분산 캐시를 적용한 실시간 검색 시스템
(Real-Time Search System using Distributed Cache)

임 건 길 [†]　　　이 재 기 [††]
(Jian-Ji Ren)　　(Jae-Kee Lee)

요 약 최근 주요 검색 엔진들의 인덱스들이 굉장한 비율로 성장하는 것처럼, 수직형 검색 서비스들은 사용자가 원하는 것을 찾을 수 있도록 지원할 수 있다. 실시간 검색은 주어진 주제를 쉽게 찾아낼 수 있기 때문에 매우 유용하다. 본 논문에서는 고성능 실시간 검색 시스템을 구현하기 위한 새로운 구조를 설계하였다. 전체적인 시스템은 실시간 검색의 특징을 기반으로, 크게 수집 시스템과 검색 시스템의 두 부분으로 나뉘어진다. 본 논문의 평가 결과에서 제안한 구조가 Replication Overhead 값을 유지하면서 실시간 검색에 있어 투명한 확장성을 제공할 수 있음을 확인하였다.

키워드 : 분산캐시, 실시간, 검색시스템

Abstract Nowadays, as the indices of the major search engines grow to a tremendous proportion, vertical search services can help customers to find what they need. Real time search is valuable because it lets you know what's happening right now on any given topic. In this paper, we designed a new architecture to implement a high performance real time search system. Based on the real time search's characters, we divided the whole system to two parts which are collection system and search system. The evaluation results showed that our design has the potential to provide the real time search transparent scalability while maintaining the replication overhead costs in check.

Key words : distributed cache, real-time, search system

1. Introduction

Traditional search approaches are still dominated by techniques developed a long time ago, for systems with hardware requirements very different from today. Real time search presents a completely different set of challenges requiring a completely different approach to tradition. The challenges are especially difficult because the scope is still enormous, the entire web and the user's expectations are for information being indexed as fast as it appears on the web, with a lag measured in seconds and (fractions of) minutes.

Modern search and its key component, indexing, have been very much influenced by historical factors such as huge difference in latency of mass storage media (hard disks) and traditional architectures. Those differences drove many decisions in distributed architectures, especially in the area of the cache based distributed architectures.

Traditional case shows that one could index a very large corpus of data with a significant lag and expense of index creation (e.g. traditional search engines), or index a small corpus of data really fast (e.g. online news), but not both. Real time search is a new case which takes the both characters for discovering what people are talking about on the internet right now.

Real time search is valuable because it lets you know what's happening right now on any given topic. Companies can use it to handle customer service. News junkies can use it to follow political events. The issue for real time search is figuring out the right balance between immediacy, popularity and relevance.

The rest of this paper is organized as follows. Section 2 talks about backgrounds, some of which has been very influential on our design. In Section 3, we present a novel architecture to implement real time search. We evaluate our system in Section 4. Section 5 concludes the paper.

· 본 논문은 2010학년도 동아대학교 학술연구비 지원에 의하여 연구되었음
· 이 논문은 제36회 추계학술발표회에서 '분산 캐시를 적용한 실시간 검색 시스템 구현'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 동아대학교 컴퓨터공학과
jimeyren@msn.com
†† 정 회 원 : 동아대학교 컴퓨터공학과 교수
jklee@dau.ac.kr

논문접수 : 2009년 12월 24일
심사완료 : 2010년 1월 28일

2. Backgrounds

2.1 Microblog

Recently, microblog became a hot topic in WWW field. The content of a microblog differs from a traditional blog in that it is typically smaller in actual size and aggregate file size. A single entry could consist of a single sentence or fragment or an image or a brief, ten second video. But, still, its purpose is similar to that of a traditional blog. User's microblog is about particular topics that can range from the simple, such as "what one is doing at a given moment," to the thematic, such as "sports cars," to business topics, such as particular products. Many microblogs[1,2] provide short commentary on a person-to-person level, share news about a company's products and services, or provide logs of the events of one's life.

2.2 Key-Value Storage System

There are some problems with traditional database, which old and very complex system, many wasted features, many steps to process the SQL query, need administration, bad performance and not scalable. In the Web Search field, Indexing is the most important part as to search engine. Collections are often so large that they cannot perform index construction efficiently on a single machine. This is particularly true of the WWW for which need large and scalable computer clusters to construct and reasonably sized web index. From long time ago many researches teams and companies discovered that the database is the main bottleneck. Building large systems on top of a traditional RDBMS data storage layer is no longer good enough. So there is another issue which was named the next generation of storage systems.

Key-Value storage system[3] is a simple data-model, just key-value pairs. Data is stored and retrieved mainly by primary key, without complex joins. Every value assigned to key. There is no complex stuff, such as: relations, ACID, or SQL quires.

2.3 Memory Based Architecture

What makes memory based architecture different from traditional architectures is that memory could be the system of record. Typically disk based architecture have been the system of record. Disk being slow we've ended up wrapping disks in complicated caching and distributed file systems to make them perform. Memory is used as all over the place as cache, but we're always supposed to pretend that cache can be invalidated at any time, the database, will step in and provide the correct values. Caching also serves a different purpose. The purpose behind cache based architectures is to minimize the data bottleneck through to disk. Memory based architectures can address the entire end-to-end application stack. Data in memory can be of higher reliability and availability than traditional architectures.

2.4 Real Time Search

Real time search means looking through material that literally is published in real time. In other words, material where there's practically no delay between composition and publishing. You take a picture and seconds later, it's posted to the world to see. You think of something, immediately tap it out on Twitter, and your tweet is shared almost as soon as you thought of it.

The entire real time start-ups search Twitter[4], and some have added in social bookmarking sites like Digg[5] and Delicious[6]. Because Twitter already has an in-house search engine through its acquisition of Surmise last year, the newer start-ups have to differentiate their products by filtering content based on relevancy and popularity. At this time, Twitter search only produces results by how recently they were published.

3. Distributed Cache Implement Real Time Search System

There are various ways to implement a search engine. The typical way uses a crawler technique. The crawler continuously scans the information, collects changes and indexes them. The indexing service is a centerpiece in the architecture and enables quick matching of keywords into search results. In real time search this technique is not applicable. The key is how fast we can put new data into the indexing server. Well, most search servers are highly optimized for fast read, however they tend to be quite heavy on write operations. How to design a real time search? There are some requirements include some issues:

- Asynchronous event-driven design: Avoid as much as possible any synchronous interaction with the data. Instead, use an event-driven approach and workflow
- Partitioning pattern: Design the data model to fit the partitioning model
- Parallel execution: Use parallel execution to get the most out of available resources. A good place to use parallel execution is the processing of users requests. Multiple instances of each service can take the requests from the messaging system and execute them in parallel. Another good place for parallel processing is using MapReduce[7] for performing aggregated requests on partitioned data.
- Replication (read-mostly): In read-mostly scenarios, database replication can help load-balance the read load by splitting the read requests among the replicated database nodes.

This is where it makes sense to put the collection and indexing server in-memory. Having an in-memory server enables both fast writes and fast searches. However, memory is limited in capacity and is not considered reliable. This is where distributed system comes to the rescue. A distributed cache system addresses the capacity and reliability of memory. Capacity is addressed by breaking the data into multiple partitions; reliability is achieved by having at least one copy of each partition available in another memory instance.

As above mentioned, what are real time search scalability challenges?

Collection (Write) -- The challenge is how to handle an ever-growing volume of publishing messaging that can lead to a viral message storm.

Search (Read) - The challenge is how to handle a large number of concurrent users that continually listen for information from users they care.

In our distributed cache system, it consists of two parts: collection system and search system. Each system has two primary layers: aggregation layer and storing layer. Aggregation layer manages the instances, replication and distribution. Storing layer consists of one or many memory-based partition key, such as a Customer ID in a CRM application or a Trade ID in a trading application. In the indexing, the pattern we'll use to perform such this task is MapReduce.

3.1 Database Architecture

The transition involved a continuous redesign and re-implementation of their initial system until it finally resembled the prototypical distributed architecture depicted in Fig. 1 (left side). The typical distributed architecture consists of a number of layers: Application logic (Ruby on Rails[8], Scala[9]), caching (Memcache[10], SQL query caching[11]) and database backend (RDBMS clusters, CouchDB [12], Google's BigTable or Amazon's Dynamo[13]) that interact through asynchronous message passing. In our distributed architecture, each layer consists of a number of machines devoted to perform their respective tasks and scaling up can be addressed by increasing the number of servers in each layer.

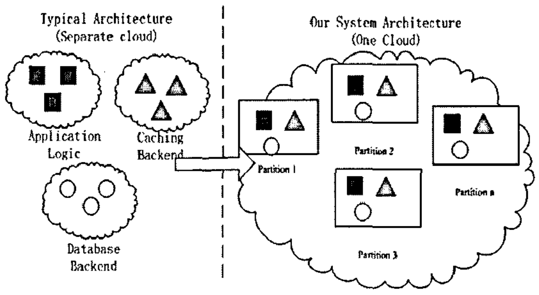

Fig. 1 Typical Architecture v.s One Cloud Architecture

3.2 Collection System Processing

Fig. 2 shows the collection system architecture. It consists of 2 parts; feed collection, information processing and storage database. Information processing and storage database both process in every machine which consists to one cloud.

We select feed-id as the partitioning key, content will be sent to a specific partition. Multiple content s may be routed to the same partition. Usually the algorithm to determine which partition fits a certain feed-id is something like:

routing-key.hash () % #of partitions

The data from the memory partitions gets replicated asynchronously into the other partitions to avoid failure.

3.3 Search System Processing

When client sends a search request to system,

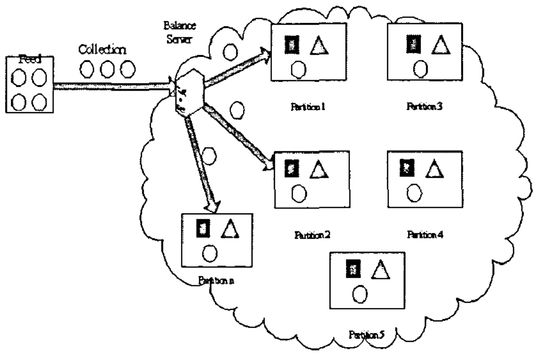

Fig. 2 Collection System Architecture

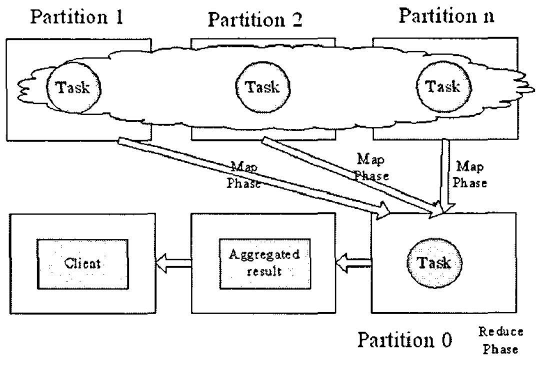

Fig. 3 Search System Architecture

aggregation layer assigns the MapReduce jobs to be executed over partitions. Because the data is stored in-memory, executing such a task is extremely fast compared with the equivalent with database and stored procedure operations. Execution is aggregated to the aggregation layer implicitly. The aggregation layer can assign Reduce jobs to aggregate the results which will be accepted by client. Fig. 3 below illustrates the processing.

4. Evaluation

For the purposes of this paper, we want to understand the trade-offs involved in using our architecture. In order to do so, we deployed our system on 15 nodes and used the Amazon EC2 trace to drive our simulation.

According to a "Social networks that matter: Twitter under the microscope [14]", their data set consisted of a total of 309,740 users, who on average posted 255 posts. Among the 309,740 users only 211,024 posted at least twice. They call them the active users. They also define the active time of an active user by the time that has elapsed between his first and last post. On average, active users were active for 206 days. We can calculate that on average a tweet is sent every 0.23 seconds. And the average the write rate of 4.4 writes/s. Unfortunately, the data set does not include information regarding search click. We choose a conservative rate of 4.4 reads/s same with write.

In order to understand the trade-offs involved in using our real-time search system, we primarily focus on read and write operations, and how these operations are affected by the number of the servers.

4.1 Replication Scheme

The traditional requirements of a single server to effectively deal with the load described earlier would be high both in specifications and cost (mainframe). But with more servers, the total cost will be lower since we could use more commodity servers or virtual machines in the Cloud. The character of distributed system is that the servers will be fault on anytime. Let us discuss the implication of the one-hop replication scheme to avoid that.

In order to gauge the overhead due to replication we define the replication ratio for feeds γ, which is the ratio between the replicated feeds and the native feeds on a given server.

Write operation: Each server need to deal with the writes of both the native feeds and their replicates. Since writes are not homogeneously distributed, each server needs to be provisioned for $(\gamma w+1)W/S$ write operations, S being the number of servers.

Read operation: After partitioning and replication, the readable feeds are local; hence the load due to the read operations is spread across the servers as N/S. The one-hop replication scheme ensures that all reads can be carried out locally, saving the network traffic.

The replication overhead results are depicted in Fig. 4. There are many differences between the read-only and read-write. We compare each other and plot the replication overhead ratio vs. the number of servers. Results are the average across the servers.

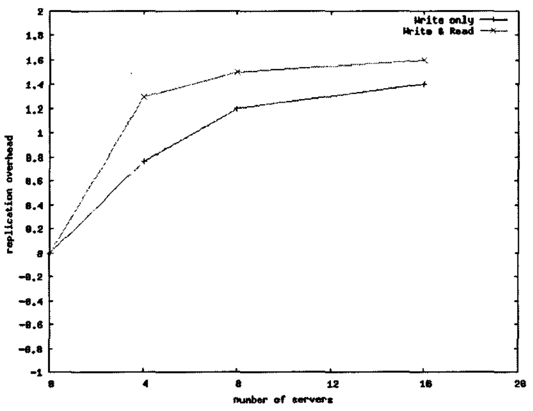

Fig. 4 Replication Overhead Results

5. Conclusions

A real time search introduces unique challenges that are quite different from a traditional database-centric search. The most profound difference is the fact that unlike with traditional sites, real time search is a heavy read/write application, and not only read-mostly. In this paper, we describe a new architecture to design the real time search system. Based the real time search's characters, we divided the whole system to two parts which are collection system and read system. The evaluation results showed that our design has the potential to provide the real time search transparent scalability while maintaining the replication overhead costs in check. We believe that, there is much scope for further research in this area and using memory-based architecture to implement cloud computing will be a very hot issue.

References

[1] http://jaiku.com/.
[2] http://friendfeed.com/.
[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, "Bigtable: A distributed storage system for structured data," In *Proceedings of OSDI'06: the 7th USENIX Symposium on Operating Systems Design and Implementation*, vol.7, pp.205-218, Nov. 2006.
[4] http://search.twitter.com/.
[5] http://digg.com/.
[6] http://delicious.com/.
[7] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol.51, no.1, pp.107-113, Jan. 2008.
[8] http://rubyonrails.org/.
[9] http://www.scala-lang.org/.
[10] http://www.danga.com/memcached/.
[11] http://devzone.zend.com/article/1258/.
[12] http://couchdb.apache.org/.
[13] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels, "Dynamo: Amazon's Highly Available Key-value Store," In *Proceedings of SOSP'07*, pp.205-220, Oct. 2007.
[14] Bernardo A. Huberman, Daniel M. Romero, and Fang Wu, "Social networks that matter: Twitter under the microscope," *Peer Reviewed journal on the Internet*, vol.14, no.1-5, Jan 2009.