Data Stream Management Technology for Accumulating and Processing Lifelogs

Tomohiro Hasegawa†*, Ichibe Naito, Takashi Menjo, Motohiro Matsuda, Hiroki Akama, and Masashi Yamamuro*

Abstract

In this article, we describe our data stream managing and processing technology, which accumulates and manages streaming log data sent from sensing devices and personal cellular phones and processes it nearly in real time. We also introduce the architecture of our Lifelog Management System (LLMS), which provides the technologies needed to support lifelog-based services.

1. Introduction

Services that offer attractive new functionalities through the use of various time-series data as lifelogs have been drawing attention. Such data includes time-series data generated from sensing devices, histories of personal activities collected from modern cellular phones, and log data accumulated in each information system. As shown in **Fig. 1**, users' location data collected by phones equipped with a GPS (global positioning system) function and users' context information such as their profiles and/or device operation histories can be accumulated. When all this data has been processed, users can be provided with personal recommendation services that suggest restaurants and tourist spots.

Such advanced services require the implementation of the following functions.

- Accumulation and management of enormous amounts of lifelog data collected over long periods of time on servers on the network
- Various types of processing in nearly real time taking into account the user's context
- Scalability for handling additional users and their requests
- Processing algorithm alteration or addition without service operation suspension

This article introduces our basic technology that helps to realize lifelog-based services.

† NTT Cyber Space Laboratories
Yokosuka-shi, 239-0847 Japan

2. Write-once read-many-times data management system

NTT Cyber Space Laboratories has been developing a write-once read-many-times data management system (DMS); it continually accumulates and manages time-series data, such as lifelog data, as a data stream and provides information to users after processing and integrating it. DMS can subject data streams to pre-registered processing algorithms every time new data arrives, so it can offer near-realtime event processing.

DMS can be run on clusters of personal computers (from one to a few hundred). It offers dynamic scalability of data processing (growth in scale) when information sources need to be added or the data volume increases. It also supports dynamic changes in processing algorithms (functionality enhancement) for data streams that arrive constantly. As shown in

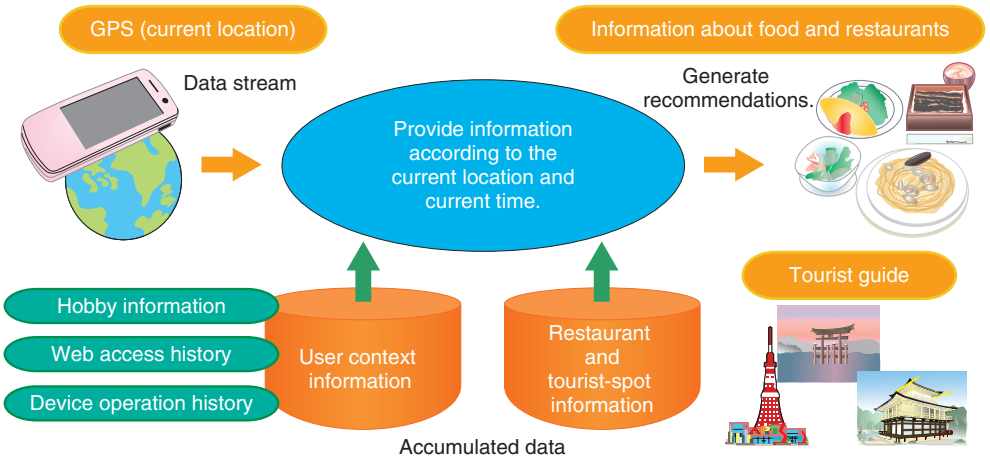


Fig. 1. Examples of lifelog services.

Fig. 2, DMS has a three-layer architecture consisting of queues, which receive and accumulate data; filters, which process received data and integrate accumulated data; and views, which provide processing results for each application.

(1) Queues

Each information source is associated with a different queue. Each queue receives and accumulates a data stream generated from the information source via *add client*. A new information source can be supported by adding a new queue.

(2) Filters

Filters pull data from queues, apply the processing algorithms set for each information source to the data, and send the processing results to views. Each filter determines which queue it should get data from on the basis of the data arrival rate. If the data volume and number of processing operations increases, the

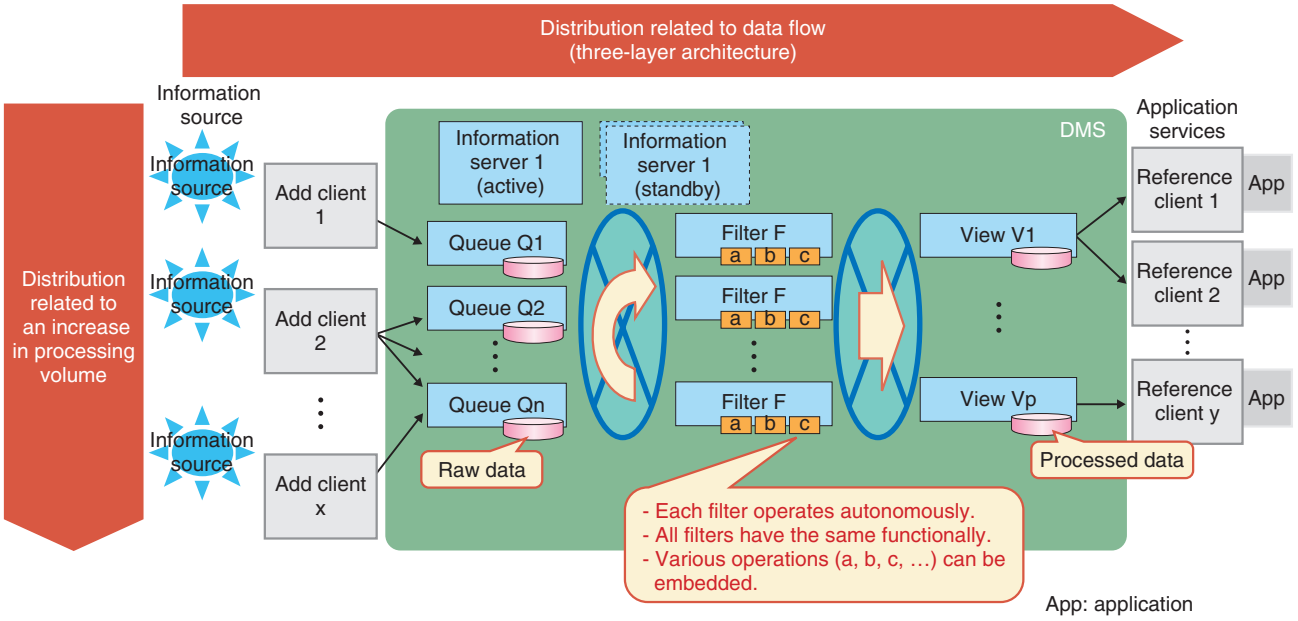
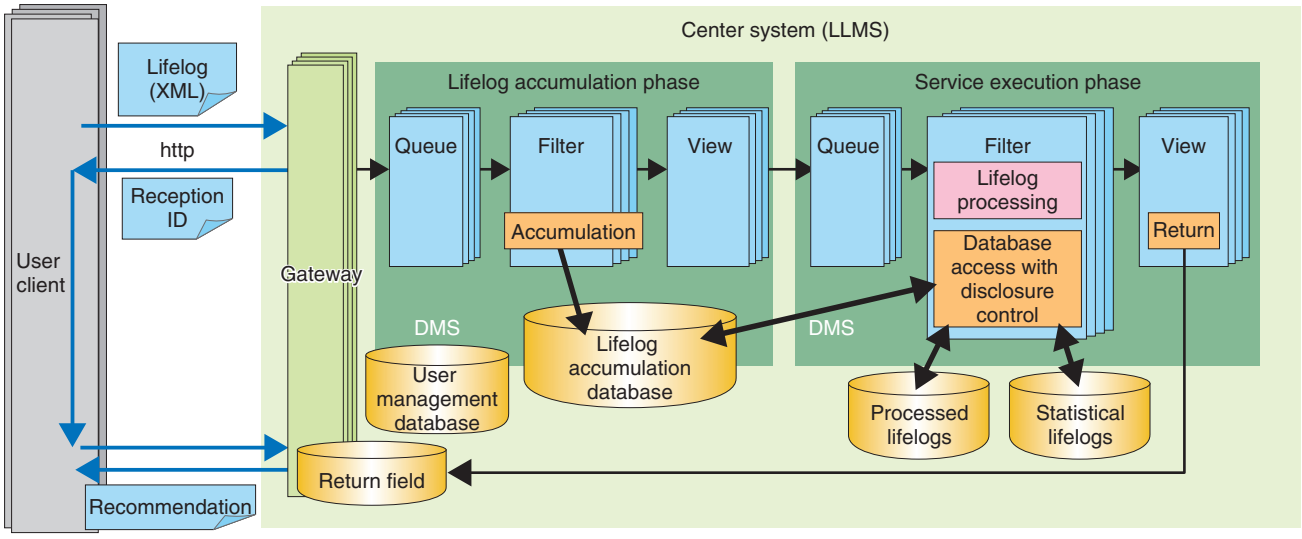


Fig. 2. Architecture of write-once read-many-times DMS.



XML: extensible markup language

Fig. 3. LLMS structure.

processing throughput can be improved by setting additional filters, without suspending operations (growth in scale). Furthermore, processing algorithms can be changed and new algorithms can be added before the next data is obtained from queues (functionality enhancement).

(3) Views

Views integrate the results yielded by the filters to offer integrated results for application services. The integrated results can be accumulated in a database or sent to another server by HTTP (hypertext transfer protocol) calls.

3. Lifelog Management System

The Lifelog Management System (LLMS) utilizes the characteristics of DMS. As **Fig. 3** shows, LLMS consists of a two-layered DMS and the lifelog database. The first layer of DMS, the lifelog accumulation phase, receives individual lifelogs from users' clients and stores them in the lifelog database. The second layer, the service execution phase, accesses the data in the lifelog database, executes the operations registered for each service, and outputs the results to the return field. Processing throughput can be improved to match an increase in the number of users by adding filters in the first layer and utilizing the growth-in-scale feature of DMS. Moreover, it can be improved to handle any increase in the number of services by adding filters in the second layer.

3.1. Workflow of LLMS

1) Each user sends lifelog data to the center system via the user's client and requests the execution of desired lifelog-based services.
2) The center system issues and returns a reception identifier (ID) to each request from a user client.
3) The user client acquires the service results from the return field, which can be accessed only by using the reception ID as the key.

Since services are executed asynchronously with respect to user requests, the user can get execution results whenever desired even if they take a long time to generate.

3.2. Lifelog accumulation database

The lifelog data sent from each user is stored in the database in the first layer of DMS, the lifelog accumulation phase. We believe that lifelog data must be handled in a unified manner under the assumption that this database will need to accumulate a large number and variety of lifelogs, such as GPS-generated user location information, names of products users bought and shops where they bought them, users' website access histories, and metadata of pictures taken by the users. Moreover, in the near future,

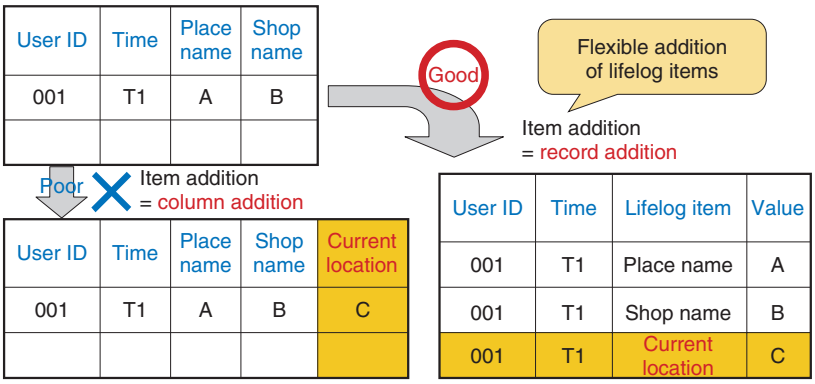


Fig. 4. Addition of new lifelog items.

many other sorts of lifelogs will emerge with the emergence of novel sensor devices and/or new services.

As just described, the lifelog database must be flexible to adapt those various lifelogs in an integrated manner and its table structure must be flexible in order to add new lifelog items easily. Thus, for our database we chose to use the idea of a *key-value store*. For example, as shown in **Fig. 4**, when the new lifelog item *current location* is added, it is handled not as a column addition to an existing table but as a *record addition*, so the table structure of the lifelog database does not need to be changed during processing.

3.3. Lifelog addition, updating, and erasure

Lifelogs received from users will be stored as log data in the lifelog database. Those newly stored lifelogs are basically used only for viewing. However, to provide flexibility in supporting requests to update or delete stored lifelog data, the lifelog database has *update time* and *erase flag* items as system management information, in addition to the lifelogs received from users. Examples of adding data to a lifelog, updating a lifelog value, and erasing a lifelog are shown in **Fig. 5**.

When new data needs to be added (Fig. 5(a)), the update time is set to the time at which the data was added (T2) and the erase flag is set to false.

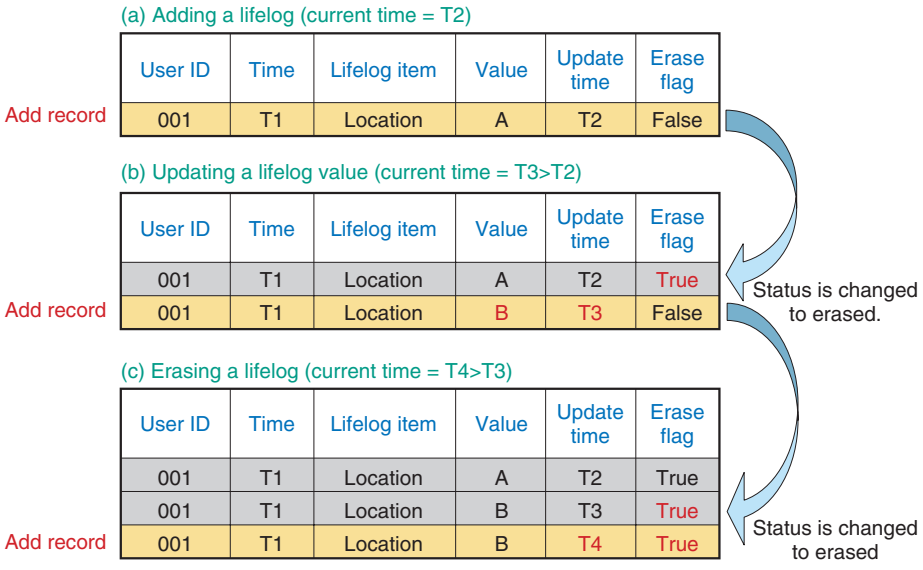


Fig. 5. Lifelog addition, updating, and erasure.

When a lifelog value needs to be updated (Fig. 5(b)), the record data whose update time is T2 is logically deleted by changing the erase flag from false to true (row 1). Then, a new record whose lifelog item value is B (instead of the previous value of A), update time is T3, and erase flag is false is added (row 2).

When a lifelog needs to be erased (Fig. 5(c)), the data's erase flag is changed from false to true to erase the data logically (row 2), in a similar way to Fig. 5(b). Furthermore, a new record whose update time is T4 and erase flag is true is added (row 3).

As you can see, the update time can be used to create a history of when lifelog records were added, updated, or erased. Moreover, the logically erased state achieved by using the erase flag enables lifelog updating and erasure to be supported while the lifelog-operation history itself is being recorded as a lifelog.

4. Conclusion

We are currently conducting field experiments on restaurant recommendations based on user context information as one example of a lifelog-utilizing service provided from modern cellular phones with a GPS function. We will strive to develop new technologies to support extremely large-scale services while continuing the field experiments and we will continue to research and develop the basic technology needed for new lifelog-based services.

Tomohiro Hasegawa
Research Engineer, Open Source Software Computing Project, NTT Cyber Space Laboratories.
He received the B.E. and M.E. degrees from the University of Tsukuba, Ibaraki, in 1995 and 1997, respectively. Since joining NTT in 1997, he has been engaged in R&D of XML and databases, high-availability systems, and distributed data processing systems. He is a member of the Database Society of Japan (DBSJ).

Ichibe Naito
Business Network Services Division, NTT Communications.
He received the B.E. and M.E. degrees from Waseda University, Tokyo, in 2004 and 2006, respectively. Since joining NTT in 2006, he has been engaged in R&D of integrated information management. He is a member of DBSJ. He moved to NTT Communications in July 2010. The work reported in this article was performed while he was working in the Open Source Software Computing Project, NTT Cyber Space Laboratories.

Takashi Menjo
Open Source Software Computing Project, NTT Cyber Space Laboratories.
He received the B.E. and M.E. degrees from Nagoya University, Aichi, in 2006 and 2008, respectively. Since joining NTT in 2008, he has been engaged in R&D of integrated information management. He is a member of DBSJ and the Information Processing Society of Japan (IPSJ).

Motohiro Matsuda
Open Source Software Computing Project, NTT Cyber Space Laboratories.
He received the B.E. and M.E. degrees from Aoyama Gakuin University, Tokyo, in 2007 and 2009, respectively. Since joining NTT in 2009, he has been engaged in R&D of integrated information management. He is a member of DBSJ and IPSJ.

Hiroki Akama
Senior Research Engineer, Supervisor, Open Source Software Computing Project, NTT Cyber Space Laboratories.
He received the B.E. and M.E. degrees from the University of Tokai, Kanagawa, in 1988 and 1990, respectively. Since joining NTT in 1990, he has been engaged in R&D of database and multimedia information retrieval and integration. He is a member of the Association for Computing Machinery.

Masashi Yamamuro
Senior Research Engineer, Supervisor, Group Leader, Open Source Software Computing Project, NTT Cyber Space Laboratories.
He received the M.S. degree in mathematics from Waseda University, Tokyo, in 1987, the M.S. degree in electrical engineering from Columbia University, New York, in 1991, and the Ph.D. degree in electronics, information, and communication engineering from Waseda University in 1999. Since joining NTT in 1987, he has been engaged in R&D of database integration, data visualization, multimedia information retrieval, and data stream management systems. He is a senior member of the Institute of Electronics, Information and Communication Engineers of Japan and a member of IEEE Computer Society and IPSJ.