Mayank Gangrade
Fall 2019

Application of NoSQL Database in Web Crawling

This article is mainly about how one can create a web crawler using a NoSQL database and at the same time explains the difference between SQL and NoSQL database designs with respect to the web crawler.

First, it discusses relational database and table design for the relational database model and after that discuss NoSQL databases and different type of databases developed by different companies. Then, it tells how one can design the same using Mongo DB and described Mongo DB advantages like it is convenient, high performance, high availability, scalability and querying database.

After it discusses their own design and implementation of Mongo DB and their database model for a crawler. After it compares the solution between Relational and NoSQL database on Data Structure, Query, Time and scalability.

At last, it concludes that database storage affects the performance of the web engine which uses a web crawler. A relational database has multiple tables' storage with a foreign key, a sharp decline in query performance with huge amount of data, and vertical scalability with high cost. Compared to the relational database, MongoDB supports schema-free, has great query performance with huge amount of data and provides easy horizontal scalability with low cost of hardware. It is more suitable for data storage in Web crawling.

Comparing NoSQL MongoDB to an SQL

This article is about comparing performance between Mongo NoSQL and Relational SQL Server database and Results show that MongoDB performs equally as well or better than the relational database, except when aggregate functions are utilized.

This paper compares databases on the basis of Insert Speed, Update Speed, and Select Operation Speed.

In conclusion, when comparing SQL to MongoDB, MongoDB has better runtime performance for inserts, updates and simple queries. SQL performed better when updating and querying non-key attributes, as well as for aggregate queries.

MongoDB could be a good solution for larger data sets in which the schema is constantly changing or in the case that queries performed will be less complex. Since MongoDB has no true schema defined and SQL requires a rigid schema definition, MongoDB would easily handle a dynamic schema such as a document management system with several dynamic fields and only a few well known searchable fields. In conclusion, MongoDB is definitely the choice for users who need a less rigid database structure. MongoDB could be a good solution for larger data sets in which the schema is constantly changing or in the case that queries performed will be less complex.

For those users that have a strict schema defined and a modest amount of structured data we also found MongoDB to perform better than SQL in general. However, there were downsides to MongoDB, such as its poor performance for aggregate functions and querying based on non-key values. Also, MongoDB required additional effort in implementation compared to SQL and required decisions that affected its performance. Lastly, SQL is the industry standard and much more widely supported over MongoDB.

Mayank Gangrade
Fall 2019

Data Aggregation System

This paper is about Data Aggregation System, a system for information retrieval and aggregation from heterogeneous sources of relational and non-relational data.

In 2010, data taking at the CMS experiment began in earnest, with approximately 42pb−1 of proton-proton collision data recorded at the time of writing. At full data-taking rates, this is expected to produce around 5PB of raw and reconstructed data per year. Associated with this is around 1TB per year of metadata, such as data locations, dataset descriptions, and machine conditions.

The architecture of DAS was designed as a series of independent components, which can scale from all running on a single node to multiple nodes and duplicate components. The web server handles user sessions, whether they originate from a browser or automated scripts. Queries made here are passed onto the cache server for processing, during which time the webserver periodically polls for the current status and displays it to the user. The cache server consists of a pool of worker threads that handle the DAS queries received from the web front-end. Each query is handled entirely by a single thread. The cache consists of one or more MongoDB shards.

DAS queries are made in a custom text-based language. Work originally centered on an extended version of the query language. Almost any source of information can be incorporated in DAS, such as an SQL database cursor, a command called in a subshell or an HTML page scraper, but in practice, most data services used by DAS are APIs accessed over HTTP and returns data in JSON or XML format. Data in DAS is principally stored in two separate collections; the raw and merged caches.

The DAS analytics system is a daemon that schedules and executes small tasks that access the DAS document store. This can be used to perform prosaic maintenance tasks, such as cleaning out expired data or pruning still-valid data if space becomes limited, but the main function is to analyze the queries received by DAS to provide information to developers and to perform automated cache optimization.