SNS Project Document

Writer: 史磊
Student ID: 5120379047

1.Abstract

This paper is an introduction to my project ‘FollowMe’, which is a social network system designed for massive data and frequent usage. I’ll list its functions , analyse its architecture and test its performance.

2.Functions

I implemented basic functions demanded. For I focused on the optimization of database and designing of the project , you can’t see that many fancy functions in the client end.

You can:

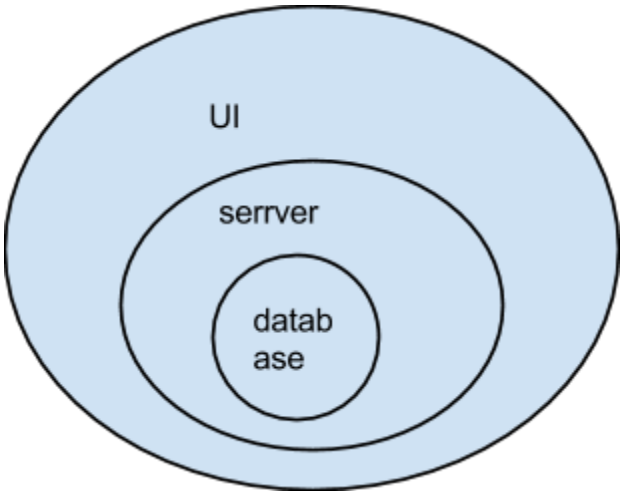
- sign up
- sign in
- sign out
- quit the system
- modify personal information (including password)
- publish news
- see news published or shared by your friends
- share your friend’s news
- search for friends
- follow another person
- see your friend list

There will be instroductions everywhere to tell you what to do. Just follow them and make choice by inputting the numbers between a pair of square brackets. It’s easy to master.

3.Architecture

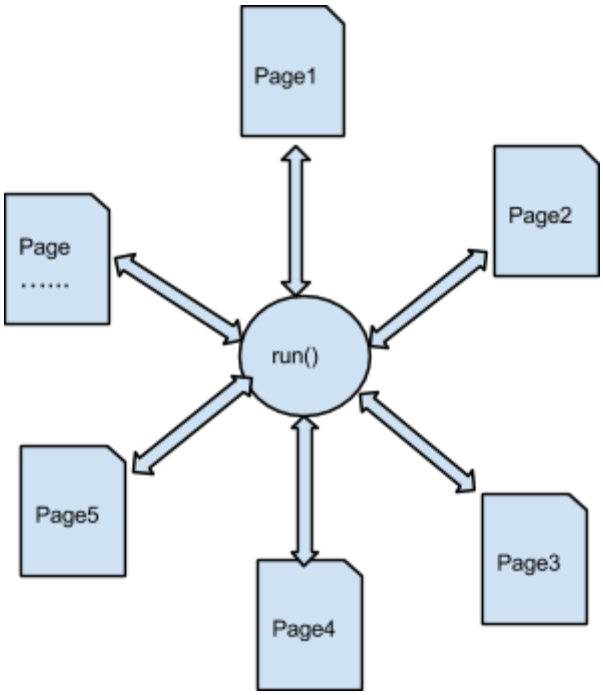
0.Overview

Similar to a simple web app , his project consists of 3 parts: UI , Server , Database.



UI interact with user and call server. Server call database and offer API to UI. Database manage data in disk and memory and offer API to server. Lower layers are free from upper layers , yet they are specially designed for the same purpose.

1.UI

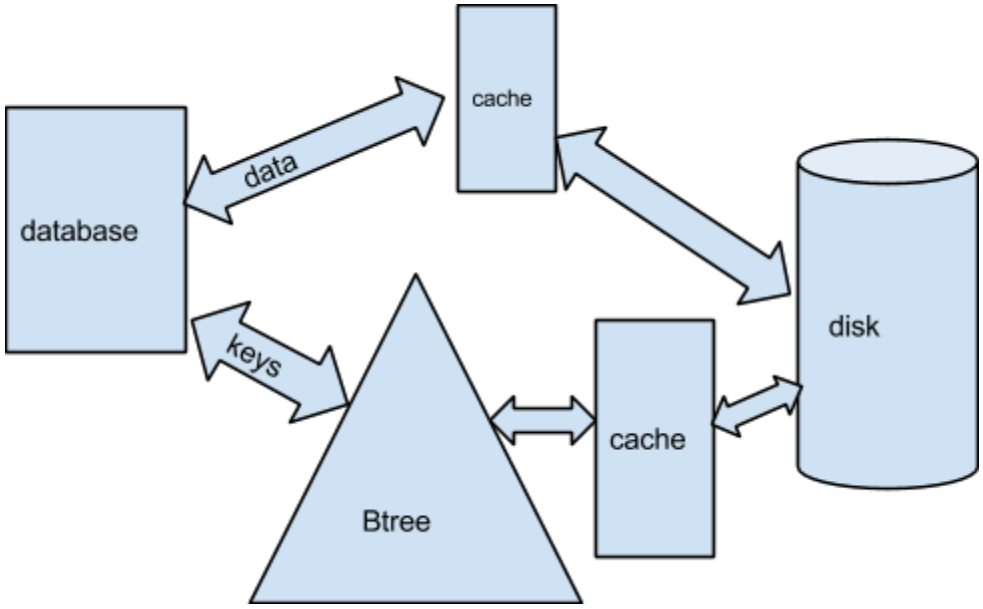


UI has many pages. Each page is a function and is corresponded to a number. Interactions with users and the server are done within pages. A page returns the number intending the next page. The function run() manage pages. It continuously calls pages according to the return value of previous pages.

2.Server

Server is quite simple. It offers many functions , each dealing with a requirement , manageing data using database and return required infomation.

3.Database



Database stores data and their keys. Each data has an unique index number (well , actually is the order they are added) and some keys.

For simplicity and generality , it is required that data and keys must be transformed to string before being stored to database.

Key-index pairs are stored in B-trees to improve searching speed. Both node reading in B-tree and data reading in database access the disk through cache , of course , to improve speed.

The elimination rule for them are different.

B-tree cache: Eliminate the least recent touched node.

Data cache: Each table has a central key (in a SNS , it could be set to user). I believe that data of the most active users will be accessed more often. So I rank data in cache by central key and eliminate those with least active central key. This concerns a heap.

Both caches are stored in a hash.

There's a special B-tree where you can find position of a data entry with its index.

4.Performance

This part tests the performance of database.

1.Correctness

With 1,000,000 random data add , 500,000 delete and 500,000 get , no error occurs.

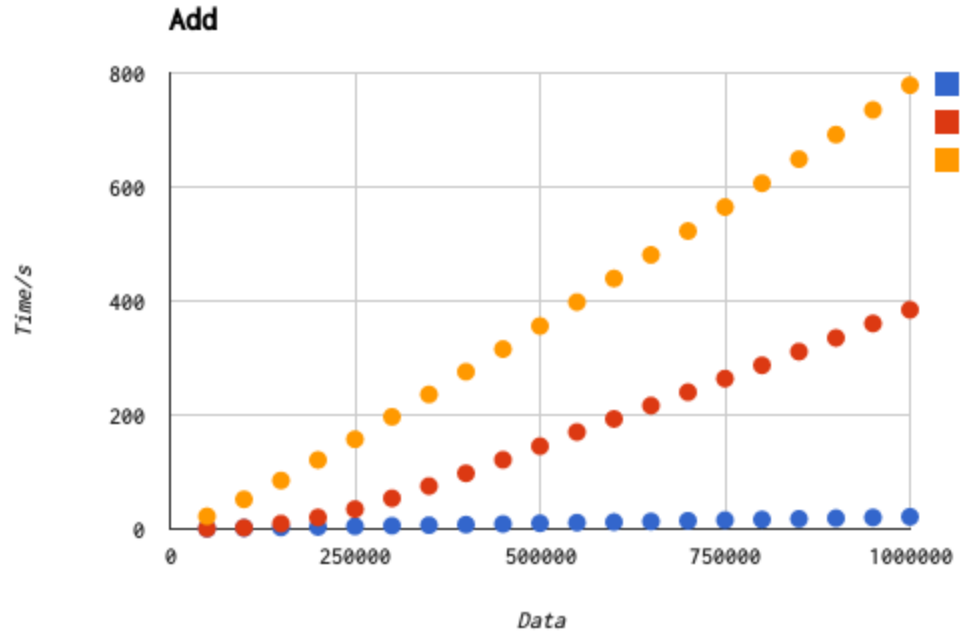
2.Time complexity

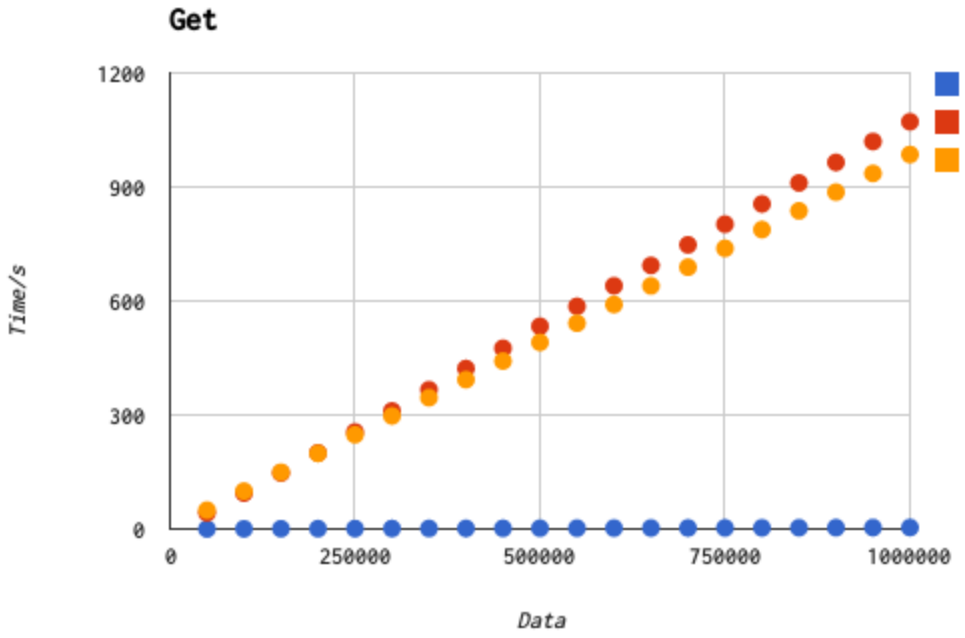
Searching , inserting , deleting and modifying actions concerning data are direct , so they have a complexity of O(1). These actions concerning keys are through B-tree , so they have a complexity of O(logN) where N is the number of data. However, there is a huge constant here due to disk I/O. So the real time spent depends on cache size and miss rate. I think in a real SNS, miss rate is smaller than random data, but I'm unable to test that :-(.

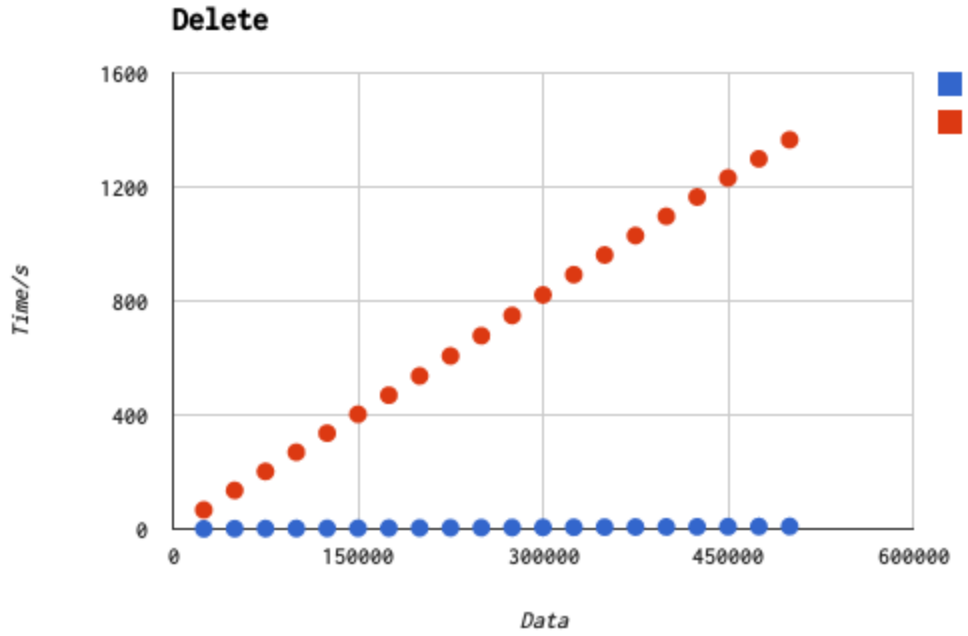
3.Comparison

Charts below show a comparison between database with different cache size.
Blue node: 2,000,000 cache entry (Big cache 600MB memory cost)
Red node: 100,000 cache entry (Small cache 180MB memory cost)
Yellow node: 1,000 cache entry (Almost no cache 150MB memory cost)

(memory cost above may include overhead used to store testing data)


Add
800
600
400
200
0
Time/s
0
250000
500000
750000
1000000
Data


Get
1200
900
600
300
0
Time/s
0
250000
500000
750000
1000000
Data



(Yellow nodes are missing in this chart because it really costs too long time)

4.Conclusion

This database can run correctly and fast enough given enough cache. Cache makes a big different in the speed of database.