBase. HBase can store the StoreFile in HDFS through DFS client to HDFS Datanode.

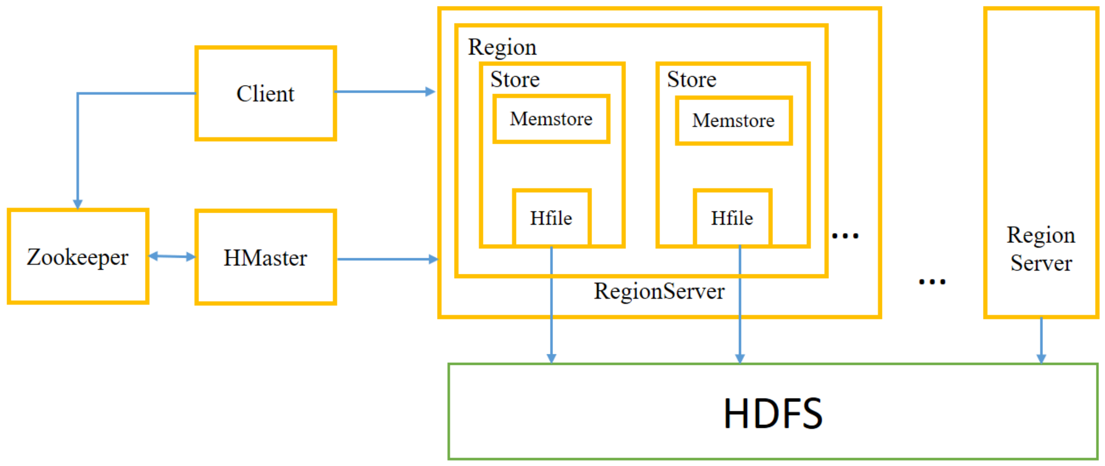


Figure 2.3 HBase Architecture

In this thesis, HBase is used for storing both metadata and small encrypted PHR data. In case of small encrypted PHR data, HBase store the data as a binary large object (BLOB). BLOB is considered to be a large file size for storing into the database because the database is designed for storing text which is smaller than BLOB. So, HBase configuration tuning can provide a better performance in comparison with the default configuration.

Hconfig [26] analyzes HBase when facing a massive data loading. Hconfig can improve HBase performance in handling a massive data loading by tuning the HBase configuration. Hconfig also suggests the good configuration for practical situations. A result shows that a good configuration can improves HBase throughput around 2-3 times. This thesis follows the suggested configurations in the work.

A driven policy configuration [27] is a continuous work from Hconfig [26]. The work provides a practical configuration for various situation such as write only, mostly write and mostly read. The result shows that the good configuration can improve HBase performance in handling massive data. The suggested configurations in the work are adopted in this thesis to gain more throughput.

2.6 Cluster Monitor (Ganglia)

To measure the proposed system performance, monitoring tool is necessary. Ganglia monitor [28] is a monitoring tool for a distributed system such as Hadoop or HBase. Both Hadoop and HBase support a direct exporting of their status to Ganglia. Ganglia can also monitor the status of the node in the cluster although there are no Hadoop and HBase are installed. Ganglia has 3 types of services including ganglia-monitor, gmetad and ganglia-web. Ganglia-monitor is responsible for monitoring a machine status such as cpu usage and network throughput. Ganglia-monitor will send the machine status to the gmetad continuously. Gmetad is a service collecting the metadata from the ganglia-monitor and saving the data to a round robin database (RRD). The RRD is a time series database that is suitable for storing the time series data such as network throughput, CPU load, HDD capacity. Ganglia-web is a PHP web application which visualizes the monitored data in RRD in a graph format. Ganglia-web also supports a exporting of the monitored data to a CSV or JSON file format. Therefore, Ganglia is used for collecting the status of Hadoop, HBase and nodes in all experiments, conducted in this thesis. The CSV exported monitored data is used to analyze the performance of DSePHR, which is given in chapter 4.

2.7 CryptDB

CryptDB [29] is a system that provides security for the data in a traditional relational database system such as MySQL and PostgreSQL. CryptDB claims that the system can keep the confidentiality of the data in the database even when the database and the application server are fully controlled by an attacker. Both data in the database and user’s query are encrypted by CryptDB proxy server that situates between application server and database server. The user’s password is used to generate a key that uses to encrypt the data at the proxy server. If the attacker does not know the user’s password, the attacker cannot decrypt the data in the database. Therefore, CryptDB focuses on the method to encrypt the data in the traditional relational database and it also has limitation when handling big data due to the nature of relational database. CryptDB does not provide the storage to store the actual data. The PHR storage should also support storing a large amount of data due to its size. The proposed system in this thesis provides the storage that can be scale to store a large amount of PHR data. With the proposed system design in this thesis, although the attacker can fully access the DSePHR storage, the attacker cannot read the data because the data is already encrypted from the source before being uploaded to the storage. The proposed system in this thesis is only designed for storing encrypted PHR data, and does not store any key for decrypting the data.

CHAPTER 3

METHODOLOGY

In this chapter, the features of the proposed system namely, Distributed Storage for Encrypted Personal Health Record (DSePHR) data API are explained. The DSePHR has been written in python under flask micro-framework. Both Hadoop and HBase are used as the underlying framework. The DSePHR interface applies REST style architecture. The overview, design and implementation of the DSePHR are described and the experimental design of DSePHR is in the last of the chapter.

3.1 PHR System Overview

The PHR system must allow an authorized user to upload and download the authorized PHR to and from the system. An authorized user can either uses the PHR system directly or access the PHR system via other application. The DSePHR must allow its users to store the encrypted PHR data on a distributed storage. The PHR data must be encrypted from the source. Thus, the PHR system overview is shown in Figure 3.1. The DSePHR is shown in the red square box, including a set of APIs and a distributed storage. The DSePHR API is an interface of the DSePHR. The PHR can be stored to the DSePHR via the uses of the DSePHR API. The distributed storage of the DSePHR is used for storing the encrypted PHR data.

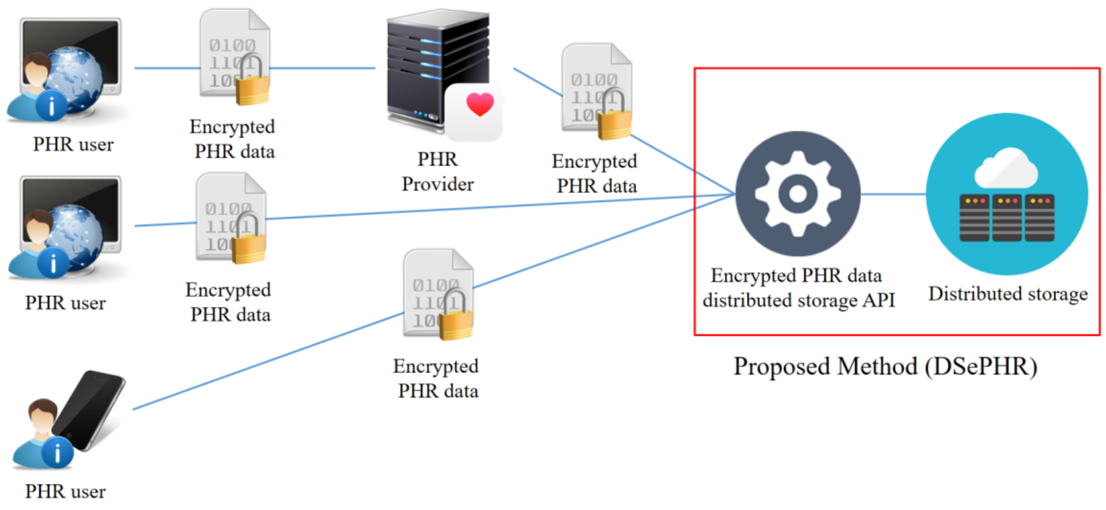


Figure 3.1 PHR System Overview

The dataflow on a write operation is shown in Figure 3.2, the PHR user must pass an authentication process first. If the authentication is successful, the DSePHR will return a token key. The token key is a string containing information to identify the user. When the user requests to upload the data, the user need to attach the token key with the encrypted PHR data to the DSePHR. The DSePHR will verify the token key and deny if the token key is not valid. If the token key is valid, the DSePHR will classify the data, create the metadata of the data and send both metadata and the data to be stored on the distributed storage.

The DSePHR prototype currently returns the data id of the uploaded PHR data as the metadata. This design aims for a fast experimental setting. However, the returning metadata can be kept as a part of the PHR user information on the system only for a later access.

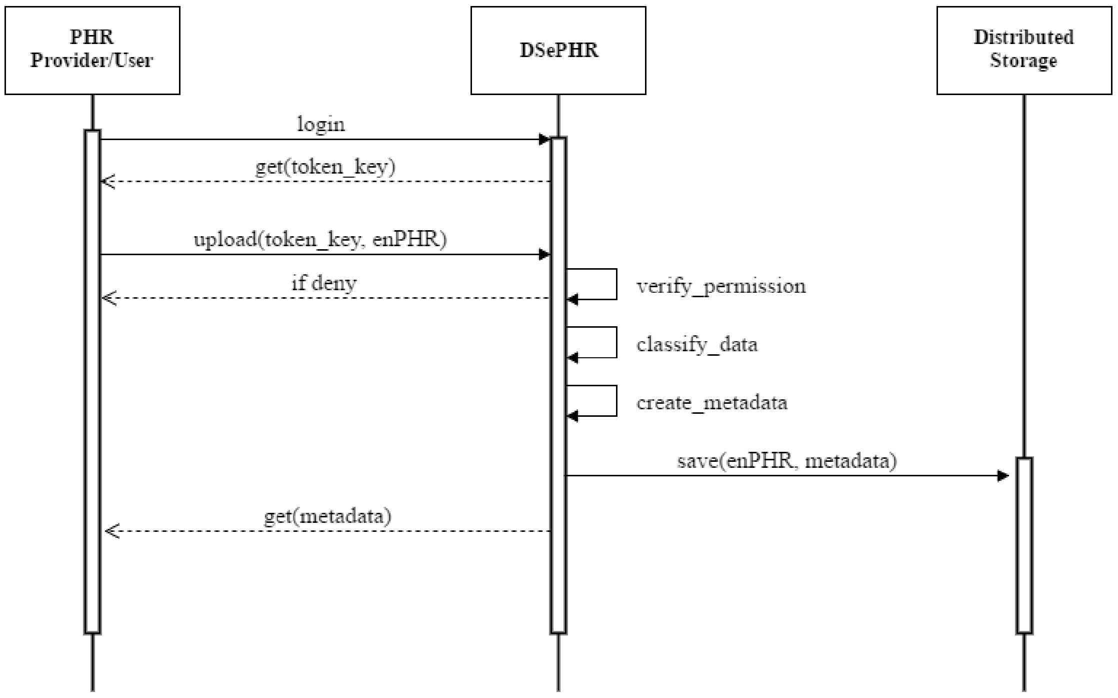


Figure 3.2 Dataflow on a write operation or PHR upload process

The dataflow on a read operation is shown in Figure 3.3. Similarity, the user must pass the authentication process first. To download the PHR, the user must provide the token key and the data id as the download request. The DSePHR verifies the token key. If the token key is valid, the DSePHR will send a request to the distributed storage and download the data from the distributed storage and send the data back to the PHR user.

For experimental purpose, the current DSePHR prototype requires the data id inside the request. In the real usage, a process for the user to select a request PHR must be done in order to retrieve the correct data ID of the request. This step can be done using the DSePHR APIs provided.

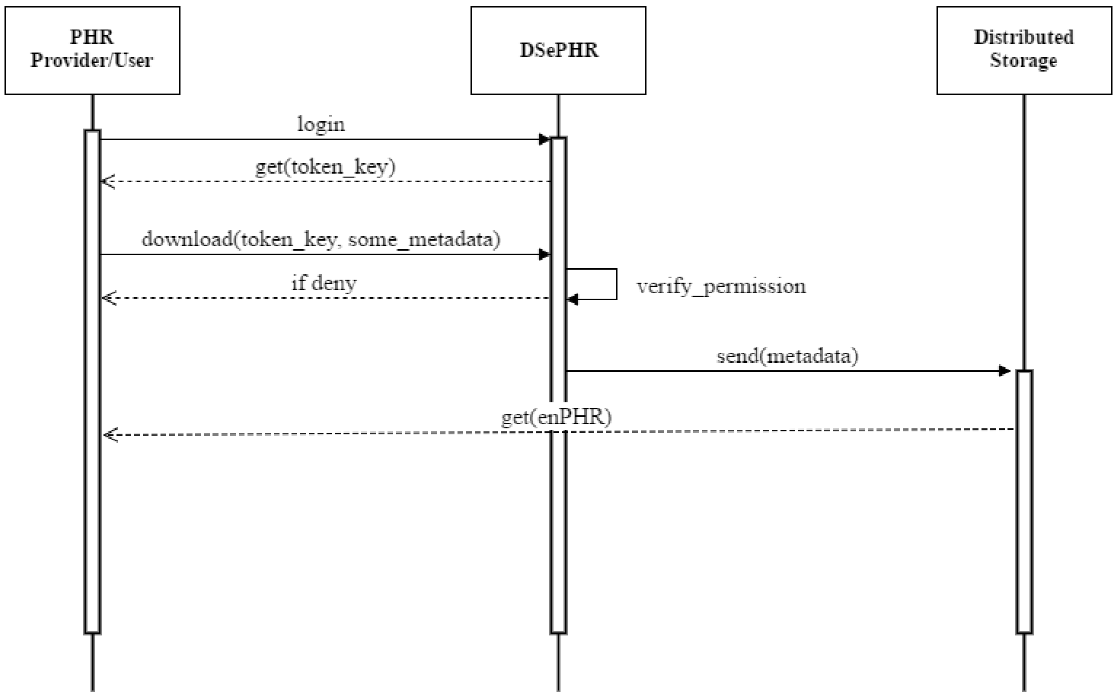


Figure 3.3 Dataflow on read operation

3.2 DSePHR API

DSePHR API is developed for users, PHR systems and applications that require to store encrypted PHR data to a distributed storage. DSePHR provides ability to store the data, to index the data and to create metadata of the encrypted PHR data automatically. The users of the DSePHR do not require to know how the data is stored in the distributed storage. The users use the API to upload their health data to the distributed storage and the DSePHR will return a data id and information of its data to the user. The data id is used to indicate and retrieve the data when the user requires. For more system feature details, will be described in next section.

3.2.1 Encrypted Communication and Authentication

The communication between a user and DSePHR is secured and only authorized user can access the data. DSePHR enables https with TLS 1.2 to perform the communication. Token based authentication is adopted in DSePHR. The DSePHR forces all incoming requests to be done according to https mechanisms and token based authentication. Therefore, the encrypted PHR data and its metadata will not be revealed or accessed by any unauthorized user, assuming all mechanisms are not compromise.

3.2.2 Retrieving and Storing Data

Users can store and retrieve their data from the distributed storage using DSePHR. DSePHR does not limit the exact size of files to be stored on the distributed storage. The distributed storage is optimized for storing both small size files and large size files without any memory issue, that is a concern in other distributed storage as discussed in Section 2.3. When the user uploads the data using the DSePHR, the API will return the data id to the user. The data id is used to indicate the required data in the DSePHR and to download the data from the DSePHR.

3.2.3 Data Accuracy

The user can ensure the data accuracy which are downloaded from the distributed storage using DSePHR. If the data in the distributed storage is incorrect, the distributed storage has a mechanism to recovery the data. When the user downloads the data from the distributed storage, DSePHR provides a data accuracy verification to check the downloaded data by means of a hash value. Since there is at least 3 replicates on the system, the DSePHR can retrieve and recover the data from another replicate.

3.2.4 Fast Search the Encrypted PHR Data

Search any encrypted PHR data is difficult because the data is lack of any related information. However, the proposed system, DSePHR, provides some related metadata of the encrypted PHR data for searching. Section 3.3.2 provides the details of all metadata stored by the DSePHR.

3.2.5 Editing the Uploaded Data

In order to update or edit any existing PHR data in the DSePHR, there are two cases. The first case is to update the data. The second case is to update only some description of the existing data. For updating the data, the user must replace the existing data by specify the data id. For updating the metadata, the data will not be touched. Only the metadata will be updated.

3.3 DSePHR Design

The architecture of the DSePHR will be described in this section including architecture and metadata design while the storage design is presented in Section 3.4.

3.3.1 DSePHR Architecture

The DSePHR architecture is shown in Figure 3.4, consisting of two main parts including my proposed API and Hadoop as a framework for our distributed storage. My proposed API contains four components, including access interface (AcI), authentication (AUTH), encrypted data manager (EDM) and metadata manager (MM). Hadoop contains two components, including Hadoop distributed file system (HDFS) and HBase. For the connection path, the user of the DSePHR will connect to the system via AcI. The encrypted PHR data will be sent over the HTTPS (HTTP with TLS) to the AcI and the AUTH will verify the request permission. The AcI classifies the request. If the request is to upload the data, the AcI make a call to MM for generating a metadata and writing the metadata to HBase. EDM is also called by the AcI for storing the data on HDFS. For a download request, the MM is used for searching the data corresponding to the requested metadata. The EDM pulls the data from Hadoop and sends the data to the requester.

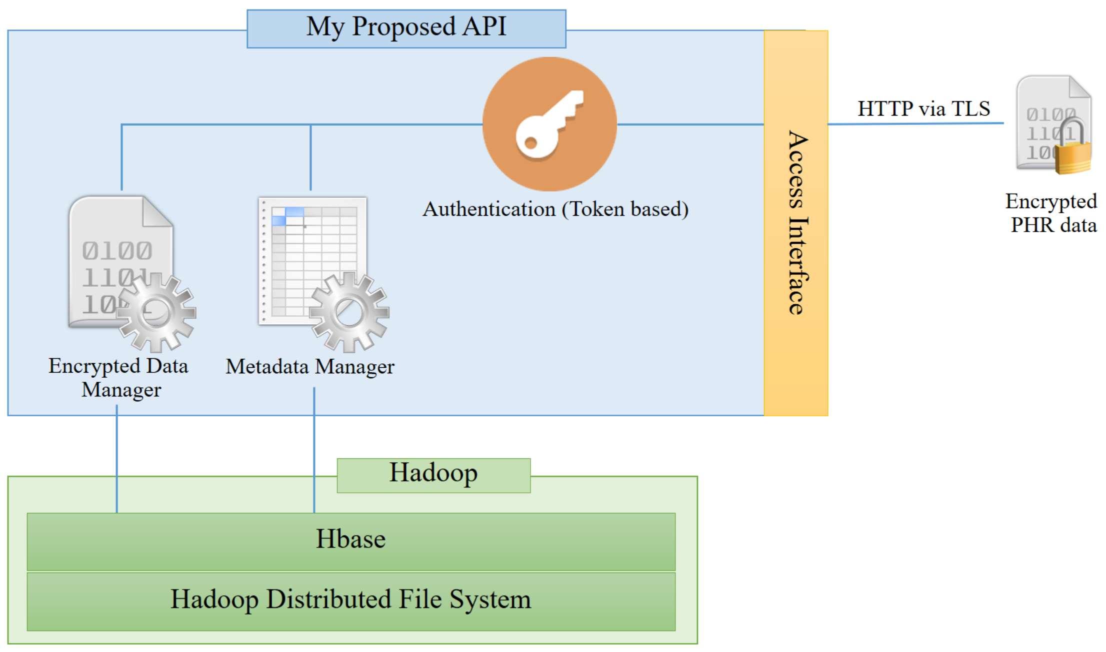


Figure 3.4 DSePHR Architecture

The flowchart of the DSePHR operations is shown in Figure 3.5. First, DSePHR receives an incoming request from a client. DSePHR will classify a type of the incoming request. For an incoming upload request, DSePHR will (1) receive the data from the incoming request and write the data to its DSePHR disk, (2) generate the metadata of the data, (3) add the data to the uploading queue to wait for storing to the distribution storage and (4) send the metadata back to the client. When the uploading queue is not empty, the uploading queue checks the data size. The data will be stored on HDFS if the data is large and stored on HBase otherwise. However, the metadata of the data will be stored on HBase. After storing the data, DSePHR will delete the data from its disk and check the data in the uploading queue again. For an incoming download request, DSePHR will retrieve the metadata of the incoming download request from HBase first to indicate the location of the request data. If the requested data is on HDFS, DSePHR will retrieve the data from HDFS. If the requested data is on HBase, DSePHR will retrieve the data from HBase. After retrieving the data, DSePHR will store the requested data to its disk first, then send the data back to the client.

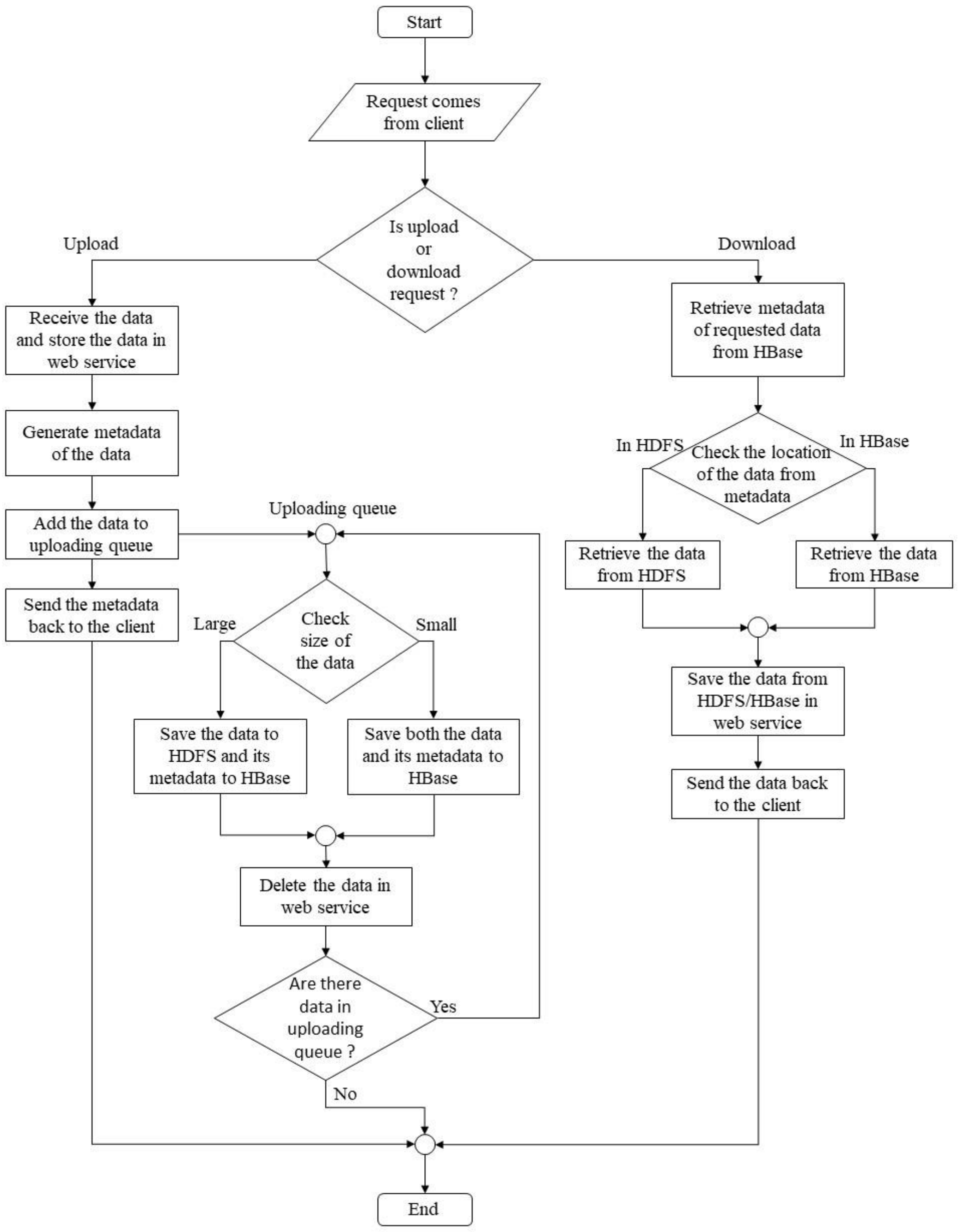

Start
Request comes from client
Is upload or download request ?
Upload
Download
Receive the data and store the data in web service
Generate metadata of the data
Add the data to uploading queue
Send the metadata back to the client
Uploading queue
Check size of the data
Large
Small
Save the data to HDFS and its metadata to HBase
Save both the data and its metadata to HBase
Delete the data in web service
Are there data in uploading queue ?
Yes
No
Retrieve metadata of requested data from HBase
Check the location of the data from metadata
In HDFS
In HBase
Retrieve the data from HDFS
Retrieve the data from HBase
Save the data from HDFS/HBase in web service
Send the data back to the client
End

Figure 3.5 DSePHR working flowchart

Another design challenge is that the encrypted PHR data is hard to preprocess or categorize. The DSePHR does not understand the encrypted PHR data because the information concerning the data is hidden. There are three properties of the encrypted data that are explicit: owner of the data, size, incoming time of the data. The DSePHR uses those explicit properties to design the database schema and the distributed storage for conveniently accessing the data by its members.

The design of the distributed storage for storing encrypted PHR data contains 2 parts including metadata design and storage design. The part of metadata design describes how to adopt the encrypted PHR data properties to index the data for convenient accesses. The part of storage describes how to store small and large files by avoiding a high memory consumption issue of distribution storage (HDFS).

3.3.2 Metadata Design

In this part, the necessary metadata for the encrypted PHR data are identified. HBase is used for storing the whole metadata of the DSePHR. The list of metadata includes user id, system id, timestamp, data id, name of the data, checksum value, size of the data, HDFSpath and description. The user id, system id, timestamp and data id are gathered and it is used as a rowkey. The remaining metadata are used as the column qualifier that persists in the column family named properties. The metadata schema is shown in Table 3.1 and the details of each metadata are described in Table 3.2.

Table 3.1 MetaTable Schema

Rowkey	properties				
<UserId-SysId-Timestamp-DataId>	name	checksum	size	HDFSpath	description

Table 3.2 MetaTable Description

ColumnFamily:ColumnQualifier	Description
UserId (Rowkey)	The user id of the data uploader
SysId (Rowkey)	The system id used by the data uploader
Timestamp (Rowkey)	Timestamp when the data arrive at the DSePHR
DataId (Rowkey)	Data id is a random gernerated by uuid4
properties:name	The name of the data file
properties:checksum	Checksum (SHA-3) value of the data
properties:size	Size of the data file
properties:HDFSpath	Location of the data file in HDFS (only large file)
properties:description	Description of the data file

The rowkey design is the important part of the distributed storage, because the data in HBase can be accessed by using the rowkey only. The rowkey must be designed in corresponding to the PHR access pattern. The owner of the PHR data can access the encrypted PHR data in the distributed storage by specifying time range of the data. The rowkey is composed of user id, system id, timestamp and data id respectively. The design of the rowkey is shown in Figure 3.6. The rowkey in HBase is in lexicographical order. Thus, the data in HBase are sorted by first the user id and then the system id. The data of the same user and the same system id are resided close to each other and sorted by time. The advantage of this design is when the user requires to access his/her data, the user can access it from one place. Accessing the data using a range of time is convenient because the data are already sorted by time.

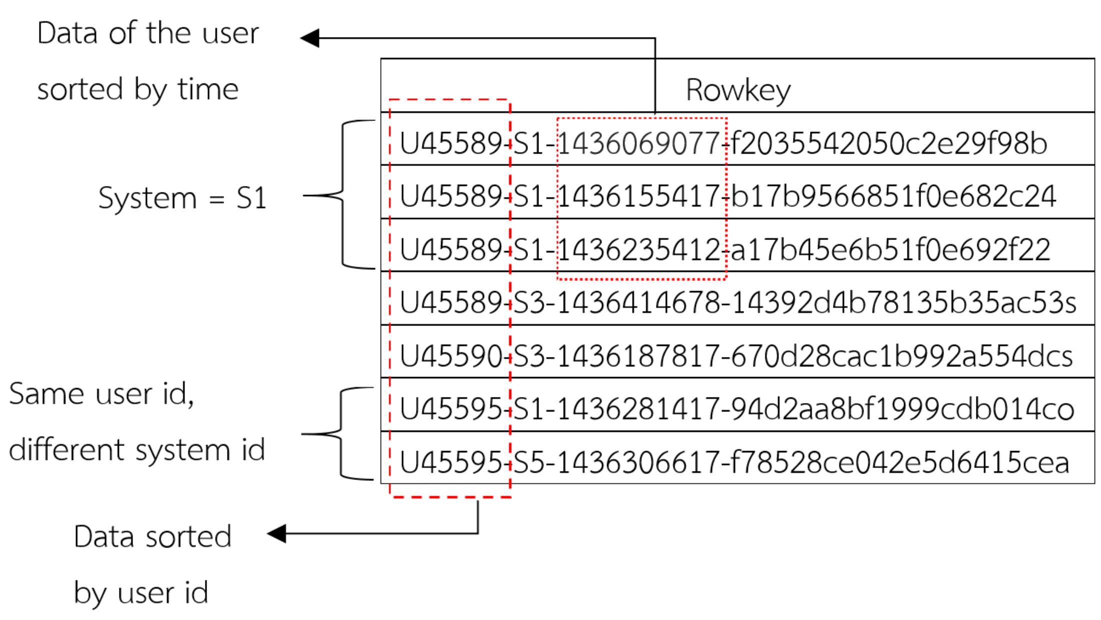

Data of the user
sorted by time
Rowkey
System = S1
U45589-S1-1436069077-f2035542050c2e29f98b
U45589-S1-1436155417-b17b9566851f0e682c24
U45589-S1-1436235412-a17b45e6b51f0e692f22
U45589-S3-1436414678-14392d4b78135b35ac53s
U45590-S3-1436187817-670d28cac1b992a554dcs
Same user id,
different system id
U45595-S1-1436281417-94d2aa8bf1999cdb014co
U45595-S5-1436306617-f78528ce042e5d6415cea
Data sorted
by user id

Figure 3.6 Rowkey Design

3.4 Storage Design

In the part of the storage design, the storage is divided to 2 parts including HDFS and HBase. Large files are stored on HDFS and small files are stored on HBase. The reason to store the small files on HBase is to eliminate the high memory consumption of Namenode as discussed earlier in Section 2.4. By the storing small files on HBase, the HBase optimization is required to achieve the performance.

3.4.1 HDFS Storage

The HDFS storage will store only large file. That is, the data that is larger than the cut-point between small and large file is directly saved to HDFS. The cut-point in this work is 10MB (10MB is the default maximum HBase cell). The value can be configured. This metadata is stored in HBase using the same schema as the metadata design shown in Table 3.1.

3.4.2 HBase Storage

Storing a lot of small files in HDFS can cause the high memory consumption problem. For instance, 10 million of 1 KB files take memory around 4GB while 128 MB of 800 files take memory around 320 KB, Although both situations take an equal capacity of the disk space. Storing small files in HBase is an appropriate solution because HBase has a compaction operation to merge many small files to a large file to be stored on HDFS automatically. The schema to store small files using HBase is shown in Table 3.3. A binary large object (BLOB) technique is adopted in HBase. The DSePHR stores a binary file in column qualifier named “ data” and column family named “EncryptedData”. The rowkey used is as same as that of the metadata design.

Another advantage of storing small files in HBase is saving time to implement the file merging operation and a CRUD operation, which stands for create, read, update and delete operations. Merging files is needed for gathering small files to a large file. The less number of files can reduce the Namenode memory. Storing small files on HDFS without HBase, merging method and CRUD operations are needed because accessing the small file in merging file cannot be performed without CRUD operations. By using HBase method, HBase merges small files automatically and also provides CRUD operation to access the data.

Table 3.3 EncTable Schema

Rowkey	EncryptedData
<UserId-SysId-Timestamp-DataId>	Data

3.4.3 HBase Optimization

HBase does not mainly design for BLOB storage but HBase is able to store the data in binary form. HBase can store both text data and binary data. Binary data such as the encrypted PHR data is large when it is compared with text data. Storing the data as BLOB can be considered a high data loading operation. Therefore, the HBase optimization is necessary. There are 2 parts of optimization including HBase configuration tuning and presplit table.

The HBase configuration tuning is an easy operation to support high write throughput. Most configurations are in hbase-dir/conf/hbase-site.xml and some configurations are in hbase/conf/hbase-env.sh. The explanation of each configuration is shown in Table 3.4.

Table 3.4 HBase Configuration Description

Configuration Name	Description	Default	Adjust
Heapsize*	Memory size of the JVM that can be used by HBase.	1 GB	5.5 GB
hbase.regionserver.global. memstore.size	Size of the whole memstores (write cache)	0.4 (40%)	0.6
hfile.block.cache.size	Size of the block cache (read cache)	0.4 (40%)	0.2
hbase.hregion.memstore. flush.size	Size of the memstore	128 MB	256 MB
memstore.block. multiplier	Block the data when size of the whole memstore exceed (memstore.block. multiplier x hbase.hregion.memstore. flush.size)	2	4
compactionThreshold	Number of hfiles to perform compaction operation	3	15
blockingStoreFiles	Block the data when the number of hfiles exceed the threshold.	10	25
-XX:CMSInitiating- OccupancyFraction*	Amount of the data in the heapsize that encourage the garbage collector to work.	92 %	70 %

* configurations are in hbase-env.sh

Pre-split table is an important technique to spread the incoming data to every node. Basically, HBase starts with a single region as a table. When the size of the data in the table increases and reaches the threshold, HBase will split the region to two regions. At the start, the whole incoming requests to HBase will go to a single machine which can cause a bottle neck problem. A presplit table can provide a solution to such problem. The pre-split mechanism divides regions to distribute the data among machines from the start. The presplit table can perform using HBase shell command when the table is created. For instance, “ create 'MetaTable', 'properties', {SPLITS => ['u1','u4', 'u7']} ; create 'EncTable', 'EncryptedData', {SPLITS => ['u1','u4', 'u7']}". A text "'u1','u4', 'u7'" is a split point. The recommended number of split point is a number of region server minus one.

3.5 Synthetic PHR Workloads

Since there is no standard for PHR workloads, this thesis also proposes a set of synthetic PHR workload based on real data files downloaded and collected from various sources. According to 11 types of PHR originally given in [3] and discussed again in Section 2.1, most PHR data are in the form of a document file such as continuity of care document (CCD) file, continuity of care record (CCR) file, and a document file type. In addition, the PHR data can exist in a form of a media file such as an image file (e.g., MRI, X-ray or ECG image file), an audio file (e.g., a doctor visit conversation) and a video file (e.g., an operation video or a healthcare instruction video). As a result, various document and media file sizes and types are downloaded as a PHR workload collection.

In this thesis, there are a total of 12 files including document files and media files, collected as a set of synthetic PHR workload as a representative of the PHR data types discussed above. Each data is explained below:

- A MRI image file: a 20KB jpeg file collected from imaging.cancer.gov as a representative of the patient information as an overview, while the large image file size will also be stored in the PHR system for details information. This file also represents other small size image files to be stored on the PHR system.
- A CCD file: a 27KB xml file collected from www.ehrdoctors.com as a representative of the several PHR data type discussed in Section 2.1 such as

problem list, procedure, major illness, provider list, family history, medications, immunizations, and laboratory test. This file represents a small size CCD file.

- A patient information file: a 30KB xlsx file created as a representative of several PHR data type. This file represents a small size document file.
- A heartbeat sound file: a 154KB ogg file collected from commons.wikimedia.org as a representative of a media file type such as the home-monitored data, the laboratory data and the family history data.
- An ECG picture file: a 393KB jpg file collected from en.ecgpedia.org as a representative of a media file type such as the home-monitored data, the laboratory data and the family history data.
- A patient information file: a 431KB docx file created as a representative of several PHR data type. This file represents a medium size document file.
- A CCD file: a 617KB xml file collected from www.myhealth.va.gov as a representative of a CCD file of different size and type.
- A CCD file: a 679KB pdf file collected from www.myhealth.va.gov as a representative of a CCD file of different size and type.
- A X-ray file: a 4MB png file collected from commons.wikimedia.org as a representative of a large image file.
- An audio file: a 8.65MB mp3 file created as a representative of a voice conversation file.
- A video file: a 27MB mp4 file of a standard-definition video collected form youtube.com as a representative of a small video file.

- A video file: a 232MB mp4 file of a high-definition video collected form youtube.com as a representative of a large video file.

Table 3.5 shows a summary of all twelve file including the size, the type for the source of each file. Since the PHR in this thesis is assumed to be encrypted from the source, all files are encrypted using the encryption presented in [10]. The first column shows the size of the original file (before the encryption process). The second column shows the size of the encrypted file. The third column shows the overhead produced by the encryption process on each file. The fourth column shows the type of files. The last column shows the description of the file including the detail of the file and the source of the file.

Table 3.5 Workload Files

Original filesize	Encrypted filesize	Increased size from original size	File type	Description
20 KB	30 KB	10 KB	JPG	MRI image Source: imaging.cancer.gov
27 KB	31 KB	4 KB	XML	CCD example Source: www.ehrdoctors.com
30 KB	41 KB	11 KB	XLSX	Patient information
154 KB	160 KB	6 KB	OGG	Heartbeat 66bpm sound Source: commons.wikimedia.org
393 KB	400 KB	7 KB	JPG	ECG picture graph Source: en.ecgpedia.org
431 KB	440 KB	9 KB	DOCX	Patient information
617 KB	620 KB	3 KB	XML	Large CCD example Source: www.myhealth.va.gov
679 KB	690 KB	11 KB	PDF	CCD PDF example Source: www.myhealth.va.gov
4.00 MB	4.01 MB	11 KB	PNG	Chest X-ray PA image (4.7MP) Source: commons.wikimedia.org
8.65 MB	8.65 MB	7 KB	MP3	Conversation sound 9 minute
27 MB	27 MB	9 KB	MP4	SD Video of operation 9 minute Source: youtube.com
232 MB	232 MB	11 KB	MP4	HD Video of operation 17 minute Source: youtube.com

All files can be classified into 4 types by its size including very small size (less than 100 KB), small size (100KB -1MB), moderate size (1MB – 10MB) and large size (more than 10MB). Base on the collection above, the size distribution of the synthetic PHR workload is shown in the Figure 3.7

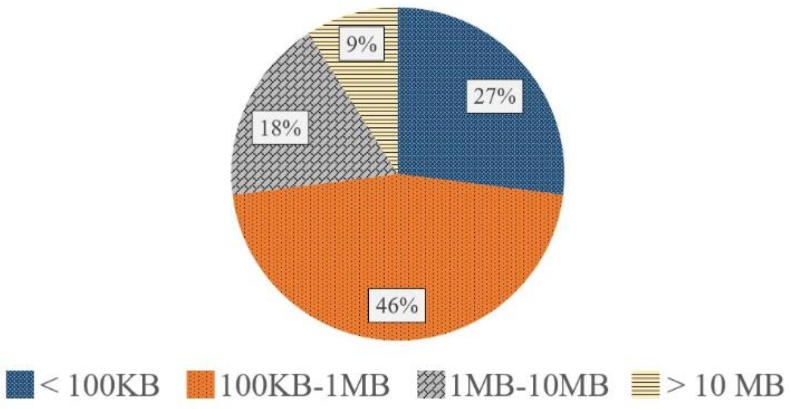


Figure 3.7 PHR workload size distribution

The discussion above only gives the details of the PHR data of the proposed synthetic workloads. To conduct an experiment in this thesis, the user must perform a download or a download of a file in the PHR workload. Therefore, the mixture of read and write requests on each file in the PHR workload is another parameter to be considered. According to the discussion in Section 2.2, the PHR requests contain a large number of write requests than that of the read requests. Therefore, the ratio of write to read requests in the experiment includes 100:0, 75:25 and 50:50.

3.6 Experimental Design

DSePHR is developed for storing and retrieving encrypted PHR data by designing an index from the data attributes and solving the memory issue of the distributed storage when storing a lot of small size files. The experiments are designed to (1) evaluate the memory consumption of the DSePHR and (2) to evaluate the DSePHR performance on various situations. The detail of experiments will be described in the next section.

3.6.1 Memory Consumption Issue

Although, HDFS has no limitation on the file size for storing but storing a lot of small size files can cause a memory consumption on the Namenode as discussed in Section 2.4. The DSePHR is mainly designed for storing the PHR data which contains both small files and large files. The result of DSePHR approach will show the DSePHR can solve the high memory consumption of HDFS when it stores a lot of small files. In this experiment, the memory consumption of the Namenode between traditional HDFS and DSePHR will be compared. A number of 25,000 to 100,000 files will be fed to both traditional HDFS and DSePHR.

The experiment is conducted on 13 machines. The detail of the experimental setup is shown in Table 3.6.

Table 3.6 Setup Detail in Memory Issue Experiment

Machine Name	Service	Specification
Namenode (1 machine)	Namenode of HDFS HQuorumpeer (Zookeeper)	OS: Ubuntu 14.04.3 LTS CPU: Core-i5 3740s 2.9Ghz HDD: 320 GB (7200 RPM, 64MB cached) RAM:DDR3 8 GB All machines are physical machine and connected via a local area network 100 Mbps on Cisco Catalyst 2960-48TT-L Switch. Total: 13 machines
Datanode (9 machines)	DataNode HRegionServer (Region Server)	
Master2 (1 machine)	Hmaster HQuorumpeer SecondaryNameNode	
WebService	The DSePHR Service API	
Measurement Machine	DSePHR client	

The PHR workload shown in Table 3.5 is used in this experiment. The PHR workload is uploaded to the traditional HDFS and the DSePHR at the same time. When the number of files reach 25,000, 50,000, 75,000 and 100,000, the memory usage at the Namenode of the traditional HDFS and the DSePHR is calculated based on the number of files in the system.

3.6.2 DSePHR Performance

In this experiment, DSePHR will be evaluated during operations. The experiments are divided into four parts including baseline experiment, effect of limited storage space, effect of write-read request ratio and effect of the file type mixture. The objective of baseline experiment is to measure the performance of DSePHR to perform an upload and a download operation on an empty system. That is, the DSePHR will only service the request command alone. This way, the performance of this part can be used as a reference point for other situations. The remaining three experiments are conducted to evaluate the effect of other factors on the DSePHR performance. The details of each experiment are given next.

3.6.2.1 Baseline Experiment

Under the baseline experiment, each PHR file of the synthetic workload described in Section 3.5 is uploaded to and download from the DSePHR. The process is done when the DSePHR is empty. Thus, the DSePHR will only service the request. As a result, the performance observed here can be used for comparison with the result of the remaining experiments.

There are 3 storage nodes, 1 web service and 1 client to conduct the experiment. Every machine is connected by LAN 100 Mbps fast Ethernet. The number of storage nodes is 3 because HDFS makes 3 replicas of its data. The client uploads to or downloads from the single web service, then a web service will store to or retrieve from storage nodes. The overview of the baseline experiment is shown in Figure 3.8 and more detail of each machine is shown in Table 3.7.

The workload used in this experiment is the PHR workload presented in Section 3.5. For the file size less than 1 MB, the file will be repeatedly uploaded or downloaded for 10 times. For the 3xray.png which is a 4.01 MB, the file will be repeatedly uploaded or downloaded twice. Other files will be uploaded or downloaded only 1 time. The repeating number is considered by the multiplication of repeating number and file size that is not over 10 MB due to transmission rate of LAN 100 Mbps fast Ethernet. For the repeated files, the average value will be used for the result of each file instead of the single value.

The experiment will be repeated for 3 times and the average value is calculated. The storage nodes will be formatted, then each file will be uploaded to the storage. If the repeating number is 10, the file will be continually uploaded to the storage 10 times. For the download measurement, each file will be uploaded to the storage first for repeating number times, then each file will be randomly selected to download for repeating number times.

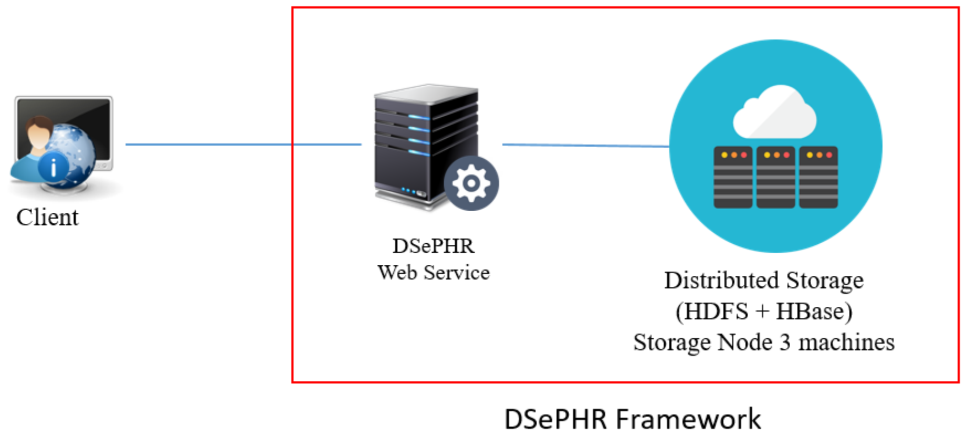


Figure 3.8 Overview experiment setup for workload baseline experiment

Table 3.7 Experiment setup detail of workload baseline experiment

Machine Name	Service	Specification
Namenode (1 machine)	Namenode of HDFS	OS: Ubuntu 14.04.3 LTS CPU: Core-i5 3740s 2.9Ghz HDD: 60 GB (7200 RPM, 64MB cached) RAM: DDR3 8 GB All machines are physical machine and connected via Lan 100 Mbps with Cisco Catalyst 2960-48TT-L Switch. Total: 10 machines
Storage Nodes (3 machines)	DataNode HRegionServer (Region Server)	
Zookeeper Node (1 machine)	HQuorumpeer	
SecondaryNameNode (1 machine)	SecondaryNameNode	
Master2 (1 machine)	Hmaster	
DSePHR web services	The DSePHR Service API	
DNS server	Bind9	
Client	The client to write or read the data to DSePHR	

3.6.2.2 Effect of the Limited Storage Space

In this experiment, the DSePHR performance are measured when the storage space is running out. This experiment is conducted on a small size cluster (3 storage nodes).

The experiment setup is shown in Figure 3.9. According to Figure 3.9, a number of clients, a number of DSePHR web services and a number of storage nodes are presented by N, X and Y respectively. Clients will upload the data to or download the data from DSePHR web services. The DSePHR web services will store the data to or retrieve the data from the distributed storage. The detail of each machine is shown in Table 3.8

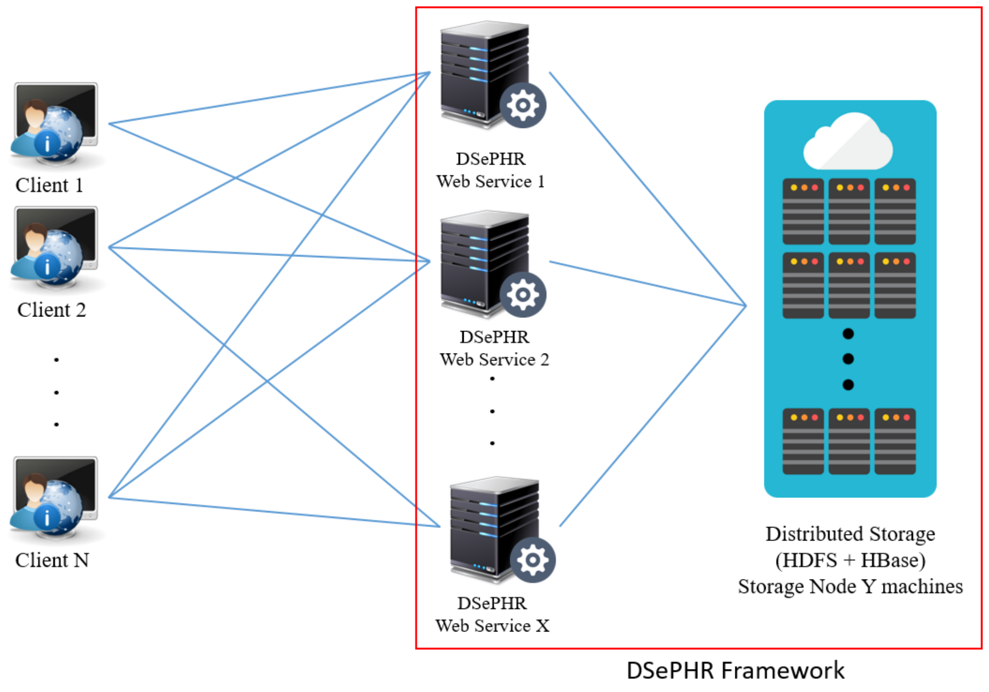


Figure 3.9 Effect of limited storage space and effect of the write-read request ratio experimental setup

Table 3.8 Effect of limited storage experimental setup details

Machine Name	Service	Specification
Namenode (1 machine)	Namenode of HDFS HQuorumpeer (Zookeeper)	OS: Ubuntu 14.04.3 LTS CPU: Core-i5 3740s 2.9Ghz HDD: 60 GB GB (7200 RPM, 64MB cached) RAM:DDR3 8 GB All machines are physical machine and connected via Lan 100 Mbps with Cisco Catalyst 2960-48TT-L Switch. Total: 20 machines
Storage Nodes (3 machines)	DataNode HRegionServer (Region Server)	
Master2 (1 machine)	Hmaster HQuorumpeer SecondaryNameNode	
DSePHR web services (6 machines)	The DSePHR Service API	
DNS server (1 machine)	Bind9	
Clients (8 machines)	The client to write or read the data to DSePHR	

The synthetic PHR workload described in Section 3.5 is used as the workload in this experiment. The mixture of the file type is as described in Section 3.5 while the ratio of write-read request is 100:0, 75:25 and 50:50. The workload in this study consist of 6,000 files for each client. Therefore, there are 48,000 files total from 8 clients.

The workload will be fed to the DSePHR until the system is almost running out of resources. After that, the capacity will be added to the system afterward. The workload is fed continually to the DSePHR until the system runs out of resource. The measurements during this experiment include the amount of data on each client and each storage node. The measurement points are the system resource usage at 75%, 80%, 85%, 90% and 95%. The response time of the DSePHR in using the additional storage nodes is also measured.

3.6.2.3 Effect of the Write-read Request Ratio

In this experiment, DSePHR will be tested on a larger scale with a warm-up and cool-down period in order to simulate the background work and the measured work. Since the PHR data source can be wearable devices, mobile phones and users from PHR system [30], [31], there is a lot of data that can be exported from sensors of such device. Resulting in a higher number of write-request than that of read-request. Three situations are mimicked such situations by using a ratio of write to read request of the workload as 100:0, 75:25 and 50:50.

The large cluster is used to measure the DSePHR performance when the DSePHR handles both upload and download requests from clients. The large cluster has a number of storage nodes more than the small cluster. A high number of storage nodes means that the DSePHR can support high amount of throughput because there are many nodes to support a lot of incoming requests. The incoming requests can be distributed to all storage nodes. The detail of machines used in the experiment shows in Table 3.9.

The overall picture of the experimental setup is shown in Figure 3.9 as same as the effect of limited storage space experiment. There are three main parties including client, DSePHR web service and distributed storage. Assuming the clients are real users who want to store or retrieve the data from DSePHR. The clients are used for accessing the DSePHR web service including write-operations and read-operations. All clients will know a list of DSePHR web service by getting it from the DNS Server. The client can directly send the request to each DSePHR with the DNS round robin mechanism. Each DSePHR web service processes the incoming requests and saves the data to the distributed storage. The distributed storage contains many machines for supporting the data from DSePHR web service.

Table 3.9 Effect of the write-read ratio and the mixture of file type

Machine Name	Service	Specification
Namenode (1 machine)	Namenode of HDFS HQuorumpeer (Zookeeper)	OS: Ubuntu 14.04.3 LTS CPU: Core-i5 3740s 2.9Ghz HDD: 460 GB (7200 RPM, 64MB cached) RAM:DDR3 8 GB All machines are connected via Lan 100 Mbps. Total: 32 machines
Storage Nodes (15 machines)	Datanode HRegionServer (Region Server) HQuorumpeer (Zookeeper)* *only machine#1, #5, #9	
Master2 (1 machine)	Hmaster HQuorumpeer SecondaryNameNode	
DSePHR Web service (6 machines)	The DSePHR Service API	
DNS Server (1 machine)	BIND9	
Clients (8 machines)	The client to write or read the data to DSePHR	

The workload described in Section 3.5 is used in this experiment. Each client has 6,000 files and there are 8 clients. Resulting in 48,000 files. However, the first 500 files of each client will be used as the warm-up files. That is, the performance of these files is not included in the result. Each file in the set of 6,000 files will be set as either read or write according to the write-read ratio. For example, the 100:0 ratio will result in 6,000 write requests. That is, all files are write requests. For the 75:25 ratio, there are 4500 write requests and 1,500 read requests. Since the first 500 files are the warm-up files and the last 500 files are the cool-down files. The measurement files are the 5,000 files in the middle. A throughput of each client is measured in megabyte per second. The throughput of each storage node will also be measured. The throughput results will show on both write and read operation.

3.6.2.4 Effect of File Type Mixture

The synthetic PHR workload described in Section 3.5 consists of 73% small files (i.e., the file size is less than 1 MB) and only 9% of files have the size larger than 10 MB. Thus, 91% of the files will be stored on HBase while only 9% of the files will be stored on HDFS due to the DSePHR design. In the future, however, the data source may be able to generate more large files. Thus, this experiment changes the mixtures of the PHR file type to increase the number of large files in order to evaluate the DSePHR performance.

All experiment settings are exactly the same as that of section 3.6.2.3 with the exception of the workload. The workload in this experiment contains 75% of small files and 25% of large files. While, the write-read ratio of the workloads is 100:0, 75:25 and 50:50.

CHAPTER 4

RESULTS AND DISCUSSIONS

This chapter presents the details of the DSePHR APIs and the experimental results including DSePHR Namenode memory usage, the workload baseline performance, the effect of limited storage space, the effect of varying write-read request ratio of the workload, and the effects of file type mixture of the workload.

4.1 DSePHR API Description

The DSePHR API is developed using REST style architecture on HTTP. The use of DSePHR API is described in Table 4.1. Column "method" shows a HTTP method, "URI" stand for Uniform Resource Identifier to the API operation, Parameter and description explain the detail of the URI. Every request requires to attach a token key for authentication by determining HTTP request as "header={'Authentication-Token':token_key}". The token key can be found from the login URI.

Table 4.1 DSePHR API Description

Method	URI	Parameter and description
POST	/login	**Description :** sign in to the system and get a token key to be attached with the request. Send data by JSON style **Parameter :** email, password email : registered email password : used password example {"email":"system1@example.com", "password": "system1"}

Table 4.1 DSePHR API Description (cont.)

Method	URI	Parameter and description
GET	/logout	**Description :** sign out from the system
POST	/upload	**Description :** use to store the data to a distributed storage. If uploading success, the system will return the metadata using JSON Send data by FORMDATA style **Parameter :** sysid, userid, file, timestamp, description sysid : registered system id such as “s1” userid : registered user id such as “u1252” file : the health data to store timestamp : time of the data. If does not specify, time is current uploaded data time. description : description of the data
GET	/download/<rowkey>	**Description :** use to retrive the data from the distributed storage. **Parameter :** rowkey *The rowkey get from /upload when the upload operation is successful
POST	/search	**Description :** use to search the data in the distributed storage. Send data by FORMDATA **Parameter :** sysid, userid, filename, starttime, endtime, description sysid: specify system id of the data to search userid: specify user id of the data to search starttime: specify start of time of the data endtime: specify end of time of the data * Both starttime and endtime can specify together.
GET	/infor/<rowkey>	**Description :** Show information concerning the data. **Parameter :** rowkey

4.2 Memory Consumption Issue

This experiment is conducted to observe the DSePHR memory behavior when the DSePHR handles a lot of data. A number of files in the system is 100,000 files with 1,200 GB of disk capacity. Encrypted PHR data with a lot of small files especially the document file type can cause a high memory consumption on the Namenode of the HDFS system. The proposed DSePHR, however, can reduce the memory consumption problem.

This experiment is performed to measure the memory consumption of the Namenode when stores a lot of files. Table 4. 2 shows the memory consumption of the proposed DSePHR and the HDFS system when the number of input files is at 25,000, 50,000, 75,000 and 100,000 files.

Table 4.2 Memory comparison between DSePHR and traditional HDFS

A number of files in the system	Systems	
	DSePHR (MB)	Original HDFS (MB)
0	56.00	56.00
25,000	57.60	72.20
50,000	59.10	83.40
75,000	60.58	104.59
100,000	61.16	120.79

The memory consumption is calculated using the formula given in [25]. At 0 file point, HDFS takes an initial memory of 56 MB. According to the results shown in Table 4.2, 100,000 files in the system, the proposed DSePHR consumes the amount of memory similar to the initial amount while the HDFS consumes almost twice the initial amount. The key solution of the proposed DSePHR is to store a lot of small files using the HBase compaction mechanism. Thus, HDFS can avoid the memory consumption issue because a lot of small files are packed in order to reduce the number of files. Without this approach, the original HDFS suffers a high memory consumption problem and lead to a state of unavailable if a large number of small files is sent to be stored, which is an important point because the encrypted PHR data is mostly a small file type such as document.

In conclusion, the experimental results show that the DSePHR can store and retrieve the data without a Namenode memory issue observed under the original HDFS system.

4.3 DSePHR Performance

To observe the DSePHR system performance, four experiments are conducted. The first experiment is to measure the real performance of DSePHR (baseline), the second experiment is the effect of limited storage space performance, the third experiment is the effect of write-read ratio and fourth experiment is the effect of file type mixture. The results of the baseline experiment, effect of limited storage space, effect of write-read ratio and effect of file type mixture are presented in Section 4.3.1, 4.3.2, 4.3.3 and 4.3.4 respectively.

4.3.1 DSePHR Workload Baseline

According to the DSePHR policy, small files will be stored in HBase and large files will be stored in HDFS. However, the small file can be stored in HDFS directly. To measure the performance between DSePHR policy and the storing in HDFS only, DSePHR system under DSePHR policy and storing in HDFS only are conducted. Both baseline performances use DSePHR system with different configuration. For the DSePHR policy, the small file is stored in HBase and large file is stored in HDFS. For the HDFS only, every file is stored in HDFS. Table 4.3 and Table

4.4 show the upload and download time baseline using both DSePHR policy and HDFS only. Every value in both upload and download baseline table is an average value of 3 repeating experiments. For the upload time table, the column “CL-WS” is the amount of time in seconds that it takes for the client to upload the data to the web service. The column “WS-SN” is the amount of time in seconds that it takes for the web service stores the data to the storage nodes. Due to the DSePHR policy, 11videosmall.mp4 and 12videobig.mp4 are stored in HDFS. Other files are stored in HBase. The “Gen Meta” column is the amount of time to generate a metadata. The “Repeat” column means that the file is continually upload for a number of repeat times in order to calculate the average values in CL-WS and WS-SN. The number to repeat depends on the multiplication result of the number of repeat and the file size. The multiplication result must not be over 10MB due to the transmission rate. The small file must be repeated because the small file can be easily affected by the network latency or OS service. For the download time baseline table, the “SN-WS” column is the amount of time that the storage nodes send the requested data to a web service in responding to the client request. The “WS-CL” column is amount of time that web service send the requested data to client. The “Repeat” column is times to repeatedly download similar to upload time baseline table.

According to the results shown in Table 4.3, the time values in CL-WS column are always less than that of the WS-SN column for both DSePHR policy and HDFS only. That can be explained by the fact that storing the data in a storage node requires more operations than storing the data in the web service. For example, in case of storing in HBase, the web service must retrieve a list of available region servers from the Zookeeper, then the web service will send the data to a specified region server. In case of storing in HDFS, the web service must retrieve a list of available Datanodes from the Namenode first, then the web service will send the data to a specified Datanode. The column WS-SN in HDFS only is clearly larger than that of the DSePHR policy for all small files (less than 10 MB). To store the data in a storage node, the system under the DSePHR policy spends time at least 0.7 seconds while the system under the HDFS only spends time at least 2.4 seconds. It can be concluded that storing data in HDFS takes more time than storing data in HBase for small files. Because HBase stores the data in the memory first while the HDFS stores the data in disk immediately, the DSePHR policy can reduce the time for storing the data. This supports the idea that storing the small files in HBase and storing the large files in HDFS can improve the performance of the upload time. The metadata generating time is increased with the file size because the metadata generating process must create the hash value of the file (SHA-3). To generate the hash value, the hash function must read the whole file and the time to create the hash value increases with the file size.

According to Table 4.4, the time values of SN-WS and WS-CL column in DSePHR policy are slightly different while that in HDFS only are clearly different for the file size less than 1 MB. The SN-WS of DSePHR policy is less than that of HDFS only for the file size less than 1 MB because the data will be retrieved from the memory of the storage node for the DSePHR case while, the data will be retrieved from disk of storage node. The retrieving time from the memory is smaller than the retrieving time from the disk. However, the retrieving time of the DSePHR policy is larger than that of the HDFS only when the file size is larger than 4 MB.

Table 4.3 Upload time baseline (DSePHR policy and HDFS only)

Filename	Size	Repeat	DSePHR policy			HDFS Only		
			CL-WS(s)	WS-SN (s)	Gen Meta (s)	CL-WS(s)	WS-SN (s)	Gen Meta (s)
2mri.jpg	30KB	10	0.016	0.788	0.001	0.018	2.426	0.001
4ccd.xml	31KB	10	0.016	0.786	0.001	0.018	2.432	0.001
8excel.xlsx	41KB	10	0.017	0.800	0.001	0.020	2.477	0.001
9sound.ogg	160KB	10	0.027	0.823	0.002	0.030	2.413	0.002
1ecg.jpg	400KB	10	0.049	0.947	0.003	0.053	2.517	0.004
7word.docx	440KB	10	0.063	0.979	0.003	0.056	2.482	0.004
5ccd.xml	620KB	10	0.070	0.893	0.004	0.073	2.623	0.005
6ccd.pdf	690KB	10	0.078	1.011	0.004	0.081	2.559	0.005
3xray.png	4.01MB	2	0.386	1.610	0.028	0.394	3.026	0.033
10sound.mp3	8.65MB	1	0.808	2.619	0.065	0.846	3.370	0.068
11videosmall.mp4	27MB	1	2.481	5.267	0.172	2.594	5.322	0.166
12videobig.mp4	232MB	1	21.006	24.482	1.066	21.897	24.838	1.065

Table 4.4 Download time baseline (DSePHR policy and HDFS only)

Filename	Size	Repeat	DSePHR policy		HDFS Only	
			SN-WS(s)	WS-CL(s)	SN-WS(s)	WS-CL(s)
2mri.jpg	30KB	10	0.007	0.005	0.030	0.005
4ccd.xml	31KB	10	0.007	0.004	0.018	0.005
8excel.xlsx	41KB	10	0.006	0.006	0.018	0.006
9sound.ogg	160KB	10	0.017	0.016	0.030	0.017
1ecg.jpg	400KB	10	0.038	0.038	0.051	0.053
7word.docx	440KB	10	0.042	0.042	0.053	0.042
5ccd.xml	620KB	10	0.058	0.058	0.069	0.059
6ccd.pdf	690KB	10	0.064	0.065	0.074	0.065
3xray.png	4.01MB	2	0.375	0.375	0.376	0.376
10sound.mp3	8.65MB	1	0.837	0.805	0.791	0.806
11videosmall.mp4	27MB	1	2.580	2.506	2.440	2.507
12videobig.mp4	232MB	1	20.949	21.248	20.816	21.257

Table 4.5 shows the upload and download time baseline from the client perspective. The time is measured from the point when the request is sent by the client until the client gets the response from the DSePHR web service on the

request. For the small files, the upload time is less than 1 second while the download time is less than 2 seconds. There is a linear relationship shown from the file size larger than 4MB. For example, the size of 10sound.mp3 is 2.16 times that of 3xray.png. The upload time of 10sound.mp3 is also 2.11 times that of 3xray.png. The download time of 10sound.mp3 is 2.16 times that of 3xray.png. The download time is almost twice the upload time because the web service must retrieve the data from storage node first, then the web service sends the data to the client and no metadata generating process. For the largest files, 12videobig.mp4, the upload time is less than half a minute and the download time is not over 1 minute. Both the upload and download time also show the linear relationship similar to that of the file size larger than 4 MB of the small files.

Table 4.5 Upload and download time baseline from client perspective view

Filename	Size	Upload time (s)	Download time (s)	Repeat
2mri.jpg	30KB	0.017	0.031	10
4ccd.xml	31KB	0.018	0.026	10
8excel.xlsx	41KB	0.019	0.025	10
9sound.ogg	160KB	0.029	0.047	10
1ecg.jpg	400KB	0.054	0.091	10
7word.docx	440KB	0.067	0.099	10
5ccd.xml	620KB	0.076	0.131	10
6ccd.pdf	690KB	0.084	0.144	10
3xray.png	4.01MB	0.416	0.766	2
10sound.mp3	8.65MB	0.876	1.658	1
11videosmall.mp4	27MB	2.661	5.102	1
12videobig.mp4	232MB	22.100	42.214	1

4.3.2 Effect of the Limited Storage Space

The objective of this experiment is to observe the DSePHR behavior when the DSePHR storage is almost filling up. Two issues are studied in this experiment including the limited resource performance and the DSePHR performance during limited resource.

4.3.2.1 Limited Resource Performance

This experiment aims to investigate the behavior of DSePHR when the capacity of the storage is limited, because extra capacity can be added to the system without shutting down the system. This feature provides the availability feature for the DSePHR system. Although this feature is already achieved by HDFS. To ensure the continuous operation of the system, the amount of time each extra storage is filled up and the amount of data to the DSePHR web service, the amount of data sent to each storage nodes are measured. The workload in this experiments consist of three scenarios, including 100:0, 75:25, and 50:50 write-read request ratio workloads of the file type mixture as described in Figure 3.7.

The initial capacity of the system is 60 GB of 3 storage nodes. There are eight clients feeding into the system until the system capacity is at 95%. Then, three storage nodes will be added to the system.

The experimental results show that the time for all three extra storage nodes are filled up with the data after the storage nodes are added are shown in Table 4.6. According to the results shown in Table 4.6, the system uses 1 to 3 minutes to fill up all three extra nodes.

Table 4.6 Time to fill up all three extra storage nodes

Workload	Time to fill up (minutes)
100:0-write:read	2
75:25-write:read	1
50:50-write:read	3

Another advantage of adding extra storage nodes is that the system can handle more data. Initially, there are only 3 storage nodes to support the data. After adding 3 extra storage nodes, the data can be distributed among six storage nodes. Table 4.7 and Table 4.8 show the amount of data sent to each DSePHR web service before and after the extra storage nodes are added respectively. The amount of data is measured over an hour of the experiments with 30 minutes prior to the addition of the storage node and 30 minutes after that. Column “WebService” refers to a name of the web service, “MAX” refers to maximum, “AVG” refers to average, “MIN” refers to minimum, “STD” refers to standard division, “VAR” refers to variance, “95th” and “99th” refer to 95 and 99 percentiles respectively. The average amount of data to each web services increases after the extra storage addition, except on the LAB_DS6. However, the data transmission rate of all 6 web services are increased after the extra storage addition.

Table 4.7 Amount of data to all six DSePHR web services

on 100:0-write:read on original PHR mixture before the extra storage addition

WebService	Amount of data (MB/s)						
	MAX	AVG	MIN	STD	VAR	95th	99th
LAB_DS1	2.421	1.032	0.049	0.741	0.548	2.219	2.393
LAB_DS2	3.308	1.408	0.380	0.966	0.934	3.081	3.251
LAB_DS3	2.717	1.689	0.432	0.667	0.445	2.538	2.670
LAB_DS4	4.063	2.057	0.426	1.039	1.080	3.887	4.041
LAB_DS5	3.343	1.884	0.580	0.694	0.482	2.881	3.220
LAB_DS6	6.540	2.969	0.545	1.361	1.852	5.561	6.371

Table 4.8 Amount of data to all six DSePHR web services

on 100:0-write:read on real PHR usage after the extra storage addition

WebService	Amount of data (MB/s)						
	MAX	AVG	MIN	STD	VAR	95th	99th
LAB_DS1	3.686	1.533	0.398	0.758	0.575	2.504	3.339
LAB_DS2	6.006	2.719	0.155	1.441	2.076	5.046	5.934
LAB_DS3	5.466	2.108	0.431	1.157	1.339	4.350	5.244
LAB_DS4	4.462	2.252	0.221	1.079	1.164	3.976	4.389
LAB_DS5	6.085	2.004	0.362	1.174	1.378	3.582	5.465
LAB_DS6	5.100	1.957	0.473	1.112	1.236	4.322	5.008

Table 4.9 The amount of data sending to each storage node on 100:0-write:read with original PHR mixture before the extra storage addition

Storage Nodes	Amount of data (MB/s)						
	MAX	AVG	MIN	STD	VAR	95th	99th
LAB_RS1	11.551	10.943	10.061	0.402	0.162	11.455	11.529
LAB_RS2	11.669	11.371	10.802	0.235	0.055	11.649	11.665
LAB_RS3	11.663	10.866	8.934	0.746	0.556	11.661	11.663
LAB_RS4	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LAB_RS5	0.000	0.000	0.000	0.000	0.000	0.000	0.000
LAB_RS6	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 4.10 The amount of data sending to each storage node on 100:0-write:read with original PHR mixture after the extra storage addition

Storage Nodes	Amount of data (MB/s)						
	MAX	AVG	MIN	STD	VAR	95th	99th
LAB_RS1	11.210	5.038	0.694	2.573	6.622	10.025	10.995
LAB_RS2	11.418	4.999	1.805	2.584	6.678	10.617	11.306
LAB_RS3	11.639	5.914	2.886	2.792	7.797	11.322	11.612
LAB_RS4	11.689	9.966	0.000	2.739	7.500	11.682	11.687
LAB_RS5	11.699	9.447	0.093	2.444	5.972	11.575	11.696
LAB_RS6	11.705	9.381	0.000	2.835	8.035	11.670	11.698

Table 4.9 and Table 4.10 show the amount of data sending to each storage node each second. The measurement is captured at the input of each storage node over the same one hour period. The data is measured every second and the statistical calculation is done over the period of one hour.

According to the results in Table 4.9, the amount of data sending to each storage node (LAB_RS1, LAB_RS2, LAB_RS3) is almost at the full capacity before the extra storage nodes are added, while the extra storage nodes (LAB_RS4, LAB_RS5, LAB_RS6) have no incoming data. Table 4.10 shows that the extra storage nodes (LAB_RS4, LAB_RS5, LAB_RS6) receive the incoming data with the higher average value meaning that shows most new incoming data are sent the new storage nodes.

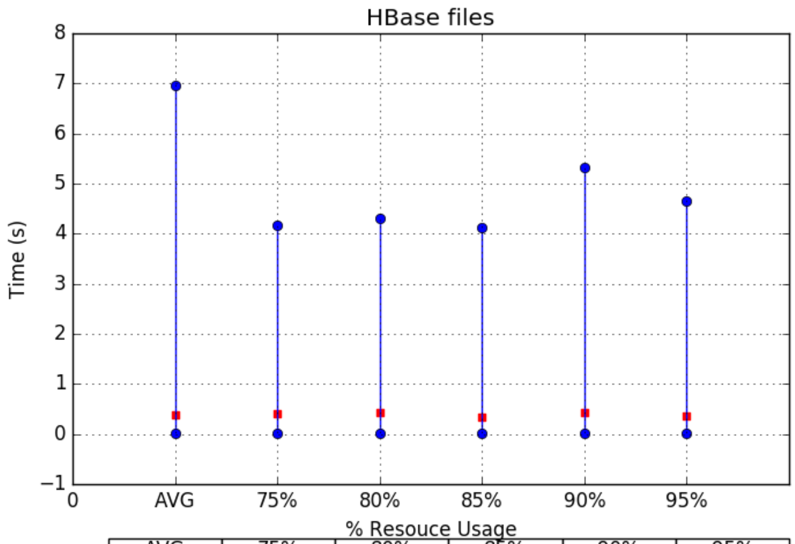
It can be concluded that the DSePHR can handle the storage capacity addition. Next, the performance of the system in terms of the retrieving time will be investigate to demonstrate the performance during the limited storage.

4.3.2.2 DSePHR Performance during Limited Resources

In this section, the performance of the data upload to and download from the DSePHR is investigated. The time to upload and download the data to the DSePHR at 75%, 80%, 85%, 90% and 95% system resource usage is analyzed. The overall performance of all files in the experiments are also presented for comparison purposes. The data at each system usage situation is collected from all active files during the two-minute period over the time when the system resource usage reaches the defined level. For example, at 1:04 time is when the system resource reaches 75%, the active files during the two-minute period from 1:03-1:05 will be used as the performance during the 75% system resource usage situation. In some cases, however, there is no file of the measured type during the two-minute period such as the read-operation requests of the 75:25-write:read operation workload. Thus, the two-minute period is changed to be the ten-minute period in order to cover a longer period.

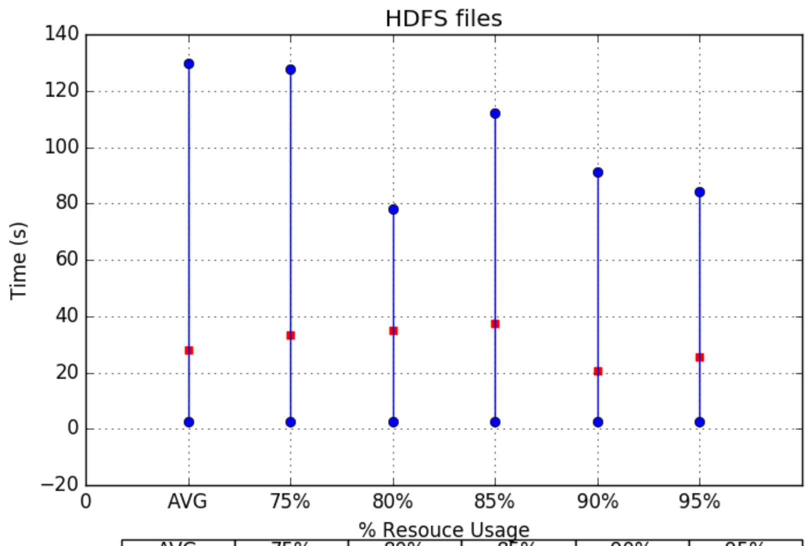
Figure 4.1 and Figure 4.2 show the time to upload the data to the DSePHR system, where Figure 4.1 shows the upload time performance of the small files stored on the HBase part of the DSePHR while Figure 4.2 shows the upload time performance of the large files stored on the HDFS part of the DSePHR. The graphs show the maximum (Max), the average (Mean) and the minimum (Min) upload time of each situation including the overall performance (AVG) of the whole experiment, and the performance when the system resource usage is at 75%, 80%, 85%, 90%, and 95%. Once, the system resource usage reaches 95%, three extra storage nodes are added to the system. The workload in this experiment is the 100:0-write:read. According to the results shown in Figure 4.1, the upload time to the HBase part of the DSePHR does not show any significant difference among each other even when the system resource usage is high. Similar trend is observed from Figure 4.2 which shows the upload time to the HDFS part of the DSePHR.

To investigate further, Figure 4.3 and Figure 4.4 show the upload time of two small files in the 100:0-write:read workload. Figure 4.3 shows the upload time of 3xray.png which is an x-ray image file of 4.01MB. Figure 4.4 shows the upload time of 10sound.mp3 which is an audio file of 8.65MB. The results of each individual file types show similar trend as that of the small files shown in Figure 4.1. Other remaining file types in the small file category is also showing the same trend. The complete results of other 8 file types of small files are presented in the APPENDIX section.



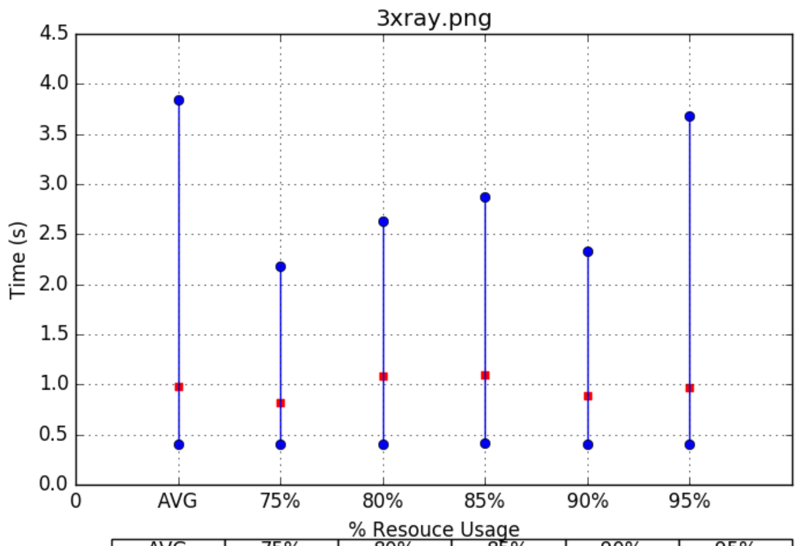
	AVG	75%	80%	85%	90%	95%
Max	6.959	4.159	4.316	4.112	5.320	4.650
Mean	0.389	0.419	0.427	0.330	0.437	0.373
Min	0.014	0.014	0.014	0.016	0.017	0.015

Figure 4.1 Upload time of small files in 100:0-write:read workload



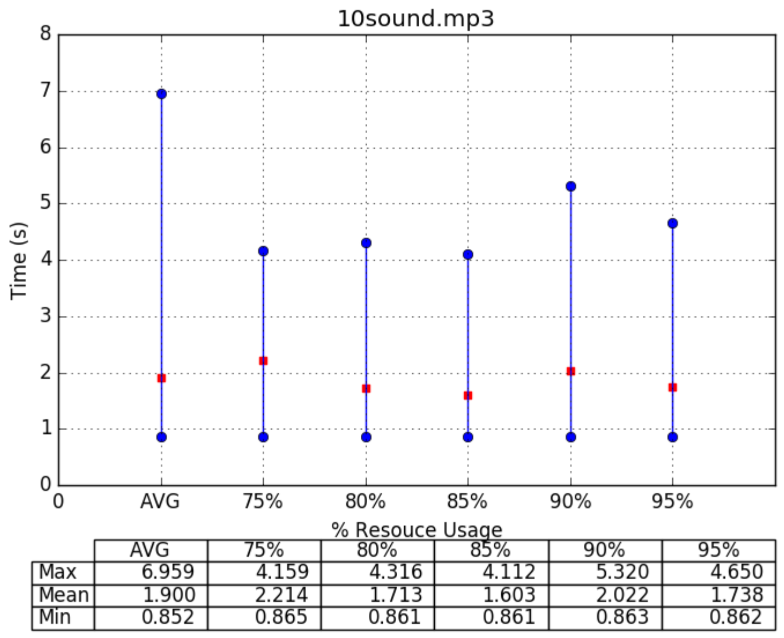
	AVG	75%	80%	85%	90%	95%
Max	129.562	127.578	77.961	112.296	91.059	84.232
Mean	28.082	33.290	34.849	37.544	20.421	25.378
Min	2.620	2.635	2.633	2.636	2.635	2.621

Figure 4.2 Upload time of large files in 100:0-write:read workload



	AVG	75%	80%	85%	90%	95%
Max	3.846	2.180	2.636	2.878	2.328	3.682
Mean	0.977	0.816	1.079	1.095	0.891	0.965
Min	0.403	0.403	0.409	0.411	0.403	0.405

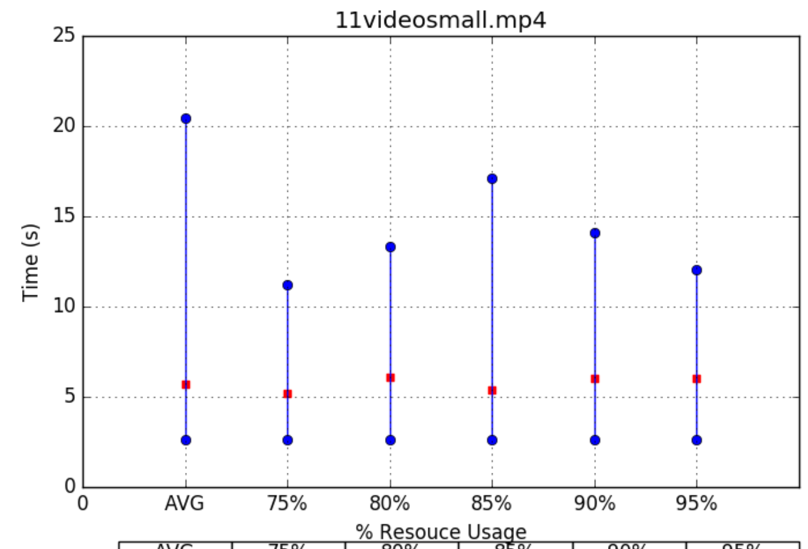
Figure 4.3 Upload time of an X-ray image file which is a small file in 100:0-write:read workload



	AVG	75%	80%	85%	90%	95%
Max	6.959	4.159	4.316	4.112	5.320	4.650
Mean	1.900	2.214	1.713	1.603	2.022	1.738
Min	0.852	0.865	0.861	0.861	0.863	0.862

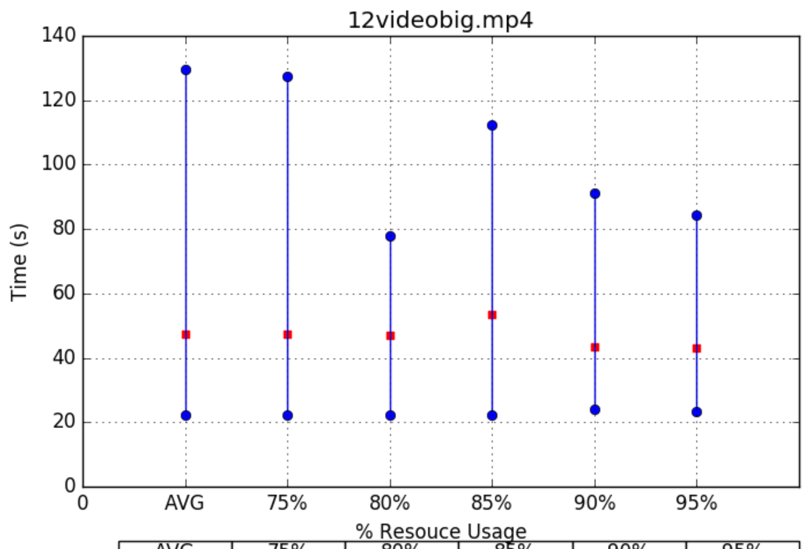
Figure 4.4 Upload time of an audio file which is a small file in 100:0-write:read workload

Figure 4.5 and Figure 4.6 show the upload time of all two-large files in the 100:0-write:read workload. Figure 4.5 shows the upload time of 11Videosmall.mp4 which is a video file of 27MB while Figure 4.6 shows the upload time of 12Videobig.mp4 which is also a video file of 232MB. According to the results shown in Figure 4.5, the upload time performance at each state of the system resource usage shows no significant difference. However, the average upload time at 90% and 95% system resource usage situations seem to be slightly larger than that of the overall performance. However, the similar average upload time at 75% system resource usage situation is observed. According to the results shown in Figure 4.6, the upload time performance at each state of the system resource usage show no significant difference. However, the minimum upload time at 90% and 95% system resource usage situations seem to be slightly larger than other situations.



	AVG	75%	80%	85%	90%	95%
Max	20.469	11.229	13.347	17.128	14.101	12.071
Mean	5.679	5.223	6.061	5.359	6.051	6.055
Min	2.620	2.635	2.633	2.636	2.635	2.621

Figure 4.5 Upload time of a video file (a larger file) in 100:0-write:read workload



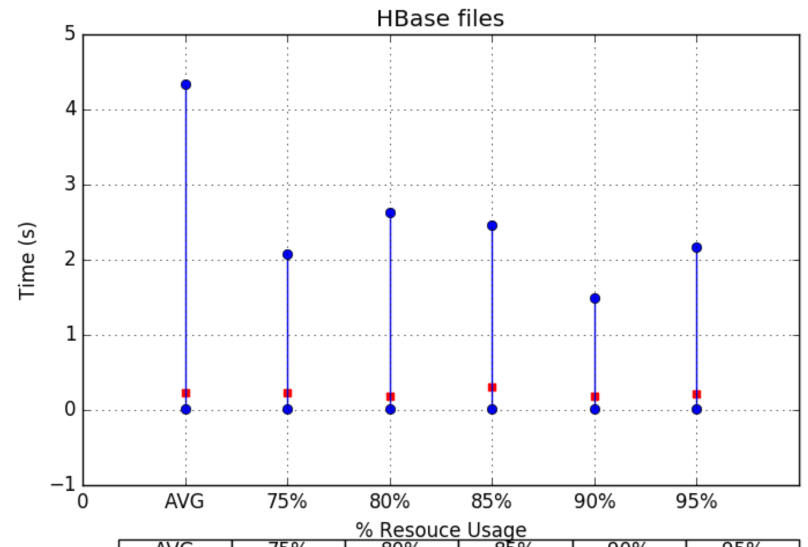
	AVG	75%	80%	85%	90%	95%
Max	129.562	127.578	77.961	112.296	91.059	84.232
Mean	47.318	47.323	47.029	53.636	43.282	43.021
Min	22.141	22.429	22.394	22.243	24.101	23.261

Figure 4.6 Upload time of a large video file (a larger file) in 100:0-write:read workload

It can be concluded that the system resource usage does not have any significant effect on the write operations of both large files and small files on the proposed DSePHR when the workload consists of only write operations. Next, the analysis on the 75:25-write:read operation workload is presented.

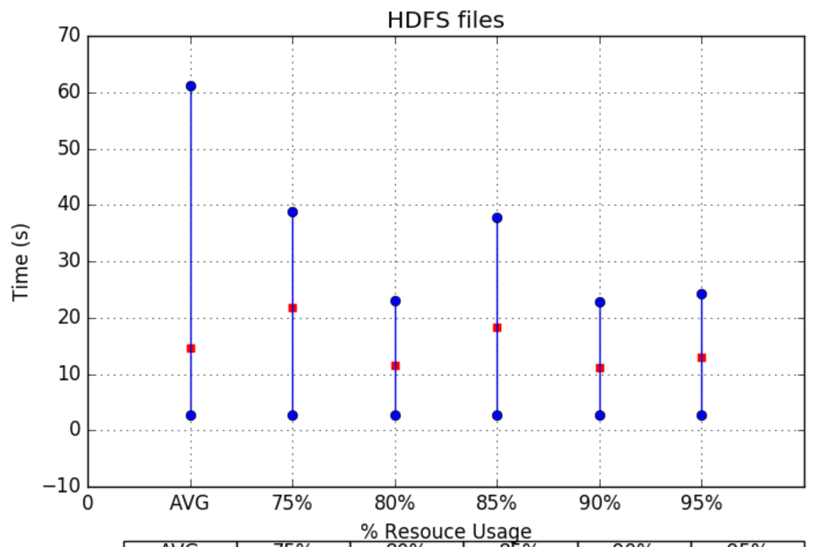
Figure 4.7 and Figure 4.8 show the time to upload the data to the DSePHR system, where Figure 4.7 shows the upload time performance of the small files stored on the HBase part of the DSePHR while Figure 4.8 shows the upload time performance of the large files stored on the HDFS part of the DSePHR. Figure 4.9 and Figure 4.10 show the time to download the data from the DSePHR system, where Figure 4.9 shows the download time performance of the small files stored on the HBase part of the DSePHR while Figure 4.10 shows the download time performance of the large files stored on the HDFS part of the DSePHR. The workloads in this experiment is the 75:25-write:read with original PHR file types mixture, meaning 75% of the requests in the workload are write operations while 25% of the requests in the workload are read operations. Due to the less number of read operation in the workload, the Figure 4.9 and Figure 4.10 use a ten-minute period to collect the data at each system usage situation.

The upload time results shown in Figure 4.7 and Figure 4.8 show similar trend as those in Figure 4.1 and Figure 4.2. In comparison, however, the average and the maximum upload time of both small and large files in 75:25-write:read workload are better than those of small and large files in 100:0-write:read workload. The detail performance of each file types is shown in APPENDIX section.



	AVG	75%	80%	85%	90%	95%
Max	4.335	2.072	2.635	2.468	1.499	2.169
Mean	0.224	0.233	0.177	0.309	0.192	0.222
Min	0.015	0.017	0.017	0.016	0.016	0.018

Figure 4.7 Upload time of small files in 75:25-write:read workload

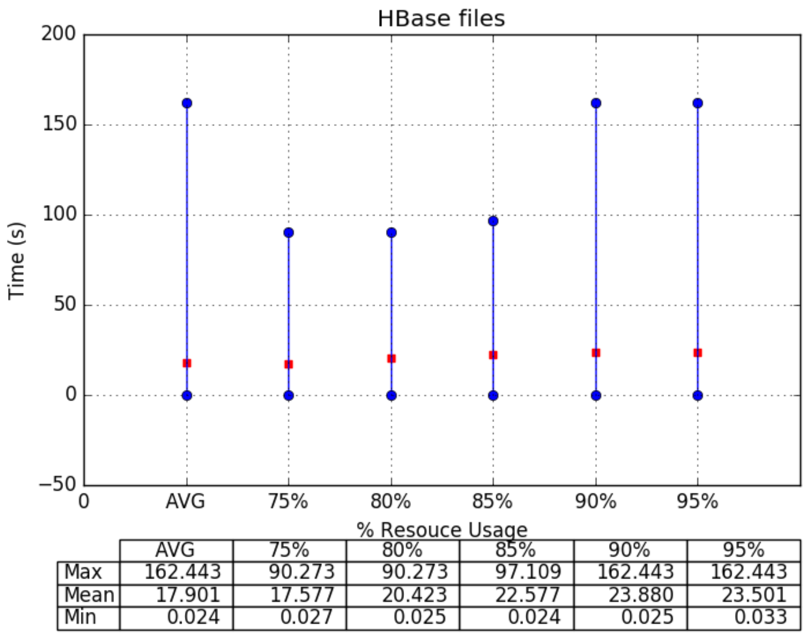


	AVG	75%	80%	85%	90%	95%
Max	61.222	38.771	23.010	37.816	22.918	24.161
Mean	14.676	21.738	11.543	18.348	11.179	13.042
Min	2.621	2.637	2.641	2.630	2.637	2.637

Figure 4.8 Upload time of large files in 75:25-write:read workload

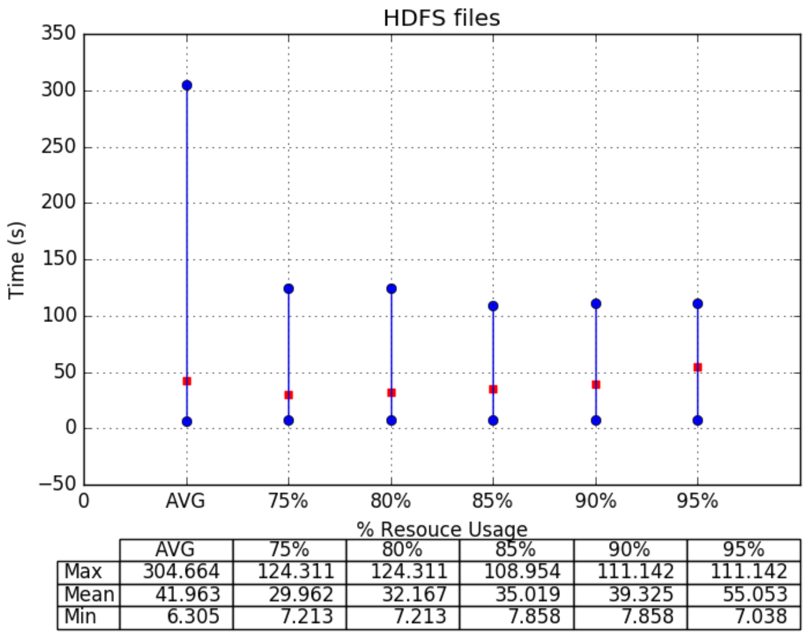
The download time of small files shown in Figure 4.9 are larger than the upload time of small files shown in Figure 4.7. This situation occurs because the files will be downloaded from the distributed storage to the DSePHR web service before sending to the client. Thus, the download time in this experiment includes all the queue time at the DSePHR web service, and the download time from the distributed storage to the DSePHR web service. Since, the data must be completely downloaded to the DSePHR web service before it can be sent to the client, the download time is longer than that of the upload time for the same file type.

According to the download time of small files shown in Figure 4.9, the average download time increases with the increasing of the system resource usage. Similar trend also shows in Figure 4.10. This trend does not significantly show up in any previous result. The maximum download time of small files (Figure 4.9) occurs at the 95% system resource usage situation which is also does not show up in any previous result discussed. However, the maximum download time of the large files (Figure 4.10) does not occur at the 95% system resource usage situation.



	AVG	75%	80%	85%	90%	95%
Max	162.443	90.273	90.273	97.109	162.443	162.443
Mean	17.901	17.577	20.423	22.577	23.880	23.501
Min	0.024	0.027	0.025	0.024	0.025	0.033

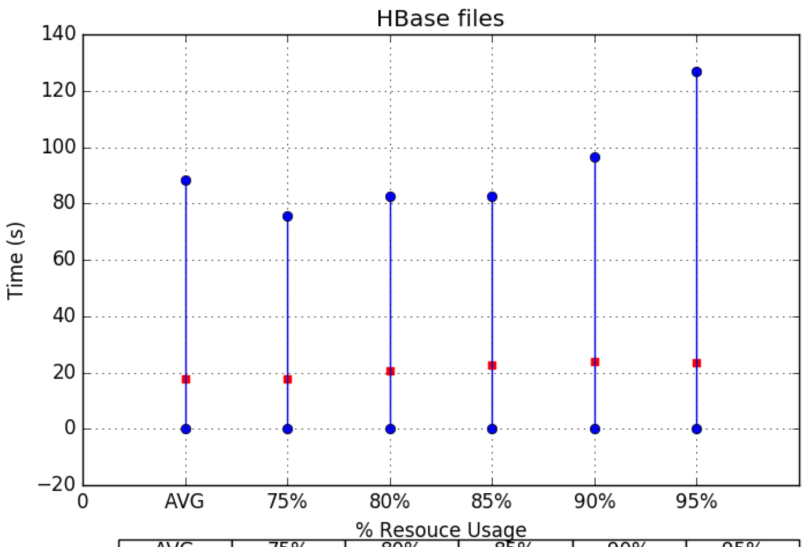
Figure 4.9 Download time of small files in 75:25-write:read workload



	AVG	75%	80%	85%	90%	95%
Max	304.664	124.311	124.311	108.954	111.142	111.142
Mean	41.963	29.962	32.167	35.019	39.325	55.053
Min	6.305	7.213	7.213	7.858	7.858	7.038

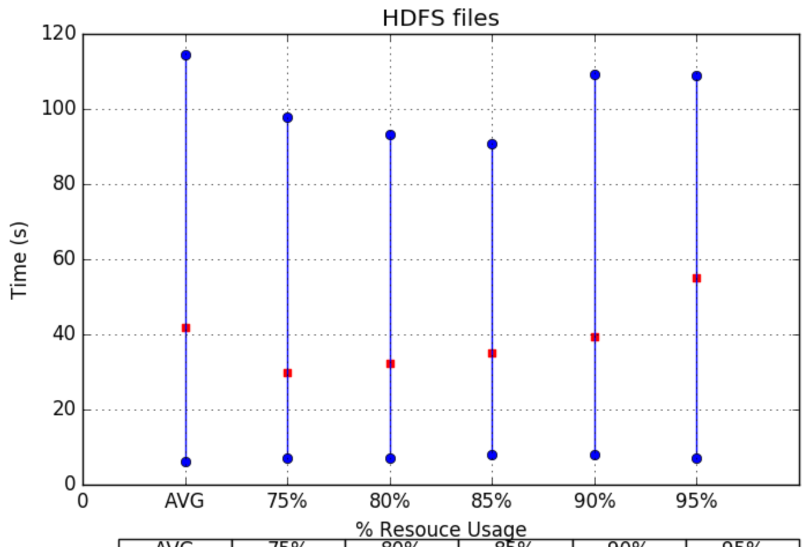
Figure 4.10 Download time of large files in 75:25-write:read workload

Unexpectedly, the maximum download time performance of the large file occurs on the small video file (Figure 4.9) with the file size of only 27MB in comparison with that of the large video file (Figure 4.10) with the file size of 232MB. The maximum download time occurs during the very early in the experiment. The situation occurs because the request of the file occurs right after the file upload request. Thus, the file is not completely uploaded to the system. As a result, the download request must be waiting until the file is completely upload to the system. This situation can be consider as a rare case. To reduce the effects of the rare cases, Figure 4.11 and Figure 4.12 show the download information by replacing the maximum value with the 95th-percentile value of Figure 4.9 and Figure 4.10, respectively. According to the results shown in Figure 4.11, the 95th-percentile download time increases with the increasing of the system resource usage. This result is clearly shown when comparing the 95th-percentile values. The same trend is not clearly shown in Figure 4.12.



	AVG	75%	80%	85%	90%	95%
95th	88.339	75.680	82.679	82.635	96.644	127.016
Mean	17.901	17.577	20.423	22.577	23.880	23.501
Min	0.024	0.027	0.025	0.024	0.025	0.033

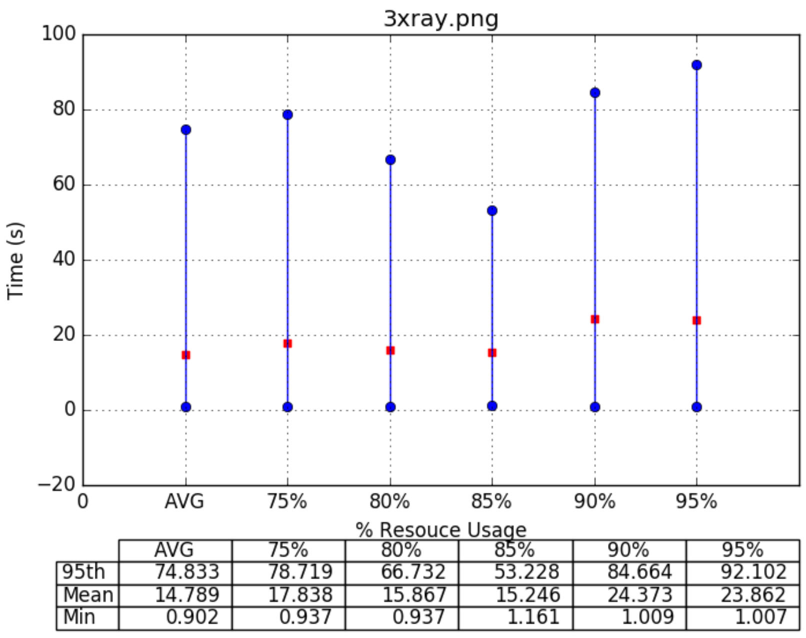
Figure 4.11 Download time of small files in 75:25-write:read workload



	AVG	75%	80%	85%	90%	95%
95th	114.434	97.802	93.128	90.647	109.350	109.024
Mean	41.963	29.962	32.167	35.019	39.325	55.053
Min	6.305	7.213	7.213	7.858	7.858	7.038

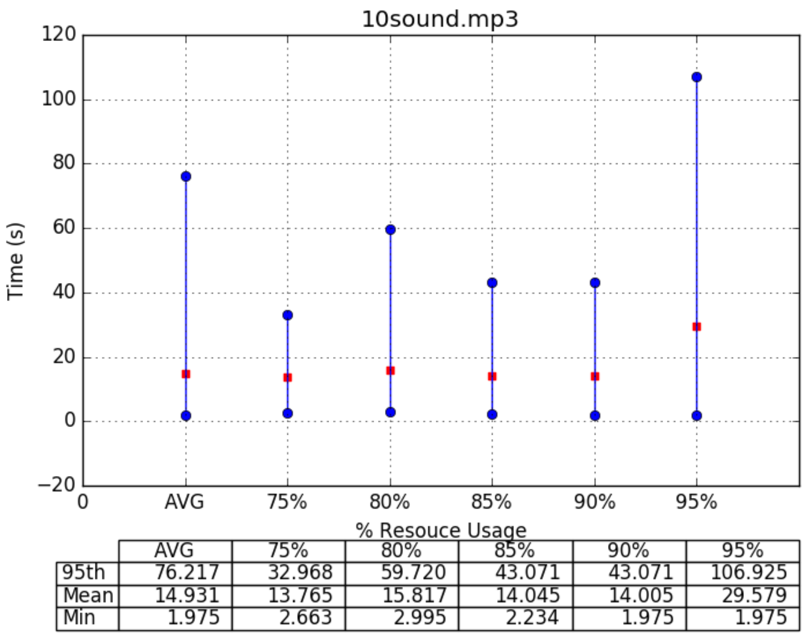
Figure 4.12 Download time of large files in 75:25-write:read workload

To investigate into the detail performance of each file types, Figure 4.13 and Figure 4.14 show the download time performance of an x-ray image file and an audio file of small files in 75:25-write:read workload, respectively. To reduce the effect of the rare cases, the maximum values will be replaced by the 95th-percentile values. According to the download time performance shown in Figure 4.13, the average and the 95th-percentile download time values do not clearly increase with the increasing of the system resource usages. However, the average and the 95th-percentile download time values at 90% and 95% system resource usage situations are clearly larger than other situations. According to the download time performance shown in Figure 4.14, the average and the 95th-percentile download time does not clearly increasing. However, the average and the 95th-percentile download time values at the 95% system resource usage situation are the highest among all. Furthermore, the maximum upload time of the audio file occurs at the 95% system resource usage as shown in Figure 4.15. This result also confirms that the download time performance of small files is affected by the system resource usage in the 75:25-write:read workload.



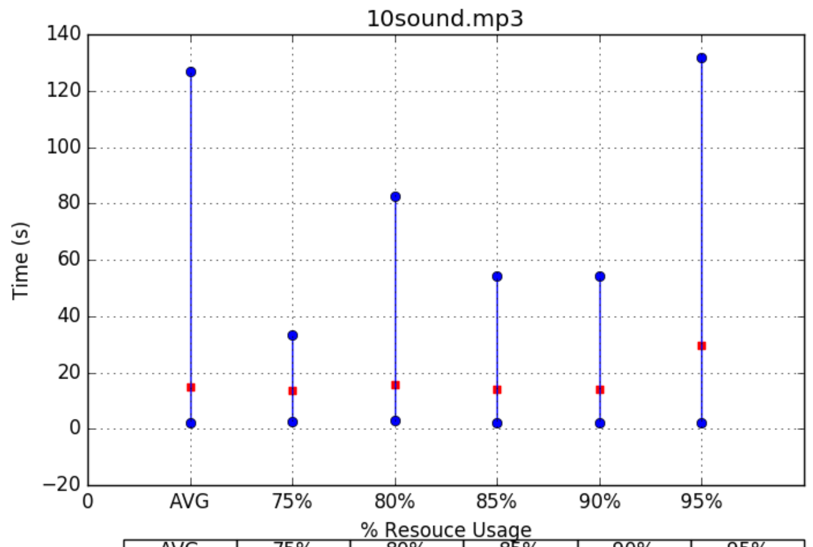
	AVG	75%	80%	85%	90%	95%
95th	74.833	78.719	66.732	53.228	84.664	92.102
Mean	14.789	17.838	15.867	15.246	24.373	23.862
Min	0.902	0.937	0.937	1.161	1.009	1.007

Figure 4.13 Download time of an X-ray image file which is a small file in 75:25-write:read workload



	AVG	75%	80%	85%	90%	95%
95th	76.217	32.968	59.720	43.071	43.071	106.925
Mean	14.931	13.765	15.817	14.045	14.005	29.579
Min	1.975	2.663	2.995	2.234	1.975	1.975

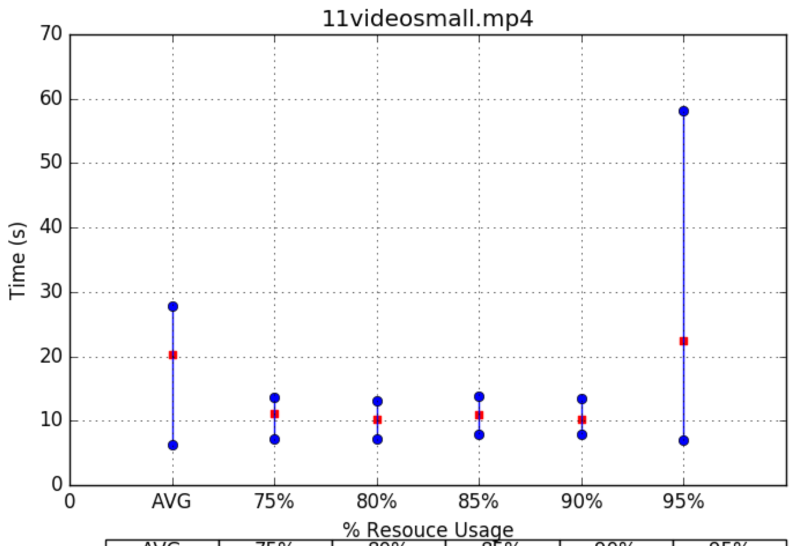
Figure 4.14 Download time of an audio file which is a small file in 75:25-write:read workload



	AVG	75%	80%	85%	90%	95%
Max	126.667	33.387	82.615	54.172	54.172	131.669
Mean	14.931	13.765	15.817	14.045	14.005	29.579
Min	1.975	2.663	2.995	2.234	1.975	1.975

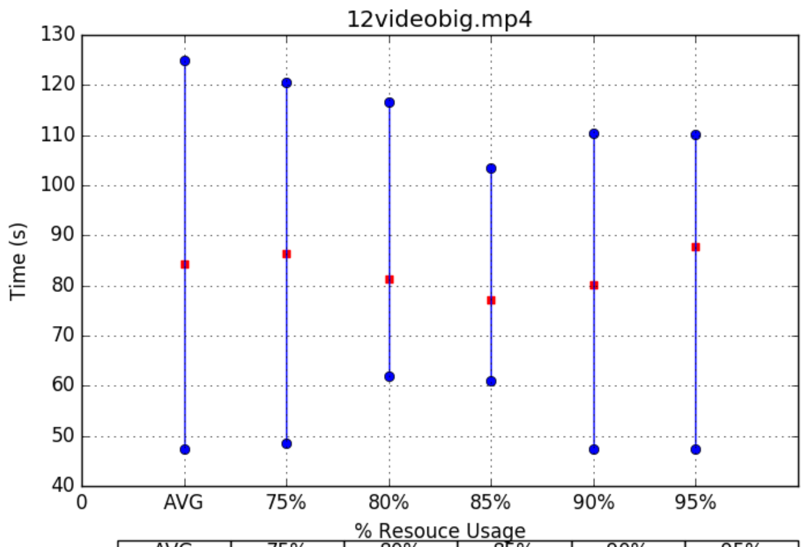
Figure 4.15 Download time of an audio file which is a small file in 75:25-write:read workload

Figure 4.16 and Figure 4.17 show the download time performance of a small video file and a large video file of large files in 75:25-write:read workload, respectively. According to the download time performance shown in Figure 4.16, the average and the 95th-percentile download time values do not clearly increase with the increasing of the system resource usage. However, the average and the 95th-percentile download time values at the 95% system resource usage are the highest values among all cases. According to the results shown in Figure 4.17, there is no significant trend to show the effect of the system resource usage on the download time performance of the large video files.



	AVG	75%	80%	85%	90%	95%
95th	27.904	13.730	13.143	13.819	13.485	58.203
Mean	20.231	11.135	10.312	10.878	10.175	22.446
Min	6.305	7.213	7.213	7.858	7.858	7.038

Figure 4.16 Download time of a video file which is a large file in 75:25-write:read workload



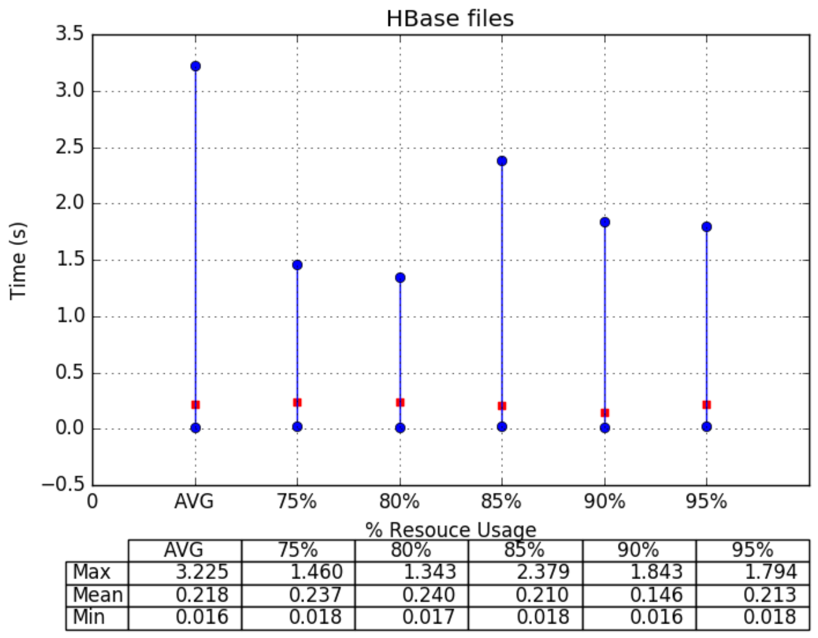
	AVG	75%	80%	85%	90%	95%
95th	124.844	120.524	116.515	103.462	110.490	110.165
Mean	84.283	86.441	81.342	77.266	80.134	87.659
Min	47.324	48.570	62.004	61.059	47.324	47.324

Figure 4.17 Download time of a large video file which is a large file in 75:25-write:read workload

In conclusion, the system resource usage has an effect on the download time performance of both small files and large files, specially the effects at the 90% and 95% system resource usage are clearly visible in many cases for the 75:25-write:read workload. However, the effect does not clearly show in the upload time performance of both file types.

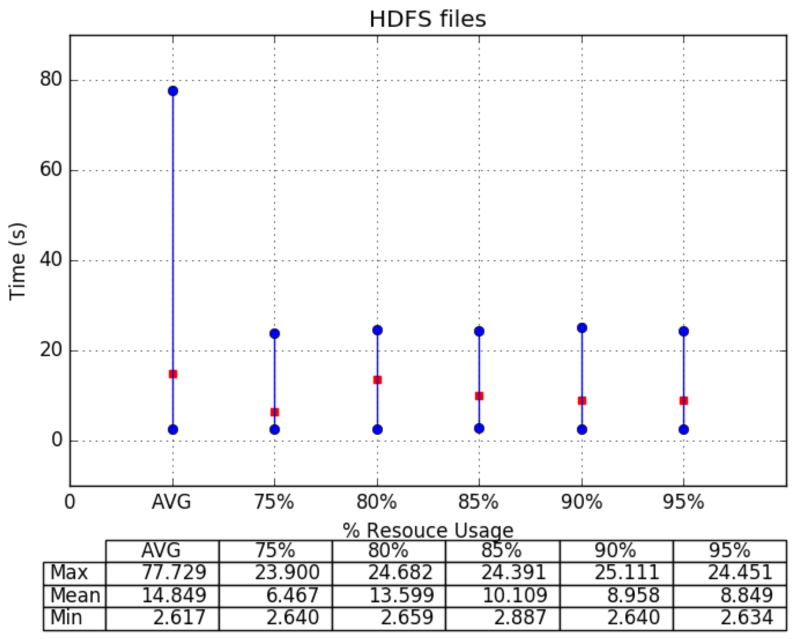
Next, the performance of the DSePHR when the workload contains 50% write operations and 50% read operations. Figure 4.18 and Figure 4.19 show the upload time performance of small files and large files to the DSePHR, respectively. Figure 4.20 and Figure 4.21 show the download time performance of small files and large files from the DSePHR, respectively. The workload used in these figures is the 50:50-write:read workload, meaning half of the requests are write operations and the other half of the requests are read operations.

The upload time results shown in Figure 4.18 and Figure 4.19 show similar trend as the upload time performance of the 100:0-write:read workload and the 75:25-write:read workload. In comparison, however, the average and the maximum upload time of both small and large file types in 50:50-write:read workload are smaller than their counterpart in 75:25-write:read workload and 100:0-write:read workload. The detail performance of each file types is shown in APPENDIX section.



	AVG	75%	80%	85%	90%	95%
Max	3.225	1.460	1.343	2.379	1.843	1.794
Mean	0.218	0.237	0.240	0.210	0.146	0.213
Min	0.016	0.018	0.017	0.018	0.016	0.018

Figure 4.18 Upload time of small files in 50:50-write:read workload

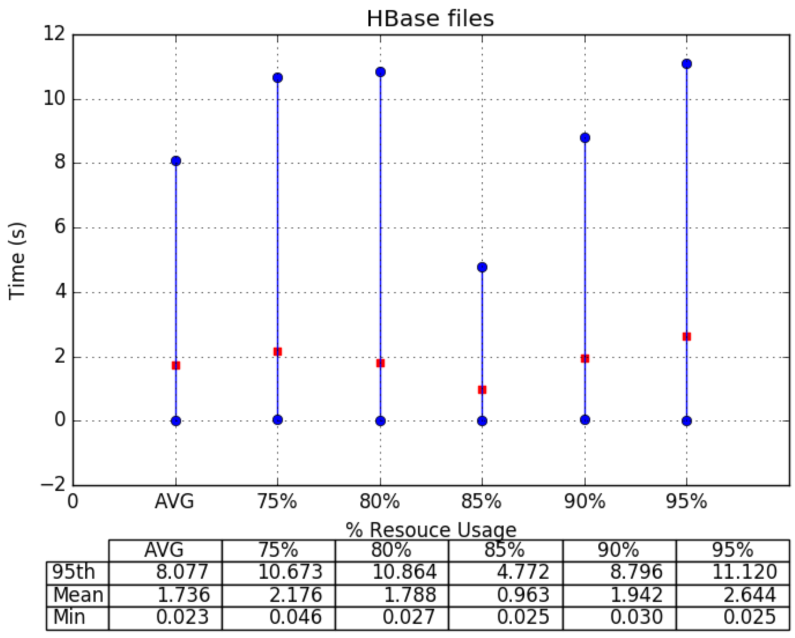


	AVG	75%	80%	85%	90%	95%
Max	77.729	23.900	24.682	24.391	25.111	24.451
Mean	14.849	6.467	13.599	10.109	8.958	8.849
Min	2.617	2.640	2.659	2.887	2.640	2.634

Figure 4.19 Upload time of large files in 50:50-write:read workload

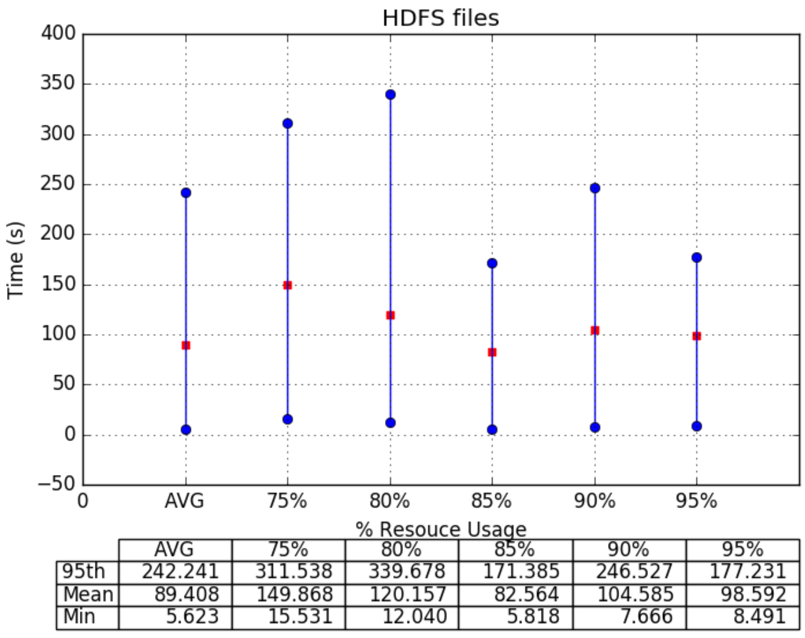
According to the download time of small file types shown in Figure 4.20 are smaller than that shown in Figure 4.11. In contrast, the download time of large file types shown in Figure 4.21 are larger than that shown in Figure 4.12. The effect of the system resource usages on the read operation is now not clearly visible. To further investigate the results of each file types, Figure 4.22 and Figure 4.23 show the download time performance of an x-ray image file and an audio file, respectively. While, Figure 4.24 and Figure 4.25 show the download time performance of a small video file and a large video file, respectively.

According to the results shown in Figure 4.22, the download time performance of the x-ray image file does not clearly show the increasing trend with the increasing system resource usages. However, the average and 95th-percentile download time values at the 95% system resource usage situation are the highest among all cases. According to the results shown in Figure 4.23, the download time performance of the audio file does not clearly show the increasing trend with the increasing system resource usages. In addition, the performance values at the 95% system resource usage are not clearly larger than other values.



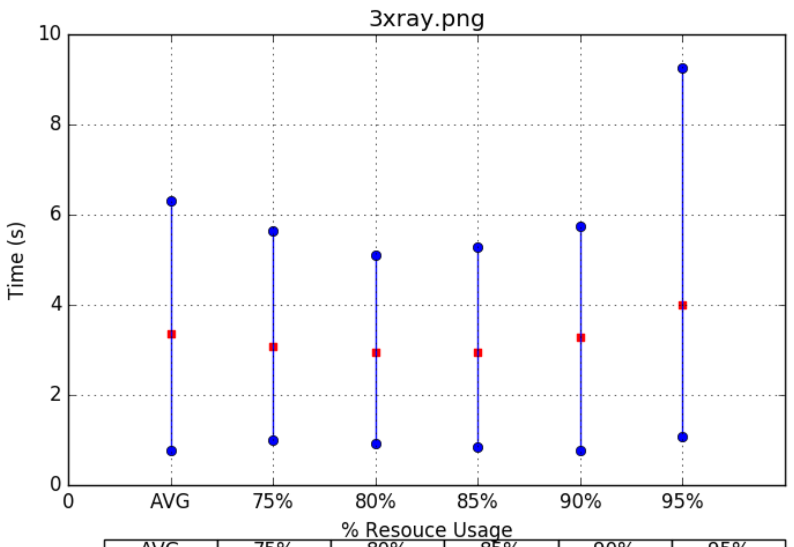
	AVG	75%	80%	85%	90%	95%
95th	8.077	10.673	10.864	4.772	8.796	11.120
Mean	1.736	2.176	1.788	0.963	1.942	2.644
Min	0.023	0.046	0.027	0.025	0.030	0.025

Figure 4.20 Download time of small files in 50:50-write:read workload



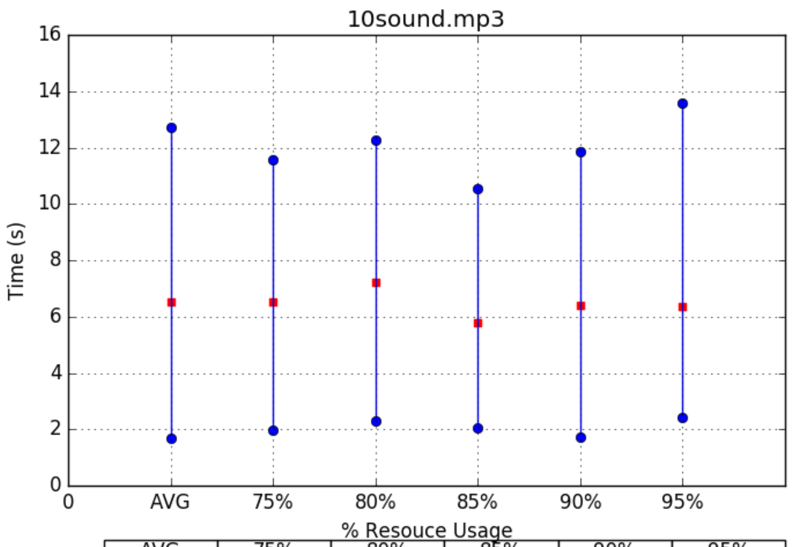
	AVG	75%	80%	85%	90%	95%
95th	242.241	311.538	339.678	171.385	246.527	177.231
Mean	89.408	149.868	120.157	82.564	104.585	98.592
Min	5.623	15.531	12.040	5.818	7.666	8.491

Figure 4.21 Download time of large files in 50:50-write:read workload



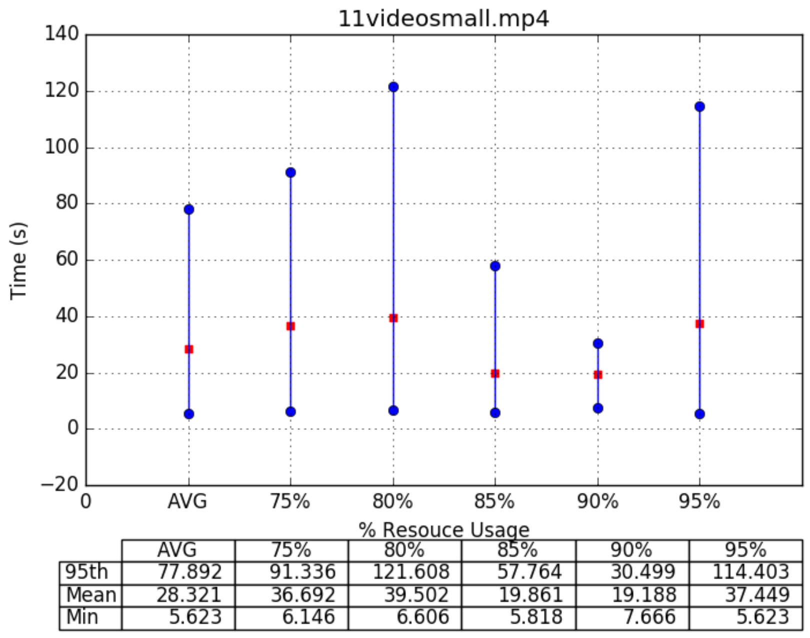
	AVG	75%	80%	85%	90%	95%
95th	6.317	5.637	5.095	5.274	5.750	9.268
Mean	3.368	3.073	2.949	2.955	3.291	4.002
Min	0.763	0.991	0.918	0.855	0.763	1.079

Figure 4.22 Download time of an X-ray image file which is a small file in 50:50-write:read workload



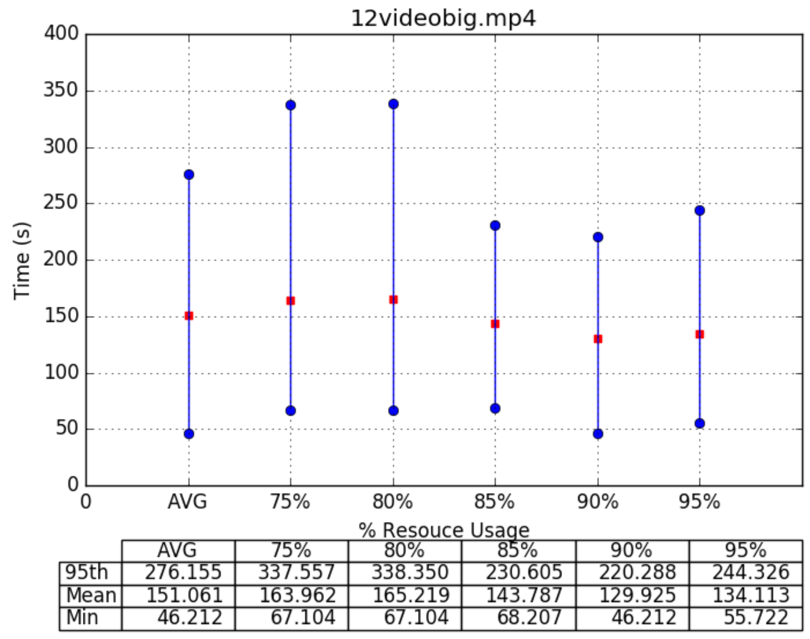
	AVG	75%	80%	85%	90%	95%
95th	12.731	11.584	12.247	10.539	11.857	13.582
Mean	6.504	6.532	7.208	5.770	6.399	6.343
Min	1.700	1.976	2.317	2.046	1.727	2.404

Figure 4.23 Download time of an audio file which is a small file in 50:50-write:read workload



	AVG	75%	80%	85%	90%	95%
95th	77.892	91.336	121.608	57.764	30.499	114.403
Mean	28.321	36.692	39.502	19.861	19.188	37.449
Min	5.623	6.146	6.606	5.818	7.666	5.623

Figure 4.24 Download time of a small video file which is a small file in 50:50-write:read workload



	AVG	75%	80%	85%	90%	95%
95th	276.155	337.557	338.350	230.605	220.288	244.326
Mean	151.061	163.962	165.219	143.787	129.925	134.113
Min	46.212	67.104	67.104	68.207	46.212	55.722

Figure 4.25 Download time of a large video file which is a small file in 50:50-write:read workload

According to the results shown in Figure 4.24 and Figure 4.25, the download time performance of the small and large video files does not clearly show the increasing trend with the increasing system resource usages. In addition, the performance values at the 95% system resource usage are not clearly larger than other values.

In conclusion, the performances of the DSePHR during the system resource usages at 75%, 80%, 85%, 90% and 95% are analyzed. According to Table 4.11, the results show that the upload time performance values are not clearly affected by the system resource usage level. However, the download time performance values can be affected by the system resource usage level, especially when the read-requests are smaller than the write-requests as shown in the results of 75:25-write:read workload. However, the effects are clearly visible at 95% system resource usages. Thus, the DSePHR is suggested to monitor the system resource usage at 90% in order to start adding more resources to the system. This way, the effects on the read-requests can be minimized.

Table 4.11 Performance effect on each situation when system resource is high

Situation/ File types	100:0-write:read	75:25-write:read		50:50-write:read	
	Upload	Upload	Download	Upload	Download
HBase files	Not effect	Not effect	Effect	Not effect	Not effect
HDFS files	Not effect	Not effect	Effect	Not effect	Not effect

In the next section, the DSePHR will be further investigated the system resource is smaller than the clients and the workload are larger. Moreover, the system will be pre-load with the data before any measurement will be taken. Also, the data will continue to arrive to the system even when the measured data requests are completely sent to the system. This method will make sure that the system performance will be measured when the system is not empty and there are requests that are not parts of the measurement data. This way, the real-world situation will be mimicked in the experiment.

4.3.3 Effect of the Write-read Request Ratio

In the realistic environment, the ratio between write and read requests can be changed. In the first state of the system, there are a lot of write requests, then a number of read requests can also increase. To evaluate the DSePHR system in a realistic environment, a cluster with 15 storage nodes are constructed in this experiment. The environment consists of 8 clients that will upload the data to and download the data from the DSePHR system. To enforce the limited resources in the real world environment, only 6 DSePHR web services are activated in this experiment. Thus, the number of DSePHR web services will be less than that of the number of clients. To measure the performance of the DSePHR, three workloads are created in this experiment including a 100:0-write:read, a 75:25-write:read and a 50:50-write:read workloads. The number of files in each workload is 6,000. The first 500 files and the last 500 files are used as the warm-up and the cool-down of the DSePHR system. Therefore, the total number of measurable files is 5,000 submitted from each client. Since there are 8 clients, the total number of measurable files in each situation is 40,000 files. The mixture of the file types is that presented in

Section 3.5 and the uniform mixture of file types. The upload and download time performances of the DSePHR system is analyzed and discussed.

In order to mimic the real environment, the DSePHR system will be warmed up with the data so that the DSePHR system will have some activities and the data inside the system before the measured workload is sent to the system. Similarly, the system will be continuously receiving the data to the system after the measured workload are completely submitted to the system. This way, the performance during the measured workload can be a representative of that in the real world situation. In this experiment, all clients will upload their data to the DSePHR web services first, then the DSePHR web services will create the metadata and later upload the data to the distributed storage. At this point, there is a 2-step operation, including the client-to-DSePHR and the DSePHR-to-storage operations. The client-to-DSePHR operation is considered an operation from the client to the DSePHR system while the DSePHR-to-storage operation is considered an inside operation of the DSePHR system.

4.3.3.1 Performance on the 100:0-write:read Workload

Table 4.12 shows the overall upload time performance of both small files and large files of 100:0-write:read workload. According to the results shown in Table 4.12, the upload time of small files are smaller than that of the large files in all measured values. The data presented in Table 4.12 include the maximum (Max), the 95th-percentile (95th-), the 99th-percentile (99th-), the average (Mean), and the minimum (Min) upload time of each file type. This is expected because the HBase files are consisting of small files while the HDFS files are consisting of large files (i.e., larger than 10MB). Thus, the upload time of the HBase files are smaller as the results.

Table 4.12 Overall upload time of 100:0-write:read workload

File types	Upload time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	7.257	3.298	1.852	0.373	0.013
HDFS files	147.917	110.310	82.868	26.202	2.625

To analyze the details information, Table 4.13 shows the upload time performance of each file in the workload sorted by the file size. As expected, the maximum upload time of HBase files is from the 10sound.mp3 which is the largest file in the HBase files while the maximum upload time of HDFS files is from the 12videobig.mp4 which is also the largest file in the HDFS files.

In HBase file group, there are three small files including an MRI image file of 30KB, a CCD xml file of 31KB and a spreadsheet data file of 41KB. The upload time of these small files takes approximately 0.5 seconds on average. There are 5 medium files in the HBase file group including a heartbeat sound file of 160KB, an ECG graph image file of 400KB, a patient document file of 440KB, a patient spreadsheet file of 620KB, and a CCD document file of 690KB. The upload time of the heartbeat sound file of 160KB is less than 0.1 second on average, while the upload time of the 400KB and 440KB files is approximately 0.14-0.15 seconds on average. The upload time of the 620KB and 690KB is approximately 0.2 seconds on average. The increment of the upload time of these files is expected because of the amount of data in each file. More data to be uploaded will result in the longer upload time. However, the increasing is not linear.

The average performance of the two large files in the HBase file group, on the other hand, shows the linear performance relationship between the file size and the average upload time. That is, the average upload time of the x-ray image file of 4.01MB takes 0.951 seconds while the upload time of the recorded sound conversation file of 8.65MB takes 1.849 seconds. Since the conversation file is approximately twice the size of the x-ray image file, the average upload time of the conversation file (1.849s) is also approximately twice the upload time of the x-ray image file (0.951s). Similar trend also shows in the HDFS file group. That is, the size of the large video file (232MB) is approximately 8.5 times of the size of the small video file (27MB). And, the average upload time of the large video file (46.65s) is also approximately 8.2 times of the average upload time of the small video file (5.687s).

The maximum upload time of the smallest file (i.e., the MRI image file of 30KB) is 3.069 seconds which is larger than the maximum upload time of all files that are smaller than 620KB. This is an unexpected event. However, a detail analysis found that the maximum upload time performance occurs due to the backlog at the DSePHR web service because there are several video files are queuing to be processed. Thus, the small MRI image files must wait in the queue. Further investigation shows that a few MRI image files are delayed because of the memory clearing process inside the DSePHR system. This inside operation occurs only on the HBase file group because the recently used data will be resided in the memory. However, the memory will be clear once the space is needed. Unfortunately, a few MRI image files are uploaded to the DSePHR during the memory clearing process. As a result, the upload time of this set of files takes longer than others. The evident is showing up on the 99th-percentile upload time performance of 0.793 seconds which is also larger than that of all files that are smaller than 620KB.

Table 4.13 Upload time of each file type of 100:0-write:read workload

File types	Filename	Size	Upload time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	3.069	0.793	0.086	0.058	0.013
	4ccd.xml	31KB	1.750	0.630	0.084	0.054	0.015
	8excel.xlsx	41KB	2.411	0.745	0.082	0.056	0.013
	9sound.ogg	160KB	2.384	0.732	0.156	0.085	0.023
	1ecg.jpg	400KB	2.639	0.777	0.298	0.146	0.047
	7word.docx	440KB	2.547	0.751	0.306	0.151	0.050
	5ccd.xml	620KB	4.181	1.263	0.484	0.236	0.071
	6ccd.pdf	690KB	3.100	1.169	0.472	0.223	0.073
	3xray.png	4.01MB	4.421	2.818	2.101	0.951	0.398
	10sound.mp3	8.65MB	7.257	5.035	3.885	1.849	0.843
HDFS files	11videosmall.mp4	27MB	19.819	15.965	12.609	5.687	2.625
	12videobig.mp4	232MB	147.917	116.734	93.745	46.650	22.114

4.3.3.2 Performance on the 75:25-write:read Workload

Table 4.14 shows the overall upload time performance of both small files and large files of 75:25-write:read workload while Table 4.15 shows the overall download time performance of the same workload.

For the upload time performance, a similar tread as that of the overall upload time performance of the 100:0-write:read workload shown in Table 4.12 is observed. That is, the upload time performance of small files (HBase files) are also smaller than that of the large files (HDFS files) in all measured values for the 75:25-write:read workload shown in Table 4.14. Moreover, the overall upload time performance of the 75:25-write:read workload are also smaller than that of the 100:0-write:read workload in all measured values, except the minimum upload time of the HBase files which is 0.001 second faster than that observed under the 100:0-write:read workload. This can be explained by the fact that the write-operations of the 75:25-write:read workload is only 75% of that under the 100:0-write:read workload because 25% of the operations are the read operations. Thus, the write-operation workload is reduced from 531.56GB to 396.35GB.

Table 4.14 Overall upload time of 75:25-write:read workload

File types	Upload time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	6.457	2.811	1.548	0.326	0.014
HDFS files	112.936	87.301	67.177	21.890	2.612

Table 4.15 Overall download time of 75:25-write:read workload

File types	Download time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	15.468	7.972	4.443	0.933	0.019
HDFS files	288.480	200.608	154.752	57.260	4.971

Table 4.16 Upload time of each file type of 75:25-write:read workload

File types	Filename	Size	Upload time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	3.066	0.367	0.075	0.047	0.014
	4ccd.xml	31KB	1.559	0.456	0.082	0.049	0.014
	8excel.xlsx	41KB	1.673	0.684	0.078	0.049	0.014
	9sound.ogg	160KB	1.304	0.539	0.146	0.072	0.023
	1ecg.jpg	400KB	1.643	0.639	0.265	0.125	0.046
	7word.docx	440KB	2.627	0.453	0.281	0.130	0.051
	5ccd.xml	620KB	2.279	0.825	0.420	0.205	0.074
	6ccd.pdf	690KB	2.433	0.884	0.424	0.199	0.074
	3xray.png	4.01MB	4.286	2.390	1.790	0.817	0.401
	10sound.mp3	8.65MB	6.457	4.326	3.295	1.566	0.834
HDFS files	11videosmall.mp4	27MB	17.524	12.190	10.076	4.688	2.612
	12videobig.mp4	232MB	112.936	93.881	76.993	38.691	22.086

To further analyze the detail upload time performance, Table 4.16 shows the upload time performance of each file in the workload sorted by the file size. Like the upload time performance of the 100:0-write:read workload, the maximum upload time of HBase files is from the 10sound.mp3 which is the largest file in the HBase files while the maximum upload time of HDFS files is from the 12videobig.mp4 which is the largest file in the HDFS files.

According to the results shown in Table 4.16, the upload time performance trend observed from the 75:25-write:read workload is similar to the trend observed from the 100:0-write:read workload. That is, the performance of all files smaller than 4.01MB is less than 0.3 seconds and the increasing of the average upload time does not shows any linear relationship with the file size. However, the linear relationship between the average upload time and the file size shows on two large files in the HBase file group and the two files in the HDFS file group. That is, the average upload time of the 8.65MB file is 1.566 seconds which is approximately twice the average upload time of the 4.01MB file which is 0.817 seconds, while the average upload time of the 232MB file is 38.691 seconds which is still 8.2 times that of the average upload time of the 27MB file which is 4.688 seconds.

For the comparison details, the average upload time of the three small files (i.e., the MRI image file of 30KB, the CCD file of 31KB and the spreadsheet file of 41KB) is approximately slightly less than 0.5 seconds which is observed under the 100:0-write:read workload. The average upload time of the heartbeat sound file of 160KB is 0.072 second which is 0.01 second less than that observed under the 100:0-write:read workload. The average upload time of the 400KB and 440KB files is approximately 0.12-0.13 seconds which is also 0.02 seconds less than that observed under the 100:0-write:read workload. The average upload time of the 620KB and 690KB is still approximately 0.2 seconds. The average upload time of the x-ray image file of 4.01 MB is 0.817 seconds which is 0.1 second less than that observed under the 100:0-write:read workload. And, the average upload time of the audio file of 8.65 MB is 1.566 seconds which is approximately 0.3 seconds less than that observed under to 100:0-write:read workload. The average upload time of the 27 MB file is 4.688 seconds which is 1 second less than that observed under the 100:0-write:read workload. The average upload time of the 232MB file is 38.691 seconds which is 8 seconds less than that observed under the 100:0-write:read workload.

Also, the maximum upload time performance of the MRI file at 3.066 seconds is occurring at the DSePHR memory clearly process time. As a result, the file experiences the worst upload time among all files with a size smaller than 620KB.

For the download time performance, the download time of the small files (HBase files) is also smaller than that of the large files (HDFS files) for the 75:25-write:read workload. However, the download time of each file type is larger than the upload time of the same type for all measured values. For the overall HBase file performance, the average download time is 0.933 seconds which is approximately 2.8 times of the average upload time which is 0.326 seconds. For the overall HDFS file performance, the average download time is 57.26 seconds which is approximately 2.6 times of the average upload time which is 21.89 seconds. This result can be explained by the fact that the data must be completely downloaded from the DSePHR storage to the DSePHR web services in order to be sent to the client. Thus, the download time will include the time that the read request is made at the client until the time that the data is completely sent to the client via the

DSePHR web service. Thus, the data transmission is occurring twice including the transmission time from the DSePHR storage node to the DSePHR web service and the transmission time from the DSePHR web service to the client. Therefore, the average download time is expected to be approximately twice the average upload time.

To further analyze the download time performance of the 75:25-write:read workload, Table 4.17 shows the download time performance of each file in the workload sorted by the file size. The average download time performance of the 75:25-write:read workload shows the similar trend as that of the average upload time. That is, the average download time of the three small files in the HBase file group is similar. The linear relationship between the file size and the average download time is not clearly shown in the files that are smaller than 4.01 MB. On the other hand, the linear relationship between the file size and the average download time of the two large files of the HBase file group is observed. That is, the average download time of the 8.65 MB file is 4.584 seconds which is approximately twice the average download time of the 4.01 MB file which is 2.328 seconds. Moreover, the average download time of the 232 MB video file is 97.871 seconds which is approximately 7.5 times of the average download time of the 27 MB video file which is 13.069 seconds.

Table 4.17 Download time of each file type of 75:25-write:read workload

File types	Filename	Size	Download time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	5.428	2.319	0.490	0.154	0.019
	4ccd.xml	31KB	7.142	2.355	0.391	0.153	0.021
	8excel.xlsx	41KB	7.907	1.686	0.494	0.156	0.023
	9sound.ogg	160KB	9.217	3.076	0.678	0.259	0.046
	1ecg.jpg	400KB	6.790	1.801	0.745	0.363	0.089
	7word.docx	440KB	7.484	2.771	0.951	0.433	0.095
	5ccd.xml	620KB	8.644	2.254	1.036	0.504	0.129
	6ccd.pdf	690KB	15.468	2.362	1.195	0.554	0.140
	3xray.png	4.01MB	8.886	6.196	4.696	2.328	0.760
	10sound.mp3	8.65MB	14.285	10.484	8.973	4.584	1.620
HDFS files	11videosmall.mp4	27MB	47.212	32.743	25.158	13.069	4.971
	12videobig.mp4	232MB	288.48	212.486	174.317	97.871	42.738

4.3.3.3 Performance on the 50:50-write:read Workload

Table 4.18 shows the overall upload time performance of both small files and large files of 50:50-write:read workload while Table 4.19 shows the overall download time performance of the same workload.

Table 4.18 Overall upload time of 50:50-write:read workload

File types	Upload time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	5.813	2.602	1.349	0.299	0.014
HDFS files	115.492	78.808	59.064	20.505	2.617

Table 4.19 Overall download time of 50:50-write:read workload

File types	Download time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	15.070	7.649	4.102	0.834	0.016
HDFS files	240.318	166.344	125.848	46.542	4.953

Similar tread as that of the overall upload time performance of the 100:0-write:read and 75:25-write:read workload shown in Table 4.12 and Table 4.14, the upload time performance of small files (HBase files) are also smaller than that of the large files (HDFS files) in all measured values for the 50:50-write:read workload shown in Table 4.18. Moreover, the average upload time performance of the 50:50-write:read workload are also smaller than that of the 75:25-write:read and that of the 100:0-write:read workload. This can be explained by the fact that the write-operations of the 50:50-write:read workload is only a half of the number of operations under the 100:0-write:read workload because 50% of the operations are the read operations. Thus, the write-operation workload is reduced from 531.56GB (100:0-write:read workload) to 273.47GB (50:50-write:read workload).

To further analyze the upload time performance of the 50:50-write:read workload, Table 4.20 shows the upload time performance of each file in the workload sorted by the file size. Similar with the upload time performance of the two previous workload, the maximum upload time of HBase files (i.e., 5.813 seconds) is form the 10sound.mp3 which is the largest file in the HBase files while the maximum upload time of HDFS files (i.e., 115.492 seconds) is from the

12videobig.mp4 which is the largest file in the HDFS files. The upload time performance trend observed from the 50:50-write:read workload is similar to the trend observed from the two previous workloads. That is, the linear relationship between the size of the files and the average upload time does not show for all files smaller than 4.01 MB. However, the relationship is clearly visible for the two large files in the HBase file group and the two files in the HDFS file group. The average upload time of the 8.65MB file is 1.455 seconds which is approximately twice the average upload time of the 4.01MB file which is 0.748 seconds. The average upload time of the 232MB file is 36.082 seconds which is approximately 8.3 times of the average upload time of the 27MB file which is 4.317 seconds.

Table 4.20 Upload time of each file type of 50:50-write:read workload

File types	Filename	Size	Upload time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	1.273	0.284	0.075	0.042	0.015
	4ccd.xml	31KB	1.185	0.285	0.078	0.042	0.016
	8excel.xlsx	41KB	1.743	0.292	0.076	0.042	0.014
	9sound.ogg	160KB	1.132	0.267	0.132	0.061	0.023
	1ecg.jpg	400KB	2.117	0.488	0.266	0.117	0.048
	7word.docx	440KB	1.170	0.542	0.280	0.122	0.051
	5ccd.xml	620KB	1.634	0.536	0.392	0.182	0.081
	6ccd.pdf	690KB	1.292	0.641	0.388	0.177	0.077
	3xray.png	4.01MB	2.716	2.175	1.632	0.748	0.406
	10sound.mp3	8.65MB	5.813	4.203	3.110	1.455	0.854
HDFS files	11videosmall.mp4	27MB	15.258	11.610	9.202	4.317	2.617
	12videobig.mp4	232MB	115.492	85.093	68.376	36.082	22.064

Similar with the overall upload time performance results shown in Table 4.18, the average upload time performance of all files in the 50:50-write:read workload shown in Table 4.20 is also smaller than that of its counterpart in the two previous workloads. That is, the average upload time performance of all files smaller than 4.01MB is less than 0.2 seconds which is 0.1 second smaller than that observed from the two previous workloads. The average upload time performance of the two large files in the HBase file group is also approximately 0.1 second smaller than that observed from its counterpart in the two previous workloads. The average upload time performance of the 11videosmall.mp4 and the 12videobig.mp4 is approximately 0.2 seconds and 2 seconds smaller than that observed from its counterpart in the two previous workloads.

Similarly, the download time of the small files (HBase files) is also smaller than that of the large files (HDFS files) for the 50:50-write:read workload, and the download time of each file type under the 50:50-write:read workload is larger than that of its counterpart in all measured values. Unexpectedly, the average download time performance of the 50:50-write:read workload is smaller than that of the 75:25-write:read workload in all cases even though the number of read operations under the 50:50-write:read workload (267.85GB) is twice the number of read operations under the 75:25-write:read workload (134.00GB).

This unexpected results can be explained by the fact that the DSePHR storage nodes must perform the memory flushing process in order to move the HBase files out of the memory and store the data on the HDFS storages. Thus, the

DSePHR will spend some times to perform this activity. This activity can affect the download time performance. During the DSePHR memory flushing activity, the DSePHR storage nodes will pause for approximately 0.3 seconds. The number of memory flushing activities is increasing with the number of write-operations in the workload because the number of write-operations can be referred to the amount of data in the system. In this experiment, there are 15 storage nodes and the DSePHR memory activity happens approximately 10, 5, and 3 times every minute at one of the storage node for the 100:0-write:read, 75:25-write:read, and 50:50-write:read workloads, respectively. Therefore, the number of DSePHR memory activities of the 75:25-write:read workload is larger than that of the 50:50-write:read workload.

Moreover, every 15 DSePHR memory flushing activities will create 1 HDFS file packing process of the 15 data blocks resulting from the 15 DSePHR memory flushing activities. This way, the 15 data blocks of 256MB or 3.84GB of data will be packed into a HDFS file. Thus, there are 40, 20, and 12 HDFS file packing process activities in 1 hour for 100:0-write:read, 75:25-write:read and 50:50-write:read workloads, respectively. Both DSePHR memory flushing activity and DSePHR HDFS file packing activity can result in the slightly high download time performance of the 75:25-write:read workload in comparison with that of the 50:50-write:read workload. The ratio of DSePHR memory flushing and HDFS packing activities of the 75:25-write:read workload to that of the 50:50-write:read workload is 5 to 3.

Table 4.21 Download time of each file type of 50:50-write:read workload

File types	Filename	Size	Download time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	5.960	1.269	0.294	0.102	0.016
	4ccd.xml	31KB	5.771	1.624	0.203	0.110	0.020
	8excel.xlsx	41KB	4.615	1.120	0.292	0.099	0.021
	9sound.ogg	160KB	7.785	1.239	0.294	0.174	0.042
	1ecg.jpg	400KB	6.298	1.452	0.552	0.291	0.088
	7word.docx	440KB	5.942	1.513	0.627	0.321	0.095
	5ccd.xml	620KB	9.898	1.601	0.820	0.420	0.128
	6ccd.pdf	690KB	6.600	1.887	0.938	0.459	0.139
	3xray.png	4.01MB	10.384	5.634	4.281	2.086	0.758
	10sound.mp3	8.65MB	15.070	10.730	8.633	4.227	1.620
HDFS files	11videosmall.mp4	27MB	49.537	30.840	23.840	12.038	4.953
	12videobig.mp4	232MB	240.318	199.861	143.312	80.638	42.462

Table 4.22 shows the ratio of the average download time observed from the 75:25-write:read workload to that of the 50:50-write:read workload. According to the ratio values in Table 4.22, the average download time observed from the 75:25-write:read workload is 1.08 to 1.58 times of the average download time observed from the 50:50-write:read workload. Furthermore, the effect on the small file size is larger than that of the large file size in HBase file group. That is, the ratio of the four small file sizes (i.e., the MRI image file, the CCD metadata file, the spreadsheet file, and the heartbeat sound file) is 1.51, 1.39, 1.58, and 1.49. The ratio of the four medium file sizes (i.e., 400KB to 680KB) is 1.25, 1.35, 1.20, and 1.21. The ratio of the two large file sizes (i.e., 4.01MB and 8.65MB) is 1.12 and 1.08. For the

HDFS file group, the ratio is 1.09 and 1.21, which is larger than that of the two large HBase files but it is smaller than that of other HBase files. The effect on the small size files is larger than that of the large size files because the average download time of the small size files (i.e., smaller than 400KB) is less than 0.3 seconds for both 75:25-write:read and 50:50-write:read workloads. The DSePHR memory flushing activity takes approximately 280 milliseconds or 0.28 seconds. Therefore, if the DSePHR memory flushing activity occurs during the download processing of the small size files (i.e., smaller than 400KB), the download time of the files will be greatly affected.

Because every 15 DSePHR memory flushing activities can create a DSePHR HDFS file packing activity, the download time of the large files (i.e., the HDFS file group) is also affected by the DSePHR memory flushing activities. However, the effect is not as high as that of the small size files because the DSePHR HDFS file packing activity can be done in parallel with other activities. Meaning, the DSePHR HDFS file packing activity will not stop any other DSePHR HDFS file storing or retrieving processes.

Table 4.22 The ratio of the average download time of the 75:25-write:read to that of the 50:50-write:read workload

File types	Filenames	Size	Average download time (s)		Ratio
			75:25-write:read	50:50-write:read	
HBase files	2mri.jpg	30KB	0.154	0.102	1.51
	4ccd.xml	31KB	0.153	0.110	1.39
	8excel.xlsx	41KB	0.156	0.099	1.58
	9sound.ogg	160KB	0.259	0.174	1.49
	1ecg.jpg	400KB	0.363	0.291	1.25
	7word.docx	440KB	0.433	0.321	1.35
	5ccd.xml	620KB	0.504	0.420	1.20
	6ccd.pdf	690KB	0.554	0.459	1.21
	3xray.png	4.01MB	2.328	2.086	1.12
	10sound.mp3	8.65MB	4.584	4.227	1.08
HDFS files	11videosmall.mp4	27MB	13.069	12.038	1.09
	12videobig.mp4	232MB	97.871	80.638	1.21

In conclusion, the upload and download time performance of the DSePHR system are studied in this section. Both the upload and download time performance of the HBase files is smaller than that of the HDFS files because the HBase files are smaller in size than that of the HDFS files. The linear relationship between the size of the files and the average performance does not clearly show for all files smaller than 4.01MB. However, the relationship is clearly visible for the two large files in the HBase file group and the two files in the HDFS file group. That is, the larger the file size the larger the upload/download time. The linear relationship is clearly visible for all three workloads (i.e., 100:0-write:read, 75:25-write:read, and 50:50-write:read).

The download time performance is larger than that of the upload time performance because the data must be completely downloaded from the DSePHR storage to the DSePHR web services in order to be sent to the client. Thus, the download time will include the time that the read request is made at the client until the time that the data is completely sent to the client via the DSePHR web service. Thus, the data transmission is occurring twice including the transmission time from the DSePHR storage node to the DSePHR web service and the transmission time from the DSePHR web service to the client. Therefore, the average download time is expected to be approximately twice the average upload time.

The DSePHR storage nodes must perform the memory flushing process in order to move the HBase files out of the memory and store the data on the HDFS storages. Thus, the DSePHR will spend some times to perform this activity. This activity can affect the download time performance. During the DSePHR memory flushing activity, the DSePHR storage nodes will pause for approximately 0.3 seconds. The number of memory flushing activities is increasing with the number of write-operations in the workload because the number of write-operations can be referred to the amount of data in the system. Thus, the amount of DSePHR memory flushing activity occurs in the 75:25-write:read workload is larger than that of the 50:50-write:read workload. The effect of the DSePHR memory flushing activity is clearly shown in the average download time of the 75:25-write:read workload which is larger than that of the 50:50-write:read workload in all cases.

Moreover, the DSePHR memory flushing activity can create a DSePHR HDFS file packing activity which will affect the download time of the large files (i.e., the HDFS file group). However, the DSePHR HDFS file packing activity can be done in parallel with other DSePHR HDFS file storing or retrieving activities. Thus, the effect on the HDFS file group download time performance is not as large as the effect on the download time performance of the small size files in the HBase file group.

4.3.3.4 Baseline performance comparison

The DSePHR performance in the realistic environment shown in Section 4.3.3.1 - 4.3.3.3, is comparing with the DSePHR baseline performance. Table 4.23 shows the comparison of the overall upload time of each situation and the baseline. The “Base(mean)” column is an average value of every files in HBase or HDFS files type. For the HBase files, the average upload times of 100:0, 75:25 and 50:50-write:read are 2.25, 1.96 and 1.80 times of the baseline, respectively. For the HDFS files, the average upload times of 100:0, 75:25 and 50:50-write:read are 2.12, 1.77 and 1.66 times of the baseline, respectively. Table 4.24 shows the upload time comparison of each file type with the baseline. The “Base” column is the baseline performance of the file. Most baseline performance is slightly larger than the minimum value of each file for each situation, except 5ccd.xml in 50:50-write:read and 12videobig.mp4 in 100:0-write:read.

Table 4.23 Overall upload time of each situation and baseline

File types	Upload time (s)						
	100:0-write:read		75:25-write:read		50:50-write:read		Base(mean)
	Mean	Min	Mean	Min	Mean	Min	
HBase files	0.373	0.013	0.326	0.014	0.299	0.014	0.166
HDFS files	26.202	2.625	21.890	2.612	20.505	2.617	12.381

Table 4.24 Upload time of each file type of each situation and baseline

File types	Filename	Size	Upload time (s)						
			100:0-write:read		75:25-write:read		50:50-write:read		Base
			Mean	Min	Mean	Min	Mean	Min	
HBase files	2mri.jpg	30KB	0.058	0.013	0.047	0.014	0.042	0.015	0.017
	4ccd.xml	31KB	0.054	0.015	0.049	0.014	0.042	0.016	0.018
	8excel.xlsx	41KB	0.056	0.013	0.049	0.014	0.042	0.014	0.019
	9sound.ogg	160KB	0.085	0.023	0.072	0.023	0.061	0.023	0.029
	1ecg.jpg	400KB	0.146	0.047	0.125	0.046	0.117	0.048	0.054
	7word .docx	440KB	0.151	0.050	0.130	0.051	0.122	0.051	0.067
	5ccd.xml	620KB	0.236	0.071	0.205	0.074	0.182	0.081	0.076
	6ccd.pdf	690KB	0.223	0.073	0.199	0.074	0.177	0.077	0.084
	3xray.png	4.01MB	0.951	0.398	0.817	0.401	0.748	0.406	0.416
	10sound .mp3	8.65MB	1.849	0.843	1.566	0.834	1.455	0.854	0.876
HDFS files	11videosmall.mp4	27MB	5.687	2.625	4.688	2.612	4.317	2.617	2.661
	12videobig.mp4	232MB	46.650	22.114	38.691	22.086	36.082	22.064	22.100

Table 4.25 shows the comparison of overall download time of each situation and the baseline. The “Base(mean)” column represents the average value. For the HBase files, the average download times of 75:25 and 50:50-write:read are 3.09, 2.76 times of the baseline, respectively. For the HDFS files, the average download times of 75:25 and 50:50-write:read are 2.42 and 1.97 times of the

baseline, respectively. Table 4.26 shows the download time comparison of each file type and the baseline. The “Base” column is the baseline performance. Most baseline performance is slightly larger than the minimum value of each file for most situations, except 11videosmall.mp4 in both 75:25 and 50:50-write:read situations.

Table 4.25 Overall download time of each situation and baseline

File types	Download time (s)				
	75:25-write:read		50:50-write:read		Base(mean)
	Mean	Min	Mean	Min	
HBase files	0.933	0.019	0.834	0.016	0.302
HDFS files	57.260	4.971	46.542	4.953	23.658

Table 4.26 Download time of each file type of each situation and baseline

File types	Filename	Size	Download time (s)				
			75:25-write:read		50:50-write:read		Base
			Mean	Min	Mean	Min	
HBase files	2mri.jpg	30KB	0.154	0.019	0.102	0.016	0.031
	4ccd.xml	31KB	0.153	0.021	0.110	0.020	0.026
	8excel.xlsx	41KB	0.156	0.023	0.099	0.021	0.025
	9sound.ogg	160KB	0.259	0.046	0.174	0.042	0.047
	1ecg.jpg	400KB	0.363	0.089	0.291	0.088	0.091
	7word.docx	440KB	0.433	0.095	0.321	0.095	0.099
	5ccd.xml	620KB	0.504	0.129	0.420	0.128	0.131
	6ccd.pdf	690KB	0.554	0.140	0.459	0.139	0.144
	3xray.png	4.01MB	2.328	0.760	2.086	0.758	0.766
	10sound.mp3	8.65MB	4.584	1.620	4.227	1.620	1.658
HDFS files	11videosmall.mp4	27MB	13.069	4.971	12.038	4.953	5.102
	12videobig.mp4	232MB	97.871	42.738	80.638	42.462	42.214

Table 4.25 and Table 4.26 show the ratio upload and download time between the average value of each file and the baseline under 100:0, 75:25 and 50:50-write:read. The trend of ratio decreases when the file size increases for all situations because the small file uses less time to upload and download and it is

easily affects by large files or network latency. The ratio in 50:50-write:read situation is less than other situations for both the upload and download time. It can be explained that the 50:50-write:read has less write operation than other situations. The more write operations can affect the DSePHR performance because of a small number of clean memory and flushing activities.

Table 4.27 Upload time and baseline ratio of each situation

File types	Filename	Size	Upload time (Ratio)		
			100:0-write:read	75:25-write:read	50:50-write:read
HBase files	2mri.jpg	30KB	3.412	2.765	2.471
	4ccd.xml	31KB	3.000	2.722	2.333
	8excel.xlsx	41KB	2.947	2.579	2.211
	9sound.ogg	160KB	2.931	2.483	2.103
	1ecg.jpg	400KB	2.704	2.315	2.167
	7word.docx	440KB	2.254	1.940	1.821
	5ccd.xml	620KB	3.105	2.697	2.395
	6ccd.pdf	690KB	2.655	2.369	2.107
	3xray.png	4.01MB	2.286	1.964	1.798
	10sound.mp3	8.65MB	2.111	1.788	1.661
HDFS files	11videosmall.mp4	27MB	2.137	1.762	1.622
	12videobig.mp4	232MB	2.111	1.751	1.633

Table 4.28 Download and baseline ratio of each situation

File types	Filename	Size	Upload time (Ratio)	
			75:25-write:read	50:50-write:read
HBase files	2mri.jpg	30KB	4.968	3.290
	4ccd.xml	31KB	5.885	4.231
	8excel.xlsx	41KB	6.240	3.960
	9sound.ogg	160KB	5.511	3.702
	1ecg.jpg	400KB	3.989	3.198
	7word.docx	440KB	4.374	3.242
	5ccd.xml	620KB	3.847	3.206
	6ccd.pdf	690KB	3.847	3.188
	3xray.png	4.01MB	3.039	2.723
	10sound.mp3	8.65MB	2.765	2.549
HDFS files	11videosmall.mp4	27MB	2.562	2.359
	12videobig.mp4	232MB	2.318	1.910

4.3.4 Effect of File Type Mixture

Previous results show the DSePHR performance on the original PHR file type mixture as described in Section 3.5 of 3 situations including 100:0-write:read, 75:25-write:read and 50:50-write:read. In this section, the PHR file type mixture is changed such that the amount of large files increase from 9% to 25% in all three workloads (i.e., 100:0-write:read, 75:25-write:read, 50:50-write:read). Table 4.29 shows the main differences between the original PHR data file type mixture and the uniform PHR data file type mixture. According to the data shown in Table 4.29, the number of small size files (i.e., smaller than 1MB) will be reduced from 73% to 50% while the number of large size files (i.e., larger than 1MB) will be increased from 27% to 50%. As a result, the amount of work on the HDFS file group will be increased from the workloads used in the previous three sections (i.e., Section 4.2.1-4.2.3). The number of DSePHR memory flushing activities must be increased from that of the workloads used in the previous three sections (i.e., Section 4.2.1-4.2.3). Moreover, the amount of data in the workload increases from 531.561GB to 1,363.356GB because the number of files in the workload is set to 40,000 files. Thus, the amount of data of the uniform file type mixture will be larger than that of the original PHR type mixture. Therefore, the total time to conduct the experiment on the 100:0-write:read uniform file type mixture workload (i.e., 9.8 hours) is longer than that of the 100:0-write:read original file type mixture workload (i.e., 3.8 hours).

Table 4.29 The number of files in each type of the two mixtures

File size	Original	Uniform	Effects
Larger than 100KB	27%	25%	Reduce
100KB to 1MB	46%	25%	Reduce
1MB to 10MB	18%	25%	Increase
Larger than 10MB	9%	25%	Increase

Table 4.30 shows the overall upload time performance of both small files (i.e., HBase files) and large files (i.e., HDFS files) of the 100:0-write:read uniform file type mixture workload. As expected, the maximum upload time of the HBase file group is from the 10sound.mp3 which is the largest file in the group while the maximum upload time of the HDFS file group is from the 12videobig.mp4 which is the largest file in the group. The upload time performance of the 100:0-write:read workload of the uniform file type mixture shown in Table 4.30 is larger than the upload time performance of the 100:0-write:read workload of the original file type mixture shown in Table 4.12 in all cases. This can be caused by the fact that the

amount of 100:0-write:read workload of the uniform file type mixture workload data is 2.56 times the amount of 100:0-write:read workload of the original PHR file type mixture workload data. To analyze the detail performance of each file type, Table 4.31 shows the upload time performance of each file in the 100:0-write:read uniform file type mixture workload sorted by the file size.

According to the performance of each file type shown in Table 4.31, the trend of the average upload time performance of each file type is similar to that observed from the 100:0-write:read workload with original PHR file type mixture shown in Table 4.13. That is, the linear relationship between the file size and the upload time is starting to show on the files of size 4.01MB or larger. The upload time of all files of size smaller or equal to 4.01MB is less than 1 second. The average upload time performance of each file type shown in Table 4.31 is either similar to or slightly larger than that of the 100:0-write:read workload with original file type mixture shown in Table 4.13, except the MRI image file. However, the average upload time performance improvement of the original file type mixture over the uniform file type mixture is less than 1 second. That is, the average upload time performance improvement of the original file type mixture over the uniform file type mixture is smaller than 0.07, and 0.8 seconds for the files smaller than 232MB and the 232MB files, respectively.

Table 4.30 Overall upload time of 100:0-write:read uniform file type mixture workload

File types	Upload time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	10.321	4.171	2.502	0.566	0.013
HDFS files	148.622	117.638	84.242	26.542	2.598

Table 4.31 Upload time of each file type of 100:0-write:read uniform file type mixture workload

File types	Filename	Size	Upload time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	4.164	0.693	0.103	0.057	0.013
	4ccd.xml	31KB	2.966	0.801	0.086	0.059	0.015
	8excel.xlsx	41KB	3.702	0.653	0.092	0.058	0.014
	9sound.ogg	160KB	2.295	0.768	0.158	0.087	0.023
	1ecg.jpg	400KB	1.539	0.923	0.306	0.152	0.046
	7word.docx	440KB	2.681	0.870	0.308	0.155	0.050
	5ccd.xml	620KB	4.975	1.428	0.515	0.251	0.072
	6ccd.pdf	690KB	2.121	1.267	0.521	0.240	0.075
	3xray.png	4.01MB	6.985	3.133	2.208	0.985	0.400
	10sound.mp3	8.65MB	10.321	5.490	4.282	1.913	0.843
HDFS files	11videosmall.mp4	27MB	20.470	16.305	12.740	5.743	2.598
	12videobig.mp4	232MB	148.622	127.220	101.740	47.420	22.080

In conclusion, the increasing amount of data in the workload does not significantly affect the upload time performance of the files in the DSePHR system for the 100:0-write:read workload situation.

Next, Table 4.32 shows the overall upload time performance of both small files (i.e., HBase files) and large files (i.e., HDFS files) of the 75:25-write:read uniform file type mixture workload. As expected, the maximum upload time of the

HBase file group is from the 10sound.mp3 which is the largest file in the group while the maximum upload time of the HDFS file group is from the 12videobig.mp4 which is the largest file in the group. The upload time performance of the small files is smaller than that of the upload time performance of the large files. The upload time performance of the 75:25-write:read workload of the uniform file type mixture shown in Table 4.32 is larger than the upload time performance of the 75:25-write:read workload of the original PHR file type mixture shown in Table 4.14 in all cases, except the 95th-percentile and the average values of the HDFS files. The increasing in the upload time can be caused by the increasing amount of data between the two workloads. That is, the amount of write operations of the 75:25-write:read workload of the uniform file type mixture workload data is 2.6 times the amount of write operations of the 75:25-write:read workload of the original PHR file type mixture workload data. However, the average upload time performance of the HDFS files is not increasing. To analyze the detail performance of each file type, Table 4.33 shows the upload time performance of each file in the 75:25-write:read uniform file type mixture workload sorted by the file size.

Table 4.32 Overall upload time of 75:25-write:read uniform file type mixture workload

File types	Upload time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	8.207	3.213	2.062	0.470	0.014
HDFS files	125.737	87.477	63.834	21.847	2.613

Table 4.33 Upload time of each file type of 75:25-write:read uniform file type mixture workload

File types	Filename	Size	Upload time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	1.781	0.361	0.080	0.047	0.014
	4ccd.xml	31KB	2.676	0.478	0.083	0.051	0.014
	8excel.xlsx	41KB	1.717	0.491	0.086	0.050	0.014
	9sound.ogg	160KB	1.151	0.474	0.138	0.071	0.023
	1ecg.jpg	400KB	2.361	0.809	0.272	0.132	0.047
	7word.docx	440KB	2.091	0.519	0.282	0.133	0.051
	5ccd.xml	620KB	3.463	0.908	0.408	0.208	0.073
	6ccd.pdf	690KB	1.998	1.150	0.431	0.207	0.073
	3xray.png	4.01MB	4.820	2.405	1.709	0.825	0.393
	10sound.mp3	8.65MB	8.207	4.172	3.302	1.602	0.853
HDFS files	11videosmall.mp4	27MB	19.415	12.963	10.041	4.771	2.613
	12videobig.mp4	232MB	125.737	95.856	75.111	38.994	22.075

As expected, the performance trend of the upload time performance shown in Table 4.33 is similar to that of other three previously shown workloads. That is, the performance of all files smaller than 4.01MB is less than 0.3 seconds and the increasing of the average upload time does not show any linear relationship with the file size. However, the linear relationship between the average upload time and the file size shows on two large files in the HBase file group and the two files in the HDFS file group. The average upload time performance of each file type shown in Table 4.33 is either similar to or slightly larger than that of the 75:25-write:read workload with original PHR file type mixture shown in Table 4.16, except the heartbeat sound file. However, the average upload time performance improvement of the original PHR file type mixture over the uniform file type mixture is less than 1 second. That is, the average upload time performance improvement of the original PHR file type mixture over the uniform file type mixture is smaller than 0.09, and 0.4 seconds for the files smaller than 232MB and the 232MB files, respectively.

Table 4.34 shows the overall download time performance of both small files (i.e., HBase files) and large files (i.e., HDFS files) of the 75:25-write:read uniform file type mixture workload. As expected, the maximum download time of the HBase file group is from the 10sound.mp3 which is the largest file in the group while the maximum download time of the HDFS file group is from the 12videobig.mp4 which is the largest file in the group. The download time performance of the small files is smaller than that of the download time performance of the large files. Unlike the upload time performance, the download time performance of the 75:25-write:read workload of the uniform file type mixture shown in Table 4.34 is not clearly larger than the download time performance of the 75:25-write:read workload of the original PHR file type mixture shown in Table 4.15, except the 99th-percentile, 95th-percentile and the average values of the HBase files. This is unexpected because the amount of read operation data in the 75:25-write:read workload of the uniform file type mixture is approximately 2.5 times that of the 75:25-write:read workload of the original PHR file type mixture. To analyze the detail performance of each file type, Table 4.35 shows the download time performance of each file in the 75:25-write:read uniform file type mixture workload sorted by the file size.

Table 4.34 Overall download time of 75:25-write:read uniform file type mixture workload

File types	Download time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	15.405	8.924	6.004	1.405	0.019
HDFS files	258.550	197.882	150.516	56.037	4.966

Table 4.35 Download time of each file type of 75:25-write:read uniform file type mixture workload

File types	Filename	Size	Download time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	2.861	1.339	0.400	0.133	0.019
	4ccd.xml	31KB	6.934	2.681	0.423	0.169	0.022
	8excel.xlsx	41KB	3.016	1.024	0.325	0.125	0.024
	9sound.ogg	160KB	4.873	2.162	0.748	0.263	0.046
	1ecg.jpg	400KB	3.057	1.560	0.756	0.374	0.090
	7word.docx	440KB	5.576	2.137	0.779	0.395	0.097
	5ccd.xml	620KB	6.445	1.796	0.952	0.496	0.130
	6ccd.pdf	690KB	3.396	2.047	1.069	0.543	0.140
	3xray.png	4.01MB	9.291	6.025	4.668	2.415	0.761
	10sound.mp3	8.65MB	15.405	12.052	9.123	4.816	1.620
HDFS files	11videosmall.mp4	27MB	55.009	31.207	25.844	13.265	4.966
	12videobig.mp4	232MB	258.55	204.544	174.617	97.622	42.712

As expected, the performance trend of the download time performance shown in Table 4.35 is similar to that of other three previously shown workloads. That is, the download time performance of all files smaller than 4.01MB is less than 0.6 seconds and the increasing of the average download time does not show any linear relationship with the file size. However, the linear relationship between the average download time and the file size shows on two large files in the

HBase file group and the two files in the HDFS file group. The average download time performance of each file type shown in Table 4.35 is slightly smaller or larger than that of the 75:25-write:read workload with original PHR file type mixture shown in Table 4.17. However, the performance difference is less than 1 second. That is, the performance difference is less than 0.09 second for all files smaller than 8.65MB and is less than 0.3 seconds for all files larger than 8.65MB.

In conclusion, the increasing amount of data in the workload does not significantly affect the upload and download time performance of the files in the DSePHR system for the 75:25-write:read workload situation.

Table 4.36 shows the overall upload time performance of both small files (i.e., HBase files) and large files (i.e., HDFS files) of the 50:50-write:read uniform file type mixture workload. As expected, the maximum upload time of the HBase file group is from the 10sound.mp3 which is the largest file in the group while the maximum upload time of the HDFS file group is from the 12videobig.mp4 which is the largest file in the group. The upload time performance of the small files is smaller than that of the upload time performance of the large files. The upload time performance of the 50:50-write:read workload of the uniform file type mixture shown in Table 4.36 is larger than the upload time performance of the 50:50-write:read workload of the original PHR file type mixture shown in Table 4.18 in all cases, except the maximum value of the HBase file and the minimum value of the HDFS files. The increasing in the upload time performance can be caused by the increasing amount of data between the two workloads. That is, the amount of write operations of the 50:50-write:read workload of the uniform file type mixture workload data is 2.51 times the amount of write operations of the 50:50-write:read workload of the original PHR file type mixture workload data. To analyze the detail performance of each file type, Table 4.37 shows the upload time performance of each file in the 50:50-write:read uniform file type mixture workload sorted by the file size.

Table 4.36 Overall upload time of 50:50-write:read uniform file type mixture workload

File types	Upload time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	5.375	2.981	1.902	0.436	0.014
HDFS files	116.072	82.459	59.992	20.533	2.579

Table 4.37 Upload time of each file type of 50:50-write:read uniform file type mixture workload

File types	Filename	Size	Upload time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	1.585	0.278	0.073	0.040	0.015
	4ccd.xml	31KB	1.368	0.284	0.081	0.043	0.014
	8excel.xlsx	41KB	1.192	0.286	0.080	0.044	0.014
	9sound.ogg	160KB	2.189	0.295	0.139	0.066	0.024
	1ecg.jpg	400KB	1.226	0.481	0.251	0.116	0.047
	7word.docx	440KB	1.967	0.410	0.278	0.124	0.051
	5ccd.xml	620KB	3.546	0.854	0.399	0.199	0.076
	6ccd.pdf	690KB	1.581	0.716	0.410	0.185	0.075
	3xray.png	4.01MB	4.806	2.192	1.598	0.769	0.404
	10sound.mp3	8.65MB	5.375	4.086	3.079	1.491	0.853
HDFS files	11videosmall.mp4	27MB	16.510	11.760	9.059	4.346	2.579
	12videobig.mp4	232MB	116.072	87.760	71.927	36.535	22.062

As expected, the performance trend of the upload time performance shown in Table 4.37 is similar to that of other three previously shown workloads. That is, the performance of all files smaller than 4.01MB is less than 0.2 seconds and the increasing of the average upload time does not show any linear relationship with the file size. However, the linear relationship between the average upload time and the file size shows on two large files in the HBase file group and the two files in the HDFS file group. The average upload time performance of each file type shown in Table 4.37 is either similar to or slightly larger than that of the 50:50-write:read workload with original PHR file type mixture shown in Table 4.20, except the MRI image file and the ECG image file. However, the average upload time performance improvement of the original PHR file type mixture over the uniform file type mixture is less than 1 second. That is, the average upload time performance improvement of the original PHR file type mixture over the uniform file type mixture is smaller than 0.04, and 0.5 seconds for the files smaller than 232MB and the 232MB files, respectively.

Table 4.38 shows the overall download time performance of both small files (i.e., HBase files) and large files (i.e., HDFS files) of the 50:50-write:read uniform file type mixture workload. As expected, the maximum download time of the HBase file group is from the 10sound.mp3 which is the largest file in the group while the maximum download time of the HDFS file group is from the 12videobig.mp4 which is the largest file in the group. The download time performance of the small files is smaller than that of the download time performance of the large files. The download time performance of the 50:50-write:read workload of the uniform file type mixture shown in Table 4.38 is larger than the download time performance of the 50:50-write:read workload of the original PHR file type mixture shown in Table 4.19, except the 99th-percentile, 95th-percentile and the average values of the HDFS files. This is expected because the amount of read operation data in the 50:50-write:read workload of the uniform file type mixture is approximately 2.54 times that of the 50:50-write:read workload of the original PHR file type mixture.

Similar to the download performance trend of the original PHR file type mixture workloads, the download time performance of the 50:50-write:read workload of uniform file type mixture shown in Table 4.38 is smaller than the download time performance of the 75:25-write:read workload of the uniform file type mixture shown in Table 4.35. This can be explained by the number of the DSePHR memory flushing activity because the number of DSePHR memory flushing activity of the 75:25-write:read workload of the uniform file type mixture is larger than that of the 50:50-write:read workload. That is, the number of DSePHR memory flushing activity occurs 4.23, 2.66 and 1.83 times in a minute for the 100:0-write:read, 75:25-write:read and 50:50-write:read workload of the uniform file type mixture.

Table 4.38 Overall download time of 50:50-write:read uniform file type mixture workload

File types	Download time (s)				
	Max	99th-	95th-	Mean	Min
HBase files	17.834	8.318	5.421	1.213	0.020
HDFS files	247.980	161.820	124.312	45.958	4.955

To analyze the detail performance of each file type, Table 4.39 shows the download time performance of each file in the 50:50-write:read uniform file type mixture workload sorted by the file size. As expected, the performance trend of the download time performance shown in Table 4.39 is similar to that of other three previously shown workloads. That is, the download time performance of all files smaller than 4.01MB is less than 0.5 seconds and the increasing of the average download time does not show any linear relationship with the file size. However, the linear relationship between the average download time and the file size shows on two large files in the HBase file group and the two files in the HDFS file group. The average download time performance of each file type shown in Table 4.39 is slightly smaller or larger than that of the 50:50-write:read workload with original PHR file type mixture shown in Table 4.21. However, the performance difference is less than 1 second. That is, the performance difference is less than 0.06 second for all files smaller than 27MB and is less than 0.2 seconds for all files larger than 27MB.

Table 4.39 Download time of each file type of 50:50-write:read uniform file type mixture workload

File types	Filename	Size	Download time (s)				
			Max	99th-	95th-	Mean	Min
HBase files	2mri.jpg	30KB	5.866	0.838	0.301	0.096	0.020
	4ccd.xml	31KB	6.403	0.643	0.271	0.094	0.021
	8excel.xlsx	41KB	5.497	1.087	0.277	0.106	0.020
	9sound.ogg	160KB	6.108	1.060	0.331	0.172	0.045
	1ecg.jpg	400KB	1.739	1.009	0.547	0.294	0.089
	7word.docx	440KB	4.711	0.902	0.609	0.323	0.095
	5ccd.xml	620KB	5.741	1.511	0.855	0.434	0.128
	6ccd.pdf	690KB	7.007	1.697	0.867	0.455	0.140
	3xray.png	4.01MB	7.313	5.195	4.198	2.141	0.759
	10sound.mp3	8.65MB	17.834	10.030	8.482	4.283	1.621
HDFS files	11videosmall.mp4	27MB	35.124	29.056	23.314	11.900	4.955
	12videobig.mp4	232MB	247.980	179.380	140.860	80.521	42.073

In conclusion, the increasing amount of data in the workload does not significantly affect the upload and download time performance of the files in the DSePHR system for the 50:50-write:read workload situation.

CHAPTER 5

CONCLUSION

In this chapter, the conclusions of this thesis including problem statement, motivation, requirements of encrypted PHR data storage, the design of encrypted PHR storage, approach to evaluate the storage and the experimental results are shown in section 5.1. The limitation and suggestion of this work are explained in section 5.2.

5.1 Conclusion

Personal health records (PHRs) are the data that belongs to your life. PHRs can be sensitive data such as disgraceful disease or dangerous medication. The secure cloud PHR systems have emerged recently, providing on encrypted method to PHR data and promising to store the encrypted data in cloud storage. However, a mechanism to store massive encrypted PHR data to a cloud storage is not addressed clearly. The secure cloud PHR systems assume that the encrypted PHR data can store in a general cloud storage without any issue. Although the encrypted PHR data can be stored in the general cloud storage, it is difficult to access the data. The encrypted PHR data lacks of its related information, making it difficult to pre-process the data for convenient retrieval. The general storages do not provide any index of the data by users and time because the storages provide a bucket-style storage for storing the whole data at same place.

This work provides a distributed storage for storing encrypted PHR data and an API for retrieving the encrypted PHR data. The storage has designed according to the encrypted PHR storage requirements including (1) the storage can support a high volume of the data; (2) the data can be accessed and retrieved immediately; (3) the user can access the data easily. To achieve the encrypted PHR storage requirements, the storage should provide scalability, robustness, failure recovery, fast responding and suitable interface to participate with user.

The distributed storage for encrypted personal health data (DSePHR) has been proposed in this thesis. DSePHR contains 2 parts including the API for distributed storage and distribute storage. DSePHR uses both HDFS (Hadoop distributed file system) and HBase (Non-relational database) as a fundamental storage framework. HDFS provides a replication mechanism for the load balancing feature and failure recovery feature. HDFS can also add more storage capacity without shutting down the system. HBase is a non-relational database that can persist on HDFS and support a replication feature from HDFS.

The interface of API has been implemented by REST interface which is a cross platform interface (de-facto standard). It can be used on various platform. DSePHR creates the index of the encrypted PHR data using its properties: the owner of the data, the incoming time of the data and the size of the data. The index of the encrypted PHR data is stored in HBase. With the DSePHR’s designed index, all files of the same user are located close to each other and sorted by the incoming time. The users can retrieve the whole data or a specified data of the user. The encrypted PHR data is divided into large size files and small size files. The large size files are stored directly on HDFS. The small files are stored on HBase acting BLOB (large binary object) to prevent the high consumption memory problem. The tuning configuration and pre-split table approach of HBase are adopted to improve the throughput of HBase for storing small files.

The experiment parts including the DSePHR APIs, memory consumption issue and DSePHR performance on effect of the limited storage, effect of the write-read ratio and effect of file type mixture. The DSePHR APIs provide the interface for client or PHR system to store to or retrieve from the DSePHR framework. The memory issue experimental results show that the DSePHR can solve the high consumption memory problem of Namenode by storing small files in HBase. The performance of DSePHR under limited storage space shows that the extra storage can be added to the DSePHR system without shutting down the system. The extra storages take around 1-3 minutes to respond to the new incoming data. Although the system resource usage is very high (i.e., 90%), the performance of both upload and download time shows a slight effects from such situation. However, the extra storage is recommended to be added to the DSePHR system when the resource usage of the system reaches 90%. In the DSePHR performance under realistic environment, both upload and download performance depends on a number of upload requests. That is, the high number of upload requests can affect both upload and download time performance. The results show that the performance on 50:50-write:read situation is the best performance. The download time is larger than the upload time because the data need to be completely downloaded from the distributed storage first, then sent to the client. Although the large file size is larger in the uniform file type mixture, the DSePHR still can handle the situation with a slightly performance difference.

5.2 Limitation and Suggestion

The limitation of DSePHR API is the size of the upload files. Although the DSePHR caches the incoming data on its disk, but the request of the data persists in the memory first before it is saved to disk. The large file can fill the DSePHR memory fast. The file size should be limited considering the memory of the DSePHR web service, or the memory of the DSePHR must be increased to handle larger files. Another limitation is the throughput of the DSePHR to the distributed storage and to the client. With a single LAN interface, it can affect the throughput because there is an overhead between sending and receiving the data simultaneously. The suggestion is to use 2 LAN interfaces, one for the client to DSePHR, another one for the DSePHR to the distributed storage. This way, the throughput of the system can be increased.

The limitation of HDFS is that the Namenode can be a single point of failure because the Namenode stores the metadata of all file in HDFS. The client cannot retrieve the data if the Namenode is down. However, there are some solutions to solve this limitation. First solution is to use a high quality machine to work as the Namenode server. Second solution is to use a secondary Namenode that is provided by HDFS. The secondary Namenode is a backup node of the Namenode. Third solution is to use a high availability Namenode (AvatarNode), by setting more than a single Namenode as a backup of the primary Namenode. If the primary Namenode goes down, another Namenode can catch up the work immediately.

REFERENCES

[1] N. R. Cook, J. A. Cutler, E. Obarzanek, J. E. Buring, K. M. Rexrode, S. K. Kumanyika, L. J. Appel, and P. K. Whelton, “ Long term effects of dietary sodium reduction on cardiovascular disease outcomes: observational follow-up of the trials of hypertension prevention (TOHP),” *BMJ*, vol. 334, no. 7599, pp. 885–885, Apr. 2007.

[2] W. E. M. LANDS, “ Dietary Fat and Health: The Evidence and the Politics of Prevention: Careful Use of Dietary Fats Can Improve Life and Prevent Disease,” *Ann. N. Y. Acad. Sci.*, vol. 1055, no. 1, pp. 179–192, Dec. 2005.

[3] P. C. Tang, J. S. Ash, D. W. Bates, J. M. Overhage, and D. Z. Sands, “Personal Health Records: Definitions, Benefits, and Strategies for Overcoming Barriers to Adoption,” *J. Am. Med. Informatics Assoc.*, vol. 13, no. 2, pp. 121–126, Mar. 2006.

[4] Markle foundation, “Connecting for Health-The personal health working group final report,” 2003.

[5] M. M. Hansen, T. Miron-Shatz, A. Y. S. Lau, and C. Paton, “Big Data in Science and Healthcare: A Review of Recent Literature and Perspectives. Contribution of the IMIA Social Media Working Group.,” *Yearb. Med. Inform.*, vol. 9, pp. 21–6, 2014.

[6] M. Li, S. Yu, K. Ren, and W. Lou, “Securing Personal Health Records in Cloud Computing: Patient-Centric and Fine-Grained Data Access Control in Multi-owner Settings,” Springer Berlin Heidelberg, 2010, pp. 89–106.

[7] C. Wang, X. Liu, and W. Li, “ Implementing a Personal Health Record Cloud Platform Using Ciphertext-Policy Attribute-Based Encryption,” in *2012 Fourth International Conference on Intelligent Networking and Collaborative Systems*, 2012, pp. 8–14.

[8] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, “ Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 131–143, Jan. 2013.

[9] F. Xhafa, J. Li, G. Zhao, J. Li, X. Chen, and D. S. Wong, “Designing cloud-based electronic health record system with attribute-based encryption,” *Multimed. Tools Appl.*, vol. 74, no. 10, pp. 3441–3458, May 2015.

[10] P. Thummavet and S. Vasupongayya, “ Privacy-preserving emergency access control for personal health records,” *MAEJO Int. J. Sci. Technol.*, vol. 9, no. 1,

pp. 108–120, 2015.

[11] Y. Li, L. Guo, and Y. Guo, “An Efficient and Performance-Aware Big Data Storage System,” Springer International Publishing, 2013, pp. 102–116.

[12] “My Health Record.” [Online]. Available: https://myhealthrecord.gov.au.

[13] “Google Fit.” [Online]. Available: https://www.google.com/fit/.

[14] “ Microsoft HealthVault. ” [Online] . Available: https://international.healthvault.com/.

[15] “ My Health Record report 2015-2016. ” [Online] . Available: https://myhealthrecord.gov.au/internet/mhr/publishing.nsf/Content/0F2F30876A95E7D4CA257F8A0008E337/ $File/ MyHealthRecordSystemOperatorAnnualReport-2015-16.PDF.

[16] E. Dumbill, “ Making Sense of Big Data,” *http://dx.doi.org/10.1089/big.2012.1503*, 2013.

[17] D. Sobhy, Y. El-Sonbaty, and M. A. Elnasr, “ MedCloud: Healthcare cloud computing system,” in *The 7th International Conference for Internet Technology and Secured Transactions*, 2012, pp. 161–166.

[18] A. Bahga and V. K. Madisetti, “ A Cloud-based Approach for Interoperable Electronic Health Records (EHRs),” *IEEE J. Biomed. Heal. Informatics*, vol. 17, no. 5, pp. 894–906, Sep. 2013.

[19] Y. Li, L. Guo, C. Wu, C.-H. Lee, and Y. Guo, “Building a cloud-based platform for personal health sensor data management,” in *IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, 2014, pp. 223–226.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[21] “Hadoop.” [Online]. Available: http://hadoop.apache.org/.

[22] “ Storing images in Hbase. ” [Online] . Available: http: / / apache-hbase. 679495. n3. nabble. com/ Storing-images-in-Hbase-td4036184. html. [Accessed: 30-Dec-2016].

[23] “imageshack.” [Online]. Available: https://imageshack.us/.

[24] W. Wang Lijun, H. Huang Yongfeng, C. Chen Ji, Z. Zhou Ke, and L. Li Chunhua, “ Medoop: A medical information platform based on Hadoop,” in *2013 IEEE 15th International Conference on e-Health Networking, Applications and Services (Healthcom 2013)*, 2013, pp. 1–6.

[25] B. Dong, Q. Zheng, F. Tian, K.-M. Chao, R. Ma, and R. Anane, “ An optimized approach for storing and accessing small files on cloud storage,” *J. Netw.*

Comput. Appl., vol. 35, no. 6, pp. 1847–1862, 2012.

[26] X. Bao, L. Liu, N. Xiao, F. Liu, Q. Zhang, and T. Zhu, " HConfig: Resource adaptive fast bulk loading in HBase," *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom) , 2014 International Conference on*. IEEE, pp. 215–224, 2014.

[27] X. Bao, L. Liu, N. Xiao, Y. Zhou, and Q. Zhang, " Policy-Driven Configuration Management for NoSQL," in *2015 IEEE 8th International Conference on Cloud Computing*, 2015, pp. 245–252.

[28] "Ganglia Monitoring system." [Online]. Available: http://ganglia.info/.

[29] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles - SOSP '11*, 2011, p. 85.

[30] N. Hongu, B. T. Pope, P. Bilgiç, B. J. Orr, A. Suzuki, A. S. Kim, N. C. Merchant, and D. J. Roe, " Usability of a smartphone food picture app for assisting 24-hour dietary recall: a pilot study," *Nutr. Res. Pract.*, vol. 9, no. 2, p. 207, Apr. 2015.

[31] T. Suzuki, H. Tanaka, S. Minami, H. Yamada, and T. Miyata, "Wearable wireless vital monitoring technology for smart health care," in *2013 7th International Symposium on Medical Information and Communication Technology (ISMICT)*, 2013, pp. 1–4.

APPENDIX

Table A.1 – A.5 present results each experiment of ideal situation. Unit is used in each table is second. Each of columns includes file: file name, num: a number of files, max: maximum time, avg: average time, min: minimum time, std: standard division, var: variance, 95th: 95 percentiles, 99th: 99 percentiles.

Table A.1 Upload time of 100:0-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	848	6.959	1.900	0.852	1.182	1.397	4.336	5.859
11videosmall.mp4	413	20.469	5.679	2.620	3.523	12.412	13.012	16.981
12videobig.mp4	481	129.562	47.318	22.141	25.663	658.590	101.537	117.282
1ecg.jpg	879	1.278	0.140	0.047	0.130	0.017	0.289	0.741
2mri.jpg	815	2.478	0.055	0.014	0.140	0.020	0.081	0.683
3xray.png	850	3.846	0.977	0.403	0.654	0.428	2.311	3.217
4ccd.xml	852	1.695	0.055	0.015	0.126	0.016	0.093	0.568
5ccd.xml	859	1.976	0.235	0.073	0.203	0.041	0.483	1.213
6ccd.pdf	855	2.370	0.236	0.075	0.227	0.052	0.533	1.261
7word.docx	885	1.589	0.150	0.050	0.164	0.027	0.307	0.952
8excel.xlsx	830	1.807	0.049	0.014	0.118	0.014	0.078	0.712
9sound.ogg	821	1.120	0.083	0.023	0.120	0.014	0.161	0.757

Table A.2 Upload time of 75:25-write:read situation **with original PHR file type mixture**

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	243	4.335	1.106	0.859	0.519	0.269	2.157	2.795
11videosmall.mp4	138	8.244	3.441	2.621	1.396	1.949	6.809	7.900
12videobig.mp4	123	61.222	27.281	22.110	7.770	60.376	45.624	56.287
1ecg.jpg	265	0.892	0.077	0.046	0.069	0.005	0.180	0.297
2mri.jpg	245	0.282	0.024	0.015	0.020	0.000	0.052	0.080
3xray.png	274	3.247	0.591	0.403	0.365	0.133	1.333	1.766
4ccd.xml	252	1.023	0.033	0.015	0.076	0.006	0.055	0.207
5ccd.xml	259	1.191	0.133	0.073	0.106	0.011	0.259	0.654
6ccd.pdf	247	0.692	0.120	0.074	0.087	0.008	0.307	0.493
7word.docx	256	0.815	0.091	0.051	0.077	0.006	0.212	0.392
8excel.xlsx	277	0.328	0.025	0.015	0.026	0.001	0.054	0.078
9sound.ogg	225	0.283	0.042	0.024	0.036	0.001	0.108	0.218

Table A.3 Download time of 75:25-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	53	126.667	14.931	1.975	26.986	728.223	76.217	115.325
11videosmall.mp4	37	304.664	20.231	6.305	48.203	2323.494	27.904	217.385
12videobig.mp4	19	129.643	84.283	47.324	24.457	598.158	124.844	128.683
1ecg.jpg	76	162.443	15.035	0.091	30.332	920.033	75.686	130.542
2mri.jpg	83	106.676	11.249	0.024	23.857	569.143	69.022	91.728
3xray.png	75	93.932	14.789	0.902	25.884	670.001	74.833	91.224
4ccd.xml	63	143.368	20.901	0.025	37.904	1436.717	96.954	137.578
5ccd.xml	75	149.265	21.883	0.138	37.699	1421.251	103.802	143.603
6ccd.pdf	72	90.470	17.037	0.214	26.698	712.809	74.828	88.381
7word.docx	66	157.400	25.594	0.113	42.255	1785.503	113.716	156.400
8excel.xlsx	61	135.957	20.573	0.027	33.966	1153.687	92.431	116.093
9sound.ogg	62	156.969	18.943	0.052	36.652	1343.381	113.246	150.202

Table A.4 Upload time of 50:50-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	249	3.225	1.128	0.859	0.435	0.189	2.080	2.809
11videosmall.mp4	123	14.369	3.373	2.617	1.395	1.945	5.845	7.189
12videobig.mp4	107	77.729	28.041	22.084	10.923	119.303	46.066	73.328
1ecg.jpg	254	0.813	0.079	0.050	0.063	0.004	0.172	0.251
2mri.jpg	259	0.269	0.025	0.016	0.020	0.000	0.059	0.081
3xray.png	251	1.976	0.561	0.407	0.259	0.067	1.132	1.476
4ccd.xml	280	1.029	0.032	0.017	0.064	0.004	0.070	0.090
5ccd.xml	265	0.474	0.133	0.081	0.064	0.004	0.277	0.366
6ccd.pdf	272	1.236	0.128	0.074	0.112	0.013	0.303	0.462
7word.docx	253	1.464	0.094	0.051	0.121	0.015	0.213	0.683
8excel.xlsx	244	0.261	0.026	0.016	0.020	0.000	0.059	0.091
9sound.ogg	280	0.372	0.041	0.023	0.036	0.001	0.101	0.164

Table A.5 Download time of 50:50-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	269	23.475	6.504	1.700	3.759	14.127	12.731	18.395
11videosmall.mp4	109	231.603	28.321	5.623	35.507	1260.749	77.892	216.668
12videobig.mp4	108	376.940	151.061	46.212	70.167	4923.377	276.155	347.929
1ecg.jpg	256	8.398	0.700	0.094	1.276	1.628	2.750	7.393
2mri.jpg	273	28.140	0.467	0.023	2.071	4.291	2.542	6.002
3xray.png	260	23.133	3.368	0.763	2.316	5.365	6.317	10.627
4ccd.xml	267	48.438	0.738	0.025	4.034	16.273	3.001	13.148
5ccd.xml	248	25.164	1.097	0.147	2.418	5.846	3.937	9.525
6ccd.pdf	256	39.950	1.168	0.145	3.238	10.483	3.021	15.375
7word.docx	272	90.728	1.363	0.105	6.052	36.633	4.640	15.570
8excel.xlsx	249	41.866	0.794	0.024	3.830	14.672	3.175	19.989
9sound.ogg	233	88.946	0.928	0.052	6.300	39.688	1.591	18.452

Table B.1 – Table B.10 present results each experiment of real-world situation. Unit is used in each table is second. Each of columns includes file: file name, num: a number of files, max: maximum time, avg: average time, min: minimum time, std: standard division, var: variance, 95th: 95 percentiles, 99th: 99 percentiles.

Table B.1 Upload time of 100:0-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	3459	7.257	1.849	0.843	1.045	1.092	3.885	5.035
11videosmall.mp4	1839	19.819	5.687	2.625	3.333	11.109	12.609	15.965
12videobig.mp4	1845	147.917	46.650	22.114	23.929	572.619	93.745	116.734
1ecg.jpg	3662	2.639	0.146	0.047	0.143	0.020	0.298	0.777
2mri.jpg	3631	3.069	0.058	0.013	0.150	0.022	0.086	0.793
3xray.png	3660	4.421	0.951	0.398	0.591	0.349	2.101	2.818
4ccd.xml	3621	1.750	0.054	0.015	0.106	0.011	0.084	0.630
5ccd.xml	3672	4.181	0.236	0.071	0.224	0.050	0.484	1.263
6ccd.pdf	3655	3.100	0.223	0.073	0.204	0.042	0.472	1.169
7word.docx	3696	2.547	0.151	0.050	0.148	0.022	0.306	0.751
8excel.xlsx	3628	2.411	0.056	0.013	0.128	0.016	0.082	0.745
9sound.ogg	3632	2.384	0.085	0.023	0.129	0.017	0.156	0.732

Table B.2 Upload time of 75:25-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	2716	6.457	1.566	0.834	0.879	0.772	3.295	4.326
11videosmall.mp4	1342	17.524	4.688	2.612	2.511	6.305	10.076	12.190
12videobig.mp4	1374	112.936	38.691	22.086	17.937	321.744	76.993	93.881
1ecg.jpg	2636	1.643	0.125	0.046	0.122	0.015	0.265	0.639
2mri.jpg	2830	3.066	0.047	0.014	0.112	0.013	0.075	0.367
3xray.png	2724	4.286	0.817	0.401	0.505	0.255	1.790	2.390
4ccd.xml	2671	1.559	0.049	0.014	0.092	0.008	0.082	0.456
5ccd.xml	2739	2.279	0.205	0.074	0.171	0.029	0.420	0.825
6ccd.pdf	2791	2.433	0.199	0.074	0.172	0.030	0.424	0.884
7word.docx	2805	2.627	0.130	0.051	0.118	0.014	0.281	0.453
8excel.xlsx	2668	1.673	0.049	0.014	0.103	0.011	0.078	0.684
9sound.ogg	2689	1.304	0.072	0.023	0.096	0.009	0.146	0.539

Table B.3 Download time of 75:25-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	874	14.285	4.584	1.620	2.246	5.046	8.973	10.484
11videosmall.mp4	431	47.212	13.069	4.971	6.516	42.455	25.158	32.743
12videobig.mp4	469	288.480	97.871	42.738	39.159	1533.402	174.317	212.486
1ecg.jpg	920	6.790	0.363	0.089	0.404	0.163	0.745	1.801
2mri.jpg	925	5.428	0.154	0.019	0.432	0.187	0.490	2.319
3xray.png	915	8.886	2.328	0.760	1.266	1.604	4.696	6.196
4ccd.xml	911	7.142	0.153	0.021	0.467	0.218	0.391	2.355
5ccd.xml	921	8.644	0.504	0.129	0.562	0.316	1.036	2.254
6ccd.pdf	941	15.468	0.554	0.140	0.701	0.492	1.195	2.362
7word.docx	882	7.484	0.433	0.095	0.577	0.333	0.951	2.771
8excel.xlsx	878	7.907	0.156	0.023	0.425	0.181	0.494	1.686
9sound.ogg	948	9.217	0.259	0.046	0.598	0.358	0.678	3.076

Table B.4 Upload time of 50:50-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	1837	5.813	1.455	0.854	0.810	0.655	3.110	4.203
11videosmall.mp4	917	15.258	4.317	2.617	2.245	5.038	9.202	11.610
12videobig.mp4	953	115.492	36.082	22.064	15.356	235.807	68.376	85.093
1ecg.jpg	1753	2.117	0.117	0.048	0.108	0.012	0.266	0.488
2mri.jpg	1798	1.273	0.042	0.015	0.080	0.006	0.075	0.284
3xray.png	1780	2.716	0.748	0.406	0.447	0.200	1.632	2.175
4ccd.xml	1824	1.185	0.042	0.016	0.072	0.005	0.078	0.285
5ccd.xml	1801	1.634	0.182	0.081	0.118	0.014	0.392	0.536
6ccd.pdf	1795	1.292	0.177	0.077	0.134	0.018	0.388	0.641
7word.docx	1874	1.170	0.122	0.051	0.108	0.012	0.280	0.542
8excel.xlsx	1808	1.743	0.042	0.014	0.090	0.008	0.076	0.292
9sound.ogg	1852	1.132	0.061	0.023	0.066	0.004	0.132	0.267

Table B.5 Download time of 50:50-write:read situation with ***original PHR file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	1832	15.070	4.227	1.620	2.337	5.462	8.633	10.730
11videosmall.mp4	917	49.537	12.038	4.953	6.381	40.723	23.840	30.840
12videobig.mp4	928	240.318	80.638	42.462	32.456	1053.381	143.312	199.861
1ecg.jpg	1817	6.298	0.291	0.088	0.312	0.098	0.552	1.452
2mri.jpg	1790	5.960	0.102	0.016	0.315	0.099	0.294	1.269
3xray.png	1832	10.384	2.086	0.758	1.183	1.400	4.281	5.634
4ccd.xml	1767	5.771	0.110	0.020	0.333	0.111	0.203	1.624
5ccd.xml	1831	9.898	0.420	0.128	0.437	0.191	0.820	1.601
6ccd.pdf	1810	6.600	0.459	0.139	0.368	0.135	0.938	1.887
7word.docx	1812	5.942	0.321	0.095	0.343	0.118	0.627	1.513
8excel.xlsx	1820	4.615	0.099	0.021	0.257	0.066	0.292	1.120
9sound.ogg	1852	7.785	0.174	0.042	0.368	0.135	0.294	1.239

Table B.6 Upload time of 100:0-write:read situation with ***uniform file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	5067	10.321	1.913	0.843	1.175	1.381	4.282	5.490
11videosmall.mp4	5006	20.470	5.743	2.598	3.440	11.835	12.740	16.305
12videobig.mp4	4988	148.622	47.417	22.084	25.566	653.639	101.738	127.220
1ecg.jpg	1985	1.539	0.152	0.046	0.146	0.021	0.306	0.923
2mri.jpg	3356	4.164	0.057	0.013	0.130	0.017	0.103	0.693
3xray.png	5024	6.985	0.985	0.400	0.657	0.432	2.208	3.133
4ccd.xml	3219	2.966	0.059	0.015	0.139	0.019	0.086	0.801
5ccd.xml	1993	4.975	0.251	0.072	0.281	0.079	0.515	1.428
6ccd.pdf	1974	2.121	0.240	0.075	0.224	0.050	0.521	1.267
7word.docx	2001	2.681	0.155	0.050	0.153	0.023	0.308	0.870
8excel.xlsx	3346	3.702	0.058	0.014	0.134	0.018	0.092	0.653
9sound.ogg	2041	2.295	0.087	0.023	0.141	0.020	0.158	0.768

Table B.7 Upload time of 75:25-write:read situation with ***uniform file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	3781	8.207	1.602	0.853	0.876	0.767	3.302	4.172
11videosmall.mp4	3799	19.415	4.771	2.613	2.625	6.891	10.041	12.963
12videobig.mp4	3783	125.737	38.994	22.075	17.708	313.588	75.111	95.856
1ecg.jpg	1452	2.361	0.132	0.047	0.142	0.020	0.272	0.809
2mri.jpg	2539	1.781	0.047	0.014	0.086	0.007	0.080	0.361
3xray.png	3700	4.820	0.825	0.393	0.492	0.242	1.709	2.405
4ccd.xml	2514	2.676	0.051	0.014	0.116	0.013	0.083	0.478
5ccd.xml	1581	3.463	0.208	0.073	0.177	0.031	0.408	0.908
6ccd.pdf	1549	1.998	0.207	0.073	0.186	0.035	0.431	1.150
7word.docx	1504	2.091	0.133	0.051	0.115	0.013	0.282	0.519
8excel.xlsx	2528	1.717	0.050	0.014	0.094	0.009	0.086	0.491
9sound.ogg	1457	1.151	0.071	0.023	0.085	0.007	0.138	0.474

Table B.8 Download time of 75:25-write:read situation with ***uniform file type mixture***

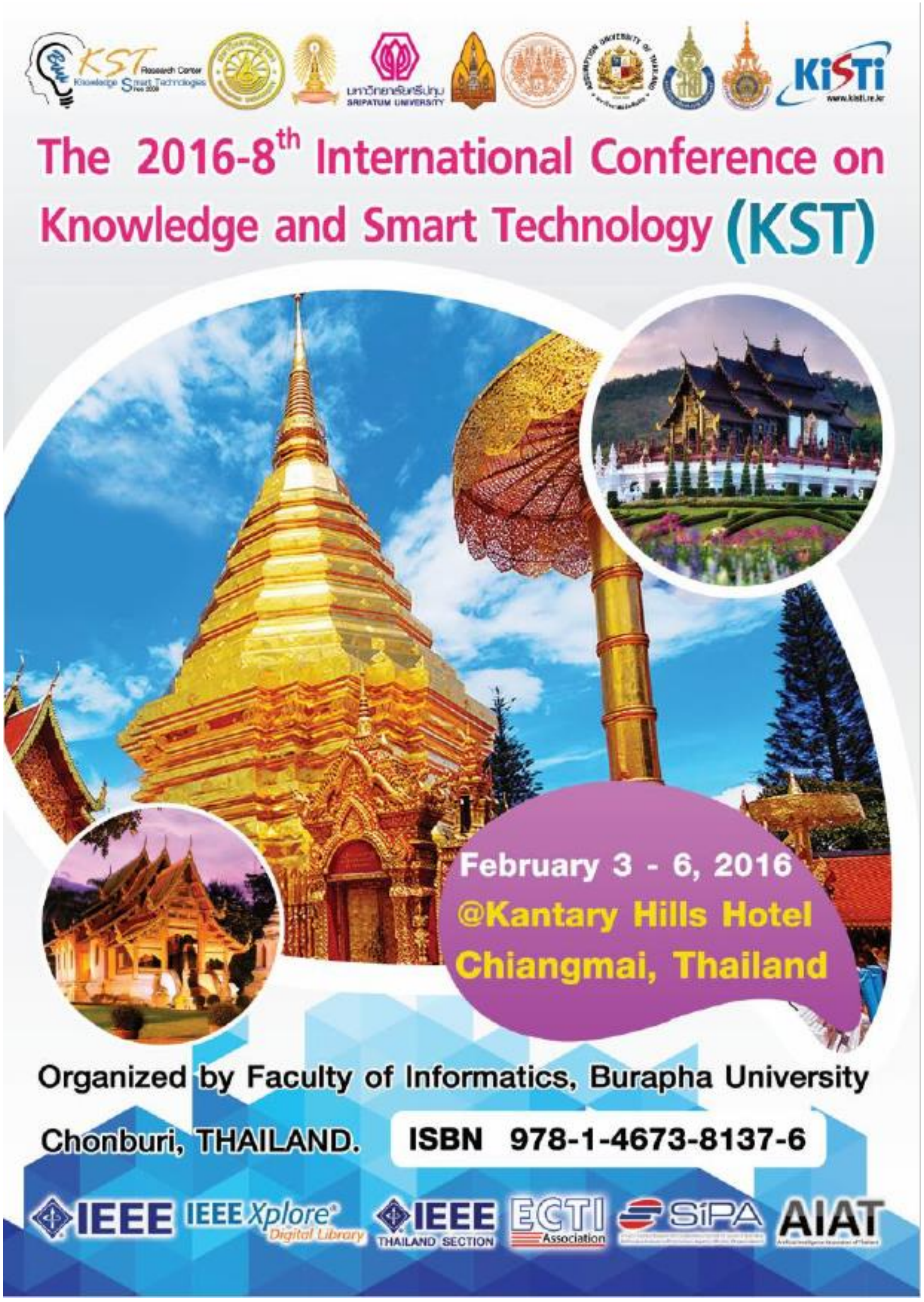
FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	1251	15.405	4.816	1.620	2.359	5.566	9.123	12.052
11videosmall.mp4	1191	55.009	13.265	4.966	6.533	42.681	25.844	31.207
12videobig.mp4	1225	258.550	97.622	42.712	37.035	1371.579	174.617	204.544
1ecg.jpg	531	3.057	0.374	0.090	0.279	0.078	0.756	1.560
2mri.jpg	779	2.861	0.133	0.019	0.244	0.060	0.400	1.339
3xray.png	1229	9.291	2.415	0.761	1.243	1.546	4.668	6.025
4ccd.xml	802	6.934	0.169	0.022	0.476	0.227	0.423	2.681
5ccd.xml	547	6.445	0.496	0.130	0.398	0.158	0.952	1.796
6ccd.pdf	487	3.396	0.543	0.140	0.371	0.138	1.069	2.047
7word.docx	510	5.576	0.395	0.097	0.381	0.145	0.779	2.137
8excel.xlsx	770	3.016	0.125	0.024	0.223	0.050	0.325	1.024
9sound.ogg	491	4.873	0.263	0.046	0.423	0.179	0.748	2.162

Table B.9 Upload time of 50:50-write:read situation with ***uniform file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	2483	5.375	1.491	0.853	0.815	0.664	3.079	4.086
11videosmall.mp4	2491	16.510	4.346	2.579	2.287	5.229	9.059	11.760
12videobig.mp4	2520	116.072	36.535	22.062	16.342	267.063	71.927	87.760
1ecg.jpg	985	1.226	0.116	0.047	0.098	0.010	0.251	0.481
2mri.jpg	1688	1.585	0.040	0.015	0.076	0.006	0.073	0.278
3xray.png	2531	4.806	0.769	0.404	0.454	0.206	1.598	2.192
4ccd.xml	1627	1.368	0.043	0.014	0.071	0.005	0.081	0.284
5ccd.xml	1013	3.546	0.199	0.076	0.184	0.034	0.399	0.854
6ccd.pdf	1007	1.581	0.185	0.075	0.153	0.023	0.410	0.716
7word.docx	1024	1.967	0.124	0.051	0.121	0.015	0.278	0.410
8excel.xlsx	1670	1.192	0.044	0.014	0.076	0.006	0.080	0.286
9sound.ogg	1006	2.189	0.066	0.024	0.094	0.009	0.139	0.295

Table B.10 Download time of 50:50-write:read situation with ***uniform file type mixture***

FILE	NUM	MAX	AVG	MIN	STD	VAR	95th	99th
10sound.mp3	2511	17.834	4.283	1.621	2.244	5.035	8.482	10.030
11videosmall.mp4	2531	35.124	11.900	4.955	5.963	35.559	23.314	29.056
12videobig.mp4	2494	247.980	80.521	42.073	31.017	962.060	140.860	179.380
1ecg.jpg	1050	1.739	0.294	0.089	0.185	0.034	0.547	1.009
2mri.jpg	1641	5.866	0.096	0.020	0.230	0.053	0.301	0.838
3xray.png	2433	7.313	2.141	0.759	1.115	1.243	4.198	5.195
4ccd.xml	1736	6.403	0.094	0.021	0.228	0.052	0.271	0.643
5ccd.xml	970	5.741	0.434	0.128	0.369	0.136	0.855	1.511
6ccd.pdf	973	7.007	0.455	0.140	0.384	0.147	0.867	1.697
7word.docx	926	4.711	0.323	0.095	0.256	0.065	0.609	0.902
8excel.xlsx	1672	5.497	0.106	0.020	0.250	0.063	0.277	1.087
9sound.ogg	1018	6.108	0.172	0.045	0.276	0.076	0.331	1.060



The 2016-8th International Conference on Knowledge and Smart Technology (KST)

February 3 - 6, 2016
@Kantary Hills Hotel
Chiangmai, Thailand

Organized by Faculty of Informatics, Burapha University
Chonburi, THAILAND.

ISBN 978-1-4673-8137-6

Distributed Storage Design for Encrypted Personal Health Record Data

Metha Wangthammang[1], Sangsuree Vasupongayya[2]
Department of Computer Engineering, Faculty of Engineering, Prince of Songkla University
Hat Yai, Songkhla 90112, Thailand
5510120085@email.psu.ac.th[1], vsangsur@coe.psu.ac.th[2]

***Abstract*—DSePHR is proposed in this work in order to manage the encrypted PHR data on a cloud storage. HBase and Hadoop are utilized in this work. The objective is to provide an API for any PHR system to upload/download the encrypted PHR data from a cloud storage. The DSePHR resolves the Namenode memory issues of HDFS when storing a lot of small files by classifying the encrypted PHR data into small and large files. The small files will be handled by HBase schema that is proposed in this work. The memory consumption and the processing time of the proposed DSePHR are evaluated using real data sets collected from various healthcare communities.**

Keywords—Hadoop; HBase; PHR; Design; Schema; Storage

I. Introduction

Today healthcare technology can improve quality of lives. Daily life activity information can be a source for predicting a disease or preventing one. Personal health record (PHR) [1] is a concept that emerges recently. The PHR owner can store any health related information into the PHR storage and the PHR system must ensure that the PHR owner has a full control over his/her data [1]. In contrast, the electronic medical record (EMR) stores the patient related information and the EMR belongs to the healthcare facilities while the PHR stores any health related information and the PHR belongs to the PHR owner [2]. This way, the user can store and retrieve his/her health information directly from the PHR storage while the user cannot access any EMR data directly. Although the patient data can be requested but the ownership of the EMR is the healthcare facility still. With the PHR concept, an individual can store any of his/her health related information of his/her entire life and the information may include some forms of EMRs. The volume of the PHR data can grow very fast because the PHR data can be any health related information and the data of each individual will be added every day. According to [1], the PHR data contains variety of health related information such as allergy data, family history, and some laboratory test results. Thus, the PHR data requires a big storage for storing the massive PHR data and the data in the storage must be correct and accessible. In other words, PHR is another application that requires a big data management [3].

Since the PHR belongs to the PHR owner, the access control on the PHR must ensure the protection of data. Moreover, the PHR may contain some sensitive data. Thus, a security mechanism to prevent any data leakage must be implemented on the PHR data. Several PHR systems provide encryption methods for securing the PHR data and the encrypted data is later stored on a cloud storage [4][5][6][7][8]. This way, the users of these systems can ensure that their PHR data is safe because the data is encrypted and only the owner of the data or the user who has the permission can decrypt it. However, these PHR systems store the encrypted PHR data directly to the cloud storage. There is an issue of the memory management because most PHR document is small. Thus, the small file size can consumes a large memory of the cloud storage system [15] which can lead to a system availability issue. Therefore, a mechanism to handle a various size of the encrypted PHR data on a cloud storage that also preserves the access control on such encrypted PHR data is needed.

Another challenge of this work is the encrypted PHR data which is not accessible by the storage system. The storage system does not have any information on the data. Any preprocessing or categorizing mechanism on the encrypted data for performances is difficult. Unlike other raw data, the storage system can perform any preprocessing or categorizing based on the content of the data. Typical cloud storages use bucket-object which is the de facto industry standard for holding objects [12]. Such technique can store the encrypted PHR data. However, such technique is not appropriate for retrieving the data by time and owner attributes because it takes a long time. The encrypted PHR data has some specific characters. That is, the PHR has an explicit owner, size and arrival time of the data. The access pattern of the PHR data typically involve these three attributes. For example, a request to retrieve the data of Mr. John which is collected last month. Therefore, this work will use these three attributes of the encrypted PHR to improve the performance of the distributed storage.

In this paper, we propose a design of a distributed storage for encrypted PHR data. The design focuses on handling various file size of the encrypted PHR data, accessing patterns of the encrypted PHR metadata and retrieving patterns of the encrypted PHR data. Both Hadoop [17] and HBase are used in our design. For comparison purposes, Hadoop Distributed File System (HDFS) which is a storage of Hadoop is used as a general big data framework to store the encrypted PHR data. The evaluation parameters include the memory consumption and the retrieving time of the proposed system.

II. Related Work

Current PHR cloud storages [10][11][12][13][14] are reviewed in this section. These storage are designed for various

978-1-4673-8139-0/16/$31.00 ©2016 IEEE

requirements of the PHR system mainly an ability to store and retrieve massive PHR data effectively.

MedCloud [10] is a health care computing system which is design to follow the HIPAA privacy and some security rules. MedCloud main goal is exchanging the health care data between health care providers. The solution is moving the health care data to a cloud computing platform as a common place for sharing the data. This way, the MedCloud user can store his/her EMR or PHR data on the system. From the technical standpoint, MedCloud uses HBase for storing and retrieving the healthcare data.

Medoop [11] is a medical platform which is developed for supporting a centralized health information exchange (HIE) in China because the size of such data can grow massively. Medoop uses Hadoop and HBase as its underlining framework. Medoop merges all files to a large one and creates an indexing file containing the information of all merged files. Medoop stores both the indexing file and the merged file on HDFS. Any frequently used data, however, will be stored separately on HBase.

The improvement of CACSS [12] is a generic cloud storage system. The organization or educational institution which needs to use a private cloud storage for processing or storing their massive data can implement such a system. The system use both Hadoop and HBase for storing the data. The system stores the data in HDFS and stores the information of the data (metadata) such as file name, file owner, and time on HBase. The system is able to store unstructured files. However, the storage does not design for fast retrieving according to the PHR owner or the PHR arrival time attributes. Querying the specific PHR data of a particular owner will take a long time because the system must scan all the metadata stored on HBase in order to filter the requested PHR data.

CHISTAR system [13] is developed as a replacement of the traditional EHR system used by many hospitals in the United States. The traditional EHR system (VistA) which is a client-server architecture has a scalable limitation and a data interoperability issue. The CHISTAR transforms the traditional EHR system to a cloud based system and enables the data interoperability. HDFS (Hadoop storage) is selected for storing the EHR data. MapReduce is selected for loading the data to HBase because HBase can solve the scalable issue efficiently. Thus, CHISTAR is suitable with the explicit EHR data type because MapReduce will classify the data for convenient retrieval before store the data on HBase.

Wiki-health system [14] is a cloud storage for PHR sensor data. Wiki-health is developed on top of a generic cloud storage system named CACSS which uses Hadoop as the underlining framework. Data such as electroencephalography (EEG) or electrocardiography (ECG) can be stored directly on the Wiki-health system. The sensor data is stored on HBase because of its fast realtime data accessing performance. Although, the Wiki-health system allows an unstructured file (an encrypted PHR data) to be attached and stored on the system. The issue raises during the data query because the database schema of the system is designed specifically for handling the sensor data.

Many cloud storage were developed on top of Hadoop and HBase because both are the main popular open source big data frameworks. Hadoop provides reliability, parallel processing capability, high write throughput, and scalable storage capability. Typically, an application that requires a huge storage and is a batch processing will use Hadoop. Real time applications will use HBase. HDFS is the storage. More details of Hadoop and HBase are given in section III.

III. Hadoop and HBase

Hadoop is a big data framework which provides high access throughput, scalable and reliable storage, and a fault tolerance feature [9]. Hadoop consists of HDFS and YARN (in Hadoop version 1 called MapReduce). HDFS is the storage while YARN is a processing tool. HBase is a non-relational database which provides a low latency data access feature and HBase runs on top of HDFS.

HDFS is a part of Hadoop storage. HDFS consists of two node types: Namenode and Datanode. HDFS Namenode stores all metadata such as a location of blocks in HDFS and a number of replication. HDFS Datanode contains the actual blocks of data. When the data arrives, it will be spited to blocks and the blocks of data will be stored on Datanode while the information related to these blocks are stored in the Namenode. The Namenode is in the memory which can reduce the metadata access time. However, this design is a limitation of HDFS because the amount of Namenode memory indicates how many files can be stored on the HDFS. Thus, small files will take up the Namenode memory space [15]. If the Namenode memory is full, HDFS may be unavailable. By default, each block of data has 3 replications located on different Datanode. The replication block is the key to provide the fault tolerance feature. If the node containing the requested data is not available, there are other nodes that can serve the requested data. If a corrupt block is found, the corrupt block will be replaced by another good block. This mechanism ensures the correctness of the data stored on HDFS. Moreover additional nodes can be assigned to HDFS without the need to shutdown the whole system. In other words, HDFS can be conveniently scaled up or down.

HBase is a column-oriented style non-relational database management system on top of HDFS. HBase schema includes rowkey, column-family, column qualifier and cell. The user must specify the rowkey in order to access the data in a row. HBase rowkey is always in a lexicography order [16]. HBase has three node types, including master (Hmaster), region server (HRegionServer), and Zookeeper (HQuorumpeer). Master is responsible for administrative operations such as creating a table, deleting a table and assigning a region server. The actual row data is stored on the region server. Zookeeper stores hbase:meta table which is the location of each region. When the data in a table grows, HBase will split the table into two regions. The information of region such as the start of region and the end of the region will be stored in the Zookeeper node. HBase is designed for supporting a large table such as millions of rows because HBase can automatic split rows to regions.

IV. Proposed Method

A distributed storage for storing encrypted PHR data (DSePHR) is proposed in this work. Firstly, the overview of the PHR system shown in Fig1 is described. The PHR data is created by the PHR user. The PHR user can be a patient or an individual who wants to store his/her health related data on the PHR system. The PHR user selects the PHR system such as MA-PHR [8] system to encrypt his/her data and the encrypted data will be uploaded to the PHR storage. Typically, the PHR systems store the data on a local storage or a generic cloud storage.

DSePHR, proposed in this work, can be used as the PHR storage. According to Fig1, the proposed method is inside the red square. DSePHR consists of two parts: the distributed storage and the API. DSePHR API is available for storing and retrieving the encrypted PHR data. Next, the detail of DSePHR is given.

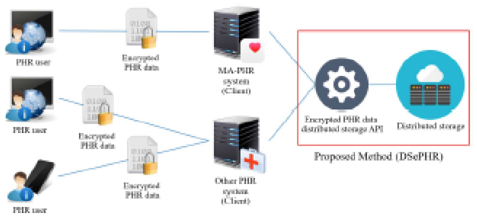


Fig1. Overview of PHR system using DSePHR

A. *Analysis of Encrypted PHR data*

The PHR data is encrypted and uploaded to the storage. The content of the data is hidden from the storage management system. Thus, any preprocessing of data for a retrieving performance is difficult. Nevertheless, the encrypted PHR data has some explicit attributes such as the owner of the data, the data arrival time and the size of the data. These three attributes are frequently used by the PHR user in order to retrieve his/her PHR data. These three attributes are utilized in the proposed design in order to store the encrypted PHR data for convenient storing and retrieving.

B. *Design of the encrypted PHR data storage*

The DSePHR API (Fig 2.) consists of the encrypted data manager (EDM), the metadata manager (MM), the authentication module, the access interface and the optimization module. The access interface is a part of the front end for communicating with the PHR system. The authentication module verifies a request from the PHR system. The encrypted data manager is responsible for managing the incoming encrypted PHR data and storing the data to the storage. The metadata manager is responsible for handling the metadata of the encrypted PHR data. The optimization module is responsible for handling the configuration and connection to both HBase and Hadoop for performance.

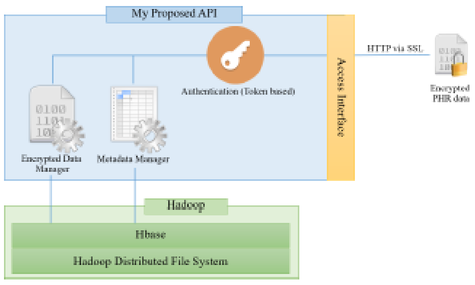


Fig 2. Overview of the proposed system

Under the DSePHR, the encrypted PHR data will be uploaded to the DSePHR through the access interface. The authentication service will first verify the permission. Then, the MM will generate a set of essential metadata for the arrival encrypted PHR data and store the metadata to HBase. The actual encrypted PHR data will be stored either on HDFS or HBase based on the EDM decision. An encrypted PHR data retrieval request will be processed by the access interface module as well. First, the request permission will be verified by the authentication service. Next, the MM will get the metadata of the requested encrypted PHR data in order to find the location that the requested data is stored. Then, the EDM will retrieve the data from its location and send the data back to the access interface which will process the response to the request issuer.

1) *Access interface*

Access interface is an interface communicating with the PHR system. The interface uses RESTful architecture style to provide the resource for the PHR system. The connection between the PHR system and DSePHR is secure by means of TLS/SSL.

2) *Authentication*

Authentication module provides all authentication related actions for all DSePHR requests. The module ensures that only the request with a valid permission can access the resource because the permission of all requests is verified by the authentication module. Token based authentication is applied in this work to provide the authentication mechanism.

3) *Encrypted Data Manager*

EDM is responsible for storing the encrypted PHR data to either HDFS or HBase. After the access interface sends the encrypted PHR data to the EDM, the EDM will classify the received data into two categories: small files and large files. To reduce the Namenode memory issues when storing a lot of small files, the small file will be stored on HBase while the large files are sent directly to HDFS. According to the discussion in [15] any file that is smaller than 4.35 MB will be considered a small file. In this work, the file smaller than 10MB (based on the maximum HBase cell capacity) will be considered a small file. The HBase schema to store the encrypted PHR data includes a column family named EncryptedData and a column qualifier named data. The rowkey consists of system id, user id, timestamp and data id. The reason behind the design of this rowkey is described in the metadata manager section. Table I shows the schema of HBase for storing a small file.

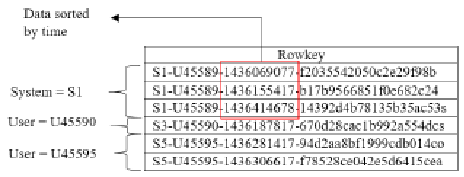
TABLE I. ENCRYPTED PHR DATA STORAGE SCHEMA

Rowkey	EncryptedData
<sysid-userid-timestamp-dataid>	Data

TABLE II. METADATA SCHEMA IN HBASE

Rowkey	Properties				
<sysid-userid-timestamp-dataid>	name	checksum	size	HDFSpath	description

TABLE III. DEMONSTRATION OF ROWKEY



Rowkey
S1-U45589-1436069077-f2035542050c2e29f98b
S1-U45589-1436155417-b17b9566851f0e682c24
S1-U45589-1436414678-14392d4b78135b35ac53s
S3-U45590-1436187817-670d28cac1b992a554dcs
S5-U45595-1436281417-94d2aa8bf1999cdb014co
S5-U45595-1436306617-f78528ce042e5d6415cea

4) *Metadata Manager*

MM is a module for handling the encrypted PHR metadata data. MM generates the metadata and retrieves the metadata for the EDM. MM uses HBase as the storage. The schema for storing the metadata contains a column family named properties and other column qualifiers such as name, size, checksum, HDFSpath, and description. Name is the encrypted PHR data filename. Size is the encrypted PHR data file size. Checksum is the hash value of the data for the data integrity verification process. In this work, SHA3 [18] is used as the checksum value. HDFSpath is presented only if the EDM decides to store the encrypted PHR data to HDFS. The HDFSpath contains no value if the data is stored on HBase. Description is a description of the encrypted PHR data. Table II shows the schema. A rowkey consists of the system id, the user id, the timestamp, and the data id. The design of the rowkey is very important in the non-relation database because a non-relational database cannot use a join operation like a relational database. The database uses rowkey as the primary key to access the data. Since the rowkey in HBase is always in a lexicographical order [12], the data from the same user will be located close to each other and the data are sorted by the timestamp. With this design, HBase can quickly map various data of the same user. Moreover, the data can be accessed conveniently according to the time attribute since the data is already stored by the timestamp. Table III shows an example of the rowkey information.

5) *Optimization*

Since the system uses HBase for storing the encrypted PHR data in case of small files. The default configuration of HBase can lead to the system unavailable because of the memory issues and the excessive garbage collection issues [21]. To prevent these problems, the HBase configuration must be changed and the java virtual machine must be configured.

To configure HBase, the memstore flush size is changed from 128MB to 32MB based on the finding in [21]. The HBase memstore flush size is a parameter that indicate when HBase will flush the data in the memory to the disk. The effect of this parameter tuning will make HBase flushing the memory more often. The HBase bloom filter parameter is enabled in order to decrease the time to access a row in HBase.

To configure the JVM, the JVM heap size is expanded from 1GB to 4GB. The JVM heap size is the size of the memory allocation from the OS for running any java application. A large heap size can support velocity data from the client. A garbage collection is an automatic tool for the JVM memory management. The memory of the JVM can be classified into two generations: young and old. A new object that is created will belong in the young generation. Later, the data will be moved to the old generation. The memory of the young generation will be cleaned however it is difficult to clean the old generation memory space. The velocity nature of the data will result in a moving of objects from young generation to old generation space. To avoid the JVM out of memory issue, the MaxTenuringThreshold is set to the maximum value (i.e., 15) in order to avoid an object in the memory to move to an old generation [22].

V. EXPERIMENT

In this section, a memory consumption of the Namenode is evaluated. The time to access the metadata of a file, the time to access the requested encrypted PHR data, and the time to retrieve the requested encrypted PHR data are measured.

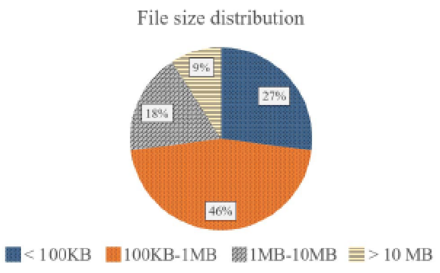


Fig 3. File size distribution

The dataset for all experiments in this work is created from health information communities including continuity of care document (CCD), electrocardiography graph (ECG), document files (pdf, docx, xlsx), x-ray image, and operating video files. The file size of these data are between 20KB to 332MB. The detail of the data set is given in Table IV. All data are firstly encrypted using the MA-PHR system [8]. Fig3 shows the distribution of the file size in this dataset. Twelve machines are used. The hardware and software specification of all machines is RAM 8 GB, HDD 320GB, Ubuntu 14.04.3 LTS, Hadoop version 2.7.1 and HBase 1.01. Table V shows the details of these machines. To simulate the real environment, the dataset is uploaded to DSePHR for 30 hours. The file are randomly uploaded to a random user during the experiment. The performance is measured after 25000, 50000, 75000 and 100000 files are successfully uploaded to the system.

TABLE IV. EXAMPLE OF DATA SET

Filesize (Original)	Filesize (Encrypted)	Increased Size	File Type	Description
393.20 KB	400.90 KB	7.70 KB	JPG	ECG picture graph Source: en.ecgpedia.org
20.30 KB	30.45 KB	10.15 KB	JPG	MRI image Source: imaging.cancer.gov
4.19 MB	4.20 MB	11.55 KB	PNG	Chest X-ray PA image (4.7MP) Source: commons.wikimedia.org
27.00 KB	38.06 KB	11.06 KB	XML	CCD example Source: www.ehrdoctors.com
617.30 KB	620.90 KB	3.60 KB	XML	Large CCD example Source: www.myhealth.va.gov
679.40 KB	690.9 0KB	11.50 KB	PDF	CCD PDF example Source: www.myhealth.va.gov
468.9 KB	480.90 KB	12.00 KB	DOC X	Patient information
30.52 KB	42.23 KB	11.71 KB	XLS X	Patient information
154.8 KB	160.90 KB	6.10 KB	OGG	Heartbeat 66bpm sound Source: commons.wikimedia.org
8.75 MB	8.75 MB	7.57 KB	MP3	Conversation sound 9 minute
27.10 MB	27.20 MB	9.23 KB	MP4	SD Video of operation 9 minute Source: youtube.com
232.01 MB	232.0 MB	11.30 KB	MP4	HD Video of operation 17 minute Source: youtube.com

TABLE V. SETUP DETAIL AND ENVIRONMENT

Name	Service Name	Specification
Master	NameNode	OS: Ubuntu 14.04.3 LTS CPU: Core-i5 3740s 2.9Ghz HDD: 320 GB RAM: 8 GB
RS1-RS9	DataNode HRegionServer (region server) HQuorumpeer (Zookeeper)* *Only RS1,RS5	
Master2	Hmaster HQuorumpeer SecondaryNameNode	
WebService	DSePHR WebService	

The DSePHR is available during the whole 30 hours of experiments and there is no dead Datanode or region server. We observe memory consumption of Namenode when the DSePHR has 25,000, 50,000, 75,000, 100,000 files in the system. Memory consumption is calculated according to the formula given in [15]. When 100,000 files are in the system, the system takes 1200GB of storage with 3 replications. Fig4 shows the memory consumption of Namenode. At the beginning, Hadoop takes an initial memory space around 56 MB which is represented by 0 file on the x-axis. At 100,000 files, DSePHR takes slightly larger memory space from the initial state while the HDFS takes approximately twice the size of the initial memory.

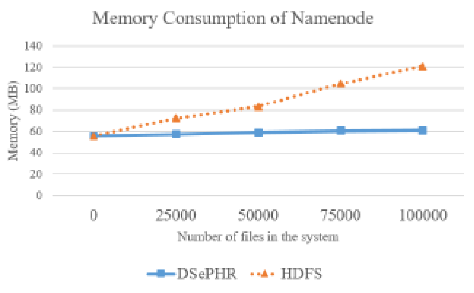


Fig 4. Memory consumption of Namenode

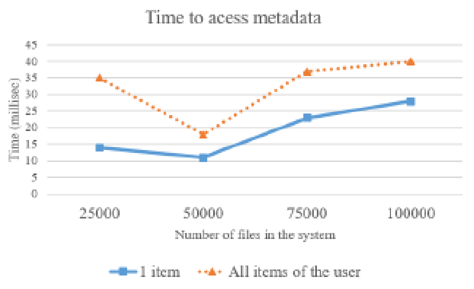


Fig 5. Time to access metadata

To measure the uploading and downloading performance of the DSePHR, a request is sent to the DSePHR in order to measure the processing time of accessing the metadata and accessing the requested encrypted PHR data. Each case is repeated 10 times the average processing time is then recorded. The result shows that the DSePHR can provide an access to the requested data immediately because the DSePHR already has the index for such PHR access patterns.

Fig5 shows the time to access the encrypted PHR metadata. There are two cases shown in Fig5. First, the average access time to the metadata when the user request only one file of a user. Second, the average access time to the metadata of a user (all files of that user). According to the results, the time to access the metadata of all files of a particular user takes approximately double the time to access the metadata of a single file of a particular user. At 50000 files in the system, the time to access the metadata drops because the metadata persists in the memory (HBase memstore). The system can read the metadata directly from the memory. Fig6 shows the average time to download various files of different sizes from the

DSePHR. The time to retrieve the file at different states of the DSePHR system is similar. The size of the file will increase the file retrieving time accordingly.

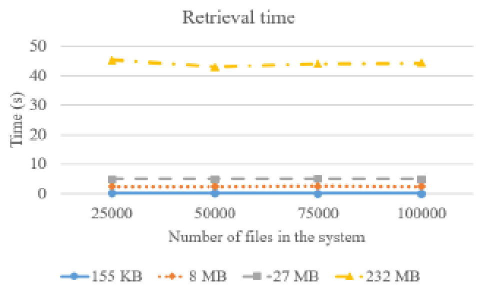


Fig 6. Retrieval time

VI. Conclusion

DSePHR is proposed in this work to handle the encrypted PHR data on a cloud storage. HBase and Hadoop are used in the proposed DSePHR. Large files will be stored on HDFS while small files are stored on HBase to solve the Namenode memory issues in HDFS. Several configuration of JVM, HBase and Hadoop are made in order to achieve the goal of the DSePHR. The encrypted PHR metadata is stored on HBase. A new HBase schema is designed and implemented in this work in order to support the PHR accessing patterns. Specifically, the PHR data is typically accessed according to the PHR owner and the PHR arrival time attributes. The experimental results show that HBase schema developed in this work can correctly classify the encrypted PHR data and sends the data to an appropriated storage. The DSePHR does not encounter the Namenode memory issues in HDFS during the whole experiment period. The average retrieving and accessing time of various files shows that DSePHR takes similar time to response to each request even when the number of files in the system is increased. Future work includes a further investigate the effect of each configuration, the effect of the HBase scheme developed on the DSePHR performance and the performance when multiple users accessing and retrieving simultaneously.

Acknowledgment

This work was supported by the Higher Education Research Promotion and National Research University Project of Thailand, Office of the Higher Education Commission (under the funding no. MED540548S at Prince of Songkla University).

References

[1] Tang, P.C., Ash, J.S., Bates, D.W., Overhage, J.M., and Sands, D.Z.: 'Personal health records: definitions, benefits, and strategies for overcoming barriers to adoption', Journal of the American Medical Informatics Association, 2006, 13, (2), pp. 121-126

[2] Häyrinen, K., Saranto, K., and Nykänen, P.: 'Definition, structure, content, use and impacts of electronic health records: a review of the research literature', International journal of medical informatics, 2008, 77, (5), pp. 291-304

[3] Margaret, M., Miron-Shatz, T., Lau, A., and Paton, C.: 'Big Data in Science and Healthcare: A Review of Recent Literature and Perspectives', 2014

[4] Li, M., Yu, S., Ren, K., and Lou, W.: 'Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings': 'Security and Privacy in Communication Networks' (Springer, 2010), pp. 89-106

[5] Wang, C., Liu, X., and Li, W.: 'Implementing a Personal Health Record Cloud Platform Using Ciphertext-Policy Attribute-Based Encryption': 'Book Implementing a Personal Health Record Cloud Platform Using Ciphertext-Policy Attribute-Based Encryption' (2012), pp. 8-14

[6] Li, M., Yu, S., Zheng, Y., Ren, K., and Lou, W.: 'Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption', Parallel and Distributed Systems, IEEE Transactions on, 2013, 24, (1), pp. 131-143

[7] Xhafa, F., Li, J., Zhao, G., Li, J., Chen, X., and Wong, D.S.: 'Designing cloud-based electronic health record system with attribute-based encryption', Multimedia Tools and Applications, 2015, 74, (10), pp. 3441-3458

[8] Thummavet, P., and Vasupongayya, S.: 'Privacy-preserving emergency access control for personal health records', MAEJO INTERNATIONAL JOURNAL OF SCIENCE AND TECHNOLOGY, 2015, 9, (1), pp. 108-120

[9] Shvachko, K., Kuang, H., Radia, S., and Chansler, R.: 'The hadoop distributed file system': 'Book The hadoop distributed file system' (IEEE, 2010), pp. 1-10

[10] Sobhy, D.; El-Sonbaty, Y.; Abou Elnasr, M., "MedCloud: Healthcare cloud computing system," Internet Technology And Secured Transactions, 2012 International Conference for , vol., no., pp.161,166, 10-12 Dec. 2012

[11] Wang Lijun; Huang Yongfeng; Chen Ji; Zhou Ke; Li Chunhua, "Medoop: A medical information platform based on Hadoop," e-Health Networking, Applications & Services (Healthcom), 2013 IEEE 15th International Conference on , vol., no., pp.1,6, 9-12 Oct. 2013

[12] Li, Y., Guo, L., and Guo, Y.: 'An Efficient and Performance-Aware Big Data Storage System': 'Cloud Computing and Services Science' (Springer, 2013), pp. 102-116

[13] Bahga, A.; Madisetti, V.K., "A Cloud-based Approach for Interoperable Electronic Health Records (EHRs)," Biomedical and Health Informatics, IEEE Journal of , vol.17, no.5, pp.894,906, Sept. 2013

[14] Yang Li; Li Guo; Chao Wu; Chun-Hsiang Lee; Yike Guo, "Building a cloud-based platform for personal health sensor data management," Biomedical and Health Informatics (BHI), 2014 IEEE-EMBS International Conference on , vol., no., pp.223,226, 1-4 June 2014

[15] Dong, B., Zheng, Q., Tian, F., Chao, K.-M., Ma, R., and Anane, R.: 'An optimized approach for storing and accessing small files on cloud storage', Journal of Network and Computer Applications, 2012, 35, (6), pp. 1847-1862

[16] L. George, HBase The Definitive Guide. Oreilly Media Incorporation, August 2011.

[17] "Apache Hadoop", https://hadoop.apache.org

[18] Bertoni, M.P.G., Daemen, J., and Van Assche, G.: 'Keccak (sha-3)', 'Book Keccak (sha-3)', 2013

[19] "Apache Thrift", http://thrift.apache.org

[20] "HappyBase", https://happybase.readthedocs.org/en/latest

[21] "Storing images in Hbase", http://apache-hbase.679495.n3.nabble.com/Storing-images-in-Hbase-td4036184.html

[22] "Garbage Collector tuning for services with large heaps", http://swish-movement.blogspot.com/2012/10/garbage-collector-tuning-for-services.html

VITAE

Name	Mr. Metha Wangthammang
Student ID	5510120085

Educational Attainment

Degree	**Name of Institution**	**Year of Graduation**
B.Eng (Computer Engineering)	Prince of Songkla University	2011

Scholarship Awards during Enrolment

The National Research University, Funding no. MED540548S

List of Publication and Proceeding

Wangthammang, Metha., & Vasupongayya, Sangsuree. (2016). Distributed storage design for encrypted personal health record data. In 2016 8th International Conference on Knowledge and Smart Technology (KST) (pp. 184–189). IEEE. http://doi.org/10.1109/KST.2016.7440505