THE DESIGN METHOD OF A MELODY RETRIEVAL SYSTEM ON PARALLEL-IZED COMPUTERS

Tomonari Sonoda, Toshiya Ikenaga, Kana Shimizu and Yoichi Muraoka
School of Science and Engineering, Waseda University
3-4-1 Ohkubo Shinjuku-ku, Tokyo 169-8555, Japan.
{sonoda,ikenaga,kana,muraoka}@muraoka.info.waseda.ac.jp

Abstract This paper describes the design method of a WWW-based melody-retrieval system which takes a sung melody as a search clue and retrieves the music title from a music database of standard MIDI files(SMF) over the Internet. The most important thing in building a melody-retrieval system on the Internet is to achieve both high matching accuracy and quick search. It was, however, quite difficult to simultaneously fulfill these two conditions since it took long time for the matching process. We propose the design of a system which consists of parallel-ized melody-retrieval servers for building a high performance service on the Internet.

Introduction

In building a melody retrieval service on the Internet, it is quite effective to take a user's sung melody as a search query for retrieving a music database.

A dp-matching method is one of effective methods for comparing melody sequences and previous system [1, 2, 3, 4, 5, 8] used it in the matching process.

It was, however, quite difficult to reduce the search time by using the dp-matching and previous systems[1, 2, 7] proposed several indexing methods for melody sequences. It was, on the other hand, difficult to keep high matching accuracy in using such indexing methods.

We therefore proposed a scalable melody-retrieval system which is implemented on parallel-ized computers and simultaneously fulfill two conditions, high matching accuracy and quick search[9]. The method was quite effective and it showed that the retrieval time of the system was 1.2 seconds on average and its matching accuracy was 70% for 1,200 inputs against a database which contained more than 26,000 melodies.

Since we had not defined the way to build such a parallel-ized melody-retrieval systems, we propose and define the design method in this paper.

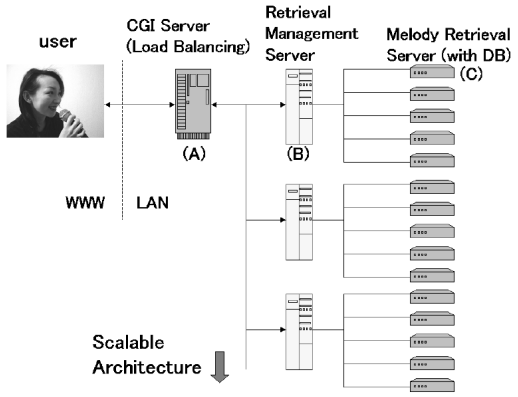

Figure 1. A melody-retrieval system on parallel-ized computers.

1. Melody-Retrieval System

This section describes the overview of a melody-retrieval system.

1.1. Overview of The Entire System

We propose a design of a melody-retrieval system which consists of several servers and client as shown in Fig.1.

The server system has at least one melody-retrieval server with a music database(C) and it can be parallel-ized by several servers(A,B) and it relays query data and the search result data between servers. This data-relaying enables the system to extend any scale of server network.

In Fig.1, each melody-retrieval server treats a matching process which compares a user's input melody with each database melody. It can also treat a text matching by using music titles or artist names. The server system returns a list of music titles for the matching result.

The client system, on the other hand, records a user's sung melody and the acoustic-signal of recorded voice is converted to small melody data and it is transmitted to a server system as a search clue. The client receives the music list from the server system and shows it to the user as a matching result.

1.2. Scalable Design of Melody-Retrieval Servers

To achieve both high matching accuracy and quick search, we propose to parallel-ize melody-retrieval servers by using two types of data-relay servers;(A)a CGI-server which relay a user's query data to one of servers. This type of server can play a role of the load-balancing function and it is quite effective to achieve quick search against a lot of user accesses and (B)a retrieval management server which relay a user's query data to all of connectable servers. This type of server enables two or more melody-retrieval servers to be parallel-ized and to use the dp-matching without index data for accurate matching since each server does not need to have a large database.

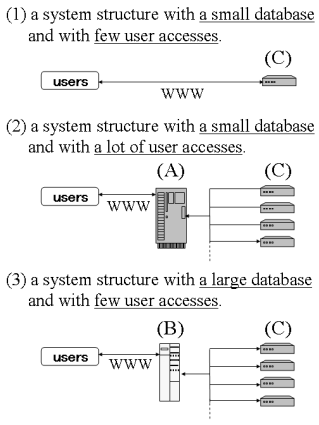

Figure 2. Scalable Server System.

Fig.2 shows images of server structures. Fig.2-(1) is the minimum structure and we used this for our previous system [3, 7] for small services, Fig.2-(2) is for the service with a small database and with a lot of simultaneous user accesses, Fig.2-(3) is for the service with a large database and with a few simultaneous user accesses, and Fig.1 is the maximum image for the service with a large database and with a lot of simultaneous user accesses.

2. Server System Design

This section describes designs of each server.

2.1. CGI Server

This section describes a logic design of a CGI-server. The main process of a CGI-server is to relay both a query data and a search result between a user and a server(B) or a server(C). A CGI-server has a server list $LIST_1$ which consists of N pairs of a host name(address) $HOST$ and a port number $PORT$.

$$LIST_1 = \{s_i \mid i = 0, 1, \ldots, N-1\}$$
$$where\ s_i = pair(HOST_i, PORT_i).$$

When a CGI-server accepts a user's query message QM, it randomly selects s_i from $LIST_1$ and sends QM to a server $SEVER_i$ represented by s_i ($SERVER_i$ is one of retrieval management servers or melody-retrieval servers). After receiving a search result message RM from $SERVER_i$, the CGI-server just relay it to the user.

When a CGI-server can not connect with s_i, it tries to connect with other $s_i (i \neq ii)$ in $LIST_1$. When there are no servers to connect with, the CGI-server returns an error message $ERROR$ to the user. The function of a $SERVER_i$

is assumed to be F_{s_i} and it can be defined by using RM and QM.

$$RM = F_{s_i}(QM)$$

The return-message of a CGI-server F_{cgi} for QM is defined as follows.

$$F_{cgi}(QM) = \begin{cases} F_{s_n}(QM), \ \ s_n \in LIST_1. \\ ERROR : when\ no\ SERVER_n\ response. \end{cases}$$

This shows that our CGI-server just relay a user's query to one of $SERVER_i$ and relay the search result of $SERVER_i$ to the user. It is quite effective for the Internet services since the CGI-server plays a role of a load-balancing server. More over, it hides a host addresses of servers and prevents users on a network with malice attacking servers.

2.2. Retrieval-Management Server

This section describes a logic design of a retrieval-management server. A retrieval management server has a list of melody-retrieval servers $LIST_2$ which contains M pairs of a host name(address) $HOST$ and a port number $PORT$. $LIST_2$ is expressed as follows like $LIST_1$.

$$LIST_2 = \{mr_j \mid j = 0, 1, \ldots, M-1\}$$
$$where\ mr_j = \ pair(HOST_j, PORT_j).$$

The retrieval management server accepts user's search query QM and transmits it to all the servers listed in $LIST_2$. Each melody-retrieval server MR_j, which is represented by mr_j, returns a music list $mLIST_j$ as the search result. The function of a melody-retrieval server F_{mr_j} can be expressed as follows.

$$mLIST_j = F_{mr_j}(QM)$$

Each $mLIST_j$ contains K_j pieces of music information $music(j)$ which consists of a matching distance D and the text information $dTEXT$.

$$mLIST_j = \{music(j)_k \mid k = 0, 1, \ldots, K_j - 1\}$$
$$where \quad music(j)_k = \ pair(D(j)_k, dTEXT(j)_k).$$

$D(j)_k$ is a dp-matching distance between $music(j)_k$ and QM. A melody similar to user's input melody has a smaller value of $D(j)_k$. $dTEXT(j)_k$ is the text information such as music title, artist name, or its genre.

When M' is the number of the sum totals of servers which response, the retrieval management server combines all music lists with one and the combined music list $mLIST_{all}$ is expressed as follows.

$$mLIST_{all} = mLIST_0 \cup mLIST_1 \cup \ldots \cup mLIST_{M'-1}$$

The retrieval management server then sorts all $music(j)_k$ in $mLIST_{all}$ in order with a smaller value of $D(j)_k$ and returns the sorted music list as the response

message for QM. The sorted music list $mLIST_{sorted}$ is expressed as follows.

$$\begin{aligned} mLIST_{sorted} &= \{music_h \mid h = 0, 1, \ldots, K_{sorted} - 1\}, \\ where \quad K_{sorted} &= \sum_{j=0}^{M'-1} K_j, \\ music_h &= pair(D_h, dTEXT_h) \in mLIST_{all}, \\ & \quad D_h \leq D_{h+1}. \end{aligned}$$

When no melody-retrieval servers response, the melody retrieval server returns an error message $ERROR$.

The return message of a retrieval management server F_{rms} for query QM is therefore defined as follows.

$$F_{rms}(QM) = \begin{cases} mLIST_{sorted} \ when\ M' > 0 \\ ERROR: \ when\ M' = 0. \end{cases}$$

As having mentioned above, a retrieval management server manages two or more melody-retrieval servers. It is effective for the quick search and the database scalability.

2.3. Melody-Retrieval Server

This section describes a logic design of a melody-retrieval server. As defined in the previous section, a melody-retrieval server returns a music list $mLIST$ for a user's query QM.

$$mLIST = F_{mrs}(QM)$$

A music database DB consists of pieces of music data which contains a pair of text information $dTEXT$ and melody data $dMELODY$. DB can be expressed by using its size K as follows.

$$\begin{aligned} DB &= \{data_k \mid k = 0, 1, \ldots, K - 1\} \\ where \quad data_k &= pair(dTEXT_k, dMELODY_k). \end{aligned}$$

The $dTEXT$ consists of such as music title, artist name or its genre and the $dMELODY$ consists of sequences of relative-pitch and relative-span value of notes which are calculated by using SMF. The information is compared with a user's query QM in a matching process.

QM can consist of both of a text data $qTEXT$ and a melody data $qMELODY$. It can also consist of only a $qTEXT$ or of only a $qMELODY$. It can be expressed as follows ($NULL$ expresses no data of $qTEXT$ or of $qMELODY$).

$$QM = \begin{cases} pair(qTEXT, qMELODY) \\ pair(NULL, qMELODY) \\ pair(qTEXT, NULL) \end{cases}$$

In the matching process the $qTEXT$ is assumed that it is a part of $dTEXT_k$ and music candidates of the search result are selected by using $qTEXT$. We express a set of the candidates $CAND$ as follows.

$$CAND = \begin{cases} \{data_k \in DB\} & : \quad where\ qTEXT(\in dTEXT_k) \neq NULL \\ DB & : \quad where\ qTEXT = NULL \end{cases}$$

After this text matching process, the melody-retrieval server compares $qMELODY$ with each $dMELODY_k$ by using dp-matching. Dp-matching calculates each distance D_k between $qMELODY$ and $dMELODY_k$[3].

$$D_k = DpMatching(qMELODY, dMELODY_k).$$
$$where\ \ 0 \leq k < K,\ \ data_k \in CAND.$$

$mLIST$ is finally defined as follows.

$$mLIST = \{music_k \mid 0 \leq k < K,\ \ data_k \in CAND\}$$
$$where\ \ \ music_k = pair(D_k, dTEXT_k),\ \ D_k \leq D_{k+1}$$

As having mentioned above, the $music_k$ in the $mLIST$ is sorted in order with a smaller value of D_k.

3. User-Side Client

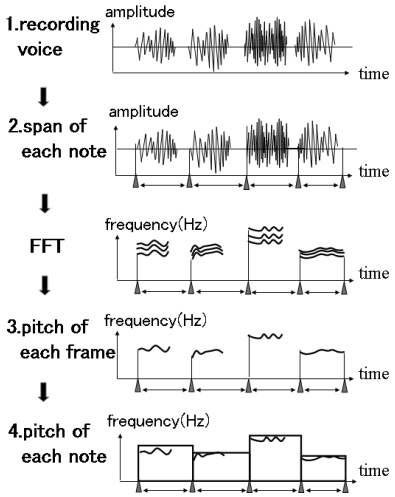

Figure 3. Pitch and Span Value Estimation for Each Note.

This section describes functions of the user-side client system. Each process of the client system is shown in the following paragraphs.

Recording Voice. In using our system, at first, a user inputs a melody into a microphone by singing. The client allows the user to input from an any part of

a piece of music and to use any key or tempo. The system assumes each input note begins with a voiceless consonant and ends with a vowel (e.g. ta, cha) [3] and that input sound is only user's voice. In A/D conversion, our system obtains a user's voice as 8 bits, monophonic recording data of which sampling rate is 8,000 Hz (Fig.3-1.).

Span-value Estimation for Each Note. To estimate a span-value of each note of a input melody, the system utilizes thresholds for the amplitude of sampled voice and detects start time of each note. The span is defined as the time-length between start times of the current and the next notes. The span of the last note is defined as its duration(Fig.3-2.) .

Pitch Detection for Each Frame . Our system then utilizes 512-points hamming window and FFT (Fast Fourier Transforms) for estimating a pitch-value of each frame (16msec). In this process, harmonics of frequency domain is used to detect the pitch value.(Fig.3-3.)

Pitch-value Estimation for Each Note. The client system then takes the highest pitch-value of frames appeared in a note span as the pitch-value of the note.(Fig.3-4.)

Conversion of Relative-value Sequences. To allow a user to choose free key or free tempo, each value of pitch and span are converted to relative-values. The relative-pitch is pitch difference between the adjacent notes in which a semitone difference is normalized to the value of 100. The relative-span value is the ratio of the span value to its previous value and is represented by percentage.

Search Query Transmission. A user can also input not only melody data but a text query, a part of artist name or of a music title. The maximum transmission data size of the query is 1500 bytes when a user inputs 128 notes.

Search Result. The search result is a list of music which has a similar part to a user's input melody. As each music information, our current system shows a pair of music title and artist name in a WWW-browser.

4. Experiments and Results

To evaluate the proposed method we had experiments in the previous paper [9] and it showed the effectiveness of our method.

Table 1 shows the performance of our current system and we confirmed that our system could keep both high matching accuracy and quick search against a large music database.

We used 6 PCs(CPU:$IntelPentium^{TM}III$ 866-MHz, RAM:512MB) for 1 retrieval management server and 5 melody-retrieval servers (e.g. Fig.2-(3)). Our database contained 26,735 melodies, consisting of 25,000 pieces of randomly generated music and 1,735 pieces of music from Japanese popular songs.

We used 1,200 input queries with only melody information from 3 people. The average number of notes of input melodies was 28.0 when we limited recording time within 15 seconds. The matching accuracy was calculated by counting the case that user's intended music is ranked in the top of a music list of the result.

Table 1. System Performance

the number of database melodies	26,735
the number of notes in the database	8,919,716
matching accuracy	70.0%
the average search time (with only melody query)	1.22 secs
parallel-ized melody-retrieval servers	5
the number of queries for testing	1,200

The average search time of 1200 inputs, which contains both melody and text information, was 0.04 seconds and matching accuracy was 84.8 %.

5. Conclusion

We have proposed the design method for a WWW-based melody-retrieval system on parallel-ized computers. The method enables a melody-retrieval system to be parallel-ized and to achieve both high matching accuracy and quick search. As the future work, we plan to design a system which can search not only SMF but actual music[6, 8].

References

[1] T.Kageyama, K.Mochizuki, and Y.Takashima *Melody Retrieval with Humming*, ICMC Proc., pp.349-351, 1993.

[2] Asif Ghias and Jonathan Logan *Query By Humming – Musical Information Retrieval in an Audio Database*, ACM Multimedia 95, Electronic Proc., 1995.

[3] Tomonari Sonoda, Masataka Goto, Yoichi Muraoka: *A WWW-based Melody Retrieval System*, pp.349-pp.352, ICMC'98 Proc., 1998.

[4] Lloyd A. Smith, Rodger J. McNab, and Ian H. Witten: *Sequence-Based Melodic Comarison: A Dynamic-Programming Approach*, pp.101-pp.116, COMPUTING IN MUSICIOLOGY 11, 1997-98.

[5] David Bainbridge: *MELDEX: A Web-based Melodic Locator Service*, pp.223-pp.229, COMPUTING IN MUSICIOLOGY 11, 1997-98.

[6] Masataka Goto, and Satoru Hayamizu : *A Real-time Music Scene Description System: Detecting Melody and Bass Lines in Audio Signals*, Working Notes of the IJCAI-99 Workshop on Computational Auditory Scene Analysis, pp.31-40, August 1999.

[7] Tomonari Sonoda and Yoichi Muraoka: *A WWW-based Melody Retrieval System -An Indexing Method for a Large Database-*, ICMC2000 Proc., 2000.

[8] T. Nishimura, H. Hashiguchi, J. Takita, J. Xin Zhang, M. Goto and R. Oka, : *Music Signal Spotting Retrieval by a Humming Query Using Start Frame Feature Dependent Continuous Dynamic Programming*, Proceedings of the 2nd Annual International Symposium on Music Information Retrieval (ISMIR 2001), pp.211-218, 2001.

[9] Tomonari Sonoda, Toshiya Ikenaga, Kana Shimizu and Yoichi Muraoka: *A Melody-Retrieval System on Parallel-ized Computers*, Proc. of IWEC2002, pp.261-268, 2002.