The Search Engine for Education and Learning

Youwei Huang

*School of Science, Software Engineering, Monmouth University, West Long Branch, NJ US 07764*

ID: S1296394

Date: 6/1/2020

# ABSTRACT

Nowadays the most common and convenient way to look for information is searching on Internet. The top Internet search engines are handful, such Google[[1]](#footnote-0), Bing[[2]](#footnote-1), Yahoo[[3]](#footnote-2), Baidu[[4]](#footnote-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 “high quality” learning resources to users. High quality 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 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. The details include the strategies designed to optimize general information querying, storage, ranking, and most importantly, finding the relevant learning resources that are “high-quality” for users. Prototyping and experimental study are conducted to conceptually prove this research.

Keywords: search engine, big data, education, learning resources, DLRV, RD

# CATALOG

[ABSTRACT 0](#_Toc897807378)

[CATALOG 1](#_Toc1536955101)

[1 INTRODUCTION 1](#_Toc1809538711)

[2 RELATED WORK 3](#_Toc656016225)

[2.1 Web Crawler 3](#_Toc594719122)

[2.2 RD and RDS 4](#_Toc1654186379)

[2.3 Distributed Storage System 4](#_Toc1499985823)

[2.4 Query and Ranking 5](#_Toc2113222068)

[2.5 Value of resources 6](#_Toc117384707)

[3 CHALLENGES AND DIRECTIONS 7](#_Toc1757425674)

[3.1 Issues of Collecting Contents 7](#_Toc431537562)

[3.1.1 Resource Crawler 7](#_Toc1648068540)

[3.1.2 Data Storage and Query 7](#_Toc1600384265)

[3.2 Issues of Defining Resource Value 7](#_Toc704069124)

[3.2.1 Value related to resources 8](#_Toc293954209)

[3.2.2 Value related to user types 8](#_Toc822821826)

[4 SYSTEM DESIGN 10](#_Toc180531962)

[4.1 Requirements 10](#_Toc1590866746)

[4.1.1 Hardware requirements 10](#_Toc1469101175)

[4.1.2 Software requirements 10](#_Toc673779195)

[4.1.3 Non-functional requirements 11](#_Toc330624248)

[4.2 Software Engineering Process 12](#_Toc310109762)

[4.2.1 System deployment structure design 13](#_Toc1669596187)

[4.2.2 Database Design 15](#_Toc813398747)

[4.2.3 Use Cases 16](#_Toc233316243)

[4.2.4 Package and Class 16](#_Toc2101575642)

[4.2.5 Sequence Diagram 20](#_Toc160628655)

[5 ALGORITHMS 22](#_Toc1986513188)

[5.1 Overcome Limitations of RD 22](#_Toc822062400)

[5.2 Storage and Query on DLRV 23](#_Toc1048464256)

[5.2.1 Relational Database for Basic Storage 24](#_Toc932238557)

[5.2.2 Non-Relational Database for Keywords 25](#_Toc1719869778)

[5.2.3 Non-Relational Database for Cache 28](#_Toc437935710)

[5.2.4 Three Layered Search 28](#_Toc594293620)

[5.2.5 Cache Switch 31](#_Toc228402355)

[5.3 Rank based on DLRV 32](#_Toc1032654832)

[5.3.1 Resource Characteristics 32](#_Toc100996351)

[5.3.2 Resource Evaluation 35](#_Toc1728388179)

[6 EXPERIMENTAL STUDY 36](#_Toc998393253)

[6.1 Purposes 36](#_Toc218381058)

[6.2 Methods 36](#_Toc1338330205)

[6.2.1 Black-box Testing 37](#_Toc1429930815)

[6.2.2 White-box Like Testing 37](#_Toc1866449598)

[6.3 Test Environments 38](#_Toc791230822)

[6.3.1 Hardware and Software Conditions Control 38](#_Toc2133999939)

[6.3.2 Experimental groups control 39](#_Toc12920160)

[6.3 Results and Analysis 40](#_Toc1614052649)

[7 COCLUSION 44](#_Toc167048253)

[7.1 Summary 44](#_Toc1603786906)

[7.2 Contribution 44](#_Toc935670176)

[7.2 Drawback and Future Work 44](#_Toc840827449)

[CITATIONS 46](#_Toc1934411154)

# 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**[[5]](#footnote-4), 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 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 labeled as ‘Ad’; some are videos; and many more 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 the essential for this search engine. Here “High quality” is an subjective judgement from users, but based on common practices in education and learning. The more users are for it, 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 this thesis.

|  |  |
| --- | --- |
| **Term** | **Definition** |
| **DLRV** | Degree of Learning Resource Value. This system 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. |
| **Cost** | The cost of finding and using a resource, both time and money cost. |
| **Popularity** | The trend of searches, clicks, and comments. |
| **Feedback** | User feedback, positive or negative comments, mark on a resource. |
| **Practicability** | Usage of a resource, being visited, 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. |

Table 1. The terminologies

Figure 0 illustrates the overview of the major components prototype in the system. The processes are divided into four parts: collecting data (collecting), resources search (recommendation or rank), data analysis (improvement), verification of valuable resources (verification or test). Normally, search engines are based on Web 2.0 technology today. Here DLRV (degree of 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 0 The general framework of the whole system and key technologies adopted by the search engine.

Finally, experimental and test study are conducted to conceptually prove this research. The details will be 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.[[6]](#endnote-0) There are a few web crawlers behind current mainstream search engines. Two of them are listed as followed.

1. **Googlebot**

The most famous search engine as we know, Google, uses a kind of web crawler named **Googlebot**. **Googlebot** collects web pages and build 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)[[7]](#endnote-1).

1. **Bingbot**

**Bingbot**is a web-crawling robot (type of [internet bot](https://en.wikipedia.org/wiki/Internet_bot" \o "Internet bot)), deployed by [Microsoft](https://en.wikipedia.org/wiki/Microsoft" \o "Microsoft) October 2010 to supply [Bing](https://en.wikipedia.org/wiki/Bing_(search_engine)" \o "Bing (search engine)).[[8]](#endnote-2) **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 (always are 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) Crawler is a kind of automated script. (4) Both of the bots have one thing in common, that is allowing users to block crawlers.

## 2.2 RD and RDS

**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.[[9]](#endnote-3) Resource Discovery Server (**RDS**) returns a set of resources as a searching result, with the links or indexes of web pages from the Internet. Various of search engines support **RD**, such as: **Google**, **Bing**, **Baidu**, and so on.

Take a Google Search of a keyword phase ‘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 does the search engine be so fast? How does it rank the result and give the matching list of the most valuable resources? These two questions are the two core tasks of traditional search engines, regarding 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[[10]](#endnote-4). **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 1 is the basic structure and work principle of **GFS**.



Figure 1 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 among tons of 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[[11]](#endnote-5), which has been well studied as an Information Retrieval (**IR**) topic. According to the **Vector-Space** model, a resource is viewed as a vector *[ w1, ..., wn ]*, where *wi* the significance of the keyword. The value of *wi* equals the number of times a keyword appears in a resource divided by the number of times the word appears in the entire collection[[12]](#footnote-5). A *wi* is zero if the keyword doesn’t appear in a resource. When a keyword appears in more resources, *wi* will be lower; otherwise, *wi* 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. Nowadays, building search engines has a lot of compliance to follow for privacy and security reasons.

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

## 2.5 Value of resources

In the paper of Identifying Valuable Resources, which is published in 2007 on European Management Journal, value of recourses was discussed, from the point of business and management. It is difficult to identify resources in a firm if there is no agreed definition of what ‘valuable’ means.[[13]](#endnote-6) A valuable resource can be rare, inimitable, and non-substitutable to be a source of sustainable competitive advantage.[[14]](#endnote-7) The problem is known by RBV (resource-based view) advocates, but the value of resources was never clearly defined. It is said, to identify the value of resources in business and management area, there are some questions to ask: What is the source of resources? Valuing one resource or many resources? The past, present, future value of a resource? Objective or subjective valuations? What’s 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, we 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, cost is an important element, mostly in monetary, but the cost advantage for learning resource is generally in time. Moreover, in our design, learning resources are endowed with more attributes, which will be 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. Take the massive amount of educational videos alone, the burden on storage system will be huge. Moreover, copyright protection definitely cannot be neglected.

### 3.1.1 Resource Crawler

Because of these concerns, different from the 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 named as “**Resource Crawler”**, instead of web crawler. As a result, this search engine will be a significantly light system to achieve high performance. The copyright issues and system over-storage problems can be both avoided.

The “Resource Crawler” collects the information of resource, in another 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, 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 and study.

### 3.1.2 Data Storage and Query

Even though only resource information are stored, the database needs to handle the potentially very high 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 from such a huge amount of data. The system built for this thesis uses a 3-layered storage structure and an encoded keyword mapping method to improve the search efficiency. For the application in the actual production environment, the storage needs to adopt the distributed system design.

## 3.2 Issues of Defining Resource Value

There are many challenges to address to collect valuable resources. First of all, how to determine whether a recourse is valuable? The Internet Crawler “spiders” are not something smart like human beings, the only thing they do is collecting and bringing the copies of the information back. Much meaningless and even fake or unhealthy information is obtained too. The search engine should not only filter bad impropriety or dangerous information, but also recommend the content according to the user's interest; however, that is still far from the definition of being valuable resources. Some issues can be magnified to different areas and user groups.

### 3.2.1 Value related to resources

In the field of education, there are many issues exposed by traditional search engines. It could be 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? How is the price?)
* How many positive feedback? (Do other users find the resource useful?)

To solve this problem, the new search engine is designed to meet educational purpose, to support different types of users, returning valuable search result and maintain query efficiency in term of speed. Several techniques are created. We summarize them into two problems to address:

* How to collect and manages education resources?
* How to define a valuable learning resource?
* How to quickly process queries from users ?

### 3.2.2 Value related to user types

Even if the resources are reliable and safe, for different user groups, they can be still be good or bad, because users of different ages, level of educations, skill experiences, and even learning preferences have different needs and expectation of what they look for. Ideally, search results should be close to the ability of different users. Below are some questions more from user perspective.

* What age group is the recourse suitable for?
* What grade level is the resource for?
* What type of resource is it? (e.g. recording, cartoon, for user preference)
* Is it for professionals in the field?
* What experience level is the resource for? (e.g. entry-level, intermediate, senior)
* Is it made for users of special ability? (e.g. language, disability)

Inspired by the advantages of valuable resources in business field as discussed in Section 2.4, we define the value of learning resources, based on the following characteristics:

* Reliability
* Practicability
* Suitability
* Popularity
* Feedback
* Cost

These 6 characteristics are used to determine whether a learning resource is reliable and with high-quality learning resource. As shown in Fig 2, the 6 characteristics are used to evaluate the value of resources, of which light blue is the dynamic characteristic and gray ones are static characteristics. ‘Dynamic Characteristics’ are generated dynamically during searching and ranking, 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 the data of resource is changed, such as feedback, cost, and etc. Dynamic characteristics are generated in real time.



Figure. 2 The characteristics for DLRV

# 4 SYSTEM DESIGN

## 4.1 Requirements

### 4.1.1 Hardware requirements

The whole search engine platform requires several high-performance servers which can potentially undertake 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 allows users use browser access to server. Generally speaking, the requirement on server performance and system configuration is to meet the demand of user number and resource volume. Therefore, in the early stage, there is no detailed requirement report for the whole set of server-side hardware, but we lay out some basic requirements presumably.

* The dual core processor E3 based on X86 system
* More than 16GB RAM
* 80GB disk storage
* Independent database server
* The distributed deployment server based on Linux is prepared with container and virtualization technology, but will not be used in the experimental time
* Data analysis server and other micro services

For user-end or test-end, we require PC and mobile devices to test all the web pages and functions to work well 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 SDK to implement.

1. **MVC design pattern**

MVC pattern is a very classic design pattern in software engineering. It was first proposed by Trygve Reenskaug in 1978[[15]](#endnote-8), and later became popular in web development area. A framework based on this design pattern can be called MVC framework. All the development and implementation described in this thesis are based on this design pattern. In another 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 part of business logic. MVC design pattern can achieve high cohesion and low coupling, and it separates data, view and business logic. MVC improves the development efficiency, code cleanliness, and has higher scalability. The purpose of using this mode is to make the search engine easy to optimize and expand the function in the experiment.

1. **Related application software**

* MVC framework based on Node and express.
* The view layer is based on VUE[[16]](#footnote-6), also called front end, which is used by users.
* Reverse proxy server and HTTP server are based on Nginx.
* MySQL database, a kind of relational database, stores a large number of data generated for the search engine system.
* Redis, a kind of non-relational database, which stores data in memory, used as cache in our design.

### 4.1.3 Non-functional requirements

To build a real search engine system, there are more to consider as requirements, such as of those that are non-functional. While we don’t take all of them in our prototyping, they are listed here for completeness.

1. **Performance**

Google answers 100 billion searches per month[[17]](#endnote-9). That means the average of a day is at least 3 billion, based on the statistics in 2012. Our learning resource search engine doesn’t need such high search performance because we are targeting at one special area, instead of all the users and resources on the Internet.

Around 2017, there are more than 30 million children use Google education apps[[18]](#endnote-10), this is not including college students and other adults, so our system needs at least double of this amount (children) for users’ requests, assumedly to accommodate 60 million users per day. More formally, DAU (Daily Active User) is at least 60 million. To accommodate extra requests from the increasing DAU, we raise the performance bottleneck to 100 million DAU in our development plan.

1. **Reliability**

The operation of all the services are 24 hours. To ensure users get search results within 1 second after starting the search, the response speed should be less than 1 second for each query.

1. **Security**

System layer security:

* Firewall between server nodes, access control on blacklist, white-list and iptables technologies.
* Data backup to prevent the data loss disaster.
* When main servers crash, use the reserved servers instead.
* Quick recovery plan for crashed servers.

1. **Business layer security:**

The security points of business logic are listed bellow

* User verification
* API requests security
* User behavior logs
* Cookie or cache security
* User privacy
* Encrypt and decrypt data

## 4.2 Software Engineering Process

In real system design, there are engineering design, system architecture deployment design, and unified modeling language. Online system and theoretical verification of the system should follow all of the design principles. In Figure 3, the design of software process follows the life cycle of software engineering and adopts agile model. Agile model is a working format that development requirements and solutions are completed through the collaborative effort of self-organized and cross-functional teams, and their customers or end users.[[19]](#endnote-11) It advocates adaptive planning, evolutionary development, early delivery, and continual improvement. This approach encourages flexible responses to changes occurred during system development, maintenance and upgrades.[[20]](#endnote-12)



Figure 3 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 specially created for this project are discussed in details 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 practically common web deployment mode as illustrated in Figure 4.



Figure 4 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 4, from top to bottom, from left to right, there are 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 different parts are to control access, for the security of data center. In our design, part of database server data is stored in 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.

The purpose of CDN is to speed up searches based on the existence of static files. CDN stands for a content delivery network, or content distribution network[[21]](#endnote-13). Static files can be distributed on multiple nodes of the Internet. Commonly applied in distributed system, 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 the access list, which can be a white list and prohibit illegal users from directly accessing the cluster.[[22]](#footnote-7)

There are many servers in the server cluster, most of them are controllers dealing with business logic, There are also some servers specialized in processing big data. For example, in the system we built, DLRV (ref: 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 DLRV algorithms and provide the core business 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 on 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 discussions are made in Chapter 5 and 6.

### 4.2.2 Database Design

ER model (Entity-relationship model) is used to present to 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 5 is the complete database (relational database only) design diagram of the search engine system.



Figure 5 ER model in UML of the relational database in the system

In Figure 5, we can read: user table is used to store user’s information; resources table stores resource information; 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. ‘ctypes’ and ‘rtypes’ are tables to store the types of courses and resources respectively.

‘Users are the owners of the resources’, this become a extremely important point which makes this search engine different from others. Users have the right to monitor the quality of the resources. This is the key point of reliability (one of six characteristics) evaluation in DLRV system.

### 4.2.3 Use Cases



Figure 6 Use case diagram for the search engine functions

Figure 6 is the use case diagram of the search engine system which covers 7 main use cases. Most are for just 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 considered as potential learners, though they can search for other users, e.g. parents for their children. Users can search resources, upload resources information manually, set up courses by group resources (resource information indeed) into a list, and evaluate courses or resources. When a resource is cited by a course, 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 DLRV 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](https://www.uml-diagrams.org/namespace.html) 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.[[23]](#endnote-14) For the server side of the search engine system as designed for this project, Figure 7 shows the main packages, mapped as different folders or collections. Package have dependencies among them.

* **Controller**. This is a package includes all the controllers. It is designed in MVC mode. Controllers deal with all the business logic, take responsibilities for connecting data and views, and accept users’ requests and responses. Controller depends on private libs and public modules.
* **Server.** It is the entrance of the whole system. This package manages the files to work as a web server. The data from user-side enter this package first. It depends on controller because the request and data from user-side need controller to serve them. It depends on public modules.
* **Config.** A package manages the connection configuration of various servers, like mail server, database server, OSS 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 libs.** The package contains all the private modules, plugins used only in this system. It depends on public modules and Model 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.[[24]](#endnote-15) It can manage the data, logic and rules in the system.
* **Modules**. Modules are the public modules. Public modules are some free software, some of which are public plugins for various developing language. They are from the Internet, generally maintained by the community or individuals. Most of them are open-sources.



Figure 7 System package diagram

In Figure 7, **Config** and **Modules** packages have the most dependence from 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 8 Class diagram of the package Controller

The controller package is the core package of the whole system. 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 9 Class diagram of the package Private

The core class of 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 functions such as get random numbers, format date, check user info, and etc, which are used with high frequency and commonly. It can be imported and used by controllers to reduce code redundancy and coupling. It is a very common design idea in software engineering. High cohesion, loose coupling[[25]](#endnote-16). Class DB provides a set of methods to operate database, such as inset, query and delete.

**C. Server Package**



Figure 10 Class diagram of the package Server

The server package is the web server start-up entry. It listens 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 11 explains what is the convention routing mode.



Figure 11 Router to Controllers

The 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 is that it does not need to configure the route files. The 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 security guarantee. Static files, like images, CSS files, fonts and other files, can be used by the website visitors or browsers, but not the source code and executable files.

### 4.2.5 Sequence Diagram

The main function of search engine system is to search valuable resources, so in many sequences series, this part mainly studies the logic of search sequence. The following is the UML design of search sequence.



Figure 12 sequence diagram of search process

The whole search process of the system: 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 DLRV 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”. It means: (1) Fast access to search results, and (2) Valuable learning resources, which is in line with the theme of the thesis. From programming point-of-view, they are ‘storage and search efficiency’ and ‘resource sorting and filtering’.

The definitions and methods in traditional search engines is about **RD**, which stands for Resource Discovery6. **RD** has been explained in chapter 2.2. The improved innovative algorithms used in the learning resource search engine system is named **DLRV**, which stands for “Degree of Learning Resource Value”. Like the approach of **RD**, **DLRV** contains two parts: storage-query and results-rank.

## 5.1 Overcome Limitations of RD

The traditional **RD** doesn’t have that much advantage when it comes to the learning resource search engine.

The distributed system takes a large place of storage and needs hundreds of distributed servers. In our system, we only save the links and main information of resources, without caching any original resources. If we distribute resources to multiple servers, it may waste a lot of physical resources, and in terms of software design, and it is also quite costly to create and maintain such a system. If we assume learning resource database is significantly lighter than those all-purpose big search engines, we do not choose to use distributed storage. But we still take the distributed storage into consideration. Unlike **GFS**, we don't need to adopt a new file system. This kind of distribution is only based on the database itself.

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 this particular area, learning resources. The traditional **RD** approach calculates the significance of each keyword *[ w1, w2 ... wn ]* in a resource (as explained Section 2.4). However, this score is too simple for learning resources. Besides the degree of matching keywords and appearing frequency, learning resources have more important attributes to determine the quality of the resources. In the section 3.2, the Figure 2 displays the 6 vital attributes of a learning resource: Suitability, Popularity, Reliability, Practicability, 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 with human judgmental 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 the 6 features. The difference of traditional RD and our DLRV in ranking characteristics is listed in Table 2. More details are presented in next sections.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Suitability** | **Popularity** | **Reliability** | **Practicability** | **Feedback** | **Cost** |
| **RD** | Yes | Yes | No | No | No | Not sure |
| **DLRV** | Yes | Yes | Yes | Yes | Yes | Yes |

Table 2 Characteristics in RD and DLRV

## 5.2 Storage and Query on DLRV

As explained in Section 5.1, There is no need to use distributed storage in a light learning resource system, especially the 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 **DLRV** system, an easy and convenient way to refer to the search engine proposed in thesis.



Figure 14 Database layers used in DLRV

In **DLRV** resources system, resources are stored in the traditional web storage mode, ‘database center driver’. It means all the basic information and relationships of resources are stored in the databases. Databases are divided to three layers in our system as shown in Figure 14: one relational database, one non-relational database in disk, and one non-relational database in memory cache. Below highlights how different layers work together to gain outstanding performance:

* The non-relational database in disk stores only pairs of keyword and its corresponding id list. The JSON data has simple structure and small size, and 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 hash map in this layer.
* Once recourse IDs are found, they are used to find complete resource information organized in the relational database. An 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.[[26]](#endnote-17) All data are logically managed in tables. Each instance of the data is called a row, which is stored in a table. In the **DLRV** system, relational database is used as our main data management format. The Figure 15 shows some relationships and entities in the database. These relationships well reflects the business logic function points of this search engine application: refer resources, publish resources, comment resources, and more.

In **DLRV** 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 **DLRV**. Next sections introduce (1) how the relational database cooperates with the non-relational database to manage data; (2) how to improve the speed of search.



Figure 15 Relational database used in DLRV 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 amount of data. Figure 14 shows a set of storage solutions containing three databases: two of them are persistent storage database stored with data on hard disks; and the other one is cache database stored data in memory. Among the three, in worst scenario, the reading speed compares as the following: Cache Database (Non-relational database) is highest; Hard Disk (Non-relational database) is in the middle; and Hard Disk (Relational database) is slowest.

The overall query performance (experiment) of different modern database systems are compared in the following table[[27]](#footnote-8). All the experiments were performed on **MacBook pro** year 2015 8GB RAM and i5 processor.[[28]](#endnote-18)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

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

Out of these database systems, Mongo, Redis and GraphQL are for all 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 the storage. More discussion will follow next.



Figure 16 Non-relational database storage structure in layer 2 in DLRV

Figure 16 shows the ‘key-value’ data structure used to manage data in non-relational database. The ‘key-value’ data structure is very common in non-relational database, storing “one key to one value”. The ‘key-value’ data structure is stored as “JSON-like” documents with optional schema by **MongoDB[[29]](#footnote-9)** in the second layer, indexed with B-tree[[30]](#footnote-10). In this database, the data type format is JSON and the search time complexity of B-tree is *O(logN)*.

Because one keyword can map to multiple resources, a keyword will filter out multiple resource entries (as IDs) in database. In a non-relational database, referring to Figure 16, the “key” is a word parsed from title, tags and description from a resource, and the “value” is a set of resource IDs stored like *[id1, id2, id3, …]*.

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.

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

Table 4 key-value stored in non-relational database

When search keywords are received, if there is only the relational database (layer 1), the search would be directly processed as a scanning of rows in the resource table sequentially*[[31]](#footnote-11)*. Sequentially accessing records to find resources matching a keyword, compared with title, tags, descriptions at the same time. When there is 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 “key-value”. **MD5** is an algorithm for inputting variable length information and outputting 128 bits of fixed length, as seen in Table 4 as examples. The purpose of storing keywords in this way is to ensure that each key stored is 128 bit long. The advantage is that the storage space for a “key” is fixed. However, **MD5** is not decodable. Does this affect searching? Take Table 4 for some examples: The left column are the “keys” as **MD5-**encoded. They can be tags, keywords, titles and any other words parsed from long search-strings submitted by users. The right column are the list of IDs to the corresponding resources.

The full searching process in layer 1 and 2 is displayed in Figure 17 as a flowchart: first, take a search keyword; then, encode the keyword by 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, so the search speed is fast too[[32]](#footnote-12).



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. Please note, non-relational database typically has advantage when data can be simply represented, so it is a best choice 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, 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 a database system based on memory. 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.

Using the cache non-relational database is to further improve search performance gained from layer 2. At layer 2, when the amount of data reaches millions rows storage each day, the storage space will become insufficient, and the query speed via B-tree will be significantly slowed. Using the cache database Redis 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 cache database like Redis include the following:

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

However, there is also layer 3 has its own limitation. Although the memory speed is much faster than the hard disk, its storage space is far less than the disk space. Take a personal computer with 16GB memory and 1TB hard disk as an example, the memory is only 1% of the hard disk. What’s more, the price of memory is much higher than that of hard disk. Therefore, we store fewer data in the third layer than in the second layer, which indicates that the query loss rate of layer 3 is higher. The solutions 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 is shown in Figure 18. Each layer has its own feature 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 as “keys” and the IDs of the corresponding resources as “values” are saved as with “key-value” data structure, just like those stored in layer 2. But the differences between layer 2 (B-tree) and layer 3 (Hash) causes big speed gap between them.



Figure 18 3-layers 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 gets to the relational database for whole resource information; otherwise, the search take a detour to the non-relational disk database. Please note, the non-relational disk database tries to hold all possible keywords and tags, but that there is no guarantee. In rare cases, if the controller still can’t find the keywords in non-relational disk database, the search will have to be done in the relational database directly.



Figure 19 3-layered search process

### 5.2.5 Cache Switch

Memory size is limited, even if having distributed storage in consideration. It is important to reserve sufficient memory space for smooth operating system operation. Special care (via algorithms/methods) is needed to keep the size of the non-relational database cache space 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 system area, similar strategies are proposed to handle the issues of this case.

Paging is concept already 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 operation system set as fixed numbers of blocks for data transferring between memory and disk[[33]](#endnote-19). 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.[[34]](#footnote-13) When searching “1” on Google, as a testing example, there are more than 25 billion result. 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. Redis can’t handle that much. Therefore, the third layer only allows the system to store limited IDs of a keyword, the rest of IDs are stored in layer 2 on the disk. Actually, it is not just the limitation of Redis itself, rather the real memory size on a server also limits the size of all IDs. To solve the gap, paging technology 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)[[35]](#endnote-20), **FIFO** (First In First Out) and **Clock**. **LFU** is found most suitable for the proposed search engine system. Following **LFU** rule, in non-relational cache database, the page with the *<key, ids>* with lowest search frequency is next to moved out cache. Table 5 shows an example of keywords, IDs and frequency in cache 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 |

Table 5. Values and keys in cache non-relational database

The switching process is shown in Figure 21.



Figure 21 Cache switch process

In a scenario, there are the <*key, ids*> entries already in the cache database and the cache space cab easily used up. Supposed there is a new search with keyword *key5*, and it is not in the cache (the example in Table 5). If the keyword *key1* has lowest frequency, then the data of *key1* will be replaced by that of *key5*.

## 5.3 Rank based on DLRV

The biggest difference with normal search engines is that a learning resources search engine needs to be evaluated more strictly and on more features. As introduced earlier, the basic DRLV idea is designed to evaluate learning resources based on their six characteristics. We call it the DLRV model. The list of resources quantifying in six characteristics are returned to users according based on their matching level, from high to low.

### 5.3.1 Resource Characteristics

Resource characteristics can be static or dynamic. Dynamic means that the value of 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. Which means they are determined by the status of a resource itself. Static is not constant. It is just that static characteristics will not be changed in real time and will not change due to user search conditions. Static value will also be changed due to the change of resource in views, evaluation, publisher, and so on.

1. **Suitability**

Resource Suitability is a dynamic characteristic of **DLRV**. It changes with each search behavior. Suitability means the results matches 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), check against all the keywords, from k(1) to k(N)*:

It adds up the frequency of each keyword in the title, tags and description of a targeted resource. Here a frequency is the number of time a keyword appears each place: title, tags, or description. The higher the frequency of occurrence, the higher value of resource suitability. Finally, 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 more than zero to be meaningful.

1. **Reliability**

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

It is a piece wise function:

* If the publisher of *user(k)* is not certified (variable *cer(k) = 0*), the *R(j)* is 0.
* If the publisher of *user(k)* certified and it is not an organizational account, the *R(j)* = *w1* the total usage of all this publisher’s resources so far. That means, 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 userk has published *Z* resources.

In the formula, *w1 and w2* are weight numbers. They should be adjusted through experiments, in order to make sure the value of Reliability be meaningful.

* If the publisher is an organizational account, *R(j)* = *w2* all the usage of the resources the user has published.
* The measurement of the usage in this system is how many times a resource itself has been referenced in courses or other resources.

1. **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.

1. **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 0.

In the function, *F(j)* is the value of overall feedback for resource *r(j)*. *p(j)* is a positive feedback and *n(j)* is a negative feedback. *t(j)* is the total feedback, which equals *n(j)* + *p(j)*. So *F(j)* can be actually viewed as the rating for a resource.

1. **Popularity**

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

*V* is the number of accumulative views of a resource. *Vd* is the accumulative views of ‘today’, *Vd-1* is the accumulative views of ‘yesterday’. The number of views added today is the value of popularity.

1. **Cost**

Cost includes time and money. *C(j)* is the value of Cost; *T(j)* is the value of time cost; and *M(j)* is the value of monetary cost.

### 5.3.2 Resource Evaluation

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

**(1) Static Value**

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

Static value of a learning resource adds Reliability, Usage, Positive feedback multiply popularity, and then the value is divided by Cost.

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

**(2) Dynamic Value**

There is only one dynamic characteristic among the six characteristics in the DLRV model. Suitability is changed dynamically when users search the resources. Different query keywords can cause different Suitability value. The calculation of dynamic value refer to Section 5.3.1 a. **Suitability**

**(3)** **General Formula**

The Value of resource:

The *V(j)* is the final value of a resource *r(j).* It equals to its Suitability times its total Static Value.

# 6 EXPERIMENTAL STUDY

In the field of software engineering, software testing and quality assurance is an important part of software development. The method of testing is to use virtual data to test the functions of software. All the data obtained in the test can be used to evaluate the software performance, functional logic, algorithms and security. This chapter is for all the above system structure, algorithm, data structure simulation test. Since the project is not online, all the data are analog data, which is generated and searched according to the user's habits. I will use script to test the developed system on the local server, and write data of order of magnitude less than one million.

## 6.1 Purposes

The main purpose of testing is to meet all the requirements and qualify the quality of the software.

We report the test in this section for the following purposes:

* To test the performance of 3-layered storage structure.
* To test the accuracy of DLRV ranking results.

The first purpose is to verify the query-storage part of the methods. The second part is to verify the DLRV ranking algorithm. These two parts are the core of this learning resource search engine, so we test them in this paper. Other requirements and functions can be verified or tested through similar methods, we will not repeat here.

The expected results of the test are

* Less search response time.
* More high-value resources are ranked topper.

## 6.2 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.

### 6.2.1 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.[[36]](#endnote-21)

Black-box testing will be used in this search engine system. It is used for testing the ranking part of the search engine. Black-box testing is very suitable for the user searching process, from keywords to the list of all the most valuable resources. The search function is a black box, testers don’t need to understand the principle of the searching algorithms and ranking methods (**DLRV**). The testers only verify the value of the resources and rank results. Resources with high value are ranked at the top in the list, while those with low value are arranged at the back. Value itself is a more subjective concept. Later in this section, we will discuss how to measure the user's recognition of resource value.

### 6.2.2 White-box Like Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) verifies the internal structures or workings of a program, as opposed to 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.[[37]](#endnote-22) [[38]](#endnote-23)

We use white-box method to test the storage-query part, especially the 3-layered storage structure. It should be noted that here is not a strict white box. According to the previous section, white-box testing asks users to choose different test paths according to the logic of the code. However, here we are not changing the input paths to get the expected results for the code logic of a function. In this part, we query the same amount of data in different storage structures, and design different experiments variables through the different keywords *[ k1, k2, k3 ... ]*, pages *[ p1, p2, p3 ... ]* and other conditions. Anyway, the logic is very similar with the white-box, we need to change the structure (for layer1, layer2, layer3) by changing the code which means the code is transparent to testers. Different from the black-box test mentioned in the previous section, testers need to understand the logic of different layers and test searching business by following different logic codes for different storage structure.

The purpose 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 3-layered storage structure.

## 6.3 Test Environments

In the test, the software and hardware environment of the project is very important. The test results are different in different environments. In this part, we stipulate the standard of test environment to prevent error and unfair data caused by different environment

### 6.3.1 Hardware and Software Conditions Control

In testing the storage and search process, we should ensure that some environments are fixed:

* Same local network
* A server with the same configuration
* A PC with the same configuration, the same browser

Test controls the network environment to prevent network problems from affecting the query speed. The server configuration is the same because different server configurations will course different test results. All test data must be based on the same set of server configuration, including hardware and software. The client used in the test must also be the same, less strict than the previous two items.

Local Network: a home router, the server and PC are connected to the same router through wired LAN port, 100M fiber.

Server Software:

* Linux operating system Ubuntu distributed, version 20.04.
* NGINX Tengine 2.2.3 proxy HTTP server.
* Relational database MySQL 5.7
* Non-relational database MongoDB
* Non-relational database in memory Redis
* Node.js latest version

Server Hardware:

* 16G DDR4 PC Memory Cards
* Intel i7 7700
* NVME SSD 512GB Samsung pm961
* ASUS motherboard
* Intel 1000M network card

Client Side:

* Chrome Browser
* JS script, automated testing

### 6.3.2 Experimental groups control

In the test, there are variables and constants in the same experimental group. The purpose is to compare the different variables in the experiments with different effects in this system and prove the performance, user experience, accuracy of this learning resource search engine.

Suppose that the generated resource content comes from 100 different word dictionaries

1. Experiment 1: Test the search time under different orders of magnitude
2. Test the average search time of 4 keywords, “a”, “is”, “2019”, “open-source”.
3. The data levels of resources are: 10, 10^0, 10^1, ... 10^6.
4. The amount of data in the database is fixed when searching.
5. Search page 1.
6. Experiment 2: Test different pages number
7. 1 million resources in database.
8. Random pages: 1, 100, 10000, 50000, end page.
9. 2 keywords as cases, get the average time of them.
10. Experiment 3: Under high query pressure
11. 1 million data is inserting or updating to databases (layer 1, 2, 3), 100-200 asynchronous requests per second.
12. 1 million resources existed in databases before start testing.
13. Search page 100.
14. 2 keywords as cases.
15. Experiment 4: Search keywords order by the value of resources
16. 1 million resources existed in databases.
17. Compare 2 conditions: rank by value or not
18. Search page 10000
19. 2 keywords as cases, we use the average time of them.

## 6.3 Results and Analysis

Chart 1. The average search time of 3 layers in different orders of magnitude

Analysis of chart 1:

This chart a line view of the results in experiment 1. It shows the the average search time of 3 layers from 1 to 1 million level random resources. For layer 1, the line shows the increasing time of search requests directly entering relational database. It is the fastest growing, with an average search time of more than 2 seconds when the data level reaches 1 million. It can be said that the bottleneck of relational database appears at this level. Compare with layer 1, the layer 2 and layer 3 are both less than one second, even a half time of layer 1. Therefore, when the data is less than 104, the speed of the three-layers query is closed, but when the data reaches more than one million, the effect of the second and third layer for keywords mapping is very significant.

Note: The search process follows the Figure 19 in chapter 5.2.4, that means layer 2 and 3 in the chart, the search time includes that search keywords in non-relational database and search resources in relational database.

Chart 2. Random pages search time of 3 layers based on 1 million data

From Chat1, we know that the performance of layer 2 is already very outstanding, so why we need a third layer for cache? Chart 2 gives us the answer. Chart 2 shows the average time taken by a query for increasing page number to simulate that user goes to random pages after searching a keyword. The statistics are based on the level of one million resources. As the number of page increases, the search time of layer 1 increases gradually, while that of layer 2 increases rapidly, from less than 0.1 second to 2 seconds. Only the third layer has almost no change of searching time, it maintains a stable search time range, less than 1 second and more than 0.5 seconds.

We have talked about the storage type of layer 2 in chapter 5.2.2, the index algorithm uses B-tree, the complexity of search time is *O(logN)*. Therefore, while the search page increasing, the search time will increase as the function *LogN*. If the data reaches billions, the time will be much longer. At this time, the role of the third layer, cache database, is reflected. *O(1)* is not sensitive to the scale of the data.

| **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 |

Table 6. Test results under high pressure query

Table 6 shows the searching time under high pressure query of data. It gives the example that when millions of data is inserting into databases (all 3 layers) and updating the resources info in databases. We can observe that data inserting has a certain impact on the relational database, layer1, while less impact on layer2 and layer3. Because the second and third layer only record the keywords and corresponding IDs of resources. The data structure is very simple. Especially the layer3, cache database, which is based on memory, write speed is much higher than the databases on the disk.

However, you may notice that when updating a large number of data, the three layer databases are almost all timeout. Here, our experimental environment is updating one million resources information asynchronously, mainly the **static value** of the resources (refer to section 6.3.2, experiment 3). This means that big data analysis and resource value analysis should pay attention to the following issues:

1. Data analysis should avoid users’ active period, maybe after midnight.
2. Data analysis requires a separate server instead of the primary server, refer to chapter 4.2.1 and figure 4.
3. Data updates need buffering and waiting.

This paper presents the feasible solutions to “timeout trouble”, but this part doesn’t belong to our topic, and is not included in the design, development and experiment.

Chart 3. Search time of 3 layers, rank or not rank based on 1 million data

According to the results of experiment 4, we have the chart 3. It shows the average search time of 3 layers while ranking value or not. And obviously, when sorting the resources, the search time is bound to increase, because the sorting algorithm will take much time on the server. Take one million data to query the resources on page 10000 as an example, the speed of the first layer is very slow even if it is not sorted. We ignore it. The second layer is the fastest without ranking because it only takes time to query the data. For B-tree, as long as the number of pages to be queried is small, the query time will not be too long, it has been discussed in Experiment 2. However, ranking needs to take out all the corresponding IDS, which is equivalent to the total number of pages, and then sort these IDs through DLRV ranking algorithm. It is very slow to query the total number of pages in layer 2. The third layer can take all the resource IDs and their values from the memory directly, and then rank them. All the processes can be completed directly in memory (except that some units that are switched to disks need paging according to section 5.2.5). Therefore, the time added in the third layer is only ranking time, which is no different from that of the first layer and the second layer.

Note: Because the second layer and the third layer depend on the complete resource information in the first layer, which means the first layer will be queried finally in any way. For the second and third layers, the ID query and ID ranking mentioned here are completed before the IDs go to the first layer.

# 7 COCLUSION

## 7.1 Summary

## 7.2 Contribution

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 high quality resources.
6. Apply user behavior and big data analysis to discover resource value.

## 7.2 Drawback and Future Work

# CITATIONS

1. https://www.google.com [↑](#footnote-ref-0)
2. https://www.bing.com [↑](#footnote-ref-1)
3. https://www.yahoo.com [↑](#footnote-ref-2)
4. https://www.baidu.com [↑](#footnote-ref-3)
5. https://www.youtube.com [↑](#footnote-ref-4)
6. Patil, Yugandhara; Patil, Sonal (2016). ["Review of Web Crawlers with Specification and Working"](http://www.ijarcce.com/upload/2016/january-16/IJARCCE%2052.pdf) (PDF). International Journal of Advanced Research in Computer and Communication Engineering. [↑](#endnote-ref-0)
7. Google LLC, ["Googlebot"](https://support.google.com/webmasters/answer/182072?hl=en). Google. 2019-03-11. Retrieved 2019-03-11. [↑](#endnote-ref-1)
8. Anonymous, ["BingBot Crawl Activity Surging?"](https://www.seroundtable.com/bingbot-crawling-much-16273.html). Retrieved 2016-07-16. [↑](#endnote-ref-2)
9. C.M. Rowman, Scalable Internet resource discovery: research problems and approaches, Communications of the ACM 37 (8) (1994) 98–107. [↑](#endnote-ref-3)
10. Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. The Google file system[C] Proc of SOSP 2003.New York:ACM,2003:29-43 [↑](#endnote-ref-4)
11. Dell Zhang \*, Yisheng Dong, An efficient algorithm to rank Web resources, 2000 Published by Elsevier Science B.V [↑](#endnote-ref-5)
12. Collection in our system are all the resources in our database. For the general search engine system, collection are the entire Web resources. [↑](#footnote-ref-5)
13. Cliff Bowman, Veronique Ambrosini, Identifying Valuable Resources, European Management Journal, Volume 25, Issue 4,2007,Pages 320-329,ISSN 0263-2373 [↑](#endnote-ref-6)
14. J.B. Barney, Journal of Management, 17 (1991), pp. 99-120 [↑](#endnote-ref-7)
15. Reenskaug, Trygve. [THING-MODEL-VIEW-EDITOR: an Example from a planningsystem](http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf), 12 MAY 1979 [↑](#endnote-ref-8)
16. A modern front-end programming framework, which combines and compiles JS, CSS, and HTML to display the views on the browser. [↑](#footnote-ref-6)
17. Sullivan, Danny. ["Google: 100 Billion Searches Per Month, Search To Integrate Gmail, Launching Enhanced Search App For iOS."](http://searchengineland.com/google-search-press-129925) Search Engine Land. August 8, 2012 [↑](#endnote-ref-9)
18. Natasha Singer. How Google Took Over the Classroom. The New York Times, May 13, 2017 [↑](#endnote-ref-10)
19. Collier, Ken W. (2011). Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing. Pearson Education. pp. 121 ff. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)" \o "ISBN (identifier)) [9780321669544](https://en.wikipedia.org/wiki/Special:BookSources/9780321669544" \o "Special:BookSources/9780321669544). [↑](#endnote-ref-11)
20. Anonymous, ["What is Agile Software Development?"](http://www.agilealliance.org/the-alliance/what-is-agile/). Agile Alliance. 8 June 2013. Retrieved 4 April 2015. [↑](#endnote-ref-12)
21. ["Globally Distributed Content Delivery, by J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman and B. Weihl, IEEE Internet Computing, Volume 6, Issue 5, November 2002"](https://people.cs.umass.edu/~ramesh/Site/PUBLICATIONS_files/DMPPSW02.pdf)(PDF). [Archived](https://web.archive.org/web/20170809231307/http://people.cs.umass.edu/~ramesh/Site/PUBLICATIONS_files/DMPPSW02.pdf) (PDF) from the original on 2017-08-09. Retrieved 2019-10-25. [↑](#endnote-ref-13)
22. 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. [↑](#footnote-ref-7)
23. OMG™ Unified Modeling Language™ (OMG UML®) specifications, Kirill Fakhroutdinov 2007-2016 [↑](#endnote-ref-14)
24. Burbeck, Steve (1992) [Applications Programming in Smalltalk-80:How to use Model–View–Controller (MVC)](https://web.archive.org/web/20120729161926/http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html) [↑](#endnote-ref-15)
25. Loosely Coupled: The Missing Pieces of Web Services by [Doug Kaye](https://en.wikipedia.org/w/index.php?title=Doug_Kaye&action=edit&redlink=1" \o "Doug Kaye (page does not exist)) [↑](#endnote-ref-16)
26. [Codd, E. F.](https://en.wikipedia.org/wiki/Edgar_F._Codd" \o "Edgar F. Codd) (1970). "A Relational Model of Data for Large Shared Data Banks". [Communications of the ACM](https://en.wikipedia.org/wiki/Communications_of_the_ACM" \o "Communications of the ACM). 13 (6): 377–387. [doi](https://en.wikipedia.org/wiki/Doi_(identifier)" \o "Doi (identifier)):[10.1145/362384.362685](https://doi.org/10.1145/362384.362685). [↑](#endnote-ref-17)
27. 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, [↑](#footnote-ref-8)
28. 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. [↑](#endnote-ref-18)
29. MongoDB is an example here, any non-relational database which can store simple “key-value” like Mongo can be used in the second layer. [↑](#footnote-ref-9)
30. 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 default used in Mongo (a kind of non-relational database) [↑](#footnote-ref-10)
31. Although most relational databases also support index like B-tree or Hash, but to match the sentences here by using LIKE, index is not supported well any more. [↑](#footnote-ref-11)
32. Primary key in relational databases, for example, MySQL will use B-tree or Hash index to enhance the searching speed. [↑](#footnote-ref-12)
33. [RAM, virtual memory, pagefile, and memory management in Windows](http://support.microsoft.com/kb/2160852/en-us). Microsoft. [2012-11-26]. [↑](#endnote-ref-19)
34. https://redis.io/topics/data-types [↑](#footnote-ref-13)
35. Donghee Lee; Jongmoo Choi; Jong-Hun Kim; Noh, S.H.; Sang Lyul Min; Yookun Cho; Chong Sang Kim. [LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies](http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=970573&isnumber=20937). IEEE Transactions on Computers [↑](#endnote-ref-20)
36. Patton, Ron (2005). [Software Testing](https://archive.org/details/softwaretesting0000patt) (2nd ed.). Indianapolis: Sams Publishing. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)" \o "ISBN (identifier)) [978-0672327988](https://en.wikipedia.org/wiki/Special:BookSources/978-0672327988" \o "Special:BookSources/978-0672327988). [↑](#endnote-ref-21)
37. Limaye, M.G. (2009). [Software Testing](https://books.google.com/books?id=zUm8My7SiakC&pg=PA108). Tata McGraw-Hill Education. pp. 108–11. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)" \o "ISBN (identifier)) [9780070139909](https://en.wikipedia.org/wiki/Special:BookSources/9780070139909" \o "Special:BookSources/9780070139909). [↑](#endnote-ref-22)
38. Saleh, K.A. (2009). [Software Engineering](https://books.google.com/books?id=N69KPjBEWygC&pg=PA224). J. Ross Publishing. pp. 224–41. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)" \o "ISBN (identifier)) [9781932159943](https://en.wikipedia.org/wiki/Special:BookSources/9781932159943" \o "Special:BookSources/9781932159943). [↑](#endnote-ref-23)