A Search Engine for Education and Learning

By

Youwei Huang

Advisor

Dr. Cui Yu

A thesis submitted in fulfillment of the requirements of the degree of

Master of Science

In

Software Engineering

Department of Computer Science and Software Engineering

Monmouth University

West Long Branch, New Jersey

August 2020

ABSTRACT

Today the most common and convenient way to look for information is searching on the Internet. The top Internet search engines are handful, such as Google[1], Bing[2], Baidu[3], and so on. They use web crawler technologies to sniff the whole visible network and then provide users simple indexes and links as the results to help users find the source of resources. Most of those Internet search engines have good performance in terms of speed and precise keyword search ability, but wide-range search engines have their cons, such as massive potential results with uncertain quality. The goal of this project is to propose a new search engine, targeting on education, to provide "valuable" learning resources to users. Being valuable could be an subjective judgement and limited to the resources available for searching. In this project, it refers to means "user-trusted" or "user-liked", based on common practices in education and learning. A few new technologies are proposed to support efficient storage and enhanced searching for valuable learning resources, in particular, to address two main issues: (1) How to build a high-performance search engine; (2) How to define the value of resources. The details include the strategies designed to optimize general information querying, storage, ranking, and most importantly, finding the relevant learning resources that are "useful" for users. Prototyping and experimental study are conducted to conceptually prove this research.

Keywords: Search Engine, Education, Learning Resource, Resource Discovery, Resource Value

[1] https://www.google.com

[2] https://www.bing.com

[3] https://www.baidu.com

ACKNOWLEDGMENT

I would like to express my appreciation to my adviser, Dr. Cui Yu, for her warm-hearted help in guiding me to complete this thesis. Thanks for her advice in both my studying and life in the United States. In this very special period, we cooperated in developing, researching and writing papers online, across the eastern and western hemispheres, across the day and night. Best wishes to her and her family.

I am also very grateful to Professor Jay Wang. He encouraged me to adhere to my research and goals in my studies, taught me the way of life and cultural customs in the United States. During the epidemic period, he was concerned with the health and living conditions of students.

Thanks to Dr. Joe Chung. Because English is my second language, he has been helping me to use correct grammar and the right words. He helped me polish my thesis.

I would also like to thank Monmouth University for giving me the best education and academic atmosphere for studying.

Thanks to my parents, they gave me the tuition fees and living expenses in the United States. They provide me an opportunity to continue to learn and get a better education abroad.

This year, 2020, is a special and challenging year. Covid-19 is sweeping across the globe. I wish all my friends, teachers, my family and other people of the countries suffering from the virus can live through this difficult time safely.

LIST OF FIGURES

TABLE OF CONTENTS

1 INTRODUCTION

There are many search engines available for online users. Some support wide range Internet search, such as **Google**, **Bing**, and **Baidu**. Some are built within systems, such as **YouTube**, which has its own search engine. There are also search engines tailored to meet certain needs, like better privacy, copyright protection, and so on. For many years, and currently, Google has been dominating with over 80% of market share. For this reason, Google is the main search engine considered for comparison in our research.

Imagine there is someone interested in learning Java. One might Google “learn java”, and then a bunch of results are displayed; some are labeled as ‘Ad’, some are videos, and many more are other links, while you can keep clicking ‘see more’ to get more results. There are huge collection of results of different kinds, different sources, and certainly different qualities. It could be quite intimidating if learning Java is something new for this user. Making a choice alone could become the big time-consuming step before actually starting to learn. We propose a new and unique search engine that targets educational resources, to promote and support self-learning. Providing “high quality” learning resources to users is essential for this search engine. Here “High quality” is a subjective judgement from users, but it can be based on common practices in education and learning. The more users favor the resource, the higher quality it is proved to be. Therefore, the score of quality is indeed limited to the resources that can be verified or tested by users. It will be an improving process over the time. High-quality can be understood as “user-trusted” or “user-liked”.

In this project, a few new search engine technologies are proposed to support efficient storage and enhanced searching for high-quality learning resources, in particular, to address two main issues:

(1) How to build such a high-performance search engine.

(2) How to define the quality of resources.

Before starting the detailed discussion, Table 1 lists the terminologies that are used in this thesis.

Table 1. The terminologies

Term	**Definition**
LRV	Learning Resource Value. This refers to the system mode that calculates the value of each resource, as the criteria for ranking and recommendation by the search engine.
Suitability	Resource title, content and tags match search keywords.
Popularity	The trend of searches, clicks, and comments.
Feedback	User feedback, positive or negative comments, mark on a resource.
Practicality	Usage of a resource, being referred and shared.
Reliability	Resource reliability refers to whether the source of resources is reliable, e.g. whether it has been verified by authority or professionals.
Cost	The cost of finding and using a resource, both time and money cost.

Figure 1 illustrates the overview of the major components prototype in the system. The processes are divided into four parts: collecting data (collecting), resources search, data analysis (analyzing), verification of valuable resources (verification or test). Here **LRV** (learning resource value) is the method to define and improve the definition of resources value (quality), which will be explained and discussed in the later chapters.

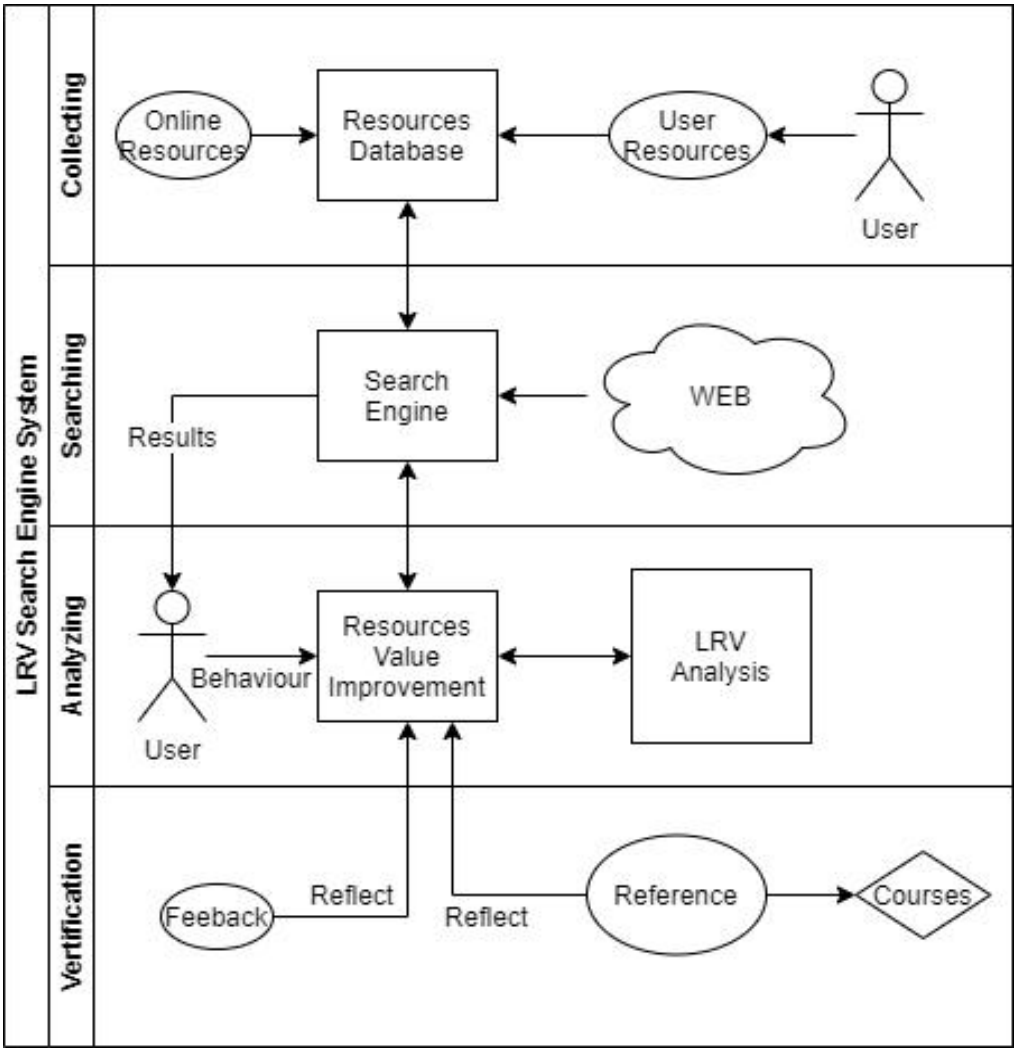

Figure 1. The general framework of the system and key technologies adopted by the search engine.

Experimental and test studies were conducted to conceptually prove this research, the details of which are presented in the later chapters.

2 RELATED WORK

Search engines have been studied for many years. All vast majority of search engine providers try to design superior algorithms to rank the quality of links and strategies to improve storage-query speed. The following is some work related to most popular search engines.

2.1 Web Crawler

Web crawler is a core component of most search engines. Web crawler provides the function of data collection, which can update the database behind the search engine to ensure that the data for user query is up-to-date.

The result of crawling is a collection of websites at a central or distributed location.[1] There are a few web crawlers behind current mainstream search engines. Two of them are listed below:

A. Googlebot

The most famous search engine, Google, uses a kind of web crawler named **Googlebot**. **Googlebot** collects web pages and builds a searchable index for Google Search engine. This name is used to refer to two different types of web crawlers: a desktop crawler (to simulate desktop users) and a mobile crawler (to simulate a mobile user)[2].

B. Bingbot

Bingbot is a web-crawling robot (type of internet bot), deployed by Microsoft in October 2010 to supply Bing.[3] **Bingbot** has the same principle and tasks with **Googlebot**. **Bingbot** collects web page information from Internet nodes and stores it in distributed system.

The work of web crawlers is very similar. They crawl the information of web pages from Internet nodes as the resource library content of search engines. The data obtained by these

crawlers is provided by the meta information of HTML pages (the title, description, keywords). There are some characteristics for these search engine bots: (1) it is impossible to crawl all the data from the Internet, (2) crawlers do not consider data correctness or quality, (3) a crawler is a kind of automated script, (4) most bots allow users to block crawlers.

2.2 Resource Discovery and Resource Discovery Server

Resource Discovery (**RD**) is a process of searching valuable information on the Internet. The study on **IETF-RD** argues that resource discovery should provide the user consistent organized view of information.[4] Resource Discovery Server (**RDS**) returns a set of resources as a search result, with the links or indexes of web pages from the Internet. Various search engines support **RD**, such as: **Google**, **Bing**, **Baidu**, and so on.

Take a Google search of a keyword phrase such as 'Learning English' as an example. The server can find over billions of resources within a second. However, regular users are only concerned with a few useful results that are hopefully most relevant. How can the search engine return results quickly? How can it rank the results and give the matching list of the most valuable resources? These two questions relate the two core tasks of traditional search engines: storage-query and results-ranking. The next sections introduce a few common solutions used in common search engines.

2.3 Distributed Storage System

Almost all commercial search engines use distributed storage system to store a large quantity of resources, Google is a good example. It has its own file system named **GFS** (Google file system). **GFS** is a scalable and classical distributed file system for large distributed data-intensive applications[5]. **GFS** has been used in Google since 2003, and it is not open source. However, the basic storage techniques used are classical and public in the technical fields. Figure 2 is the basic structure and work principle of **GFS**.

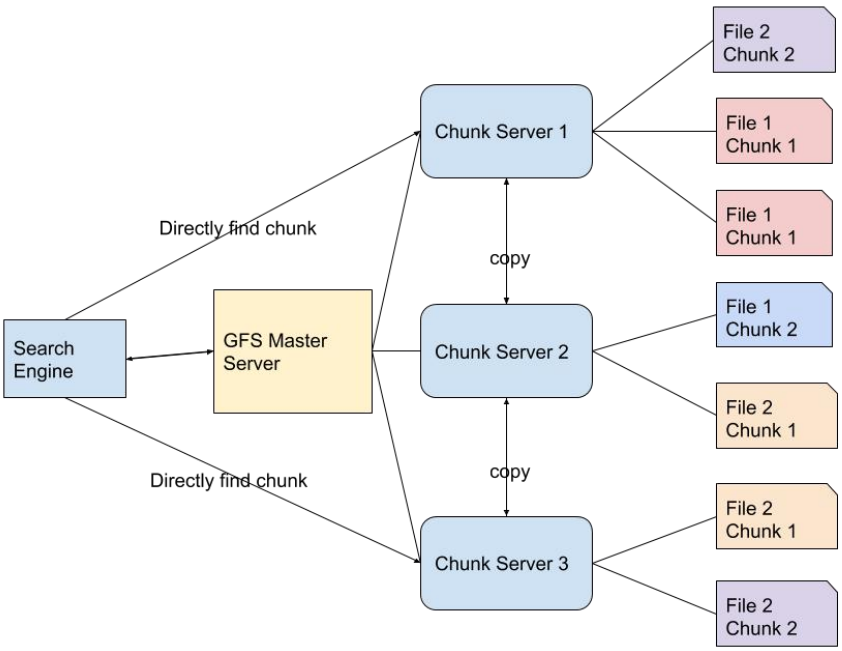

Figure 2. Google File System (GFS) storage and query processing.

When a query is submitted, the processed keywords first go to the master server. The master server only stores the file system namespaces and mapping information to the chunk locations, not database files or chunks themselves. The data files are divided into multiple chunks. The large number of resources are stored on the chunk servers. When the main server locates the corresponding addresses of the chunks, the search engine can directly access the chunks through chunk servers.

Google collects billions of resources from the Internet crawler every day. They are not stored and processed in a single server or database file. These resources, including web pages and web addresses, are divided into several small chunks and stored in the distributed file system. The chunks have many duplicates over different servers, and at the same time any chunk server can also have multiple copies to prevent loss. The master server records all the mapping relationships of chunks, so they are very easy to locate and process queries quickly.

In a distributed system, it is not very difficult to store and query millions of data. The specific search algorithms in the distributed system are skipped here. The point is that, in traditional search engine system or a resource management system, distributed storage is a good solution.

2.4 Query and Ranking

The main purpose of an efficient storage system is for efficient querying. Fast searching is one of important user experiences that all search engines strive to provide. Google, Bing, Baidu and other major search engines commonly use the following cycles to process a search request:

1. Accept a user query
2. Parse query strings
3. Figure out the keyword order
4. Look up the information in databases
5. Rank the results
6. Send back the results

In order to search for related resources in large amounts data, traditional search engine systems use cache, pre-fetching results, memory indexes and other methods to shorten or speed up the search life cycle. In Step (5), one basic ranking method for **RD** is using **Vector-Space** model[6], which has been well studied as an Information Retrieval (**IR**) topic. According to the **Vector-Space** model, a resource is viewed as a vector $[w_1, ..., w_n]$, where w_i the significance of the keyword. The value of w_i equals the number of times a keyword appears in a resource divided by the number of times the word appears in the entire collection[4]. A w_i is zero if the keyword doesn't appear in a resource. When a keyword appears in more resources, w_i will be lower; otherwise, w_i will be higher.

Modern search engines also collect user behaviors. Consequently, search results returned by search engines are highly related to users' interests and habits, unless such feature is chosen to be disabled, which by law, is an option provided to users. Today, building search engines requires compliance with privacy and security laws and concerns.

[4] Collection in our system are all the resources in our database. For the general search engine system, collection are the entire Web resources.

The storage-query model and results-ranking techniques are used in mainstream search engines. They are also very critical in the learning resources search engine proposed in this thesis. Some heavily tailored algorithms are explained in Chapter 5.

2.5 Value of Resources

In the paper "Identifying Valuable Resources", which was published in 2007 on European Management Journal, the value of resources was discussed, from the point of view of business and management. It is difficult to identify resources in a firm if there is no agreed upon definition of what "valuable" means.[7] A valuable resource can be rare, inimitable, and non-substitutable to be a source of sustainable competitive advantage.[8] The problem is known by RBV (resource-based view) advocates, but the value of resources was never clearly defined. To identify the value of resources in business and management area, there are some questions to consider: What is the source of resources? Value one resource or many resources? What is the past, present, future value of a resource? Objective or subjective valuations? What is the cost of resources? The conclusion is that valuable resources can generate three types of competitive advantage: cost advantage, the ability to premium price, and volume-based advantage.[7]

This idea of using multiple characteristics of a resource to determine the degree of value is inspiring. Similarly, the normalized resources attributes and evaluation methods can be also adopted to determine the values of learning resources, with careful design. The general evaluation method is borrowed from the business field. First, put forward the problems of learning resources. Then, give some resource attributes according to these problems. One resource is "divided" into several pieces by several attributes, and these attributes can easily be evaluated. Finally, combine all the values of the attributes into a final value of the resource. However, because learning resources are quite different for those in business and management, all characteristics need to be redesigned. For example, in business and management area, monetary cost of a resource is an important element. But the cost advantage for a learning resource is generally related to time. Moreover, in our design, learning resources are endowed with more attributes, which are discussed in Chapter 3.

3 CHALLENGES AND DIRECTIONS

3.1 Issues of Collecting Contents

An all-purpose search engine like Google requires super computing power and storage capacity. The search engine proposed in this thesis is to be used for education specifically, aiming to support users to find useful learning resources. Given the massive amount of educational videos alone, the burden on a storage system will be huge. Moreover, copyright protection definitely cannot be neglected.

3.1.1 Resource Crawler

Because of these concerns, in contrast to a traditional search engine, this learning resource search engine does not use crawlers to obtain web page contents. The system only obtains and stores the information about the learning resources. To show the difference, this solution is called a "**Resource Crawler"**, instead of web crawler. As a result, this search engine will be a significantly lighter system to achieve high performance. The copyright issues and system over-storage problems can both be avoided.

The "Resource Crawler" collects the information about resources, in other words, the meta-data of resources, such as titles, locations, tags, publishers, descriptions, comments, and etc., many of which obviously require input from users. Meta-data are organized in system database, supporting inquires. Through these information collections, the value (quality) of resources can be defined or estimated, and users can search and locate the original resource content. It helps users find useful resources for their study.

3.1.2 Data Storage and Query

Even though only resource meta-data are stored, the database needs to handle the potentially very large volume of growing data. Users need quick searches, and they depend on a high performance design of the search engine system, in particular, an efficient storage structure to support information access. Every query is filtered out from large amounts of data. The system built for this thesis uses a 3-layer storage structure and an encoded keyword mapping

method to improve the search efficiency. For application in an actual production environment, the storage needs to adopt the distributed system design.

3.2 Issues of Defining Resource Value

How to determine whether a resource is valuable? The Internet Crawler "spiders" are not something smart like human beings; the only thing they do is collect and retrieve copies of the information they crawl. Much meaningless and even fake or unhealthy information is obtained too. The search engine should not only filter inappropriate or dangerous information but should also recommend content according to the user's interest. However, that is still far from the definition of "valuable" resources.

In the field of education, there are many issues exposed by traditional search engines, directly regarding resource quality. Below are some of the practical questions related to resources:

- Is the resource reliable? (Does it provide correct information? Is the provider reliable?)
- How relevant is the resource to the user's search? (e.g., matching keywords.)
- Is the resource up-to-date? (When was the resource last updated?)
- Is information appropriate? (Is the resource safe?)
- Is the information redundant? (Are they repeated copies of the content?)
- How much time is needed to go through the resource? (e.g., length of a video.)
- What is the monetary cost? (Is it free? What is the price?)
- How many positive feedbacks? (Do other users find the resource useful?)

To answer the above questions, with inspiration from the methods to measure the value of resources in the business field as discussed in Section 2.4, we define the value of learning resources, based on 6 characteristics: reliability, practicality, suitability, popularity, feedback and cost.

These 6 characteristics are used to determine whether a learning resource is reliable and a high-quality learning resource. As shown in Figure 3, the 6 characteristics are used to evaluate the value of resources, with the light blue color representing a dynamic characteristic and a gray color representing static characteristics. Dynamic characteristics are generated dynamically during searching and ranking operations, based on input keywords, while the static characteristics are stored in the database statically. “Static” is not “constant”. It means a characteristic is only changed when an attribute of a resource is changed, such as feedback, cost, etc. Dynamic characteristics are generated in real time.

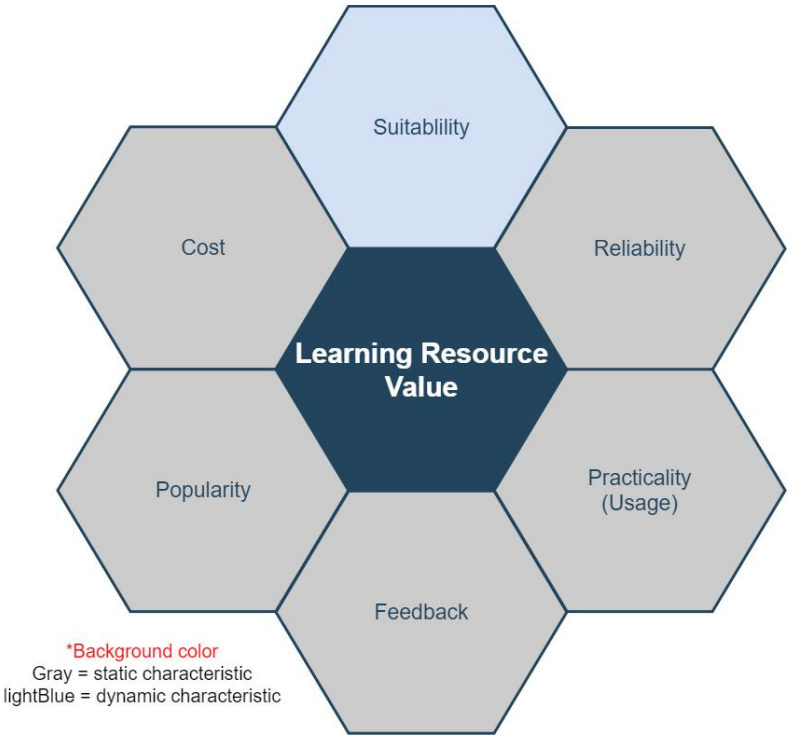

Figure 3. The characteristics for LRV

4 SYSTEM DESIGN

4.1 Requirements

4.1.1 Hardware Requirements

The complete search engine platform designed for a production environment requires several high-performance servers which can potentially process billions of requests from users. Search results are listed in the browser. We call these web applications or B/S architecture; B/S is browser and server, a kind of application which allows users to have browser access to servers. Generally speaking, the requirements for server performance and system configuration are calculated to meet the demands of user and resource volume. Therefore, in the early stage, there is no need for a detailed requirement report for the whole set of server-side hardware. Nevertheless, we lay out some basic requirements below:

- A dual-core processor, such as E3-based x86 system
- More than 16GB RAM
- 80GB disk storage
- Independent database server
- A Linux-based distributed deployment server with container and virtualization technology (not used in experimental phase)
- Data analysis server and other micro services

At the user-end or tester-end, we require PC and mobile devices to test all system web pages and functions to verify correct operation on various browsers.

4.1.2 Software Requirements

The complexity of software requirements is much higher than that of hardware. All algorithms, technical details and functional requirements are implemented and verified by

software programming. We can use the normal web development environment, tools, languages and related SDKs for implementation.

A. MVC Design Pattern

The Model-View-Controller (MVC) pattern is a classic design pattern in software engineering. It was first proposed by Trygve Reenskaug in 1978[9], and later became popular in web development area. A framework based on this design pattern can be called a MVC framework. All the development and implementation described in this thesis are based on this design pattern. In other words, the search engine system in this project adopts MVC framework. MVC separates model, view and controller. In actual development, model is data level, view is front-end, and controller is the business logic. MVC design pattern can achieve high cohesion and low coupling, and it separates data, view and business logic. MVC improves development efficiency, code cleanliness, and enables higher scalability. The purpose of using this pattern is to make the search engine easy to optimize and to expand functionality in the experimental phase.

B. Related application software

- MVC framework based on Node.js and the Express web framework.
- The view layer is based on VUE[5]; also called the front end for application users.
- Reverse proxy server and HTTP server based on Nginx.
- MySQL relational database to store large amounts of data generated for the search engine system.
- Redis non-relational database which stores data in memory; used as cache in our design.

[5] A modern front-end programming framework, which combines and compiles JS, CSS, and HTML to display the views on the browser.

4.1.3 Non-functional Requirements

To build a real-world search engine system, there are more requirements to consider, such as non-functional requirements. While we did not consider all non-functional requirements for our prototype, they are listed here for completeness.

A. Performance

Google processes 100 billion searches per month[10]. That means the average searches per day is at least 3 billion, based on 2012 statistics. Our learning resource search engine does not need such high search performance because we are targeting one special area, instead of all users and resources on the Internet.

In 2017, there were more than 30 million children using Google education apps[11],not including college students and adults. Based on this, we estimated that our system would need to accommodate 60 million users per day, i.e., two times the 2017 figure. More formally, DAU (Daily Active User) would be at least 60 million. To accommodate extra requests from increasing DAU, we raised the performance bottleneck to 100 million DAU in our development plan.

B. Reliability

The services must operate continuously with minimal interruption. To ensure that users receive search results within 1 second after starting a search, the service response speed should be less than 1 second for each query. A rapid recovery plan for crashed servers is needed.

C. Security

The layers of system security are as follows:

- Firewall between server nodes, with iptables access controls using blacklists and whitelists.
- Data backups performed to prevent data loss in case of disasters.
- Co-located servers to ensure availability, in case the main search servers crash.

D. Business layer security:

The business logic-level security protections are listed below:

- User verification
- API requests security
- User behavior logs
- Cookie or cache security
- User privacy protection
- Data encryption/decryption

4.2 Software Engineering Process

In real system design, there are engineering design, system architecture deployment design, and unified modeling language (UML). Online system and theoretical verification of the system should follow all of the design principles. In Figure 4, the design of software process follows the life cycle of software engineering and adopts agile model. Agile model is a working format in which development requirements and solutions are completed through the collaborative effort of self-organized and cross-functional teams and their customers or end users.[12] It advocates adaptive planning, evolutionary development, early delivery, and continual improvement. This approach encourages flexible responses to changes that occurred during system development, maintenance and upgrades.[13]

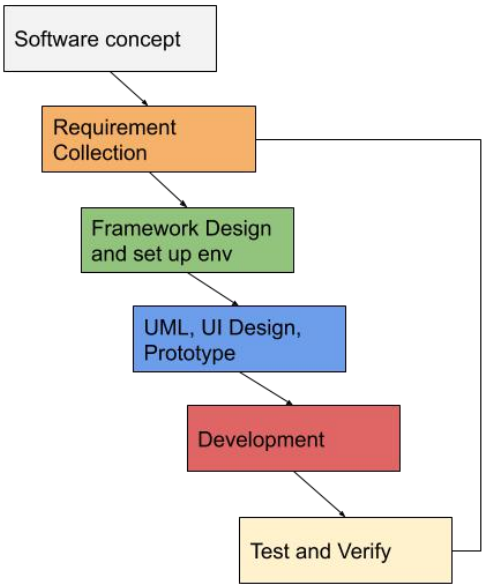

Figure 4. System development process, based on Agile

The whole implementation process is divided into 6 parts. Some of the specific requirements are explained in this section, such as Framework design and UML design. The key algorithms and methods specifically created for this project are discussed in detail in Chapter 5. Finally, testing and verification are presented in Chapter 6.

4.2.1 System Deployment Structure Design

The deployment of the whole search engine system follows the common web deployment mode as illustrated in Figure 5.

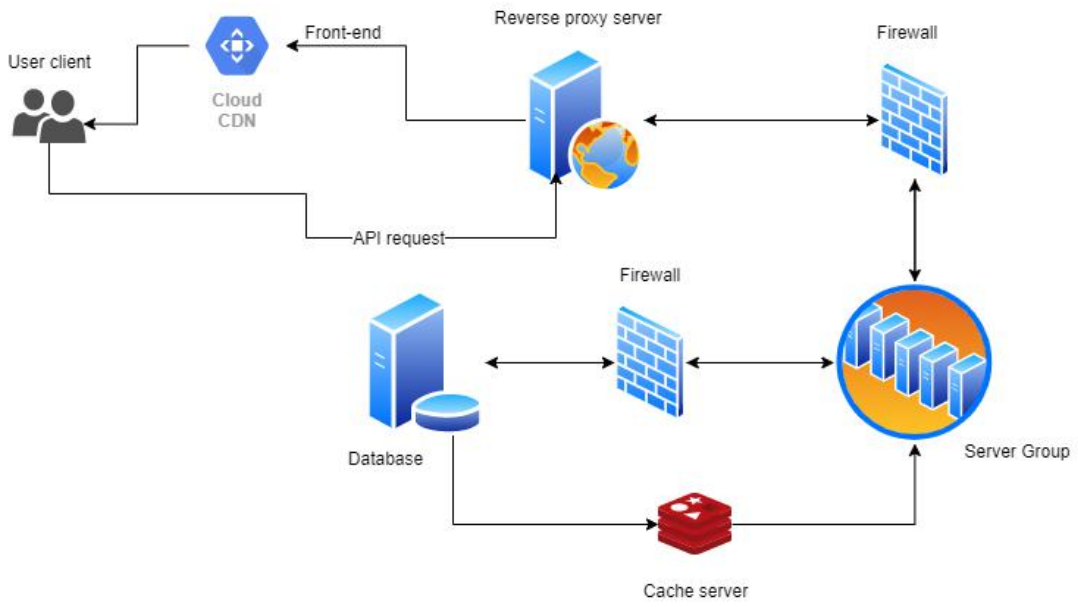

Figure 5. System deployment architecture

If server deployment is distributed in multiple servers in the same Intranet or multiple networks, each server has its own work task and provides API or open port for connections with other components or applications. This is a very popular deployment method of Web services, as it can handle large concurrent requests, reduce the coupling between services, and improve security. Multiple servers can be managed by different teams or individuals, making it easier and more efficient to cooperate with each other. In Figure 5, we show user clients (PC or mobile with browsers), CDN (content delivery network), reverse proxy server, firewalls, business logic server group, business server and database connections.

The firewalls between the servers are needed to control access to secure the data center. In our design, part of database server data is stored in a high-speed, non-relational database, such as Redis or MongoDB, to deal with some high-frequency search engine requests. High-performance storage structure is detailed in Chapter 5 to accommodate three layered searching strategy.

CDN stands for content delivery network or content distribution network[14]. The purpose of CDN is to speed up searches based on the existence of static files. Static files can be

distributed on multiple nodes of the Internet. Commonly applied in distributed systems, when users access static data, the nearest fastest server is tried first.

The reverse proxy server distributes user requests to upstream servers, which can effectively reduce the possibility of congestion. At the same time, no server downtime will affect user requests. There must be a firewall between the reverse proxy server and the cluster server to control access using either a whitelist or blacklist to prohibit illegal users from directly accessing the cluster.[6]

There are many servers in the server cluster. Most of them are controllers dealing with business logic, and there are also some specialized servers for processing big data. For example, in the system we built, **LRV** (see Chapter 5) exists here. These servers, which are responsible for data processing, run continuously, sorting and classifying the resources and tags from the database, scoring the resources based on **LRV** algorithms to provide the core business function for users to search for the valuable resources.

Search engines have high requirements for the speed of search and data acquisition, and the structure of relational database can be very complicated. For some simple tag searches, non-relational database and even cache database based in memory can provide search engines with greatly improved performance. In our design, non-relational database, such as Redis and MongoDB, are considered for performance optimization. Redis can save high-frequency search keywords in memory based on some page switching algorithms, which can effectively improve the search speed. More discussion of this topic can be found in Chapters 5 and 6.

4.2.2 Database Design

ER model (Entity-relationship model) is used to present the logic of the entities and relationships among them. Modern web application development is typically database driven, and the design of relational database follows ER model design. Figure 6 is the complete database design diagram of the search engine system (relational database only).

[6] Cluster, or server cluster. It is a kind of deployment method that combines multiple servers into a server group to provide different services. The servers in the cluster can exchange data through API.

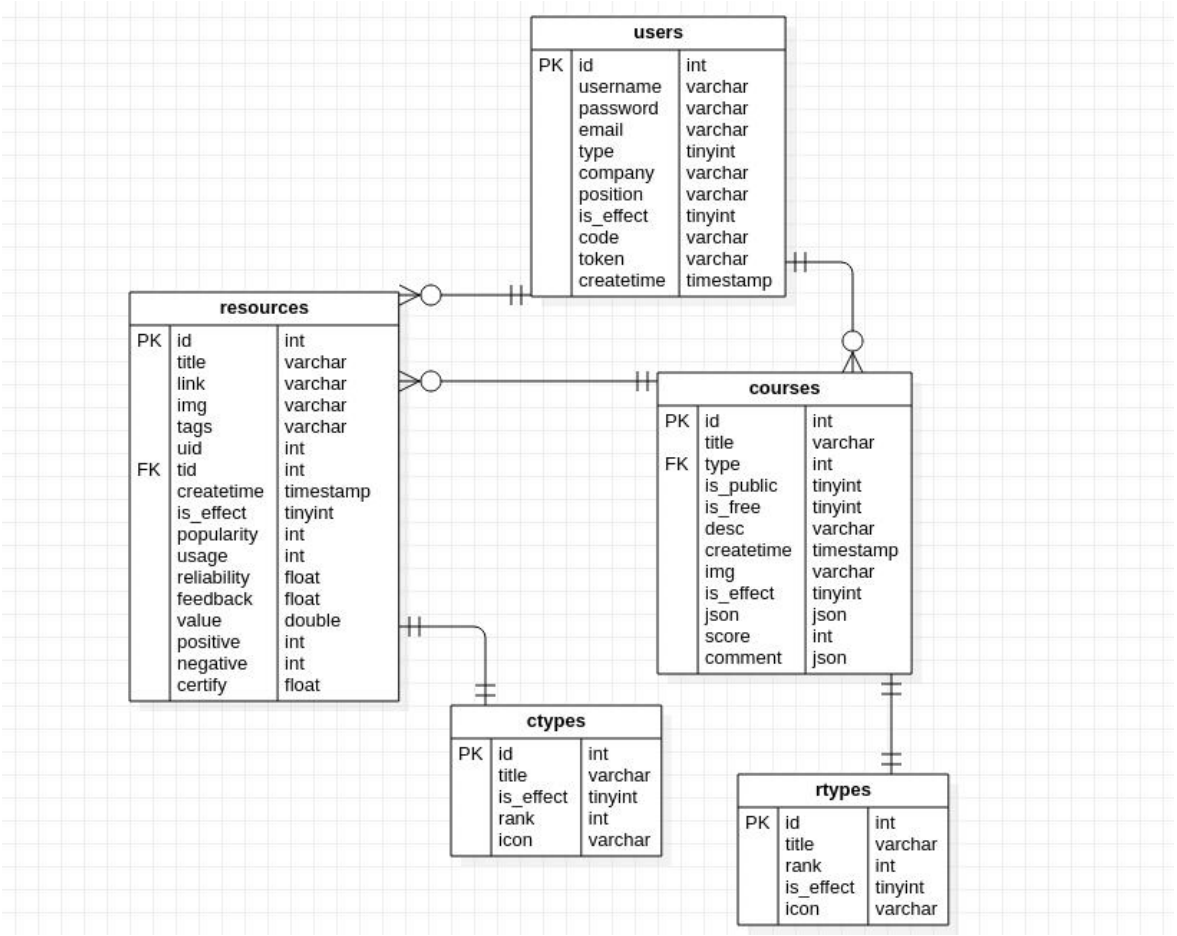

Figure 6. UML diagram of ER model of the system relational database

In Figure 6, we see that the user table is used to store user's information, the resources table stores resource information, and users are the owners of the resources. One user can recommend many resources. A user can pick multiple resources to organize a course. One course can include many resources. If a resource is used in a course by a user, "usage" will increase by 1 for that resource. The 'ctypes' and 'rtypes' entities are tables to store the types of courses and resources respectively.

That "users are the owners of the resources" is a crucial point which makes this search engine different from others. Users have the right to monitor the quality of the resources. This is the key component of reliability evaluation (one of the six characteristics) in the **LRV** system.

4.2.3 Use Cases

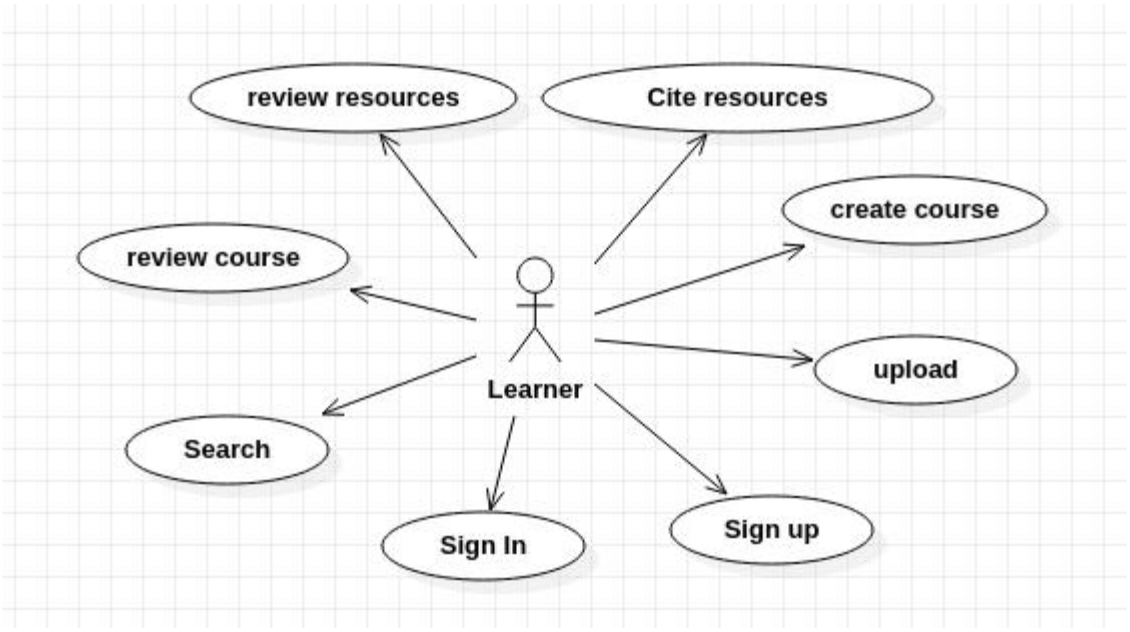

Figure 7. Use case diagram for the search engine functions

Figure 7 is the use case diagram of the search engine system which covers 7 main use cases. The use cases cover the basic functions, and the core use case is for the search function. Other small or trivial functional details are omitted here. Users can generally be considered potential learners, though they can search for other users, e.g., parents for their children. Users can search resources, upload resource information manually, set up courses by grouping resources into a list, and evaluate courses or resources. When a resource is cited by a course, the system will add 1 to usage value of this resource.

These use cases, including “review resources”, “cite resources”, “review course” and “upload”, will affect the **LRV** system in defining the values of the six characteristics. For example, they can change the value of usage, feedback, reliability, and so on. These are called “user behavior effects”.

4.2.4 Package and Class

Package is a namespace used to group elements together that are semantically relevant or might change together. It is a general purpose mechanism to organize elements into groups to provide better structure for system model.[15] For the server side of the search engine system

as designed for this project, Figure 8 shows the main packages, mapped as different folders or collections. Package have dependencies among them.

Controller. This is a package that includes all the controllers of the MVC framework. Controllers deal with all the business logic, take responsibilities for connecting data and views, and accept users' requests and responses. Controller depends on private libraries and public modules.

Server. This package is the entrance to the whole system. This package manages the files to work as a web server. The data from the user-side enters this package first. It depends on controllers because the request and data from the user-side need controllers to serve them. It depends on public modules.

Config. This package manages the connection configuration of various servers, like mail server, database server, OSS (Object Storage Service) server and all other servers needed in the search engine system. At the same time, this package contains some configuration of the system itself. Config package doesn't depend on any other packages.

Private libraries. This package contains all the private modules, plugins used only in this system. It depends on public modules and Models package.

Models. Models as a unit is the central component of MVC. It is the application's dynamic data structure, independent of the user interface.[16] It can manage the data, logic and rules in the system.

Modules. Modules are the public modules. Public modules are free software, some of which are public plugins for various programming languages. They are downloadable from the Internet, generally maintained by a community or individuals. Most of them are open-source.

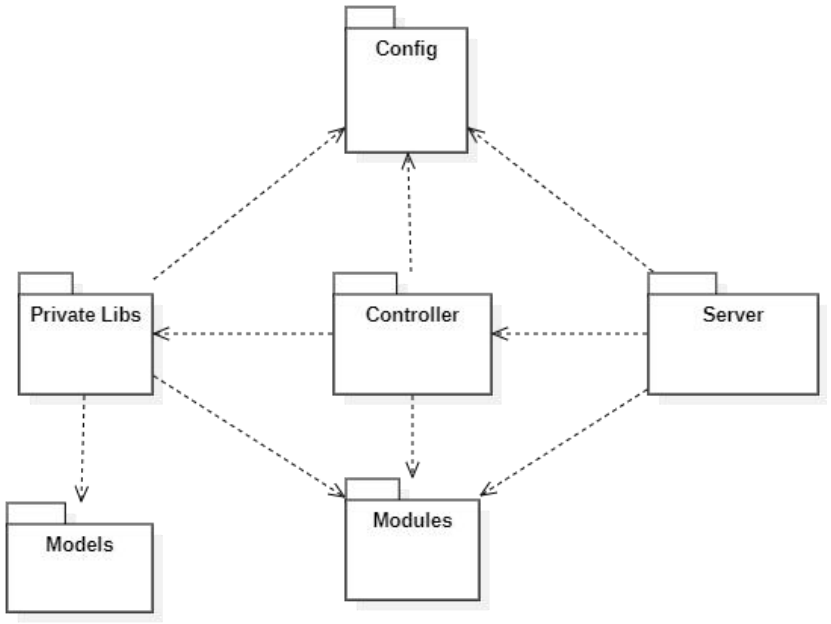

Figure 8. System package diagram

In Figure 8, **Config** and **Modules** packages have the most dependence on other resources. **Config** contains all the configuration of the whole system. **Modules** here are public plugins downloaded from the Internet for this project.

A. Controller Package

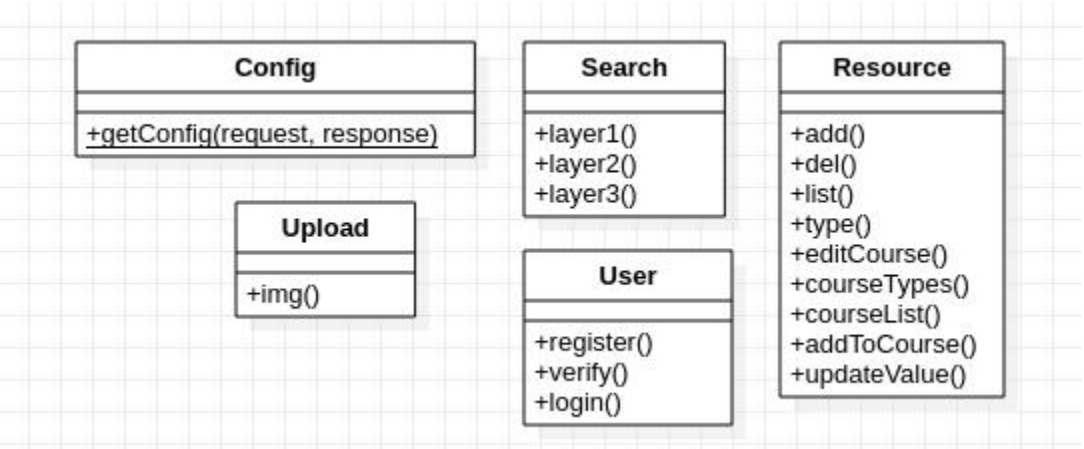

Figure 9. Class diagram of the Controller package

The controller package is the core package of the whole system. As shown in Figure 9, it contains classes which are used to deal with the user's business logic. The **Search** class solves the search requests from all users. The **Config** class returns the configuration of the site to the front end. The **Resource** class is responsible for data collection, resource classification, deletion and other functions. Through the **Resource** class, users can also edit courses and get course lists.

B. Private Package

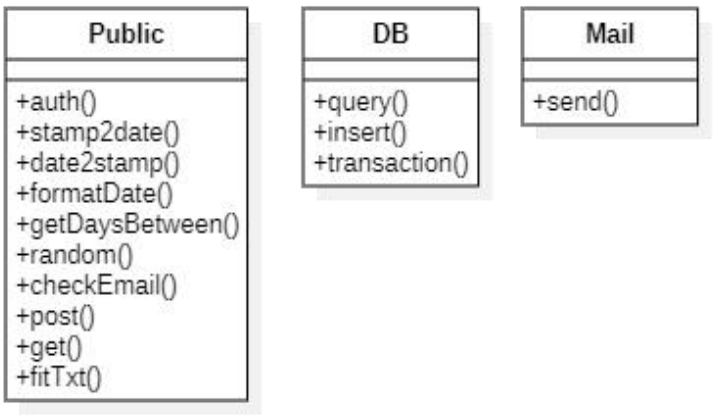

Figure 10. Class diagram of the Private package

The core class of the Private package is the class **Public**, which means public library in a private project. It is a public library specially developed for only this system (search engine). It includes functionality such as get random numbers, format date, check user info, etc., which are commonly and frequently used. It can be imported and used by controllers to reduce code redundancy and coupling. This is a very common design idea in software engineering: high cohesion, loose coupling[17]. The DB class provides a set of methods to operate a database, such as inset, query and delete.

C. Server Package

Figure 11. Class diagram of the Server package

The Server package is the web server start-up entry. It listens for and distributes the user's request to the controllers. The mapping mode used between the Server package and the controllers is called **Convention Routing**. Figure 12 depicts what the convention routing mode looks like.

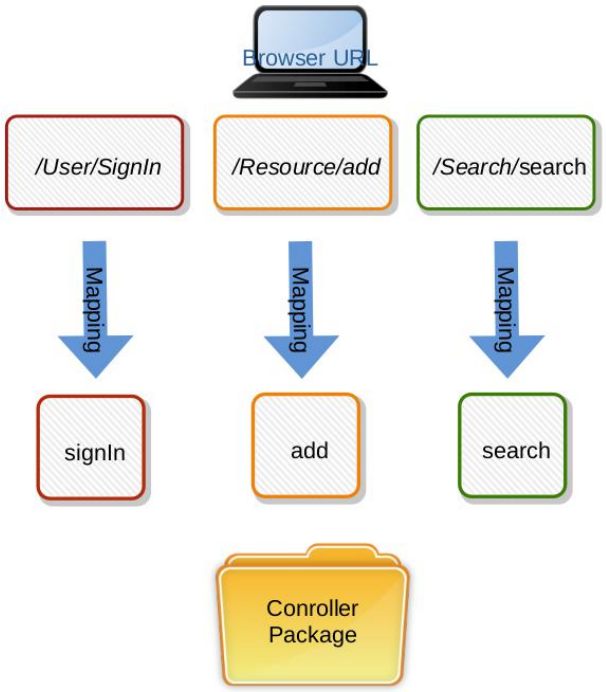

Figure 12. Router to Controllers

The routing solution is to automatically map the user's router to the same class name and action name under the controller. The characteristic and advantage of the convention routing method is that it does not need to configure the route files. This helps reduce the development

time and the writing of method documents. It can also reduce the possible misunderstanding between the front-end and back-end communication.

In this system, the static file has its own unique mapping method, different from the controller, so it can isolate the access of code and media files, for guaranteed security. Static files, like images, CSS files, fonts and other files, can be used by the website visitors or browsers, but the source code and executable files cannot.

4.2.5 Sequence Diagram

The main function of search engine system is to search valuable resources, and this search process needs to happen in a specific sequence. The following is the UML diagram of the search sequence.

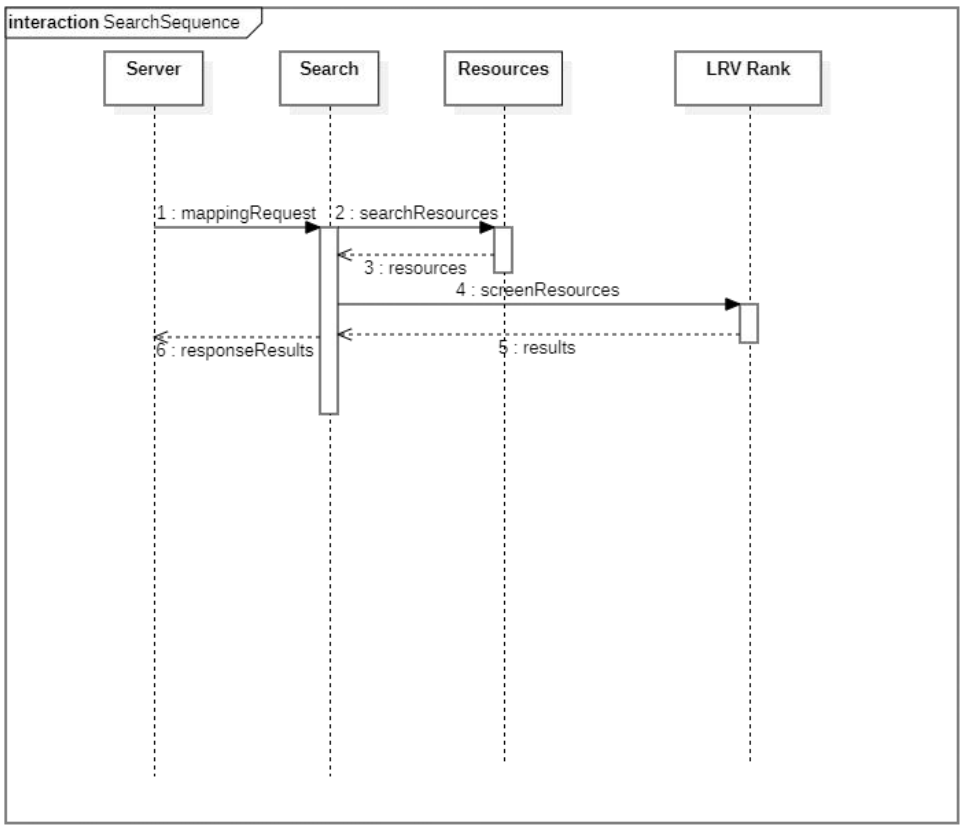

Figure 13. Sequence diagram of search process

The search process of the system proceeds as follows: After getting the user's request from the server, the keywords are sent to the **Search** controller to screen the data in resource database. When the matching and approximate data results are found, the **Search** controller calculates and verifies the value of the resources through the **LRV Rank** module service, and then returns the organized results to the controller. Finally, the controller returns a sorted list of results to the user browser through the HTTP server. Users get valuable resources that match their requests.

5 ALGORITHMS

This chapter describes key algorithms and methods proposed for this learning resource search engine, to achieve the goal of "good user experience", which means: (1) fast access to search results, and (2) valuable learning resources, which is in line with the theme of the thesis. From a development point-of-view, the goals are "storage and search efficiency" and "resource sorting and filtering".

The definitions and methods in traditional search engines enable **RD**, which stands for Resource Discovery[6]. **RD** has been explained in chapter 2.2. The algorithms that we developed for the learning resource search engine system are collectively called **LRV**, which stands for "Learning Resource Value". **LRV** has advantages over **RD** as detailed below. Like the approach of **RD**, **LRV** contains two parts: storage-query and results-rank.

5.1 Improvements over RD

The traditional **RD** approach has certain limitations when applied to the learning resource search engine.

The distributed system needed for RD requires large amounts of storage and needs hundreds of distributed servers. The need to distribute resources to multiple servers can result in the waste of many physical resources, and in terms of software design it is also costly to create and maintain such a system. In contrast, in our system we only save the links and vital information about resources, without caching any original resources. If we assume that the learning resource database is significantly lighter than those of all-purpose search engines, we can choose not to use distributed storage for it, and even if we choose to use distributed storage, we do not need to adopt a new file system such as **GFS** to distribute our database.

The learning resource search engine also needs its specially designed ranking methods to achieve good performance. In Chapter 3, we have listed a number of search engine problems

in the particular area of learning resources. The traditional **RD** approach calculates the significance of each keyword $[w_1, w_2 \ldots w_n]$ in a resource (as explained Section 2.4). However, this score is too simple for learning resources. Besides the degree of matching keywords and frequency of appearance, learning resources have more important attributes to determine the quality of the resources. In the section 3.2, Figure 3 displays the 6 vital attributes of a learning resource: Suitability, Popularity, Reliability, Practicality, Feedback and Cost. Among those attributes, the traditional **RDS** can measure only two of them: suitability and popularity.

The proposed learning resource search engine allows users to "like", "comment", "cite" and "recommend" resources. These features that incorporate human input give the proposed search engine more power to measure the value (quality) of resources. The learning resource system has its own ranking calculation systems to measure all of the 6 features. The differences between traditional **RD** and our **LRV** in ranking characteristics is listed in Table 2. More details are presented in the next sections.

Table 2. Characteristics in RD and LRV

	Suitability	**Popularity**	**Reliability**	**Practicality**	**Feedback**	**Cost**
RD	Yes	Yes	No	No	No	Unknown
LRV	Yes	Yes	Yes	Yes	Yes	Yes

5.2 Storage and Query on LRV

As explained in Section 5.1, there is no need to use distributed storage in a light learning resource system, especially a distributed file system. Instead, we use a combination of relational database and non-relational database to save data for high speed query processing. The database saves the information and attributes about the original resources, including the title, introduction, link, file type, length, and thumbnail of the resources. The following is a diagram of the storage system for the **LRV** system.

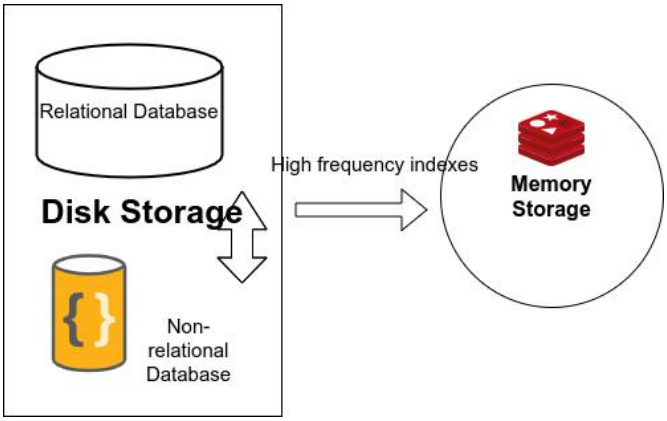

Figure 14. Database layers used in LRV

In **LRV** resources system, resources are stored in the traditional web storage mode, i.e., "database center driver", which means all the basic information and relationships of resources are stored in the databases. Databases are divided into three layers in our system as shown in Figure 14: one relational database, one non-relational database on disk, and one non-relational database in memory cache. The different layers work together as outlined below:

- The non-relational database on disk stores only pairs of keywords and its corresponding id list. The JSON data has simple structure and small size and is indexed for quick query. The complexity of search time at this layer 2 is *O(logN)*.
- The non-relational database in memory cache is an additional layer, providing faster access because of media advantage. The data is stored as a hash map in this layer.
- Once resource IDs are found, they are used to find complete resource information organized in the relational database. A primary index on resource IDs may help improve search speed; otherwise, the time complexity of "select" operations is *O(n)*.

5.2.1 Relational Database for Basic Storage

A relational database is a digital database based on the relational model of data.[18] All data are logically managed in tables. Each instance of the data is called a row, which is stored in a table. In the **LRV** system, a relational database is used as our main data management format. Figure 15 shows some relationships and entities in the database. These relationships reflect the business logic functions of this search engine application, i.e., refer resources, publish resources, comment resources, etc.

In **LRV** system, the relational database stores all the resources information and relationships among various entities. So the relational database in this system is called the "layer 1 database". It stores the most comprehensive data and it is the basic layer of storage structure in **LRV**. Next sections introduce (1) how the relational database cooperates with the non-relational database to manage data and (2) how to improve the speed of search.

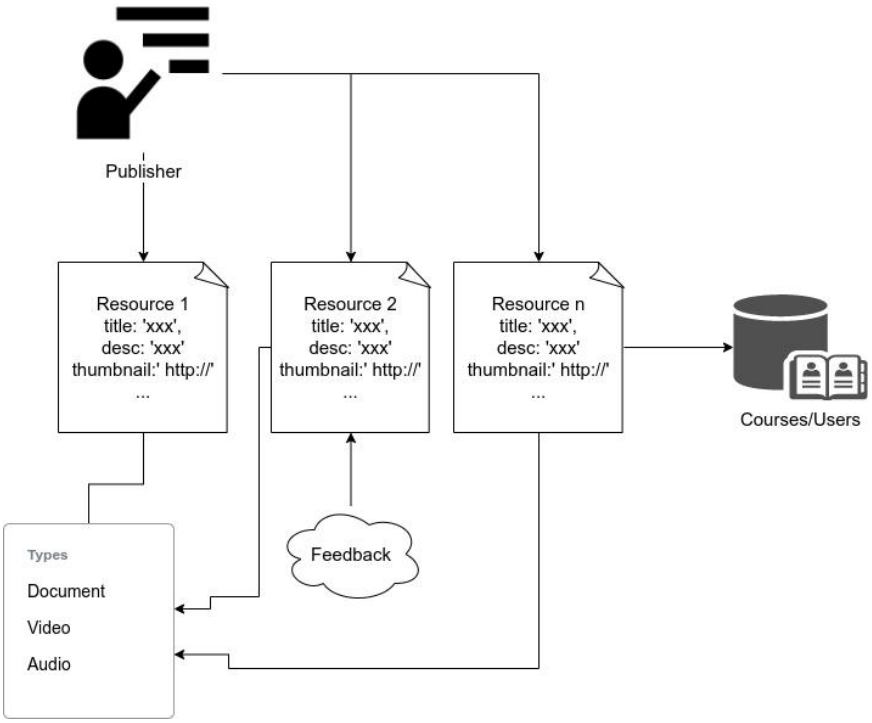

Figure 15. Relational database used in LRV with entities and relationships in it

5.2.2 Non-Relational Database for Keywords

The structure of non-relational database is very simple, without the need of recording dependent relationships among data. When retrieving a piece of data, its reading speed is relatively faster, especially in the case of large amounts of data. Figure 14 shows a set of storage solutions containing three databases. Two of them are persistent storage databases on hard disks, and the other is a cache database in memory. Among the three, the worst case reading speed would be the following: cache database (non-relational) is highest, on-disk database (non-relational) is intermediate, and on-disk database (relational) is slowest.

Overall empirical query performances of different modern database systems are compared in the following table[7]. All the experiments were performed on a 2015 **MacBook Pro** with 8GB RAM and Intel i5 CPU.[19]

Table 3. Query performance of databases with 10,000 records in milliseconds

Operation	Oracle	MySql	Mongo	Redis	GraphQL
Insert	0.091	0.038	0.005	0.010	0.008
Update	0.092	0.068	0.009	0.013	0.012
Delete	0.119	0.047	0.015	0.021	0.018
Select	0.062	0.067	0.009	0.015	0.011

Of these database systems, Mongo, Redis and GraphQL are non-relational databases. They can be chosen as the non-relational database layer (layer 2) in our search engine system. Searching keywords at layer 2 is very fast, and these keys can be further encoded to reduce storage.

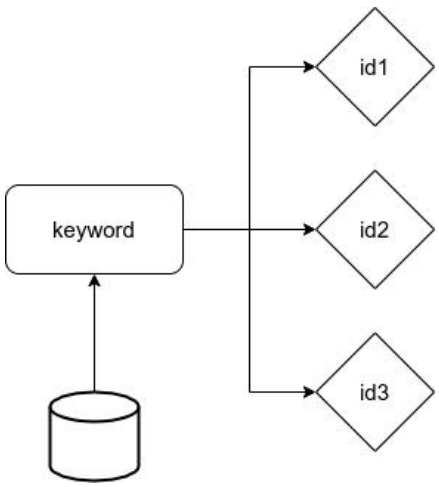

Figure 16. Non-relational database storage structure in layer 2 in LRV

[7] The database performance table is from Roman Čerešňák, Michal Kvet, Comparison of query performance in relational a non-relation databases, Transportation Research Procedia, Volume 40, 2019, Pages 170-177, ISSN 2352-1465,

Figure 16 shows the 'key-value' data structure used to manage data in the non-relational database. The 'key-value' data structure is very common in non-relational databases, storing "one key to one value". The 'key-value' data structure is stored as "JSON-like" documents with optional schema by **MongoDB**[8] in the second layer, indexed with a B-tree[9]. In this database, the data type format is JSON and the search time complexity of the B-tree is *O(logN)*.

Because one keyword can map to multiple resources, a keyword will filter out multiple resource entries (as IDs) in the database. In a non-relational database, referring to Figure 16, the "key" is a word parsed from the title, tags and description from a resource, and the "value" is a set of resource IDs stored in an array $[id_1, id_2, id_3, ...]$.

Although there may be many resource IDs (value) to a keyword (key) in the non-relational database, the resources IDs can be stored in JSON as an array, which are compatible with most, if not all, mainstream programming languages. An example of storing keywords and resource IDs in the non-relational database can be found in Table 4, where keys are encoded.

Table 4. Key-value pairs stored in non-relational database

Key (keywords, tags)	**Value (resource ids as an array)**
ba0a6ddd94c73698a3658f92ac222f8a	1, 2, 3
c31b32364ce19ca8fcd150a417ecce58	4, 5, 6, 7, 8, 9, 11, 12
4dbe9ff7f2742c912b53b9feab9f343e	6

[8] MongoDB is used as an example here; any non-relational database which can store simple "key-value" pairs like MongoDB can be used in the second layer.

[9] In computer science, a B-tree is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. B-tree storage is the default used in Mongo.

When search keywords are received, if there was only a relational database (layer 1), the search would be directly processed through a scan of rows in the resource table[10], sequentially accessing records to find resources matching a keyword, and comparing with title, tags, descriptions at the same time. With the addition of a non-relational database (layer 2) with an index built in, the search speed can be significantly improved. Here are two steps working together in the simplest scenario considered:

(1) Find resource IDs at layer 2.

(2) Find detailed resource information at layer 1.

In addition, **MD5** is used to encode the key in the "key-value" pair. **MD5** is an algorithm for inputting variable length information and outputting 128 bits of fixed length, as seen in Table 4. The purpose of storing keywords in this way is to ensure that each key stored is 128 bits long. The advantage is that the storage space for a key is fixed. In Table 4, the left column contains **MD5-**encoded keys. They can be tags, keywords, titles and any other words parsed from long search strings submitted by users. The right column contains the list of IDs corresponding to the resources.

The full search process in layers 1 and 2 as displayed in Figure 17 is as follows: first, take a search keyword; then, encode the keyword using **MD5** to get the "key"; then, use the "key" to get the IDs of relevant resources; and finally retrieve the resources' related information according to the ID list. Here, ID is the primary key in the resource table, which also serves to speed up the search[11].

[10] Though most relational databases also support non-simple indexes like B-tree or Hash, matching sentences using LIKE with such indexes is not well-supported.

[11] The primary key in relational databases (such as MySQL) will use a B-tree or Hash index to enhance the search speed.

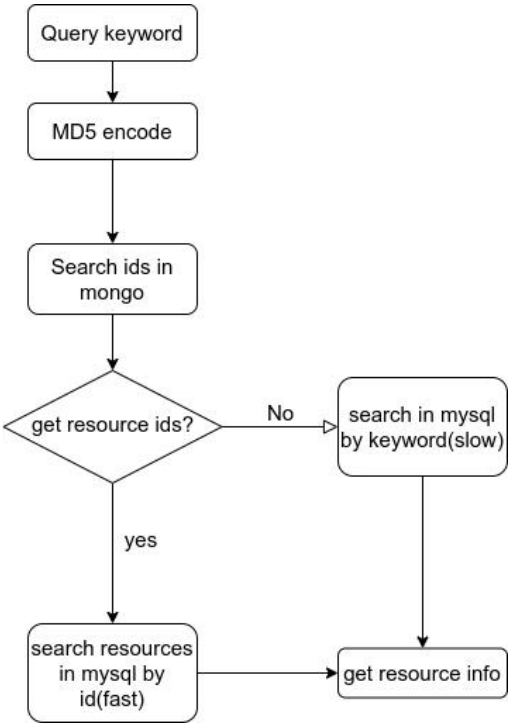

Figure 17. The search process from layer 2 to layer 1

As explained earlier, non-relational database (layer 2) only stores the correspondence between keywords and resource IDs. The advantage is from the simple structure and fast search process in layer 2. However, a non-relational database must rely on the data center of the relational database (layer 1) to achieve the resource details. The core effect of second layer is saving the time to find matching keywords. It should be noted that non-relational database typically has advantage when data can be simply represented, so it is a best practice to hold the whole database as non-relational. Further, non-relational database does not support fuzzy search very well because of possible "result loss". Therefore, in our design, the complete data are stored in a relational database in layer 1 to ensure data integrity.

5.2.3 Non-Relational Database for Cache

Cache database is often used to speed up searches. Our system adopts **Redis**, which is an in-memory database system. Redis has several advantages:

- Redis can store a large amount of data. It supports the 2^32 keys in hash map, and the maximum size of each key or value is 512M.

- Redis can be scalable and distributed.

The purpose of using a cache non-relational database is to further improve search performance gained from layer 2. At layer 2, when the amount of data reaches millions of rows of storage each day, the storage space will become insufficient, and the query speed via B-tree will be significantly slowed. The cache database can take advantage of the hashing structure supported in Redis, whose search time complexity is *O(1)*. So, layer 3 is the fastest layer.

The benefits of using a cache database like Redis include the following:

- Using distributed storage allows dividing data onto several servers, to balance the storage pressure.
- The speed of reading and writing in memory is far higher than that of hard disk, which is especially beneficial for frequently accessed data.

However, layer 3 has its own limitations. Although the memory speed is much faster than the hard disk, its storage capacity is far less. Take a personal computer with 16GB memory and 1TB hard disk as an example. The memory capacity is only 1% of the hard disk capacity. What is more, the price of memory is much higher than that of hard disk. Therefore, we store less data in the third layer than in the second layer, which indicates that the query loss rate of layer 3 is higher. Ways to work around this limitation are presented in Sections 5.2.4 and 5.2.5.

5.2.4 Three-Layered Search

The data storage system of the search engine is implemented with three layers, as shown in Figure 18. Each layer has its own features and purpose, working together to improve the speed of keyword search. To sum up, the data stored in memory, layer 3, are the keywords with high search frequency. These keywords (keys) and the IDs of the corresponding resources (values) are saved in “key-value” form, just like those stored in layer 2. But the

differences between layer 2 (B-tree) and layer 3 (Hash) causes a large speed gap between them.

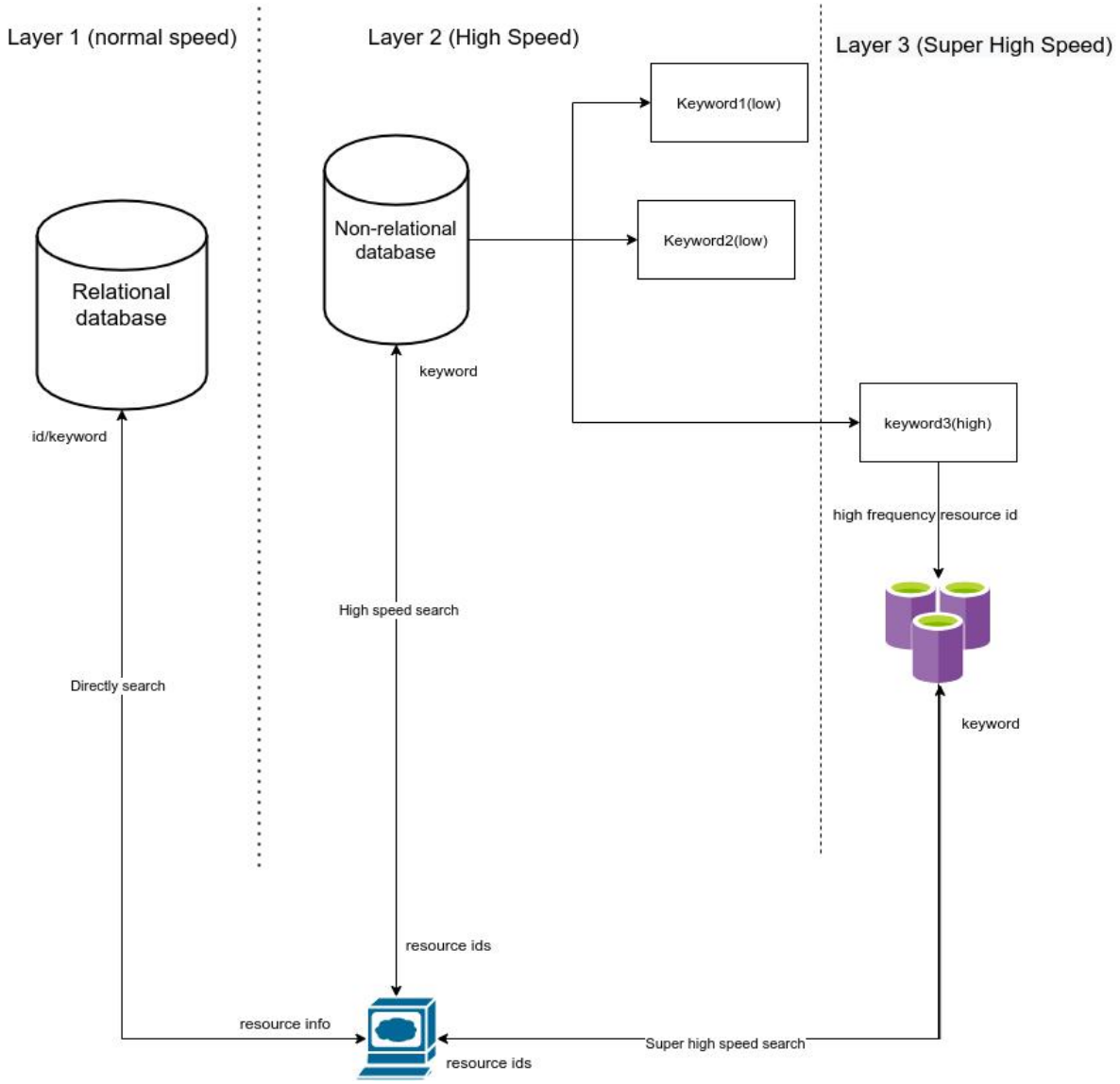

Figure 18. Three-layer storage system with different speeds

As shown in Figure 19, when a user searches for resources, the input keywords first reach the non-relational cache database to find the IDs of the relevant resources. If found, the search goes to the relational database for whole resource information retrieval. Otherwise, the search takes a detour to the non-relational disk database. It should be noted that the non-relational disk database tries to hold all possible keywords and tags, but there is no guarantee that it will

contain them. In rare cases, if the controller still cannot find the keywords in non-relational disk database, the search will have to be done in the relational database directly.

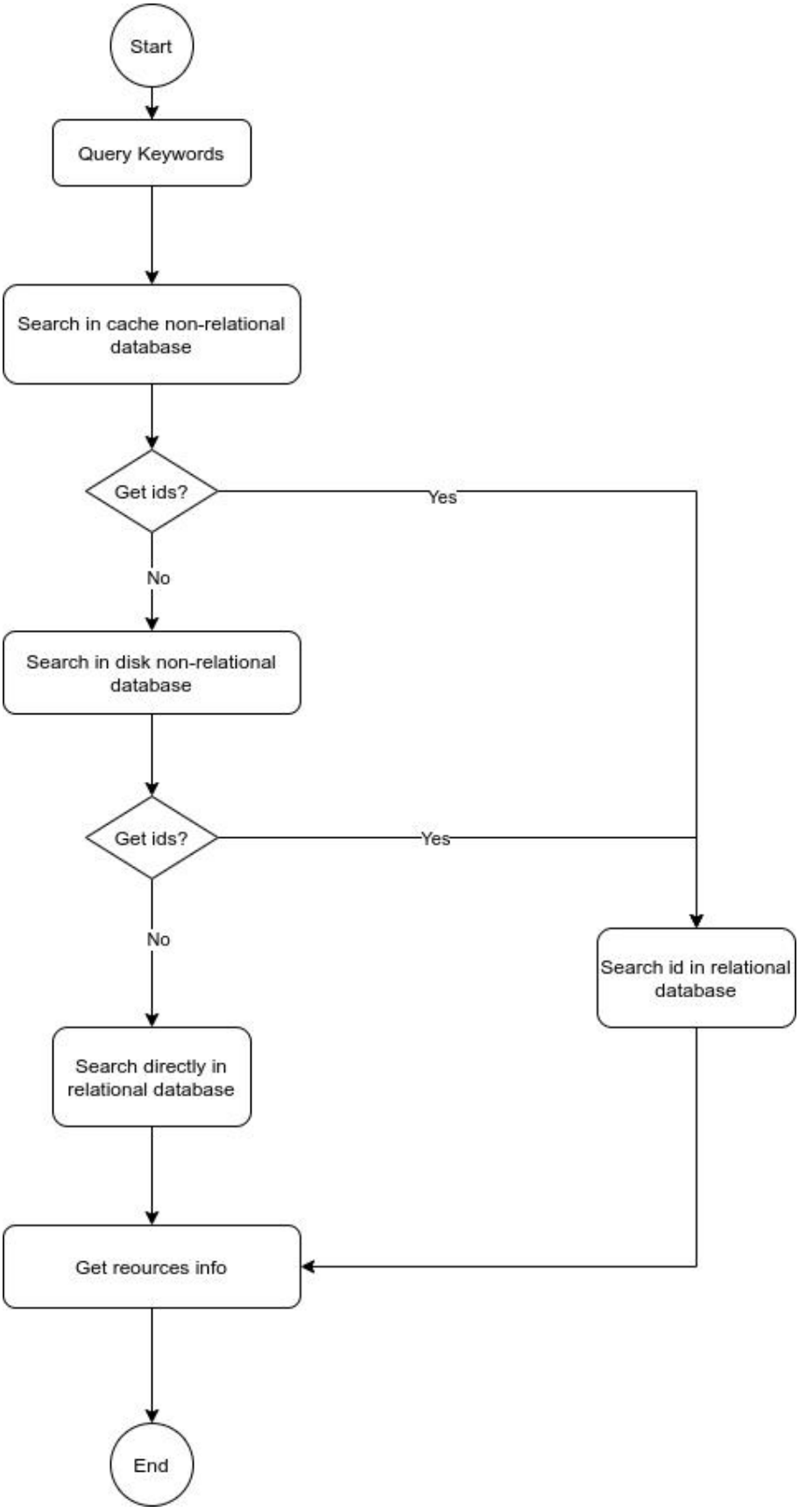

Figure 19. Three-layer search process

5.2.5 Cache Switch

Memory size is limited, even taking distributed storage into consideration. It is important to reserve sufficient memory space for smooth operating system function. Special care via algorithms and methods is needed to keep the size of the non-relational database cache within a reasonable range. As a result, it is impossible to index all the keywords in the non-relational cache database, considering the growing amount of data. Exploiting the idea of page switching in operating systems, similar strategies are proposed to manage the non-relational database cache.

Paging is a concept used in memory management of traditional operating systems. It enables the main memory of a computer to effectively manage data stored in auxiliary memory, typically hard disk. Page is the unit that the operating system sets as fixed numbers of blocks for transferring data between memory and disk[20]. In our search engine system, one "page" is a unit with the <*key, ids*> entries in Redis. In Redis, one hashing file can store up to more than 4 billion entries.[12] As an example, when searching "1" on Google, there are more than 25 billion results. If we take this number as the max search results in our system, a <*key*, i*ds*> entry can store up to 25 billion IDs, which Redis cannot normally handle. Therefore, the third layer only allows the system to store limited IDs of a keyword, and the rest of the IDs are stored in layer 2 on the disk. Actually, it is not just the limitation of Redis itself, but rather the real memory size on a server also limits the size of all IDs. To solve this problem, paging is used between layer 2 and layer 3.

In order to achieve effective page switching (key-ids unit switching), we exploit the common replacement policies like **LFU** (Least Frequently Used), **LRU** (Least Recently Used)[21], **FIFO** (First In First Out) and **Clock**. **LFU** was found to be most suitable for the proposed search engine system. Following the **LFU** rule, in non-relational cache database, the page with the <*key, ids*> with lowest search frequency is next to be moved out of cache. Table 5 shows an example of keywords, IDs and frequency in the cache database.

[12] https://redis.io/topics/data-types

Table 5. Keys, values and frequency in the cache non-relational database

Key	**ID**	**Frequency**
Key1	*1,2,3,4,5*	10000
Key2	*6,7,8,9,10*	20000
Key3	*1,3,9,11,12*	30000
Key4	*12,23,45,222,657,12321*	50000

The cache switching process using **LRU** is shown in Figure 20.

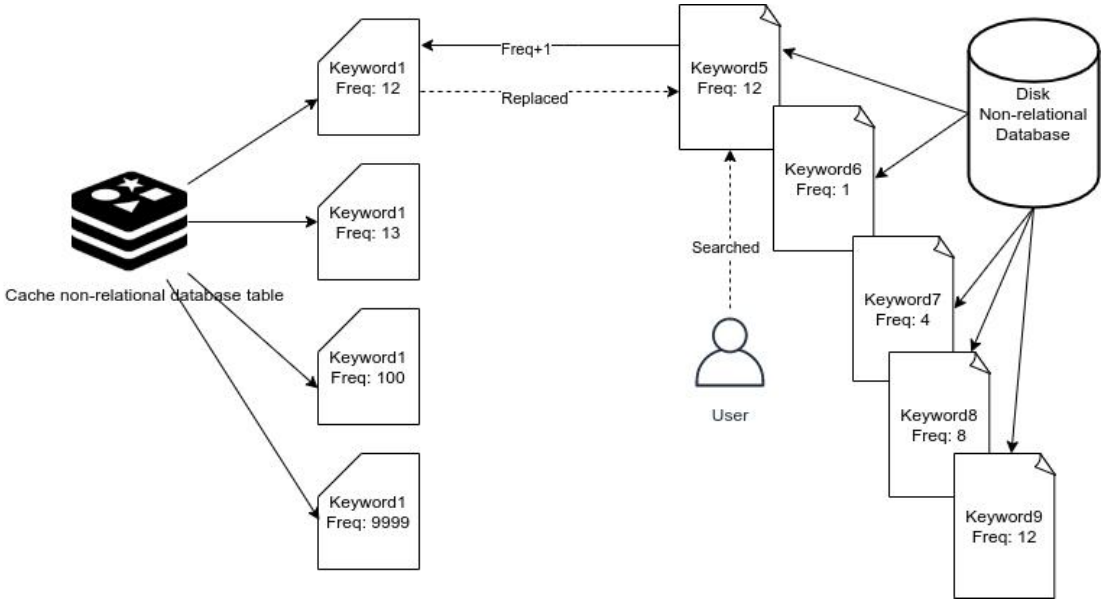

Figure 20. Cache switch process

In the scenario shown in Figure 20, there are the <*key, ids*> entries already in the cache database. Suppose there is a new search with keyword *key5*, and it is not in the cache. If the keyword *key1* has lowest frequency, then all data of *key1* (*key1* and its IDs) will be replaced by that of *key5*.

5.3 Resource Ranking based on LRV

A learning resources search engine needs to be evaluated more strictly, given that it has more features than a normal search engine. As introduced earlier, the basic DRLV idea is designed to evaluate learning resources based on their six characteristics. We call it the **LRV** model. The list of resources quantified using six characteristics are returned to users, from highest to lowest relevance.

5.3.1 Resource Characteristics

Resource characteristics can be static or dynamic. Dynamic means that the value of a characteristic is generated dynamically, and different values are generated each time under different conditions. It has no direct relationship with the resource itself. Different conditions can produce different results even on the same resource. A dynamic characteristic is never stored in any database. In contrast, a static characteristic is stored in the database.

Among the six characteristics, only “suitability” is dynamic. It is related to the user's search keywords. The other characteristics, “reliability”, “practicality”, “popularity”, “feedback” and “cost”, are all static. This means that they are determined by the status of a resource itself. Static does not mean constant. It is just that static characteristics will not be changed in real time and will not change due to user search conditions. Static values can be changed due to the change of resource in views, evaluation, publisher, and so on.

A. Suitability

Suitability of a resource is a dynamic characteristic of **LRV**. It changes with each search behavior. Suitability means the results match the user's search keywords and purpose. If it is not what the user wants, no matter how good the result is, it is still considered as an unsuitable resource. How to measure what users want? We can put all the *N* keywords into an array: *[k(1), k(2), ... k(i),...k(N)]*, and list the *M* resources as *[r(1), r(2), ... r(j),..., r(M)]*. We use keywords array to match the information of resources, because these keywords represent the general purposes of the users. For each resource *r(j),* to calculate its Suitability *S(j),* we check against all the keywords, from *k(1) to k(N)*:

$$S(j) = \sum_{1}^{i=\mathrm{N}} \mathrm{count1}(i) + \mathrm{count2}(i) + \mathrm{count3}(i)$$

$$\mathrm{count1}(i) = \#\text{ of times k}(i)\text{ appears in the } \mathbf{title} \text{ of r}(j),\ 1 < i < \mathrm{N}$$

$$\mathrm{count2}(i) = \#\text{ of times k}(i)\text{ appears in the } \mathbf{tags} \text{ of r}(j),\ 1 < i < \mathrm{N}$$

$$\mathrm{count3}(i) = \#\text{ of times k}(i)\text{ appears in the } \mathbf{description} \text{ of r}(j),\ 1 < i < \mathrm{N}$$

This adds up the frequency of each keyword in the title, tags and description of a targeted resource. Here, a frequency is the number of times a keyword appears in each item: title, tags, or description. The higher the frequency of occurrence, the higher the value of resource suitability. Finally, we accumulate the number of times keywords appear in the title, description and tags of *r(j)* to get its suitability. The value of the Suitability should be greater than zero to be meaningful.

B. Reliability

Reliability of a resource is determined by the source of resources, which we call resource publishers. Resource publishers are of two different types: **personal** accounts and organizational accounts. A personal account can be **certified** or **uncertified**. The organizational accounts must be verified. The Reliability value for a resource *R(j)* is calculated as follows (*r(j)* is a reference to a resource, while R(j) refers to Reliability of *r(j)*. A publisher is shown as user(*k*)):

$$R(j) = \begin{cases} 0, & \mathrm{cer}(k) = 0 \\ w1 \times \mathrm{user}(k), & \mathrm{cer}(k) = 1 \cap \mathrm{org}(k) = 0 \\ w2 \times \mathrm{user}(k), & \mathrm{org}(k) = 1 \end{cases}$$

The above formula is a piecewise function:

If the publisher *user(k)* is not certified (variable *cer(k) = 0*), the *R(j)* is 0.

If the publisher *user(k)* is certified and it is not an organizational account, the *R(j) = w1* × the total usage of all of this publisher's resources so far. This means that if a user publishes resources *[r(1), r(2),... r(k),..., r(Z)]*, the usage of each resource is *[usg(1), usg(2),... usg(k), ..., usg(Z)],* assuming user_k has published *Z* resources.

$$\text{user}(k) = \sum_{1}^{k=Z} \text{usg}(k)$$

In the above formula, *w1 and w2* are weight numbers. They should be adjusted through experiments, in order to ensure that the weight of Reliability towards the total Resource Value is reasonable. Our experiment shows that they should be between 0-1, referring to Experiment 5 in Section 6.3.5.

If the publisher is an organizational account, *R(j) = w2* × all the usage of the resources the user has published.

The measure of usage in this system is how many times a resource itself has been referenced in courses or other resources.

C. Practicality

Practicality is basically about usage. The more times a resource is used, the more practical (useful) it is. The value of Practicality of *r(j)* is *usg(j)*, which is the total number of times resource *r(j)* is referenced.

D. Feedback

User feedback is a direct reflection of a resource value. Users' evaluation of resources can be simply labeled as positive or negative. Positive feedback can add resource value and the negative feedback reduces resource value. We use the proportion of positive comments to reflect the value of resources. If a resource has no feedback, we set this value to 0.

$$\text{F}(j) = \begin{cases} 0, & \text{t}(j) = 0 \\ \dfrac{\text{p}(j) - \text{n}(j)}{\text{p}(j) + \text{n}(j)}, & \text{t}(j) > 0 \end{cases}$$

$$\text{where } t(j) = p(j) + n(j)$$

In this function, F(*j*) is the value of overall feedback for resource *r(j)*, p(*j*) is the positive feedback value, and n*(j)* is the negative feedback value. For example, 3 positive feedbacks on *r(j)* means p(*j*) is 3, and 2 negative feedbacks on r(*j*) means n(*j*) = 2. Also, *t(j)* is the total number of feedbacks, which is equal to p(*j*) + n(*j*). F(*j*), the overall Feedback value, can actually be viewed as the overall rating for a resource. F(*j*) can be a negative value if the number of negative feedbacks are more than the number of positive feedbacks.

E. Popularity

Popularity represents the trend of increasing views for a resource. The Popularity of resource *r(j)* is P*(j)*.

$$P(j) = V_d - V_{d-1}$$

V is the number of cumulative views of a resource. V_d is the number of cumulative views for 'today', and V_{d-1} is the number of cumulative views for 'yesterday'. The number of views added today is the value of popularity.

F. Cost

Cost includes time and money. In the formula below, C*(j)* is the value of Cost, t*(j)* is the value of time cost, and m*(j)* is the value of monetary cost. *K* is a constant used to control the weight of Cost value, since Cost is not as significant as the other characteristics.

$$C(j) = (t(j) + m(j)) \times K$$

5.3.2 Resource Evaluation

Resource value is calculated based on the six characteristics introduced in 5.3.1. First, we find the 'static' value and the 'dynamic' value.

(1) Static Value

The characteristics reliability, practicality, feedback, popularity and cost are static characteristics and are stored with resources. The value of all the static characteristics is calculated using the formula:

$$\mathrm{Static}(j) = \frac{\mathrm{R}(j) + \mathrm{Usg}(j) + \mathrm{P}(j) \times \mathrm{F}(j)}{\mathrm{C}(j)}$$

Static value of a learning resource is the sum of Reliability, Usage (same as Practicality), Popularity times Feedback, divided by Cost.

Note, in the static function, we multiply Popularity and Feedback values. That means, if P*(j)* keeps increasing but there is no Feedback F*(j)=0*, P*(j)* will have no effect on the static value.

(2) Dynamic Value

Suitability is the only dynamic characteristic among the six characteristics in the **LRV** model. Suitability is changed dynamically when users search the resources. Different query keywords can cause different Suitability value, as explained in Section 5.3.1.

(3) General Formula

The Value of a resource is:

$$\mathrm{V}(j) = \mathrm{S}(j) \times \frac{\mathrm{R}(j) + \mathrm{Usg}(j) + \mathrm{P}(j) \times \mathrm{F}(j)}{\mathrm{C}(j)}$$

V(j) is the final value of a resource *r(j)*. It is equal to Suitability multiplied by the total Static value.

6 EXPERIMENTAL STUDY

Software testing and quality assurance is an important part of software development. Since the project is not formally released, and there are no real user data that can be collected, the experiential study conducted is based on simulated data. This chapter presents the experimental study done with generated data to test storage-query performance and **LRV** ranking algorithm.

The main purpose of testing is to meet all the requirements and assess the quality of the software. To test the core components of this learning resource search engine, the testing plan was the following:

> Test the performance of the 3-layer storage structure. Verify the query-storage methods.
>
> Test the accuracy of **LRV** ranking results. Verify the **LRV** ranking algorithm.

The expected results of testing are (1) good search response time, and (2) more valuable resources ranked higher.

6.1 Methods

Software testing is based on requirements and specifications of design. There are some common and mature testing methods in software engineering, and we briefly introduce and apply some methods to our learning resource search engine.

6.1.1 Black-box Testing

Black-box treats the software as a "black box", examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it.[22]

Black-box testing is used for this search engine system. It is used for testing the ranking component of the search engine. Black-box testing is particularly suitable for the user

searching process, from sending keywords to the listing of all the most valuable resources. The search function is treated as a black box, and testers do not need to understand the principle of the searching algorithms and ranking methods (**LRV**). The testers only need to verify the resources returned, checking their ranking. Resources with high value are ranked at the top of the list. Value itself is a more subjective concept. Later in this chapter, we will discuss how to measure the user's recognition of resource value in the experiments.

6.1.2 White-box Testing

White-box testing (also known as clear box, glass box, transparent box, and structural testing) verifies the internal structures or logic of a program, as opposed to only the functionality exposed to the end-user. In white-box testing, an internal perspective of the system (the source code), as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.[23] [24]

We use white-box testing to test the storage-query component of the search engine, especially the three-layer storage structure. However, here it is not a strict white box. Normally, white-box testing asks users to choose different test paths according to the logic of the code, but we do not change the input paths to get the expected results for the code logic of a function. Instead, we query the same amount of data in **different** storage structures, with **different** experiment variables through **different** keywords *[k1, k2, k3 ...]*, pages *[p1, p2, p3 ...]* and other conditions. The idea is similar to the white-box, changing the structure (layer 1, layer 2 or layer 3) within the code and making it transparent to the testers. In contrast to the black-box test mentioned in the previous section, the testers need to understand the logic of the different layers and test the search process logic for the different storage structures. The purpose here is to test the query performance and storage performance of the search engine system through different experimental control groups and prove the reliability of the three-layer storage structure.

6.2 Testing Environments

The hardware and software used for the testing environment is very important because the test results (and related analysis) may vary depending on the testing environment. In this section, we stipulate a standard testing environment to try to prevent errors and biased results.

6.2.1 Hardware and Software Condition Control

In the experiments of testing the storage and search process, some environments need to be fixed:

- Same local network
- A server with the same configuration
- A PC with the same configuration and the same browser

The network environment for testing must be controlled to prevent network problems from affecting the query speed. The server configuration must be the same because different server configurations will likely cause different testing results. All testing data must be based on the same set of server configurations, including hardware and software. The client used in the test should also be the same, but can be less strictly controlled than the previous two requirements. Detailed configurations and settings for testing are listed below:

Network:

- A home network router, server and client PC are connected to the same router through wired LAN ports, operating over 100Mbps fiber.

Server Software:

- Linux operating system: Ubuntu distribution, version 20.04
- Proxy HTTP server: the NGINX-based Tengine 2.2.3
- Relational database: MySQL 5.7
- Non-relational database: MongoDB
- Non-relational database in memory: Redis
- Node.js, latest version

Server Hardware:

- RAM: 16GB DDR4
- CPU: Intel i7 7700
- Storage: NVME SSD, 512GB Samsung pm961
- ASUS motherboard

- Intel 1000Mbps network adaptor

Client Software:

- Google Chrome Web browser
- Automated testing script, written in JavaScript

6.2.2 Experimental Group Control

Different variables and conditions are tested in each experimental group. The purpose is to compare the effect of different settings in the experiments and to evaluate the performance, cost, user experience and accuracy. The experimental data are simulated. The resource contents are artificially generated from a “100 different words” dictionary. The “100 words” are randomly combined into the title, content and various initial information of a new resource through the testing script.

Figure 21 is the “simulator web page”, used to generate simulated resources for the experiments. This is an automated test script that can insert or update resource data in the databases and simulate user behaviors. The “dictionary text area” is for the tester to enter keywords. All contents of each random resource are generated from this tester-entered dictionary, so the terms in titles, descriptions and tags are from this dictionary.

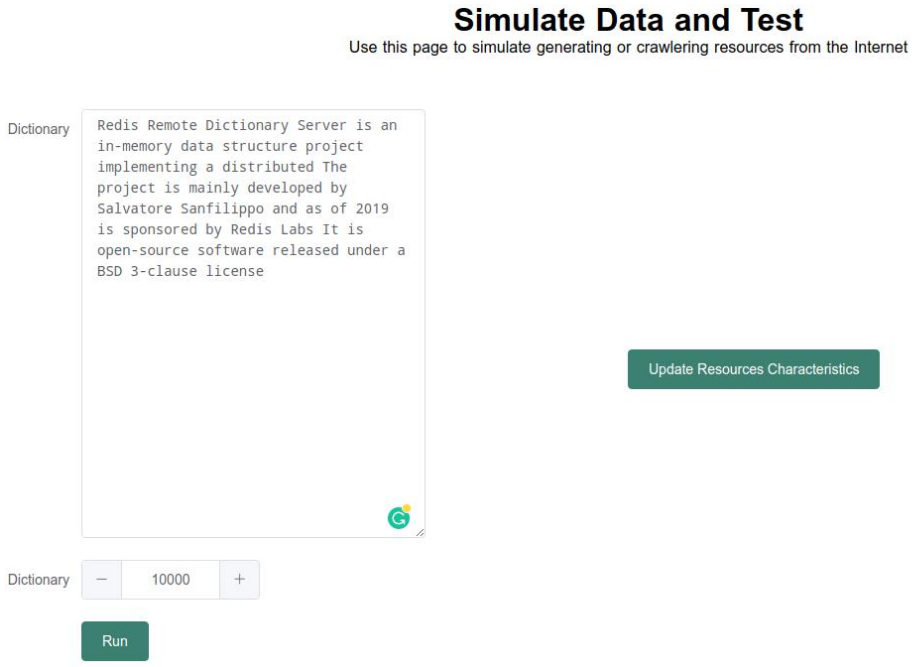

Figure 21. Resources query simulator

A. Experiment 1: Measure the search time under different magnitudes of data volume.

1) Test the average search time of 4 keywords, “a”, “is”, “2019”, “open-source”.
2) The data size of resources in databases range from tens, hundreds, thousands, to millions of resources.
3) No query operations (insert, update...) during searching.
4) We choose the first page of all the search results for a keyword.

B. Experiment 2: Measure the search time when locating resources at different depths (pages within returned list results).

1) Set 1 million resources in database.
2) Calculate the average time to locate resources returned in different pages: page 1, page 100 … page 50,000, last page.
3) Two keywords are used as experimental samples to calculate their average search time.

C. Experiment 3: Test on busy system with heavily dynamic data

1) 1 million resources are inserted into or updated on databases (layers 1, 2 and 3) and 100-200 asynchronous requests per second are issued.
2) 1 million resources inserted in databases before start of testing.
3) Search for results located on page 100.
4) Two keywords are used as experimental samples to calculate their average search time.

D. Experiment 4: Compare the search times for finding ranked and unranked results.

1) 1 million resources in databases.

2) Compare two conditions: 1) ranked search results, 2) unranked search results.

3) Search for results in page 10,000.

4) Two keywords are used as experimental samples to calculate their average search time.

E. Experiment 5: Verify the rank list based on **LRV** algorithm.

1) 10,000 resources in databases.

2) All the resources are randomly updated with several attributes.

3) Simulate "user behavior" data for the 5 static characteristics:

   I. Generate positive or negative feedbacks (fewer than 10,000 feedbacks).

   II. Generate two numbers: number of visits of a resource today, number of visits of a resource yesterday.

   III. Generate the number of references a resource has.

   IV. Randomly set user's accounts to uncertified personal, certified personal, or organizational.

   V. The above values are simulated using the simulator web page shown in Figure 21.

4) Search using 4 sample keywords.

6.3 Performance Analysis

6.3.1 Search time for Varying Resource Volume

Figure 22 is a line graph of the results for Experiment 1. It shows the average search time for different numbers of layers of storage. With increasing numbers of resources, we see the search time of all cases clearly increasing after about 10,000 resources. As expected, when using only Layer 1, relational database storage, the speed is affected most. When all 3 layers are used, the search speed is not affected until about 100,000 resources. When the number of

resources exceeds 100,000, using 2 layers starts to become multiple times slower than using 3 layers, but about half as slow as using just 1 layer. In fact, in our tests, when there were millions of resources to search, using only 1 layer resulted in search times greater than 2 seconds. Therefore, our multiple layer storage model is helpful to achieve good query speed of the proposed search engine.

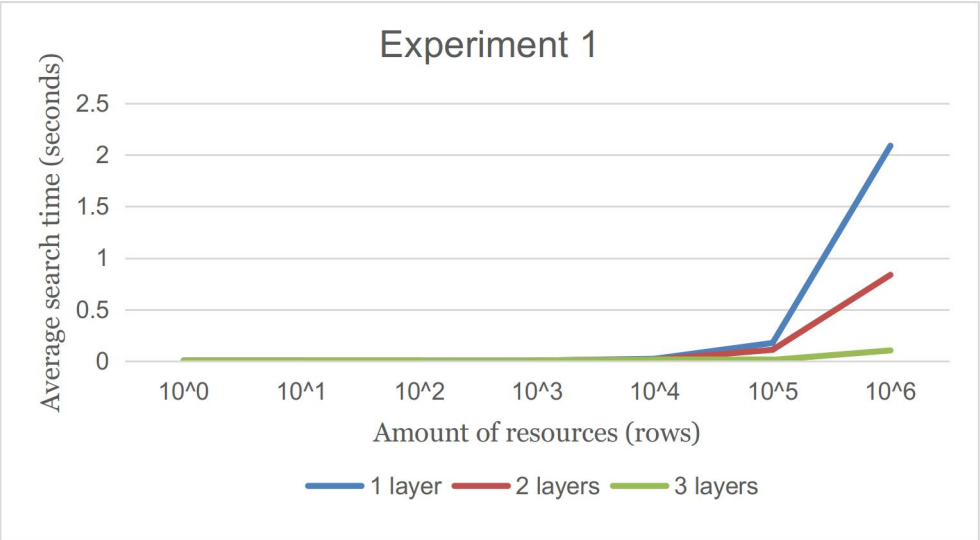

Figure 22. Average search times for 3 layers of storage with increasing resources

6.3.2 Search Time for Different Search Result Page Depths

Figure 23 shows the average search time when querying results located on different pages. The screenshot shown in Figure 26 shows page numbers of the resources found, but not yet accessed. Page number is used to simulate which page a user goes to after making a search. The test is conducted using one million stored resources. When the user goes on to a later page, the search time spent on the first storage layer increases gradually. The search time starts from 2 seconds, which means accessing resources on Page 1 is already significantly slow. The search time when using 2 layers is low at the start (0.1 seconds) and then increases rapidly to 2 seconds. Only when using 3 layers is there almost no change in search time to access from the first page to the last page; a stable search time range is maintained, between 0.5 and 1 second. The explanation for this is that for matching keywords in layer 2, which uses the hard disk with B-tree indexing, the complexity is *O(logN)*. When the search depth of pages increases, the match time will increase by *LogN*. If the number of resources is in the billions, the time will be much longer. Layer 3 is a cache database, which has *O(1)* time

complexity to match keywords, so it is not sensitive to the scale of the data. These 2 layers together take much less time to match keywords than layer 1 because matching keywords in layer 1 is *O(n)*.

Note that the page numbers displayed in the footer of the search results page is generated by dividing the total number of resources by the size of one results page. So when a user does not locate a result that he wants on a page, that page will not be accessed. So generating page numbers has no effect on search time.

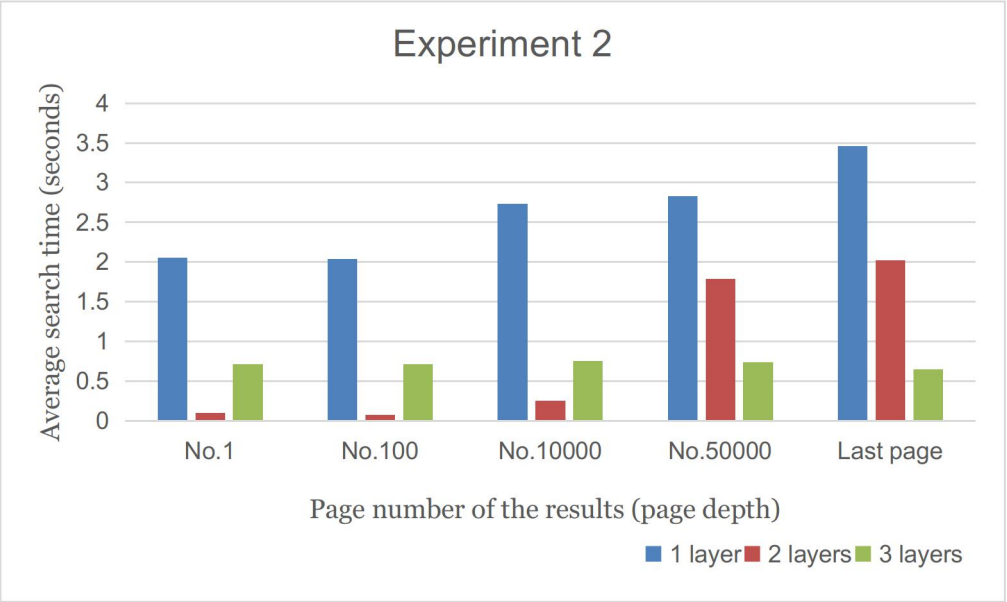

Figure 23. Average search times for 3 layers at various page depths, for 1 million stored resources

Also note that even when using 2 or 3 layers to search results located at any page depth, the system will eventually have to search in layer 1. This means that a certain minimum search time will always be required, that is, the time used to search in the relational database for the full resource information, based on the given set of IDs in the current result page.

6.3.3 Search Time on Busy System

Table 6 shows the search times during busy operation. It shows the performance impact while millions of resources are inserted into the databases (all 3 layers) and also while the resources are updated in the databases. We can observe that data insertion has a certain impact on the

relational database in layer 1, but less of an impact on layers 2 and 3. This is because the second and third layers only record the keywords and corresponding IDs of resources, and the data structures used are very simple and small, especially in layer 3, the fast cache database.

Table 6. Test results under high pressure query

id	keyword	operation	layer1	layer2	layer3
1	a	insert	1.449	1.615	0.872
3	implementing	insert	2.09	1.193	0.702
1	a	update	207.016	338.566	89.327
2	implementing	update	timeout	timeout	timeout

We observed that if a large number of resources are being updated at the same time, the three layer databases all reach timeout. Our experimental environment was updating information for one million resources asynchronously, mainly on the **static value** of the resources. We did not try to resolve the timeout issue through changes in system design or implementation as it is beyond the scope of this study. But we can make the following suggestions for big data analysis and resource value analysis that may be helpful for avoiding timeout conditions:

1. Data analysis should be done when users are least active, perhaps after midnight.
2. Data analysis may be best performed on a separate server, instead of on the primary server (see Section 4.2.1).
3. Data updates should be buffered and performed under low system loads.

6.3.4 Search Time for Ranked vs. Unranked Results

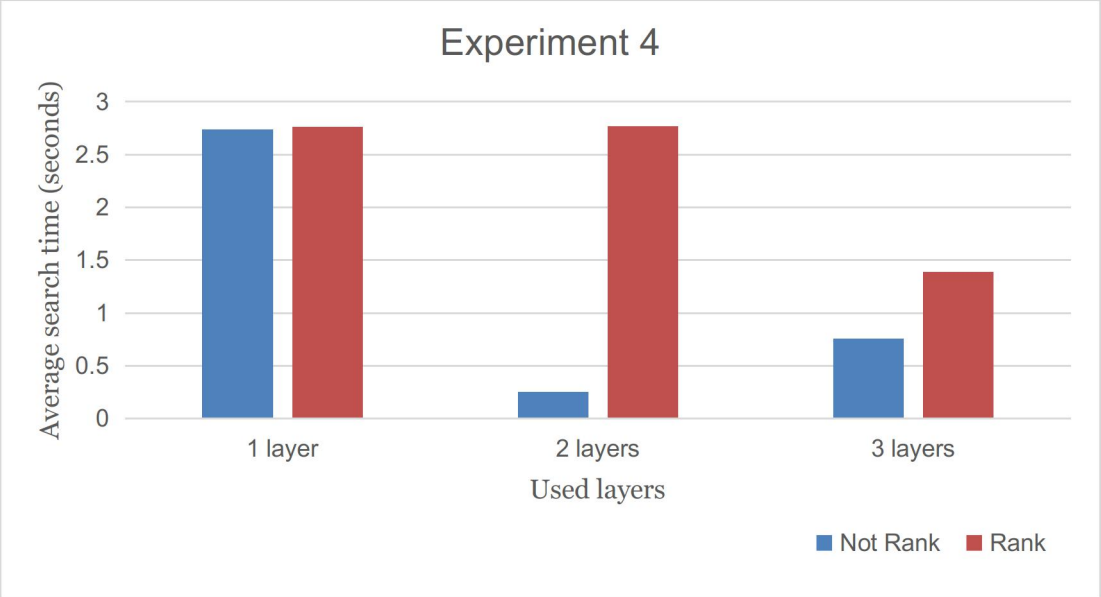

Figure 24. Average search times for 3 layers, returning ranked or unranked results, for 1 million stored resources

Figure 24 shows the average Experiment 4 search times for three-layer storage when the search results are ranked and when they are unranked. When sorting the resources by rank, the search time is bound to increase because the sorting algorithm naturally takes more time on the server. This experiment was conducted to observe how much of a difference sorting makes for search time. There were one million resources in the databases for this experiment. The processing time with one layer is very high even if sorting by rank is not done. Using 2 layers is very fast with no ranking because layer 2 only spends time to process queries; for a B-tree, as long as the number of pages to be queried is small, the query time will not be too long. However, the addition of ranking requires extracting all the corresponding IDs, which is equivalent to accessing all the pages of matching resources, and then sorting these IDs through **LRV** ranking algorithm. So, accessing all resource information in layer 2 is slow when a large number of pages needs to be sorted. Adding the third layer speeds up ranked searching significantly because layer 3 can take resource IDs and their values from memory directly. Ranking is fast in memory, except that some data may need to be swapped to disk. Therefore, the time added in layer 3 is basically only ranking time, which is very different from that of layers 1 and 2.

Note that because layers 2 and 3 depend on the complete resource information stored in layer 1, layer 1 will always be searched eventually. For layers 2 and 3, the ID query and ID ranking mentioned here are completed before the IDs are passed to layer 1.

6.3.5 LRV Ranking Verification

Experiment 5 was conducted to verify that the ranked results returned by the search engine are reasonable, that is, correctly match the expected LRV resource ranking values. Figures 25 and 26 are webpage screenshots of search results generated by our search engine. The bottom of the webpage shows the number of pages of resources found. A long list of pages means many resources were found.

These search results displayed on the browser are ranked according to the **LRV** value of the resources. The higher the value, the higher the ranking. To verify the rank list of a resource, we would measure the number of references that the resource has, as well as the weight of publishers and the amount of positive feedback.

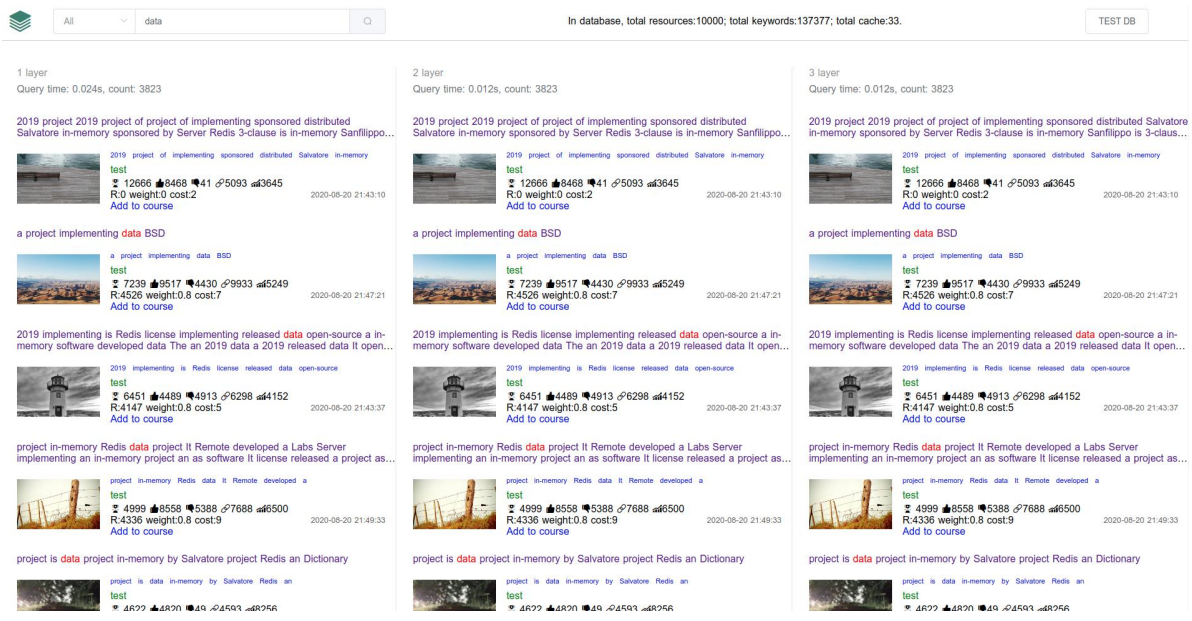

Figure 25. Search results ordered by LRV algorithm, first page

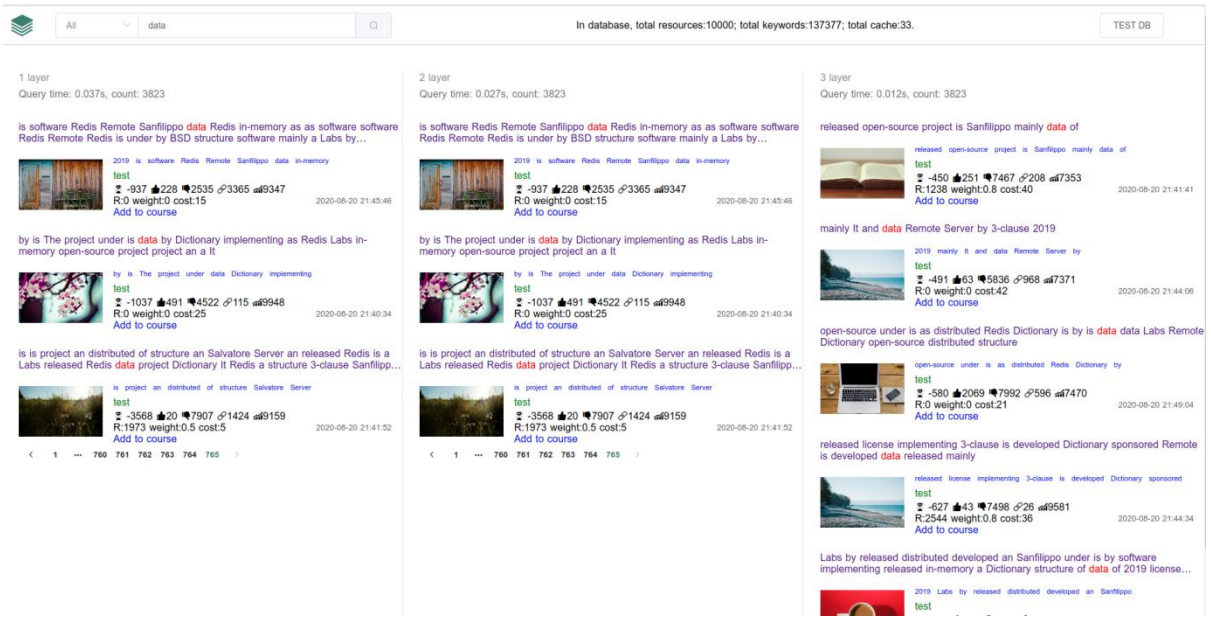

Figure 26. Search results ordered by LRV algorithm, last page

Figures 25 and 26 are the first and last pages of matching resources returned by the search engine. The value of the first resource on the first page is the highest, so it is ranked at the top. On the first page (Figure 25), most of the resources have much more positive feedback than negative feedback, especially the first resource, which has 8,468 positive feedbacks and 41 negative feedbacks. On the last page (Figure 26), all the values become negative numbers. In addition to feedbacks, the number of references and the reliability values are also shown. The number of resource references on the first page is generally greater and the reliability values are also higher. One of the reasons for higher reliability values is that the user account weighs more. The weight numbers for different accounts were varied in the experiments, and a set of weight values that emerged (0, 0.5 and 0.8) were found to be reasonable for uncertified (personal account) user, certified user, and organizational user, respectively.

It should be noted that the above analyses are of experimental results and are limited as a result. First, testing data were mostly artificially generated. Second, resource ranking characteristics such as reliability, user likes, and other judgments of resource value can be quite subjective. This study demonstrated possible experimental approaches for system evaluation. More objective verification can only be obtained through real users' feedback after releasing the system for more public usage.

Lastly, it needs to be pointed out that there is “keywords loss” in layers 2 and 3. This is mainly because layer 2 stores keywords, but some keywords are not parsed. Layer 3 stores “key-values” in memory, and some keywords can be paged out of the cache due to insufficient storage, as discussed in Section 5.2.5 (Cache Switch).

7 CONCLUSIONS

Although there are many kinds of search engines that are now an important part of people's daily lives, there is still room to change and improve. This project aimed to create a search engine that is unique and more useful in certain areas. This thesis proposes a novel search engine, specifically for the field of learning and education. It aims to help users to search for learning resources, and not just any learning resources, but those that are “good”, that is, valuable to support individual users’ learning goals.

This search engine’s advantages are the following: (1) ensure the quality of search results, (2) allow users to focus on their learning goals, and (3) ease the development and operation of the search engine system.

This study focused on two main concerns in the area of search engine research and development: (1) search speed and (2) result ranking. This search engine adopts a three-layer storage structure to improve search speed. It uses the **LRV** ranking algorithm to usefully rank search results. A prototype search engine was successfully built and tested for this research.

7.1 Research Contributions

To summarize, the contributions of this research are:

1. Research search engine technologies used on learning resources.
2. Define the value of learning resources via quantitative methods.
3. Propose resources storage and rank in a search engine system.
4. Build a web application that supports searching of education and learning resources.
5. Design a verification method to locate valuable resources.
6. Apply user behavior to discover resource value.

7.2 Limitations and Future Work

During the research on designing **LRV** algorithms, two issues were unresolved. A "query timeout" occurs when the database is updated under high pressure. The "Reliability" characteristic of a resource has two weights for the certified personal accounts and organizational accounts which cannot be accurately measured for the time being; more experimental data may be helpful in determining these weights.

The future direction of this research will continue to focus on solving two basic issues: storage-query and ranking resources based on **LRV**. To improve the paging strategy in the three-layer storage structure, large pages may be further divided into smaller pages. The "query timeout" issue when simultaneously updating resources might be solved by adding ECS (Elastic Compute Service) servers like Amazon EC2[25]. The Reliability characteristic weight values could be made more reasonable through more experiments or by using real user feedback.

As a software application, the system must pass the actual test of running live on the Internet. If more students, teachers, and educational institutions use this system, more resources will be collected and recognized. A practical search engine system for learning and educational resources can benefit many people in this massive and ever-changing knowledge world.

REFERENCES

[1] Y. M. Patil and S. Patil. “Review of Web Crawlers with Specification and Working.” 2016.

[2] Google LLC. (2019) "Googlebot". [Online]. Available: https://support.google.com/webmasters/answer/182072?hl=en

[3] B. Schwartz. (Jan 28, 2013) "BingBot Crawl Activity Surging?" [Online]. Available: https://www.seroundtable.com/bingbot-crawling-much-16273.html

[4] C. M. Bowman, P. Danzig, and M. Schwartz. “Scalable Internet resource discovery: research problems and approaches.” Communications of the ACM. Vol. 37(8), 2001.

[5] S. Ghemawat, H. Gobioff, and S. T. Leung. “The Google file system,” Proc of SOSP 2003. New York: ACM, 2003: pp. 29-43

[6] D. Zhang and Y. Dong. “An efficient algorithm to rank Web resources.” Computer Networks 33 (2000): pp. 449-455.

[7] C. Bowman and V. Ambrosini. “Identifying Valuable Resources,” European Management Journal, Vol. 25(4), 2007, pp. 320-329

[8] J. B. Barney. Journal of Management, Vol. 17 (1991), pp. 99-120.

[9] T. Reenskaug. (12 MAY 1979) “THING-MODEL-VIEW-EDITOR: an Example from a planningsystem” [Online]. Available: http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf

[10] D. Sullivan. (August 8, 2012) "Google: 100 Billion Searches Per Month, Search To Integrate Gmail, Launching Enhanced Search App For iOS." [Online]. Available: https://searchengineland.com/google-search-press-129925

[11] The New York Times. (May 13, 2017) “How Google Took Over the Classroom”. [Online]. Available: https://www.nytimes.com/2017/05/13/technology/google-education-chromebooks-schools.html

[12] K. Collier. Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing. Pearson Education. 2011.

[13] Agile Alliance. (8 June 2013) "What is Agile Software Development?". [Online]. Available: https://www.agilealliance.org/agile101/

[14] J. Dilley, B. M. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. “Globally Distributed Content Delivery.” IEEE Internet Computing. Globally distributed content delivery. Carnegie Mellon University. Journal contribution. 2018.

[15] K. Fakhroutdinov, OMG™ Unified Modeling Language™ (OMG UML®) specifications. 2007-2016

[16] S. Burbeck. (1992) “Applications programming in smalltalk-80: how to use model-view-controller (mvc).” [Online]. Available: https://web.archive.org/web/20150518095937/http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

[17] D. Kaye. Loosely Coupled: The Missing Pieces of Web Services, RDS Strategies LLC, 2003, ISBN 1881378241, 9781881378242

[18] E. F. Codd. “A relational model of data for large shared data banks,” Communications of the ACM. Vol. 13:6 (1970), pp. 377-387

[19] R. Čerešňák and M. Kvet. “Comparison of query performance in relational a non-relation databases,” Transportation Research Procedia, Vol. 40 (2019): pp. 170-177.

[20] Microsoft. (2012-11-26) “RAM, virtual memory, pagefile, and memory management in Windows.” [Online]. Available: https://support.microsoft.com/en-hk/help/2160852/ram-virtual-memory-pagefile-and-memory-management-in-windows

[21] D. Lee, J. Choi, J.H. Kim, S.H. Noh, S.L. Min, Y. Cho, and C. S. Kim, “LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies,” IEEE Trans. Computers 50 (2001): pp. 1352-1361.

[22] R. Patton. Software Testing (2nd ed.). Indianapolis: Sams Publishing (2005). ISBN 978-0672327988.

[23] M.G Limaye, Software Testing. Tata McGraw-Hill Education (2009). pp. 108–11. ISBN 9780070139909.

[24] K. A. Saleh. Software Engineering. J. Ross Publishing (2009). pp. 224–41. ISBN 9781932159943.

[25] M. LaMonica. (March 27, 2008) "Amazon Web Services adds 'resiliency' to EC2 compute service". [Online]. Available: https://www.cnet.com/news/amazon-web-services-adds-resiliency-to-ec2-compute-service/