Lifelog Mashup with Implementations of Various Databases

Luz Lopez
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
lopez477morris.umn.edu

ABSTRACT

This paper will introduce readers to the commonly used activity of *lifelogging*. Many of our lifelogging activities have become a norm and we do not even realize where a lot of information about our behavior is going and where it is being stored. In this paper I will mainly focus on the comparison of two implementations of databases that manage the data provided by lifelog services. Limitations and implementations will be discussed and evaluated for each implementation. Lastly, there will also be mashup application experiments conducted for each that demonstrate the feasibility of the two different databases being compared. In order to achieve these implementations the reader will be introduced to a lifelog common data model and a lifelog API. Other terms and tools will be explained in the process of learning about these two implementations.

Keywords

Lifelog mashup, query, LLAPI, LLCDM, MySQL, NoSQL

1. INTRODUCTION

Lifelog is a social act to record and share human life events in an open public form [1]. There are various services provided on the Internet as well as devices that allow for people to record their daily activities. Lifelogging technology include wearable devices and mobile applications that track our steps, heart rates, diets, check-in at specific places, or even devices or systems that track health related information about ourselves. Some examples of these technologies are Twitter, Digifit, the Fitbit, HonestBaby, and many others. We may not even realize how much of a norm these things have become in our lives.

As various services collect information we may not know where all this newly collected information is being stored, nor how it's being managed and returned to us, the users. Sometimes two or more of these services merge together and create a new service that implements a combined platform and returns valuable data visualizations or a service to interact with. This is known as a *lifelog mashup platform* which was created to bring all of it together, and allow for users to enjoy more sophisticated features that would be more useful if it was together rather than all separate. As a more formal definition, a lifelog mashup platform is an integration of lifelogs to create a value-added service[1].

Having a tool that allows you to count calories or check heart rate when working out can help you achieve more goals than if you were unaware of this information. As mentioned briefly earlier, lifelog has a lot of information about users all over the place, whether that be in their companies' databases or servers, or if it's across the globe being stored in a random database, lifelog can only be helpful if users can read and analyze their own data. The amount of data generated is growing due to more users being involved in lifelogging activities, and more devices or services collecting lifelog data, therefore there's more data that has to be organized, processed, analyzed, and finally displayed. Takahashi et al [1,2], from Kobe University, initially developed a system with an XML platform, was later improved with a MySQL RDB implementation, and lastly improved with a NoSQL MongoDB integration. Their goal was to support developers by providing a set of tools to make the creation of LLCDM more efficient. In this paper I will explain three of their studies, but put a heavy focus on the two last ones.

There are various parts to how lifelogging works, such as the actual functionality and features of certain services and devices, the data gathering, the platform of which the data is being managed, and finally, how the data is being presented to the user through *mashup applications*. Mashup applications are applications created by developers that actually demonstrate how multiple services work together and create visuals for the user. In this paper I will focus on the data gathering process and on the platforms on which the data is being managed. Like I mentioned before, there were three different platforms that Takashi et al [1,2] developed, implemented and tested. I will explain the limitations of each one, and go over more detail about the implementation of the two last databases used for each experiment. I will first present you with the limitations on the XML platform to understand the importance of the upgrade to MySQL. Then I will explain the process of how MySQL was implemented, show the evaluation process along with the TabetaLog mashup application developers created and conclude with some of the limitations MySQL could not resolve. A similar process will be presented for NoSQL implementation, the only difference is their mashup application was a different which was creating a SensorLoggingService. With all these implementations of different databases to the LLAPI, they found that in most cases the use of LLAPI took fewer lines of code than the previous version, and in all cases took less time for development

UMM CSci Senior Seminar Conference, November 2017 Morris, MN.

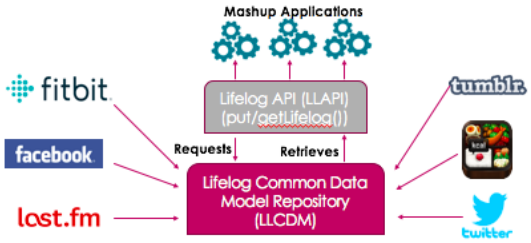

Figure 1: Lifelog mashup architecture

2. LIFELOG MASHUP

As mentioned before, lifelog mashup was created to organize and make all scattered information be in one location to improve the accessibility. First I would like to introduce a few things that pertain to lifelog mashup. Figure 1 shows several important parts of Lifelog Mashup and it is to help you understand how parts are connected. I have selected this set up because it is the closest to the kind that Takashi et al. [1] have used for their implementations. We can see some examples of the services and devices given in section 1 on the far left and far right of the architecture. Lifelog mashups store all information from the services in databases that then retrieve the data and organize it and display in a way that its users can understand and analyze it however they please. Since all of the data is public, some lifelog services are already providing APIs and/or blog components to access the lifelog data according to Takahashi et al.[1].

In order to store large amounts of data and improve the efficiency of the platform, *lifelog common data model (LLCDM)* has been proposed (see figure 1 where LLCDM is shown as the pink block). The LLCDM is designed based on an interrogative analysis, deriving standard attributes from viewpoints of what, why, when, who(m), where, and how *application-neutral* attributes, which are commonly interpreted among different lifelog services [1]. The LLCDM is responsible for collecting the data using those attributes. It is in the analysis service where the LLAPI will do its work and process the data through the database and then display it.

I will explain how the LLCDM retrieves the data then puts it into the database in the next section. This was a simple explanation to have a basic understanding of how things are connected before I go into detail on the individual experiments and the LLAPI.

3. COMMON LIFELOG DATA MODEL (LLCDM)

It is important to understand why an LLCDM platform is helpful. Figure 2 is an example of two sets of data records, one for Twitter and the second for a Sensor Logging Service. We can see how the format for these two data sets are different. Specifically time and date. The common data model eliminates such problem, taking data with various formats, and transforming it to the LLCDM format and then storing it as an raw XML file. When the data is all stored in the same format, it is much easier to create attributes with the viewpoints mentioned in section two.

The following is the process on how to import data into


```
{
 "created_at": "Fri Jul 05 03:45:35 +0000 2013",
 "id": 353155450876854300,
 "text": "It's sunny, but Im writing a paper in my room...",
 "source": "<a href="http://krile2.star....Krile2</a>",
 ....
 "user": {
  "id": 48441032,
  "screen_name": "koupetiko",
  ....
 "geo": {
  "type": "Point",
  "coordinates": [ 34.72579115, 135.23622368 ]
 },
 "coordinates": { ....
}
```


(a) A data record of Twitter


```
{
 Time: "12:46:57",
 ....
 User: "koupe",
 Weather: "Sunny",
 TempF: 76.73,
 Humidity: 18,
 Brightness: 310,
 Temperature: 26.5,
 ....
 Id: 238361,
 Date: "2013-07-05",
 Light: 81,
 Logger: "PerlSensorLoggerClient.pl",
 Host: "cs27hnspc", ....
}
```


(b) A data record of Sensor Logging Service

Figure 2: Data of two different lifelog services

the LLCDM. First obtain original data from heterogeneous lifelog services and store all data in an XML file. Then transform the data to LLCDM platform. In this step according to Shimojo et al. [1] they map the original data to each item of the LLCDM, based on a *conversion rule* defined for each individual service in [3]. For the last step, the XML file is then parsed, extracts the attributes and inserts the data into the appropriate tables.

4. TOWARD SUPPORTING EFFICIENT DEVELOPMENT OF LIFELOG MASHUP APPLICATIONS

It is important to understand the differences of the three different designs for the mashup platform, figure 3 shows the order in which they were designed. The first design was with an XML platform. This platform was a first step and had two major limitations. Although I will not go into detail on this implementation due to greater progress of the two last ones, it is important to understand how these limitations lead to the implementation of the next experiment. The limitations for this experiment were that it could not specify application-specific attributes for a data query and it had a scalability problem. When they found these problems for this experiment they re-designed the platform using MySQL, a relational database (RDB). They then re-designed creating the last and third platform implementing it with a NoSQL Mongo database.

I will introduce each experiment in its own section followed by their corresponding evaluation application and lastly with the limitations briefly presented in this subsection.

Figure 3: Order of experiments

5. MYSQL WITH RBD IMPLEMENTATION, EVALUATION, AND LIMITATIONS [1]

5.1 Introduction to the RBD Design

This design seeks to fix the limitations of the previous XML platform. As a reminder the two major limitations were, poor portability and low performance. The poor portability issue came from the developers using Perl as the development language, due to this developers had no other choice than to use Perl for building mashup applications. The low performance limitation was due to the LLCDM repository being just a file system containing converted data as raw XML file causing an overhead [1]. To overcome these limitations Shimojo et al [1] set two goals:

G1 Improve performance and portability of the prototype

G2 Evaluate the practical feasibility of the LLAPI

To achieve goal 1, they would put all the lifelog data in a relational database, instead of having the data as a raw XML file. This would improve a faster search and accessibility for the data. To achieve goal two, they would conduct an application development experiment. In the experiment, they ask subjects to implement two versions of a mashup application, with and without the proposed LLAPI and the LLCDM [1].

5.2 Evaluating Performance

Shimojo et al. [1] conducted an experiment. They compared the old prototype to the new implementation. They evaluated a total if 1,591 lifelog data records. They then store the data in the MySQL database or in a XML files. They developed Perl clients to invoke the LLAPI [1] with four different queries. The first one was for lifelog data for a full day, second query was for 30 days, third query was for an hour and 15 minutes, and the fourth query was all lifelog data for one user. Figure 4 is the table with the execution times. The experiments used SOAP and REST web protocols for the new LLAPI implantation. "Old" refers to the old implementation. The time of each execution is seconds, and we can clearly see the reduction in time for SOAP and REST compared to the old implementation.

	November 15-November 16	September 1-September 30	9:00:00-10:15:00 On any date	User – "Shimojo"
SOAP (sec)	0.131	1.006	0.281	0.422
REST (sec)	0.015	0.100	0.019	0.025
OLD (sec)	4.238	4.028	4.254	0.581
# OF ITEMS	36	119	195	449
DATA SIZE (kB)	118	381	1,450	630

Figure 4: Execution times

6. MASHUP EXAMPLE: TABETALOG

An example of how data from multiple lifelog sources was brought together to create something manageable to understand for the user is the TabetaLog application [1]. It was created to evaluate the feasibility of the LLAPI. Which if you recall is the second goal in section 5.1. The table shows an example of two lifelog service used. One was Flickr and the second was BodyLogService, one containing picture data and the other containing weight data.

6.1 Steps to create TabetaLog[1]

1. Obtain original lifelog data from LLAPI
2. Extract necessary data by *parsing*, breaking data blocks into smaller chunks so that it can be more easily interpreted.
3. Combine data items that are then dumbed into a JSON file.
4. Visualize the JSON data using ActionScript

Figure 5: Example of a TabetaLog

In figure 5, I have replicated the idea of what the TabetaLog should look like. The purpose of this mashup application is to combine two web services that are different in functionality, Flickr and FoodLogService. Flickr is a web service that provides a public platform for sharing memorable photos and videos. In contrast, the FoodLogServices application are used to track daily caloric intake by storing large amounts of data regarding nutritional information of numerous foods in one easy-to-access, easy-to-use application. The Tabetalog works to combine data from these two different applications to create a new, more sophisticated platform for the user to evaluate the changes in weight gain based on the type of food consumed. The idea is for a user to take daily photos of all meals for one year, upload the images to Flickr and for each photo, record their weight on a FoodLogServices application. The Tabetalog gives users a new opportunity to visualize their meals and how their food

choices affect their weight. Combing Flickr and Foodlogservices allows users to more accurately track the effectiveness of a new diet on weight loss.

In this experiment they asked five developers to create the TabetaLog that had the similar features to what I replicated. They had to use the new LLAPI and the conventional (conv) LLAPI, which is simply what they named the old LLAPI. Each developer was assigned to create the Tabetalog using the conventional or new implementation first. Figure 6 shows a table with the results of the experiment.

One again proving that the time and lines of code to create the mashup application takes a lot less time and effort to develop with the new implementation.

7. NOSQL IMPLEMENTATION WITH MONGODB, PROCESS, EVALUATION, AND RESULTS

7.1 Reasons to Choosing MongoDB

MongoDB is a schema-less document orientated database developed by 10gen and is open source [1]. The following list shows the advantages and disadvantages of MongoDB for lifelog mashup. It is important to understand how and why some of the functionalities might be helpful for the implementation of lifelong data.

Adv 1: **Document-oriented storage** meaning MongoDB stores data in BSON (binary JSON) objects that are binary encoded JSON like objects.

Adv 2: **High Availability and Easy Scalability** Mongo supports the distribution of documents over servers. It also supports replication with automatic failure and recovery as well [1].

Adv 3: **Full Index Support** MongoDB indices are explicitly defined using an ensureIndex call and any existing indices are automatically used for query processing [1]. This means that it looks for data with certain properties rather than looking through entire documents that can have many records with the same attributes. This makes queries more efficient.

Adv 4: **Supports MapReduce** MapReduce is supported by MongoDB. This is significantly important since MapReduce is a programming model for processing and generating large datasets, that is amenable to a brand variety of real world tasks [3].

As good as these advantages might seem, we also have to take in account the characteristics that might not be as helpful for lifelog platform:

Con 1: **No Transition** MongoDB has no version concurrency control, which is an advanced technique for improving databases performance in a multi-user enviorment.

Con 2: **No JOIN support** If some of the data has joins, columns with similar data, it will combine and store the data into one document to remove the joins.

Con 3: **New technology** MongoDB has only been around since about 2009. It does not have much history and it is constantly releasing new versions.

It may not be clear yet why these pros and cons demosnstrate the effectiveness of MongoDB to be implemented with lifelog-mashup. In the next section I will explain its validity and why it was chosen by Takahashi et al[1] implement MongoDB with lifelog mashup in the next few sections.

7.2 Modeling LLAPI with MongoDB

"Application information", in other words an attributes, will contain things that pertain to the service or device that is collecting lifelog information. In the RDB (MySQL) "Application information" was the label for part of the database that stored application-specific information. It was a catch all for information that did not have a particular place to go. Due to the way that information was stored, it was difficult to extract the application-specific information from a query. Instead, the program developer would need to get a lot more information from the database and parse within the application. Since MongoDB is able to create more attributes that can be more specific to data such as temperature, time, date, etc., then they don't have to dump many extra unrecognized records. For example, knowing the application is Twitter rather than Bodylog, we will know what kind of information to retrieve or what to look for more specifically. With MongoDB we can create more attributes that can become more specific to more services. Figure 7 shows the new attributes as the blue, and the removed as the red attribute.

Username information is very important because this is how lifelog information will be attached to the user. These are things common services will request to create a user profile. This information is much needed to connect users with proper lifelog information they are looking for.

The important information to have is the one describing the lifelog. Although this information will be different depending on the service, like application information, it would be the most common information asked by many services. For a location based service (Twitter, Flickr, applications with weather settings, etc.) it will request a specific location. This information is typically not requested from the user but from the service. All other information is used for the efficiency the user is seeking for.

7.3 Implementation of LLAPI with new Platform

[1] developed a new LLAPI consisting of two methods: putLifeLog() and getLifelog(). The putLifeLog() stores a lifelog data into the LLCDM, and getLifelog() method retrieves lifelog data from the LLCDM depending on the given query.

[1] Implemented the LLAPI with Java. Morphia ORmapper was used for *marshaling tuples into objects*, which simply means to take data and transform the components stored in memory of a object into a data format suitable for storage or transmission. The advantage of using the ORmapper is to decouple the SQL statements from the Java code, which improves the robustness and the maintainability of the system [2]. The LLAPI was then deployed as a web service, which has many advantages being that it sets the LLAPI free from the artificial dependencies of the language and platform [2]. They deployed the LLAPI using the Apache Axis2 web service framework, which allows both web service protocols, SOAP and REST, to access the LLAPI. Web service protocols are just a standard communication set of rules specification for XML-based message exchange

Programmer	1		2		3		4		5		Correct	
System used first	*LLAPI*		*Conventional*		*Conventional*		*Conventional*		*LLAPI*		-	
	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}
Programming Language	Perl	Perl	Perl	Perl	Java	Java	Java	Java	Java	Java	-	-
Source lines of code	115	365	227	379	480	612	423	397	150	181	-	-
SLOC (w.out blank and comments)	71	223	103	188	351	426	286	263	106	125	-	-
# of source-code classes	-	-	-	-	7	7	5	5	2	2	-	-
# of source-code files	1	4	1	3	-	-	-	-	-	-	-	-
Man-hour (man minute)	114	196	54	205	96	252	147	514	132	397	-	-
# of weight records <Shimojo>	53	54	53	54	32	53	53	54	52	52	53	54
# of weight records <Togunaga>	102	101	102	101	52	103	103	104	103	115	102	101
# of picture records <Shimojo>	8	9	8	9	8	9	8	9	8	9	8	9
# of picture records <Tokunaga>	85	86	85	86	60	87	85	44	65	85	85	86

Figure 6: Results of the TabetaLog Experiment

```
getLifeLog([s_date, e_date, s_time, e_time, s_term, e_term, user,
party, object, s_alt, e_alt, s_lat, e_lat, s_long, e_long,
loc_name, address, location, application, device, content, se
limit, order, offset])
```

Figure 7: New and removed attributes

[5]

8. EVALUATING PERFORMANCE

Takashi et al. [2] developed an experiment using environmental sensor log from SensorLoggingService, deployed in their smart home. This service measures the environment inside/outside of their laboratory using various sensors including temperature, humidity, brightness, pressure, motion, and the number of people. The sensor has recorded every minute for three years, a total of 1,664,937 entries. Records are then imported to new(MongoDB, NoSQL) and old(RDB, MySQL) platform. A client application was developed where it picks out summery days, which means a day that between 9 AM and 6 PM, the maximum temperature exceeds 25 degrees Celsius. Figures 8 and 9 show the results. We can see that the newer system, LLAPI, takes roughly the same amount of time to retrieve data for most any size window of time, as opposed to the old implementation taking a lot more time with more data.

Taking a closer look at the graphs, we can see how the number of records being retrieved are so large for the old LLAPI, in part because the data in in "application information" and almost all the records need to be retrieved and then passed by the system rather than being able to do an efficient query with access to more attributes.

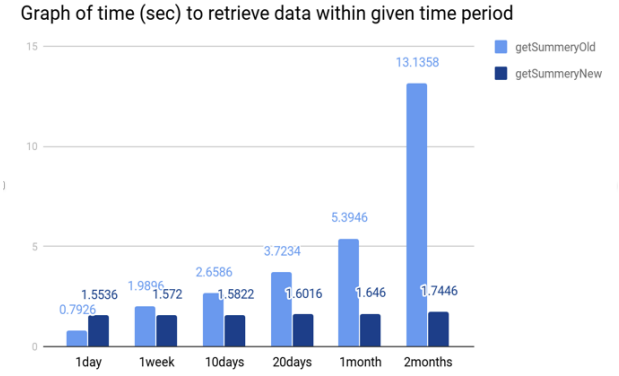

Figure 8: Graph for execution times

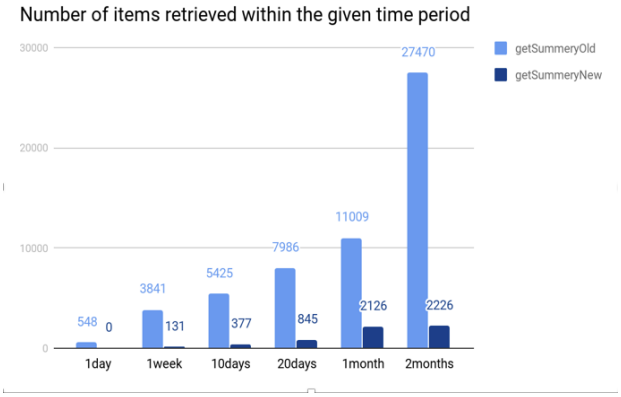

Figure 9: Graph for number of items

9. CONCLUSION

Lifelogging is a way of recording everyday activities. With more services that allow for lifelogging as well as more users using these services and devices, data continues to grow. With large amounts of data, there needs to be a system to process these large amounts of data efficiently and in an agile form. As I presented in this paper, Takahashi et al. designed and successfully implemented various databases to improve

the feasibility to create mashup applications by developers.

As a summarization of their multi-step experiment to design a lifelog common data model and an lifelog API platform to create mashup applications, we can see the improvements of each implantation for applications to gain access to specific data from different databases.

[1] [2] [3] [4]

10. REFERENCES

[1] J. Dean and S. Ghemawat. Mapreduce: A flexible data processing tool. *Commun. ACM*, 53(1):72–77, Jan. 2010.

[2] R. Padhy, M. Ranjan, P. Suresh, C. Satapathy, and O. India. Rdbms to nosql: Reviewing some next-generation non-relational database's. 10 2017.

[3] A. Shimojo, S. Matsumoto, and M. Nakamura. Implementing and evaluating life-log mashup platform using rdb and web services. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '11, pages 503–506, New York, NY, USA, 2011. ACM.

[4] K. Takahashi, S. Matsumoto, S. Saiki, and M. Nakamura. Design and evaluation of lifelog mashup platform with nosql database. In *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, IIWAS '13, pages 133:133–133:139, New York, NY, USA, 2013. ACM.