MiniGoogle

Keren Ye, Xinyue Huang

1. Introduction

In this project, we have designed and implemented a basic data-intensive application to index and search large documents. More specifically, we have designed a simple search engine, referred to as **MiniGoogle**, to retrieve documents relevant to simple search queries submitted by users.

The MiniGoogle includes three basic components: 1) UIShell, a simple user interface, is responsible for submitting Index task and initiating query and retrieve requests; 2) Master Node, served as both task coordinator and normal server, is responsible for assigning tasks to worker nodes and responding query and retrieve requests; 3) Worker Node, a multi-threaded process, can simultaneously process "Word Count" for given documents meanwhile respond to retrieve requests.

In the following chapters, we will first introduce our main components in more details. Then, a comprehensive description of the three functionalities - index, query, and retrieve - will be presented. Our designs of considering the single node failure of Master Node will be described in the next chapter. Finally, a conclusion of our MiniGoogle projects will be given.

2. Main Components

There are mainly three components in our MiniGoogle system, including 1) one or more **UIShell** processes submitting and requesting the Master Nodes concurrently; 2) a **Master Node** along with its **backup processes**; 3) **Worker nodes** responsible for concurrently computing "Word Count".

2.1 UIShell

UIShell is a command line process used for 1) submitting indexing tasks to master, and 2) initiating query and retrieve requests.

When an indexing task is submitted via UIShell, the UIShell would first traverse among the specific directory recursively and put each of the individual files into a sending queue. Multithread is applied for sending the files simultaneously.

When a query or retrieve task is submitted via UIShell, the UIShell would parse the parameters and send them to Master through a network packet. After receiving the reply from master, the UIShell parsed the packet and output the information in a human-readable fashion.

Fault-tolerance is an important consideration in the designs of UIShell. The UIShell processes request for available backup processes (refer to as the Slave Nodes) of the original Master periodically. For each of the query and retrieve requests, they choose among all of the available servers randomly.

2.2 Master

The Master node handles requests from both UIShells and Workers. It serves as a coordinator in the system.

When an indexing task is submitted to the Master via "task_put" interface, the master simply appends the indexing task into a FIFO queue. Available workers check the task queue periodically via a "task_poll" interface, popping up a task if applicable. After finishing computing the "word count", a worker sends result back to the master via the "report" interface, along with the ip and port indicating its location. Upon receiving the report packet, the master can update the inverted index for the specific document.

Processes like query and retrieve are much simpler. Upon receiving a query request, the master just splits the query into keywords, recalls document ids using its own knowledge represented in the memory, then reply the requesting UIShell. For retrieve request, extra communication between Master and Worker must be done because the actual files are stored in Worker nodes.

The Slaves we referred is functionally the same as the Master. It appears to be a master except it does not reply to requests from workers. The Master exposes an interface called "backup". When slaves launch, they copy data from Master via this interface in order to synchronize the memory data. After launching, the slaves register to the master, telling master to notify them updated indexing information. It means the master would send updated information to all of the registered slaves upon receiving a "report" packet.

The interfaces provided by the master nodes are described as follows:

Requester	**Interface**	**Description**
Slaves	register	It provides the interface which a slave node can register/unregister the ip and port to the master node.
	backup	It provides the interface which a slave node copies the data from.
UIShells	query_slaves	It provides the interface which the uishell asks for available slaves when the master node crashes.
	task_put	It provides the interface which the uishell puts all of the html files into the task queue of the master for indexing.
	query	It provides the interface which the uishell asks for the file ids which contain the query keywords.
	retrieve	It provides the interface which the uishell asks for the file content given specific file id.
Worker	task_poll	It provides the interface which a worker removes and processes the first

	task from the task queue of the master.
report	It provides the interface which a worker reports the word count result to the master node.

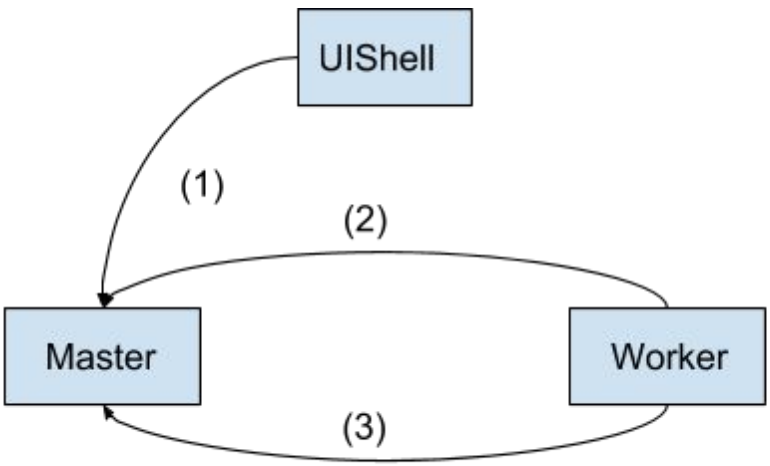
2.3 Worker

Worker node serves as both client and server in the system. One one hand, worker serves as client because it sends “poll_task” requests to the master periodically asking about the availability of tasks. On the other hand, worker serves as servers because it waits for retrieve requests from master in a passive manner, and then reply the contents of files reside in the local machine.

3. Implementation

3.1 Index a File

The indexing functional graph is show as follows:



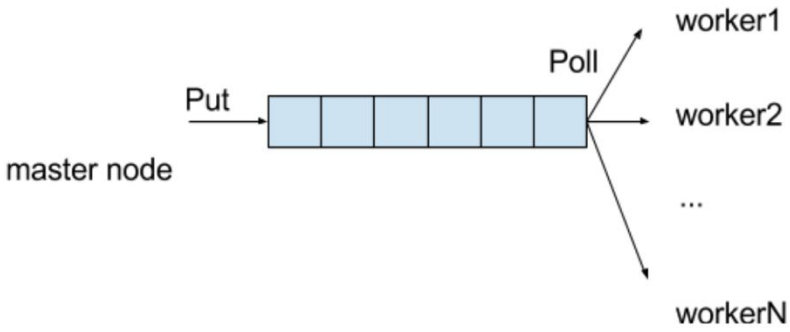
Interactions between uiShell, master node and worker nodes for indexing.

(1) UIShell sends index request to the master node with a file. Then master node gives a file id to the file and stores the file into the task queue.
(2) A worker node will fetch a file at one time from the task queue of the master node and implement the word count calculation on the file.
(3) After operations on the file, a worker node will send a report request to the master node and master node will update lookup table and invert index table.

For indexing a file, a client first submits one file to the master node. Upon receiving the file, the master node creates a task for handling the file and appends the task to the end of the task queue.

The **task queue** in master node is a FIFO queue. Each time when “task_put” request is received, the master node append the task to the end of the queue. Each time when “task_poll” request is received, the

master node remove the first task from the queue and return the first task to the requesting slave node. If the task queue is empty, “task_poll” gets failure immediately without blocking the slave node.

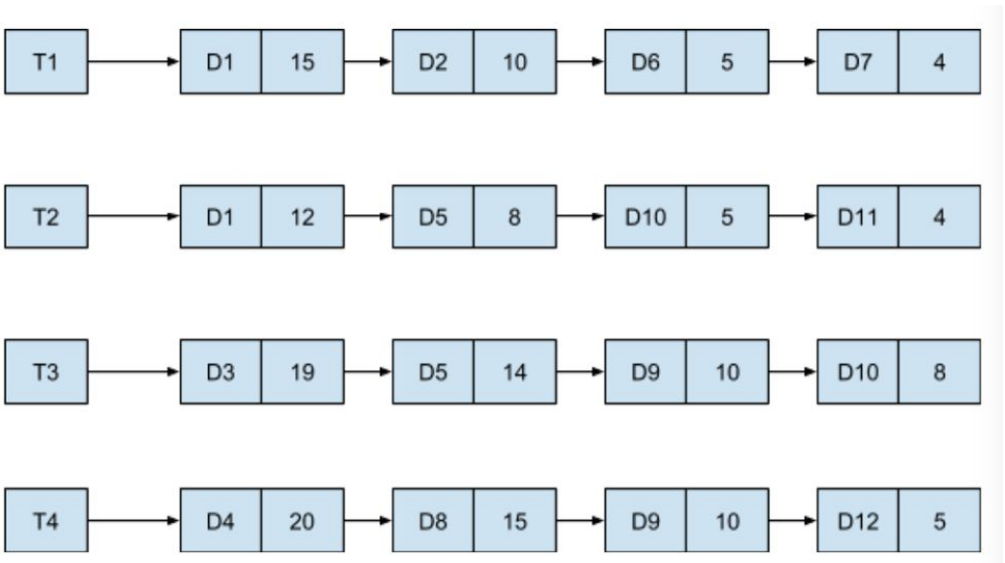


For implementing the two functionalities of retrieval and query, data structures for both index and inverted index are maintained in the master node.

The **index** is a hash-like data structure, in which each entry stores the actual location (ip and port) of the slave node for a specific document specified by its identifier. We should notice that the actual files reside on worker nodes, thus the master node only maintains the information regarding “where does the file reside”.

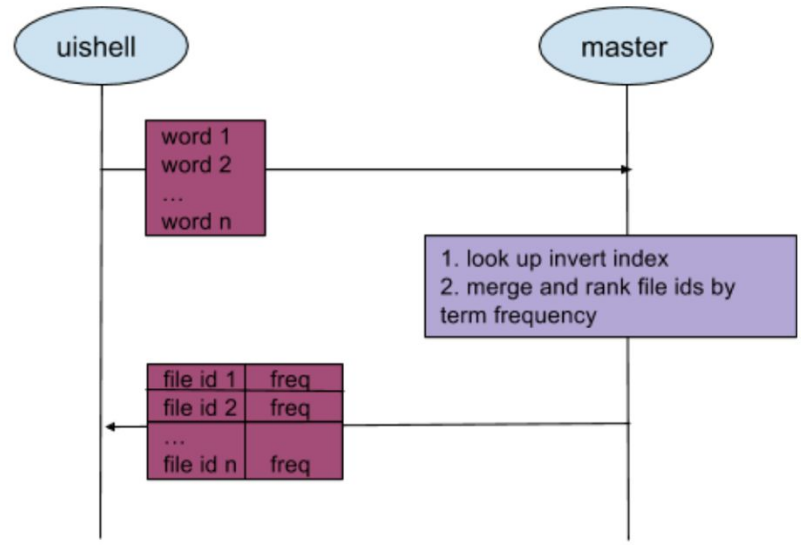
KEY	VALUE
doc_id1	slave1 addr
doc_id2	slave2 addr
...	...

The **inverted index** is also a hash-like data structure. However, the entry for a specific key, the term, is pointed to a sorted list sorted by the term frequency of corresponding document.



3.2 Query Keywords

When the master node accepts the query request, it looks up the inverted index to find a set of document ids containing the keyword. Then the document ids will be filtered by judging whether a file contains all of the words in the query. The returned files will be sorted by the word occurrence frequency. If multiple words occur in a file, the frequency for this file will be based on the minimum frequency of query word occurred in the file. The design schema is shown in the following graph:



Interaction between uishell and master in query request

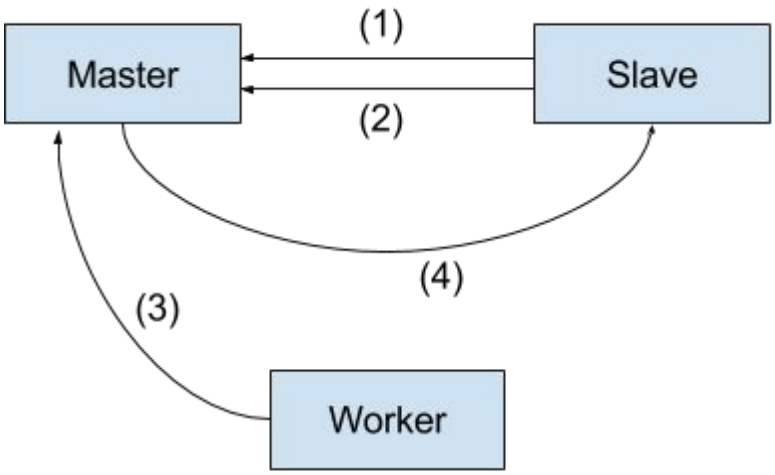
3.3 Retrieve a File

When the master node receives a retrieve request from the UIShell, it firstly looks up the index to find where the resource resides. If the location is not recorded in its lookup table, it means the file is not already indexed, thus master would return a failure in this situation. Otherwise, if the location is recorded, the master node would then initiate a request to the worker node who has the resource. After receiving the reply from worker node, the master can now reply to the UIShell and return the file content.

4. Design of backup

To keep our system running in a consistent way when the master node crashes, we design a backup mechanism. This mechanism ensures that the system can provide consistent service in the situation of single node failure of master node. Also, this mechanism helps to improve the scalability because it alleviates the “bottleneck” problem by applying multiple replicas of master nodes.

In our solution, some specific servers (referred as slave nodes) can be registered to the master node as alternative nodes. Slave nodes copy data from the master node at the very beginning, and they keep update with master as updates happen in the original master. The UIShells selects among all alive master and slaves, choose one to initiate query and retrieve requests. The mechanism can be presented using the following graph.



Interactions between Master, Slave, Worker for backup

(1) Slave node registers to master, and keeps heartbeat with master node
(2) Slave requires master to duplicate its memory to network packet. Upon receiving this packet, slave reconstructs data structures that is the same as master node
(3) The master node receives a report packet from worker
(4) The master node duplicates the report packet sent by worker, and sends the duplicated packet to each of the slave nodes registered

4.1 Master-backup

To implement the mechanism of backup, these four functionalities are required in master:

1. Register - The master node accepts register requests from slave nodes, it also accepts heartbeat and unregister requests from slave nodes.
    a. it automatically removes a slave node from its list if this slave node expires to keep heartbeat.
    b. it removes a slave node from its list if an unregister request is received.
2. Report - Upon receiving a report request, the master will dispatch the report request to each of its slave nodes registered.
3. Backup - Upon receiving a backup request, the master will dump its memory to network packets, including the lookup table and the inverted index table.
4. Query for slaves - Upon receiving a "query-slaves" request, the master returns the list of registered slave nodes.

4.2 Slave-backup

To implement the mechanism of backup, these three functionalities are required in slave:

1. When one slave node is about to start, it sends a "backup" request to the master for building its own lookup table and invert index. Then, a "register" request is also sent.
2. Slave node sends "heartbeat" periodically to the master node.

3. When a slave node is about to quit, it sends an “unregister” request to the master node.

4.3 UIShell

To implement the mechanism of backup, the UIShell is responsible to keep track of status of both the master node and the slave nodes.

1. UiShell will cache the list of slave nodes periodically by sending “query-slaves”request to the master node.
2. When UIShell sends a request to the master node and detects the master node is died, the UIShell will automatically forward the request to one of the slave nodes.

Conclusion

In this project, we have implemented three functionalities for MiniGoogle, including index, query and retrieve. In short, index is responsible for producing an index for a file and calculating the occurrence of the words. Query is responsible for returning the ids of files which contain the query keywords. Retrieve is responsible for fetching file contents from worker nodes. Besides this, we achieve some extra works including:

1. Slave nodes can backup from the master and serve as alternative masters in case of single node crashing.
2. Replica helps to improve the scalability of the system because work is divided among master and slave nodes.

User Manual

Launch Master

./mini_google_master -i 127.0.0.1 -p 8000

1. IP: '-i 127.0.0.1' denotes the ip address of the local machine.
2. Port: '-p 8000' denotes the port number that the master server is listening.

Launch Worker

./mini_google_slave -i 127.0.0.1 -p 8001 --master-ip 127.0.0.1 --master-port 8000

1. IP: '-i 127.0.0.1' denotes the ip address of the local machine.
2. Port: '-p 8001' denotes the port number that the worker server is listening.
3. Master IP: '--master-ip 127.0.0.1' denotes the ip address of the master node, after computing 'wordcount', the worker reports result to this address.
4. Master Port: '--master-port 8000' denotes the port that the master is listening.

Submit Indexing Task

./uishell -i 127.0.0.1 -p 8000 -n -d webdata/

1. Master IP: '-i 127.0.0.1' denotes the ip address of the master node.
2. Master Port: '-port 8000' denotes the port that the master is listening, the uishell will send file through <master ip, master port>.
3. Indexing Task: '-n' denotes that the uishell is running under indexing mode, which means it will exit after uploading all the files in the local directory.
4. Directory: '-d webdata' denotes that the uishell will upload all the files in this directory recursively.

Query and Retrieve

./uishell -i 127.0.0.1 -p 8000

1. Master IP: '-i 127.0.0.1' denotes the ip address of the master node.
2. Master Port: '-port 8000' denotes the port that the master is listening, the uishell will send file through <master ip, master port>.

Using this command will guide users into the command line mode. In the command line mode, users can use commands such as "query keren ye xinyue huang" to submit query request; or use commands such as "retrieve 036e8cb6f36d474aa70d2d4cf0d1e07b" to submit retrieve request.

Launch Slave

./mini_google_master -i 127.0.0.1 -p 8003 --master-ip 127.0.0.1 --master-port 8000

1. IP: ‘-i 127.0.0.1’ denotes the ip address of the local machine.
2. Port: ‘-p 8003’ denotes the port number that the slave server is listening.
3. Master IP: ‘--master-ip 127.0.0.1’ denotes the ip address of the master node, slave node copies data from master at the beginning.
4. Master Port: ‘--master-port 8000’ denotes the port that the master is listening.