JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 18, No. 5, pp. 811-820, Aug. 2017

음원 데이터베이스의 효율적 확장을 지원하는 내용 기반 음원 검색 시스템

박 지 훈[1] · 강 현 철[2*]
[1]중앙대학교 대학원 컴퓨터공학과
[2]중앙대학교 컴퓨터공학부

A Content-based Audio Retrieval System Supporting Efficient Expansion of Audio Database

Ji Hun Park[1] · Hyunchul Kang[2*]
[1]Department of of Computer Science and Engineering, Graduate School, Chung-Ang University, Seoul 156-756, Korea
[2*]School of of Computer Science and Engineering, Chung-Ang University, Seoul 156-756, Korea

[요 약]

음원 서비스의 주요 기능 중 하나인 내용 기반 검색을 위해 음원의 지문을 채취하여 데이타베이스에 저장하고 색인하여 검색에 활용하는 기법이 널리 사용되고 있다. 그런데 지속적으로 추가되는 신규 음원의 지문이 기존의 데이타베이스에 계속 삽입되면 공간 효율 및 음원 검색 성능의 저하가 점차 초래되는 문제점이 있다. 따라서 시스템 운용 비용의 증가를 가져오는 주기적인 데이타베이스 재구성 없이 효율적인 음원 데이타베이스의 확장을 지원하는 기법이 요구된다. 본 논문에서는 샤잠의 지문 채취 알고리즘을 기반으로 클러스터 컴퓨팅 환경에서 맵리듀스 및 NoSQL 데이타베이스를 사용하여 이러한 문제를 해결하는 내용 기반 음원 검색 시스템의 설계를 제시하고 실제 음원 데이터를 이용한 다양한 실험을 통해 그 성능을 평가한다.

[Abstract]

For content-based audio retrieval which is one of main functions in audio service, the techniques for extracting fingerprints from the audio source, storing and indexing them in a database are widely used. However, if the fingerprints of new audio sources are continually inserted into the database, there is a problem that space efficiency as well as audio retrieval performance are gradually deteriorated. Therefore, there is a need for techniques to support efficient expansion of audio database without periodic reorganization of the database that would increase the system operation cost. In this paper, we design a content-based audio retrieval system that solves this problem by using MapReduce and NoSQL database in a cluster computing environment based on the Shazam's fingerprinting algorithm, and evaluate its performance through a detailed set of experiments using real world audio data.

색인어 : 클러스터 컴퓨팅 환경, 내용 기반 음악 검색 시스템, 디지탈 음원, 맵리듀스, NoSQL 데이타베이스
Key word : Cluster computing environment**,** Content-based music retrieval system, Digital audio source, MapReduce, NoSQL database

http://dx.doi.org/10.9728/dcs.2017.18.5.811

Received 14 August 2017; **Revised** 27 August 2017
Accepted 31 August 2017

***Corresponding Author; Hyunchul Kang**

Tel: +82-2-820-5306
E-mail: hckang@cau.ac.kr

Ⅰ. Introduction

With the advancement of the Internet and communication technologies, the industry of digital audio services where various mobile devices such as smart phones are commonly used is growing in the trend of simultaneous transition of PC to mobile and download to streaming [1]. In the speech recognition AI speakers (e.g., Amazon Echo), which are spreading recently due to the development of Artificial Intelligence technology, the interoperability with the audio service is one of the key components.

Much research has been conducted on music information retrieval and recommendation for audio service. The music retrieval could be basically based on metadata such as song title, name of the singer, etc., tag-based which allows music retrieval possible without knowing the exact information about the music [2][3], or content-based where the music that matches a sample query given in the form of an audio clip is retrieved [4][5]. In the content-based retrieval, one of the widely used approaches is to extract the fingerprints of the audio sources, store and index them in a database, and use them for the match with the given audio sample [6][7]. Since the number of audio sources is vast and the volume of the fingerprint database is accordingly huge, the scalability of the audio database is important. To deal with this issue, the big data processing technologies such as Google's MapReduce and NoSQL database can be employed [8][9].

In this paper, we have investigated the techniques to construct and efficiently maintain a fingerprint database for a content-based audio retrieval system. One of important characteristics in audio services is that new audio sources are continually added. The fingerprint data for these would be continually inserted into the existing database and the database and the index are accordingly updated. As a side effect, the space utilization efficiency and the audio retrieval performance are to be gradually degraded. The most basic method to solve this problem is the periodic reorganization of the database, but this would increase the system operation cost. Therefore, some techniques are required whereby the database expansion due to the addition of new audio sources could be efficiently conducted. In this paper, we design an audio retrieval system based on the fingerprinting algorithm of Shazam [10][11][12], employing MapReduce and a NoSQL database in a clustered computing environment. We evaluate its performance through experiments to show that it efficiently supports the database expansion.

The rest of this paper is organized as follows. In Section 2, we describe Shazam's algorithm, MapReduce, and NoSQL database as preliminaries, and then describe the related work on the fingerprint databases. In Section 3, we first describe the MapReduce steps for the fingerprint storage and those for the audio retrieval given an audio sample. Then, we describe NoSQL database schema designs for fingerprint storage. In Section 4, we report and analyze the experimental results. Finally, we describe further research and concludes the paper in Section 5.

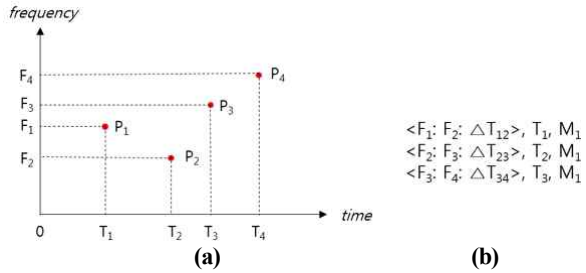
Ⅱ. Preliminaries and Related Work

In this section, we first describe Shazam's algorithm, MapReduce, and NoSQL database as background knowledge of this paper, and then describe related work on audio fingerprint databases.

2-1 Shazam's Algorithm

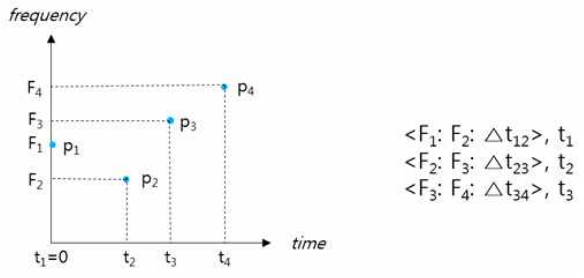
Shazam's algorithm is to extract fingerprints of the audio source represented in a 2-dimensional space of time and frequency, and to use them for audio retrieval. The outline of the technique introduced in [10] is as follows. Fig. 1(a) shows only four adjacent points P_1, P_2, P_3, and P_4 among all the points representing the time and frequency values of an audio source. The time and frequency of point P_i are T_i and F_i, respectively, where the time means the time offset from the beginning of the audio source to the corresponding point.

When P_1 is an anchor point and P_2 is a target point, fingerprint $<F_1: F_2: \Delta T_{12}>$ is extracted between the two points, where ΔT_{12} denotes the time difference between the two points, i.e., $T_2 - T_1$. Similarly, when P_2 is an anchor point and P_3 is a target point, fingerprint $<F_2: F_3: \Delta T_{23}>$ is extracted. When P_3 is an anchor point and P_4 is a target point, fingerprint $<F_3: F_4: \Delta T_{34}>$ is extracted. Given an anchor point, its target zone where several of its adjacent points are located can be set, and fingerprints can be extracted for each target point within the target zone. As the number of fingerprints extracted increases, the accuracy of audio retrieval would get higher, but the volume of the fingerprint database and its indices would increase and the maintenance overhead would increase, too.



(a) (b)

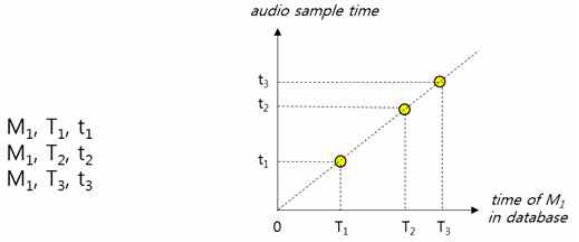
그림 1. (a) 시간-주파수 2차원 공간 (b) 지문 데이터 3-투플
Fig. 1. (a) 2-Dimensional Space of Time and Frequency (b) Ternary Tuples of Fingerprint Data



(a) (b)

그림 2. (a) 오디오 샘플의 시간과 주파수 (b) 지문 데이터 3-투플

Fig. 2. (a) Time and Frequency of Audio Sample (b) Ternary Tuples of Fingerprint Data



(a) (b)

그림 3. (a) 데이타베이스 검색으로 생성된 3-투플 (b) 대각선

Fig. 3. (a) Ternary Tuples Produced After Database Search (b) Diagonal Line

In order to make audio retrieval possible, it is necessary to store in the database not only the fingerprint but its time of the corresponding anchor point and the identifier of the corresponding audio source. That is, the data stored in the database is a ternary tuple of (fingerprint, time, audio identifier). Let M_1 be the identifier of the audio source in Fig. 1(a). Then, the three ternary tuples of fingerprint data shown in Fig. 1(b) are stored in the database.

When an audio sample is given as a query, the audio sources containing it are retrieved as follows. Fingerprints are extracted by the same Shazam's algorithm for the audio sample, and a list of (fingerprint, time) pairs is produced, where time is the time of the anchor point for the fingerprint. For each fingerprint out of these (fingerprint, time) pairs, the database is searched for the (fingerprint, time, audio identifier) tuples whose fingerprint is matched in order to produce the list of (audio identifier, database time, sample time) tuples. After these tuples are sorted on the audio identifier, a list of the (database time, sample time) pairs is produced for each audio identifier. Suppose this particular audio source in the database is the one that is matched against the audio sample. Then, plotting the (database time, sample time) pairs in a 2-dimensional space where x-axis is the database time and y-axis is the sample time would make a diagonal line. This is because the interval between adjacent times in the time sequence of the matched audio source is equal to that of the audio sample.

For example, let us consider an audio sample S composed of the four points shown in Fig. 2(a). They are the 4 points p_1, p_2, p_3, p_4, respectively, at time $t_1 = 0$, t_2, t_3, t_4 shown in Fig. 2(a). That is, p_i of Fig. 2(a) corresponds to P_i of Fig. 1(a), $i = 1, \cdots, 4$. If the same Shazam's algorithm is applied to the four points of Fig. 2(a), the list of the pairs shown in Fig. 2(b) is produced.

Searching the database with the fingerprints in Fig. 2(b), the ternary tuples in Fig. 3(a) is produced, because $\Delta T_{12} = \Delta t_{12}$, $\Delta T_{23} = \Delta t_{23}$, $\Delta T_{34} = \Delta t_{34}$. For the audio source M_1, plotting the three points (T_1, t_1), (T_2, t_2), (T_3, t_3) in a 2-dimensional space, a diagonal line of Fig. 3(b) appears because $\Delta T_{12} = \Delta t_{12}$, $\Delta T_{23} = \Delta t_{23}$, $\Delta T_{34} = \Delta t_{34}$.

2-2 MapReduce

MapReduce is a programming model presented by Google to process large data sets such as big data in a distributed and parallel way on a machine cluster [13]. Its open source implementations such as the one in the Apache Hadoop framework are widely used [14].

The MapReduce model consists mainly of a map function and a reduction function. The map function processes the input data to generate a list of (key, value) pairs. For each key out of the map functions, the reduce function groups and/or aggregates the list of its values, producing a list of values as required in the application. These operations are performed in distributed and parallel processing at each node of the cluster, and data transmission occurs between nodes in the process. The complete execution steps of the MapReduce are as follows: (1) input the data, (2) map, (3) shuffle the intermediate results of the map step, (4) reduce, and (5) output the result.

2-3 NoSQL Database

The conventional relational databases are characterized with the tabular data structure, transaction support with ACID properties, joins, SQL standard query language, etc. The NoSQL database features more flexible data structures, simple APIs, and more relaxed properties than ACID, such as BASE [15], being distributed in a cluster environment, providing horizontal scalability, and supporting big data [16].

The development of NoSQL databases has started for the applications that need to provide Web scale scalability, and its proof of concept is given through systems like Google's BigTable [17], Amazon's Dynamo [18], and Memcached [19]. Now, there are a variety of NoSQL database products, and they can be

classified on the data model into (1) wide column stores such as HBase and Cassandra, (2) key-value stores such as DynamoDB and MemcacheDB, (3) document stores such as MongoDB and CouchDB, and so on [20].

2-4 Related Work on Audio Fingerprint Database

The representative audio fingerprinting techniques include Shazam [10], Haitsma and Kalker's technique [21], and Google's Waveprint [22]. The fingerprints extracted from audio sources with any of these techniques are to be stored in a database. Because the number of audio sources is so large and the volume of fingerprint data is accordingly so large as well, indexing methods for efficient retrieval are essential [7].

In Haitsma and Kalker's technique [21], a 32-bit-length sub-fingerprint is extracted from the extracted fingerprint and used as a unit of search. In order to expedite query processing with a fingerprint block composed of 256 consecutive sub-fingerprints, a lookup table that can directly search for a sub-fingerprint is constructed as an index of the fingerprint database. The lookup table lists all possible sub-fingerprint values as its entries, meaning that there are as many as 2^{32} entries. Since it may not be feasible in terms of memory capacity, the lookup table can be implemented as a hash table. In [23], techniques are proposed to improve the retrieval performance using the lookup table of Haitsma and Kalker's technique.

Since the number of audio sources provided in an audio service is large, scalability is important for the fingerprint database. Scalability could be provided by employing big data processing technologies. In [8], the techniques to implement the fingerprint database with the Google's Waveprint algorithm on the cloud using Hadoop/MapReduce are presented. In [9], MapReduce is applied in the process of constructing the fingerprint database with the Shazam's algorithm. The proposed MapReduce generates an indexed-sequential file of fingerprint data for efficient search.

Ⅲ. System Design

This section describes the design of the content-based audio retrieval system of this paper. We first describe the MapReduce steps for storing fingerprint data in the database and for audio retrieval given an audio sample. Then, we describe the designs of the NoSQL database schema for storing large volume of fingerprint data.

3-1 MapReduce Steps

Since processing of large volume of audio data is carried out on several nodes in a cluster, MapReduce can be effectively utilized in the operations involved. Such operations include the followings: (1) Initial load: The bulk load of the fingerprint data obtained from a large number of initial audio sources into the database. (2) Additional insert: The insert of the fingerprint data obtained from additional audio sources into the database. This operation is executed every time a new set of audio sources is added for the audio service. (3) Sample search: The search of the database for retrieving audio sources that match a given audio sample.

In [9], the map and reduce steps for preparing the initial load are presented as follows. The map generates a list of (fingerprint, <time: audio identifier>) pairs from each audio file. In other words, the key is fingerprint, and the value is the combination of time and audio identifier. The reduce generates a list of <time: audio identifier> composite values for each fingerprint. In all, the list of these fingerprints and their corresponding lists of <time: audio identifier> composite values is to be loaded and indexed in the database. These MapReduce steps for the initial load can be re-used for the additional insert as well.

As for the operation of sample search, we design its MapReduce steps as follows. After a list of (fingerprint, time) pairs is obtained from the audio sample, the map generates a list of (audio identifier, <database time: sample time>) pairs by searching the database with each fingerprint *f* out of the audio sample for the composite value <time: audio identifier> of the ternary tuple in the database whose fingerprint matches *f*. In other words, the key is audio identifier while the value is <database time: sample time>. The reduce generates a list of <database time: sample time> composite values for each audio identifier. This list is used to determine if the particular audio source at hand is a match or not. The results of these map and reduce steps for the three types of operations are summarized in Table 1.

표 1. 맵리듀스 단계의 결과
Table. 1. Results of MapReduce Steps

Operation / Step	Initial Load & Additional Insert	Sample Search
Map	A list of (fingerprint, <time: audio identifier>) pairs	A list of (audio identifier, <database time: sample time>) pairs
Reduce	For each fingerprint, a list of <time: audio identifier> composite values	For each audio identifier, a list of <database time: sample time> composite values

3-2 NoSQL Database Schema

As described in Section 3.1, there are three types of main database operations: initial load, additional insert, and sample search. In audio services, the addition of a new set of audio sources continues, and the additional inserts are executed whenever such an event happens. The database is not static. Rather, it is supposed to be continually updated. In general, a storage system of a database can effectively perform an initial allocation of space for storing and indexing data to be bulk-loaded, thereby achieving very good retrieval performance. However, this initial space and search efficiency is to be gradually degraded due to the subsequent database updates. Eventually, it would suffer from lower space utilization and poor search performance. When we design a NoSQL schema for the fingerprint database, we need to consider that the additional inserts are the update operations with such effects.

In order to prevent the performance degradation in audio retrieval because of a series of additional inserts, it is necessary to periodically reorganize the database. However, considering the number of the audio sources, it takes a lot of time and might incur formidable cost of system operation. Therefore, a key point to consider in schema design is the performance trade-off between insertion of new audio sources and audio retrieval. Different schema designs are possible depending on whether the insertion performance is more important or the search performance is more important. In this paper, we take into account that the frequency of sample search operations is much higher than that of insert operations.

As described in Section 2.3, there are a number of NoSQL database models. In this paper, we consider the NoSQL database that belongs to the category of a wide column stores, one of whose original implementations is BigTable developed by Google [17]. The data in a wide column store can be modeled as consisting of a row key and multiple column families. Each column family consists of multiple columns. Column families and their columns can be added dynamically. The value of each column can store multiple versions. That is, several pairs of a data value and its timestamp as its version can be stored. Therefore, this model can be viewed as a multidimensional mapping that can retrieve the data given three values: row key, column, and timestamp.

The data to be stored in the fingerprint database are the ternary tuples of (fingerprint, time, audio identifier) as described in Section 2.1. Various designs of NoSQL database schema of the wide column stores for efficient storage of these tuples are possible as described below. We name them as Schema P, Schema T, and Schema L.

1) Schema P

The row key stores a fingerprint, and one column family is allocated. Each column of the column family stores a pair of (time, audio identifier). In BigTable, the timestamp which indicates the version may be set by the system or in the application. In the case of the NoSQL database belonging to the category of the wide column stores where the latter capability is supported, it is possible to have a variation of Schema P where only the audio identifier is stored as the value of the column, and the time is set to the timestamp as its version. Another variation is to allocate one column for all the pairs to store a list of audio identifiers with their respective times set to the timestamps as their versions. This variation is conceivable only when the NoSQL database supports a very large number of versions of data per column for a row key.

2) Schema T

The row key stores a fingerprint, and one column family is allocated. For each audio source, a column is allocated in the column family to store the time. If the same fingerprints are extracted several times from the same audio source, multiple number of different times could be stored as different versions. In the case of the NoSQL database belonging to the category of the wide column stores which allows the application to set the timestamp, it is possible to have a variation of Schema T where only a list of non-null indicator is stored as the value of the column, and the respective times are set to the timestamps as their versions.

3) Schema L

The row key stores a fingerprint, and one column family is allocated. A column is allocated in the column family to store a list of (time, audio identifier) pairs. It is possible to have variations of Schema L to deal with the case that this list is too long. First, the row key stores a composite value of <fingerprint: audio identifier> and the column stores a list of (time, audio identifier) pairs, where the audio identifier in the row key is the maximum of the audio identifiers stored in its corresponding column. The list is vertically split according to the audio identifiers. Secondly, in the case of the NoSQL database belonging to the category of the wide column stores which allows the application to set the timestamp, the column stores multiple lists of (time, audio identifier) pairs, which are disjoint with each other. Each list is treated as different version. In this variation, the list is horizontally split.

In the experiments of this paper, we also considered the conventional relational database as a repository of the fingerprint

data for performance comparison. In the case of a relational database, a table schema composed of three columns storing respectively fingerprint, time, and audio identifier is used. Since all the three columns do not have uniqueness property, the composite key consisting of all the three columns are declared as the primary key. An index is created on the fingerprint column to improve retrieval performance. As for the NoSQL schemas designed above, only the original three schemas are considered in the experiments to relax the functional requirements of the NoSQL database employed for the audio service.

Ⅳ. Experiments

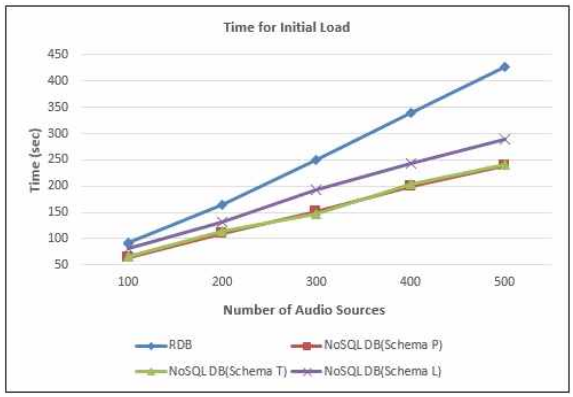
In this section, the performance of the content-based audio retrieval system designed in the previous section is evaluated through experiments using real world audio data, and the experimental results are described and analyzed.

4-1 Outline of the Experiments

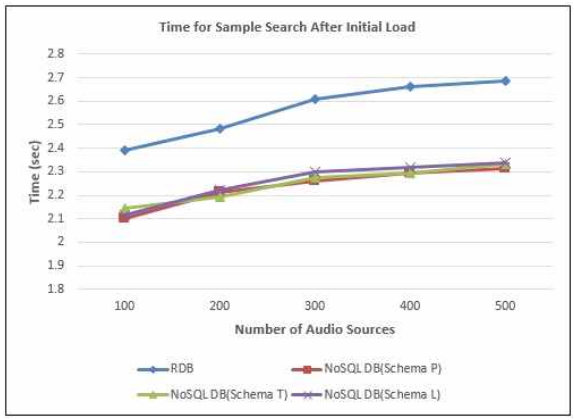
Three sets of experiments were conducted. First, as the basic experiments, we compared the efficiency of the fingerprint database between a NoSQL database and a relational database as well as among the three schemas of a NoSQL database. In this set of experiments, we compared the performance of the initial load, the additional inserts, the sample search after the initial load, and the sample search after the additional inserts. The volume of data used is 500 audio sources. Secondly, as the experiments on the scalability, we increased the volume of data up to 10,000 audio sources, and compared the scalability of a NoSQL database and that of a relational database as the data volume increases. Thirdly, as the experiments on the performance enhancement in a cluster computing environment with a NoSQL database, we increased the number of machines and that of nodes, and measured the performance improvements as the computing power increases horizontally.

All the experiments were conducted on a system equipped with 6GHz CPU, 2GB RAM, and 250GB HDD in Ubuntu 64bit OS environment. Implementation of MapReduce was done on the Hadoop platform. As a fingerprint database, one relational database and one NoSQL database belong to the category of the wide column stores were used. All the implementations were done in Java.

As for the audio data, publicly available MIDI files in MuseScore [24] were used. Among the various information stored in MIDI files, we extracted the pitches of the main melody and their corresponding time offsets, created a string



(a)



(b)

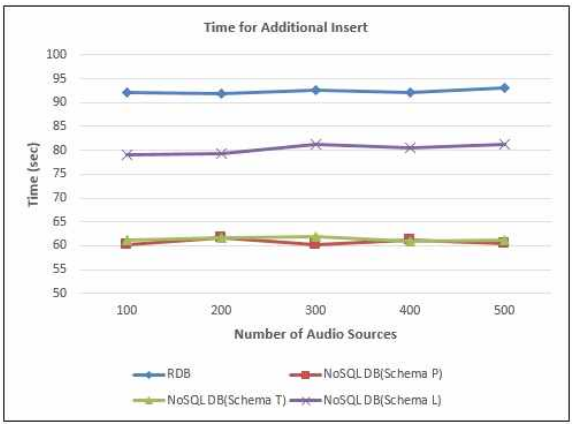
그림 4. 초기 적재 및 샘플 검색에 대한 기본 실험
Fig. 4. Basic Experiments for Initial Load and Sample Search

consisting of (pitch, time) pairs using jFugue [25], an open source programming library for Java. The pitch is one of the information directly related to the frequency of a sound, it is represented as an integer in the range of 0-127 in MIDI files. In our experiments, the pitch was used as the frequency when the fingerprints are extracted by the Shazam's algorithm.

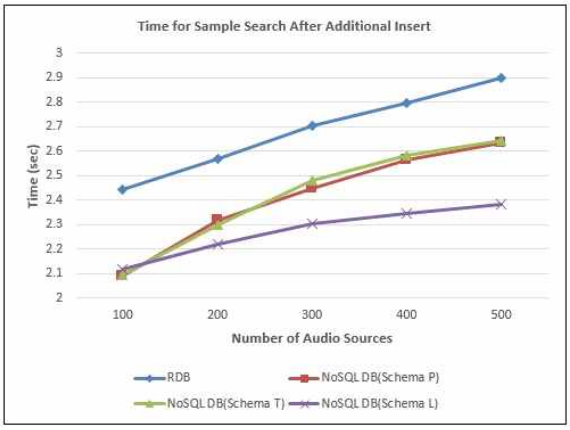
4-2 Experimental Results and Analysis

1) Basic Experiments

Fig. 4(a) compares the time taken for the initial load of the fingerprint data to a relational database and that to a NoSQL database with three different schemas. The volume of data used is 500 audio sources. Fig. 4(b) compares the time taken to retrieve the audio sources that are matched against a given audio sample right after the initial load. As the audio sample, a part of randomly selected audio source whose fingerprints are stored in the database was used. Five audio sources were selected for each experiment, and the average of the measured times was obtained. We also verified the accuracy of the searches. Fig. 5(a) compares the time it takes to insert the fi

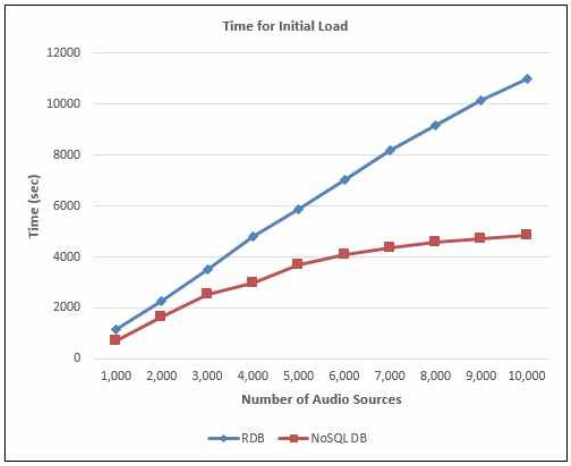


(a)

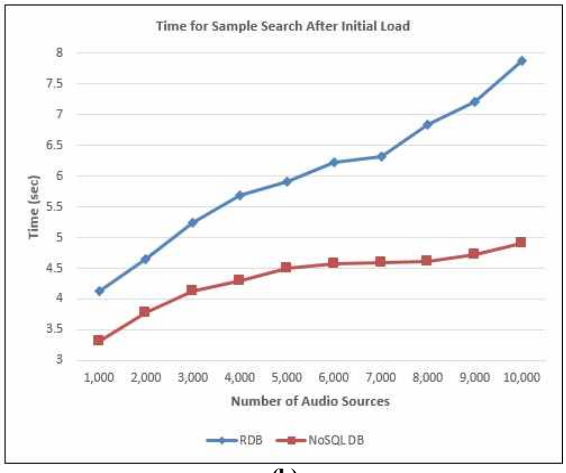


(b)

그림 5. 추가 삽입 및 샘플 검색에 대한 기본 실험
Fig. 5. Basic Experiments for Additional Insert and Sample Search



(a)



(b)

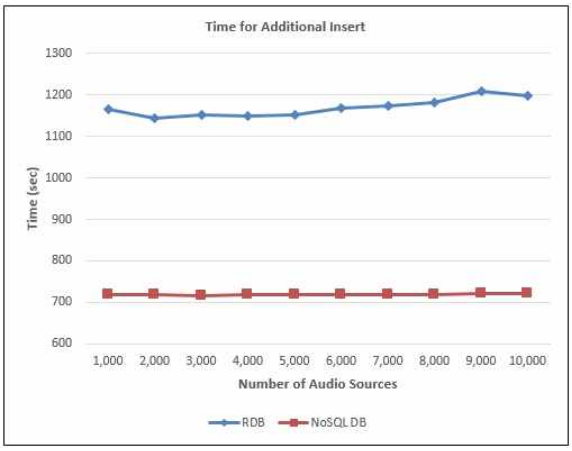
그림 6. 초기 적재 및 샘플 검색에 대한 확장성 실험
Fig. 6. Experiments on Scalability for Initial Load and Sample Search

ngerprint data out of a new set of audio sources to the existing database. The additional insert was conducted by adding 100 new audio sources to the database. Regardless of the number of audio sources stored already in the database, the number of newly inserted audio sources is equal to 100 for each additional insert. Thus, it is shown to take almost the same time for each additional insert. Fig. 5(b) compares the time taken to retrieve audio sources that are matched against a given audio sample after the additional insert.

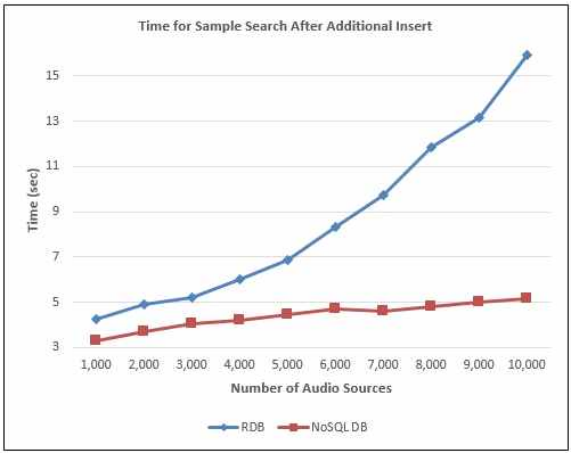
In all the experiments, the system with a NoSQL database significantly outperform that with a relational database. This means that it is more appropriate to use a NoSQL database than a traditional relational database as a repository of the fingerprint data for audio sources. Comparisons among the three schemas of the NoSQL database reveal that the time for both the initial load and the additional insert with Schema L takes longer than that with Schema P or Schema T as shown in Fig. 4(a) and Fig. 5(a). This is because, in the case of the schema L, the overhead of updating the list of (time, audio identifier) pairs in those operations incurs. However, such an overhead is amortized in the performance of audio retrieval in the audio service where new set of audio sources are continually added. In other words, considering the performance trade-off between insertion and search, the sacrifice in the insertion performance is redeemed as the effectiveness in the sample search. Such enhancement of retrieval performance is not noticeable immediately after the initial load of the fingerprint data as shown in Fig. 4(b), where there was not yet any additional insert. However, after the additional inserts in Fig. 5(b), the performance with Schema L is considerably better than that of Schema P or Schema T.

2) Experiments on Scalability

The same set of the basic experiments were carried out for a much larger set of audio sources to see if the system is scalable. In this set of experiments, we increased the volume of data up to 10,000 audio sources, while the volume of data used in the



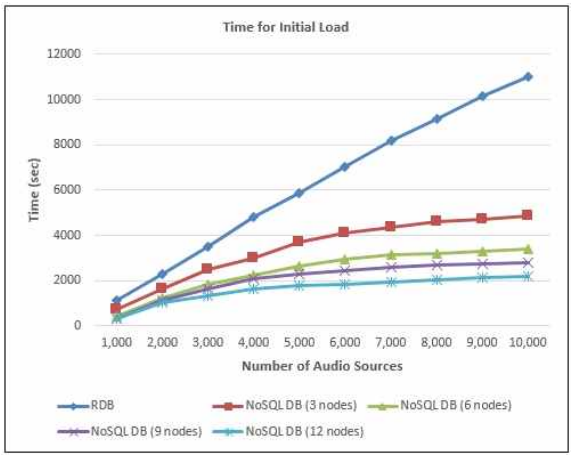
(a)



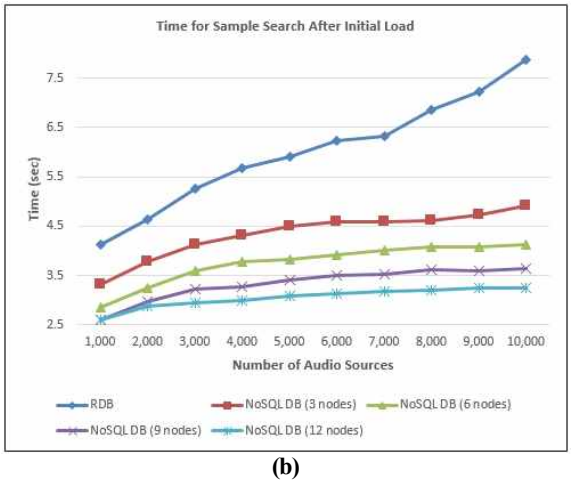
b

(b)

그림 7. 추가 삽입 및 샘플 검색에 대한 확장성 실험
Fig. 7. Experiments on Scalability for Additional Insert and Sample Search



(a)



(b)

그림 8. 머쉰 클러스터에서 초기 적재 및 샘플 검색의 성능 향상
Fig. 8. Performance Enhancement in a Machine Cluster for Initial Load and Sample Search

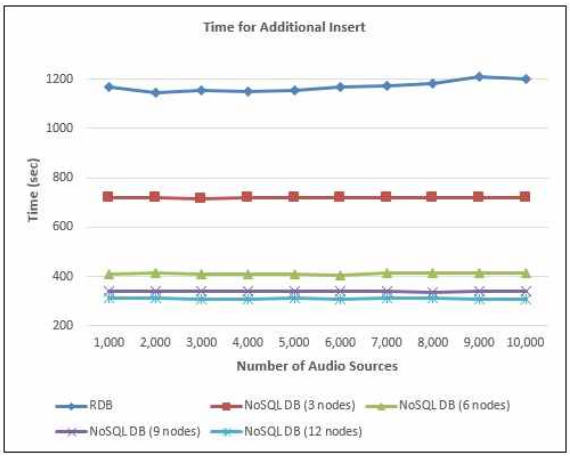
asic experiments was 500 audio sources. The additional insert was conducted by adding 1,000 new audio sources to the database. Fig. 6 and Fig. 7 show the results of the experiments, each of which respectively corresponds to the experimental results of Fig. 4 and Fig. 5 for smaller set of data in the basic experiments. As for the NoSQL database, only Schema L was considered in comparison with the relational database, because it had shown the best search performance in the basic experiments. In all the experiments, the scalability with a NoSQL database is much better than that with a relational database. While the system with a relational database turns out to be not so scalable as the number of audio sources increases, the scalability with a NoSQL database is obviously observed.

3) Experiments on Performance Enhancement in a Cluster Environment

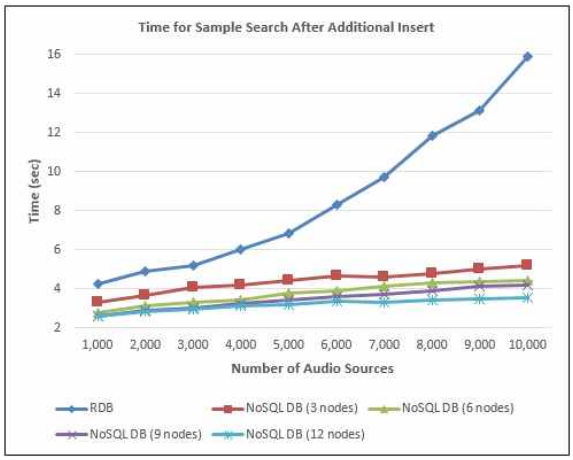
In this set of experiments, the number of machines in a cluster for the NoSQL database is varied from 1 to 4 while the number of nodes per machine was set to 3. That is, the number of nodes in the cluster changes from 3 to 12. For a set of 10,000 audio sources, the same set of experiments are carried out to measure the performance enhancement as the size of the cluster increases. Fig. 8 and Fig. 9 show the results. As the number of nodes increases, the performance of all the operations, the initial load, the additional insert, and the two cases of sample searches, against the NoSQL database are improved compared to that in the case of 3 nodes in 1 machine. In particular, we can observe that the system with the NoSQL database gets more scalable as the number of nodes increases.

Ⅴ. Conclusions and Further Research

In this paper, we presented a design of a content-based audio retrieval system that runs in a cluster computing environment



(a)



(b)

그림 9. 머쉰 클러스터에서 추가 삽입 및 샘플 검색의 성능 향상

Fig. 9. Performance Enhancement in a Machine Cluster for Additional Insert and Sample Search

with a NoSQL database to store and index the audio fingerprints, which are extracted using the Shazam's algorithm. MapReduce is used both in the process of storing the fingerprint data and in the audio retrieval given an audio sample. Using a large set of real world audio data, its performance of main database operations is evaluated. Due to the big data processing capabilities employed, the scalability of the system is observed. In particular, a good retrieval performance is achieved despite the continual expansion of the database occurred by the continual additions of new set of audio sources for the audio service, without periodical reorganization of the database.

Further research work includes the following: First, it is necessary to improve the performance of additional insert operations which are executed for the database expansion every time a new set of audio sources is added for the audio service. Further refinements of the proposed NoSQL database schemas are necessary to deal with this issue. Secondly, it is necessary to study the efficient modeling of the audio fingerprint data when a NoSQL database in the category of the key-value stores, which are also very popular in the development of web-scale applications, is used instead of the one of the wide column stores. Finally, alternative storage systems and access methods for a very large volume of audio fingerprint data need to be investigated. The storage platforms for big data processing such as distributed file systems and the techniques of parallel processing could be considered [26][27].

References

[1] Market Overview, IFPI Digital Music Report 2015, Available: http://www.ifpi.org/downloads/Digital-Music-Report-2015.pdf.

[2] D. Turnbull, L. Barrington, and G. Lanckriet, “Five approaches to collecting tags for music,” in *Proceedings of the 19th International Conference on Music Information Retrieval*, pp 225–230, 2008.

[3] S. Lee, M. Masoud, J. Balaji, S. Belkasim, R. Sunderraman, and S. Moon, “A Survey of Tag-based Information Retrieval,” *International Journal of Multimedia Information Retrieval*, Vol. 6, pp. 99–113, 2017.

[4] N. Borjian, E. Kabir, S. Seyedin, and E. Masehian, “A Query-By-Example Music Retrieval System Using Feature and Decision Fusion,” *Multimedia Tools and Applications*, 2017 [Online]. Available: https://link.springer.com/content/pdf/10.1007%2Fs11042-017-4524-1.pdf.

[5] M. Kaminskas and F. Ricci, “Contextual music information retrieval and recommendation: State of the art and challenges,” *Computer Science Review*, Vol. 6, No. 2-3, pp. 89–119, 2012.

[6] P. Cano, E. Batle, T. Kalker, and J. Haitsma, “A review of algorithms for audio fingerprinting,” in *Proceedings of IEEE Workshop on Multimedia Signal Processing*, pp. 169-173, Dec. 2002.

[7] C. Yu, R. Wang, J. Xiao, and J. Sun, “High Performance Indexing for Massive Audio Fingerprint Data,” *IEEE Transactions on Consumer Electronics*, Vol. 60, No. 4, pp.690-695, November 2014.

[8] J. Wenyu, Z. Yongwei, B. Xiaoming, and Y. Rongshan, “Cloud-based Audio Fingerprinting Service,” in P*roceedings of Asia Pacific Signal and Information Processing Association Annual Summit and Conference*, pp. 1-6, 2012.

[9] J. Lee and H. Jung, “Content-based Music Searching System

Using Hadoop," in P*roceedings of the Third International Conference on Emerging Databases*, pp. 311-316, 2011.
[10] A. Wang, "An Industrial Strength Audio Retrieval Algorithm" in *Proceedings of the 4th International Conference on Music Information Retrieval*, pp. 7-13, 2003.
[11] Shazam, https://www.shazam.com/.
[12] A. Wang, "The Shazam Music Recognition Service," *Communications of the ACM*, Vol. 49, No. 8, pp. 44-48, 2006.
[13] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pp. 137-149, 2004.
[14] Apache Hadoop, https://hadoop.apache.org/.
[15] D. Pritchett, "BASE: An ACID Alternative," *ACM Queue*, pp. 48-55, May/June, 2008.
[16] R. Cattell, "Scalable SQL and NoSQL Data Stores," *SIGMOD Record*, Vol. 39, No. 4, pp. 12-27, December 2010.
[17] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A Distributed Storage System for Structured Data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pp. 205-218, 2006.
[18] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store," in *Proceedings of ACM Symposium on Operating Systems Principles*, pp. 205-220, 2007.
[19] Memcached, www.memcached.org
[20] NoSQL Database, http://www.nosql-database.org/.
[21] J. Haitsma and T. Kalker, "A Highly Robust Audio Fingerprinting System With an Efficient Search Strategy," *Journal of New Music Research*, Vol. 32, No. 2, pp. 211-221, 2003.
[22] S. Baluja and M. Covell, "Waveprint: Efficient wavelet-based audio fingerprinting," *Pattern Recognition*, Vol. 41, pp. 3467–3480, 2008.
[23] S. Lee, D. Yook, and S. Chang, "An Efficient Audio Fingerprint Search Algorithm for Music Retrieval," *IEEE Transactions on Consumer Electronics*, Vol. 59, No. 3, pp. 652-656, Aug. 2013.
[24] MuseScore, https://musescore.org/.
[25] JFugue, http://www.jfugue.org/.
[26] J. Yoon and U. Song, "Study of Optimization through Performance Analysis of Parallel Distributed File System," *Journal of Digital Contents Society*, Vol. 17, No. 5, pp. 409-416, Oct. 2016.
[27] V. Nguyen, S. Nguyen, and K. Kim, "Design of a Platform for Collecting and Analyzing Agricultural Big Data," *Journal of Digital Contents Society*, Vol. 18, No. 1, pp. 149-158, Feb. 2017.

박지훈(Ji Hun Park)

2013: B.S., Department of Computer Engineering, DaeJeon University
2016: M.E., Department of Computer Science and Engineering, Chung-Ang University

Areas of Interest : Audio Data Processing, Distributed Processing of Big Data

강현철(Hyunchul Kang)

1983: B.E., Computer Engineering, Seoul National University
1985: M.S., Computer Science, U. of Maryland, College Park
1987: Ph.D., Computer Science, U. of Maryland, College Park

1988~Present: Professor, School of Computer Science and Engineering, Chung-Ang University
Areas of Interest : Database, Big Data, Sensor Network, Internet of Things