Name: Pravin Dhuri
NUID: 002138314

Assignment 1 (Part 2)

Application of NoSQL Database in Web Crawling

Introduction

Web crawling is the process of collecting, filtering, and storing information from the internet in order to build a search engine. As a result, the database needed for this must have a huge storage capacity and a cheap hardware cost. Scalability, performance, and availability are the primary goals. When dealing with huge volumes of data, traditional relational databases store data in 2D tables, which is inefficient. Many firms utilize their own databases, such as Bigtable from Google. Furthermore, the paper describes about the comparison between the relational i.e., SQL and non-relational i.e. No-SQL by incorporating them into a solution for a specific meteorological BBS data collection system.

Web Crawling System

A web crawler is a process that searches the internet for material, examines it, and stores it in databases. It works by widening the connective interaction between webpages to the full page. It makes use of a spider to scan web pages, this process usually takes some time so that multiple spiders can process simultaneously to improve the crawl rate and save URLs in URL databases, a controller to operate the spider by deciding which URLs to read from URL databases, and a page library to record page data.

Meteorological BBS Information Collection System

Meteorological BBS information collecting system filters and gathers postings from typical meteorological BBSs on the Internet, such as clud.weather.com.cn, www.cmabbs.com, and others. It gives a professional meteorological search engine database. The postings are recorded in text files, which are subsequently preserved in databases. The URL and title of the post appear first, followed by "#floor," which indicates the beginning of the first level, "postby," which indicates the person who posted this floor, and the time and content of this floor. "#floor postby time content" will be repeated if there are additional floors. The post's structure includes a set URL and title, as well as unfixed floors. Because each post has a varied number of storeys, their constructions change. The database structure design must be both practical for storing material from txt documents and simple to query. The system retrieves all the post's content from the database based on the ID, including all floors.

Relational Database Solution

The Cmabbs Meteorological BBS information collection system has 237,746 floors, and each BBS has more than ten thousand posts. The amount of data is huge and grows rapidly. Traditional databases store data in the form of a two-dimensional table with strictly row and column format. In the case of large amounts of data, the database must still ensure the good performance of query. For meteorological BBS, they have used a relational database to store the data on the posts and floors. Table tb_post stores the fixed information of post, including URL and title, as well as the title and number of occurrences. Each floor is stored as a record with a foreign key, called a postID, and each occurrence is queried by its post ID.

NoSQL Database

NoSQL, as Not Only SQL, is the broad definition of non-relational data storage. The first edition of Berkeley DB was published in 1991. NoSQL databases have huge storage, high performance and low cost.

Name: Pravin Dhuri
NUID: 002138314

MongoDB Solution

Document-oriented, documents can be easily converted to computer language data types, embedded documents and arrays reduce the need for joins, schema-free for easy schema evolution (Automatic sharding with order-preserving partitioning, which improves distributed reading and writing speed, easy horizontal scaling with low hardware costs, no joins and no transactions, which makes distributed query simple and quick). (MongoDB provides conditional operators, regular expressions, queries on embedded documents and arrays, and can replace most SQL queries; more complex aggregation operations may be accomplished using utility functions like as map/reduce.) According to official documentation, when the amount of data is greater than 50GB, MongoDB access performance is ten times quicker than MySql.MongoDB was designed to manage massive amounts of data storage.

Conclusion

Web crawling is one of the most essential Internet applications, and the database storage option chosen has a direct impact on search engine speed. The purpose of this paper is to explore a relational database and NoSQL database MongoDB solution for a meteorological BBS information gathering system. The data storage structure and query are constructed, and the advantages and disadvantages are explored in the data structure, query, and scalability sections. A relational database has numerous tables with foreign keys, has a quick decrease in query speed when dealing with large amounts of data, and has limited vertical scalability at a high cost. In comparison to relational databases, MongoDB allows schema-free queries, has high query speed with large data volumes, and offers simple horizontal scaling at a low hardware cost.

Name: Pravin Dhuri
NUID: 002138314

Comparing NoSQL MongoDB to an SQL DB

When working with a small amount of structural data, this article presents SQL (SQL Server) and NoSQL (MongoDB) databases comparison. By now know that relational databases perform much better with small amounts of structured data, and NoSQL databases function better with large unstructured data. There has been no comparison obtainable on the operation of NoSQL databases on small amounts of structured data, which is what this paper attempt to discover.

Introduction

Relational databases very often store data in 2D tables which are structured. Primary and foreign keys are used to store the data in different tables. Join queries are used to retrieve data from multiple tables. The number of tables used has a direct relationship with the time taken to retrieve data. Non-relational databases, lack schema and structure in their data. In a key value pair, all of the data can be stored in one file. The querying is straightforward but designing it can be challenging. On the basis of the following three factors, this paper compares the performance of relational and non-relational databases on a small amount of structured data:
insert speed, update speed, and select operation speed.

Related Work

There are a lot of papers that compare different NoSQL databases, but almost none that compare NoSQL to SQL databases. MongoDB stores data as key-value pairs. In the case of one-to-one or one-to-many relationships, the user can store all the values in the same file. In MongoDB, the user has the option of saving all files in one collection or making different collections and saving only references to one in another. Non-relational databases, on the other hand, do not have predefined queries like relational databases. As a result, when retrieving data from various collections, users must create their own queries.

As an example: MapReduce is a method in which the user provides specific instructions to the bot during the Map and Reduce phases.

Experiments

To run the experiment, three tables were created: User, Department, and Project. The department may have numerous projects, and each project may have numerous users. A manager who is also a user can supervise a large number of users. A department may have numerous managers and projects. On both the MongoDB and SQL servers, 100 queries of various types were executed.

Result

Insert: The insert time is practically same for both MongoDB and SQL
Update: MongoDB take some time to update while dealing with non-indexed fields. Then again when the updates were made on indexed fields or primary keys then MongoDB performed better compared to SQL. It could be this performance gap, because of MongoDB having a prefabricated index on the primary key of the file which is quicker than SQL Server's primary key clustered file.
Select: MongoDB fared better with straightforward select inquiries but when the information from many tables were utilized and the queries became perplexing the performance of MongoDB disintegrated drastically as the amount of data increased.

Conclusion and Future Works

MongoDB is better for inserting, updating, and running simple queries. SQL performs better when updating, querying non-key attributes, and running aggregate queries.
MongoDB is ideal for handling large amounts of data. In the future, the authors hope to run MongoDB and SQL as a distributed database.

Name: Pravin Dhuri
NUID: 002138314

Data Aggregation System

Abstract

Creating an information retrieval system using relational and non-relational data for the Compact Muon Solenoid experiment at CERN's Large Hadron Collider. The experiment presently has numerous naturally developed sources of data. DAS creates a single platform for querying all of these services. It also contains a caching surface to expedite and merge data using primary key.

Introduction

CMS began gathering data in earnest in 2010, with around 42pb1 of proton-proton collision data collected at the time of writing. At full data-taking speeds, this is expected to create around 5PB of raw and reconstructed data each year. Every year, this is accompanied by around 1TB of information, which includes data locations, dataset descriptions, and machine conditions. Each of these metadata attributes is kept in a specific system that employs multiple technologies. Each offers unique interfaces for querying the data stored. Prior to data gathering, this scenario was acceptable because most systems were used in isolation and only by specialists. Users in the digital era, on the other hand, typically need to conduct queries across many data sources.

Analytics

The DAS analytics system runs as a daemon, scheduling and executing small tasks on the DAS document storage. This can be used for normal maintenance tasks like as wiping away expired data or pruning still-valid data if space becomes limited, but its major function is to analyze DAS requests and provide information to developers, as well as to do automated cache optimization. Because the data in question often has a TTL of less than an hour, users will frequently discover that the data they seek is not in the cache and must be downloaded on demand. As a result, the analytics system makes an effort to detect the most common user queries and ensuring that they are always cached and hence available for real-time retrieval, regardless of the actual fetch latency.

Future work

DAS is currently being deployed into production for beta testing. The current development priority is to increase the range of available inquiries, both by providing new services and increasing the number of APIs supported for existing services. DBS, PhEDEx, SiteDB, Tier-0, LumiDB, RunRegistry, Dashboard, and Overview CMS services are now supported to some extent. However, the DAS mapping only reflects a subset of their APIs, and in order to be relevant to customers, they need to supply as many of the queries available through their original interfaces as feasible.