Research Paper E-ISSN No : 2455-295X | Volume : 2 | Issue : 6 | June 2016

BANKING ANALYSIS PORTAL

Rahul R. Nair [1] | Rohan E. Mhetre [1] | Avinash D. Zagade [1] | Ganesh S. Thorat [1]

Dept. of Computer Engineering, JSPM's Imperial College of Engineering and Research, Wagholi.

ABSTRACT

There are very less systems in the Application Market which connects the different banks to access the transactional data (i.e. credited and debited) of the user. A user may have many different bank accounts for which the user may find it difficult to keep a track of all the transactions he had performed from different accounts and so the user finds it difficult to organize his expenditures and earnings at the end of the month. The proposed system organizes all together expenditures and earnings of the user from various accounts.

Distributed Data Aggregation Service (DDAS): as the name itself suggests that it is nothing but aggregation of data. In simple words, the data that is present at different locations are merged together in order to achieve greater speed, security and flexibility. There are several systems for database management and one of them is Distributed data aggregation service (DDAS) system which is relying on Blobseer. It is found that it provides a high level performance in aspects such as data storage as a Blob (Binary large objects) and data aggregation. For complicated analysis and instinctive mining of scientific data, Blobseer serve as a repository backend. In this paper we review the different aspects regarding Distributed data aggregation service (DDAS) and different approaches and case study regarding it and in which aspects it is useful over the globe. The main purpose of this system is to provide the security to user sensitive data. This can be achieved applying security to three different connections of the system namely, User-API, API-Bank, API-Database using different algorithms like DDAS, Hash Coding, etc.

KEY WORDS: Distributed Data, Blobseer, BLOB (Binary Large Objects), Aggregation.

I. INTRODUCTION

We have seen an era where people have no time to calculate their expenditure and earnings. But most of us would like to keep a track of their expenditure and earnings so as to plan their savings accordingly. As when people have no time to do it themselves, but they need it. So who on earth will give their earning history to someone else to keep a track? This question arises in minds of most of us. We also have a culture of using internet for our daily use activities. We are involved into internet in such an extent that we find it difficult to go out and buy our food from outside instead ordering our food online would be a better choice. So can't we use internet to be a friend who helps us in keeping a track of expenditure and earnings so as to simple our life? This project report presents a formal model of a portal that connects all the banks in a specified country and provides these services to the users of the application in that country without compromising in security.

This illustration aims to get an idea about different security algorithms which can be used for a system which connects the different banks to access the transactional data (i.e. credited and debited) of the user. A user may have many different bank accounts for which the user may and it difficult to keep a track of all the transactions he had performed from different accounts and so the user finds it difficult to organize his expenditures and earnings at the end of the month. The proposed system organizes all together expenditures and earnings of the user from various accounts.

The main purpose of this system is to provide the security to user sensitive data. This can be achieved applying security to three different connections of namely, User-API, API-Bank, API-Database using different algorithms like DDAS, Spring Security, Hash Coding, etc.

II. Proposed System

The architecture of DDAS is made of several components each with a well-defined purpose to ensure the fast processing of a request[1]. Application requests can either require a large number of objects or huge data size for storage and retrieval operations continuously; therefore we need to efficiently isolate the system's functions when it comes to computation, storage, selection and metadata management [2].

A. Data Back-end Storage System (BlobSeer)

This is the actual BlobSeer storage system. It only communicates with the DDAS' extended client through read and write requests to store and retrieve specific objects in their serialized form therefore aggregating the distributed data [5]. For write operations, the meta- data manager returns the meta-information to identify the location of the written objects, while for read operations it returns the serialized object based on the meta-information input by the DDAS extended client. As it is the storage system for the DDAS it is not viewed to the client and is encapsulated from the user[3].

B. DDAS

This component acts as the mediator between BlobSeer and all other data-management applications that require fast and reliable storage and retrieval of data or simply the client or user. It is mainly divided into two layers: the metadata management layer and the extended BlobSeer client. The upper layer uses the Collect Gate module to update its scheme-object mappings and performs the operations requested: either storing the serialized object into BlobSeer or retrieving the mapping of an aggregation scheme from BlobSeer and delivering it to the client. For an input aggregation scheme, the DDAS will request from Collect Gate the list of object keys that fit the aggregation scheme [4]. Using the keys or the existing aggregation scheme the service will send to the extended client the meta-information based on which the retrieval of data from BlobSeer will be made and the client gets the data.

C. Collect Gate

This module is represented by a database of rules and properties that, depending on the application, generates expansion and aggregation schemes that fit one or more objects thereby improving performance of the system which in turn is useful to the user. Furthermore, Collect Gate creates lists of object keys that map to a scheme to send them to the DDAS in order to fulfill new requests [4]. This model therefore gives more flexibility compared to others. The role of this component is to perform quick searches and selections based on a few properties therefore increasing speed. As storage is required for new objects collect gate manages that and gives the required retrieval.

D. Data Flow

This block is used when a request is made by a client to store data or retrieve a set of objects based on an aggregation scheme. For a new object the DDAS queries Collect Gate for all the existing schemes that fit the new object. For read and write BlobSeer is responsible as BloobSeer gets the read operation it searches it in Blob and does the necessary action for write it executes the write operation and returns the corresponding meta-information to the DDAS. For an aggregation operation it requires a client application to input an aggregation scheme in its request to the DDAS. If this aggregation scheme is not already in the metadata management layer of the DDAS, a request to Collect Gate is made to determine the list of object keys that match this scheme. After Collect Gate returns the list or if the scheme already exists in the DDAS, the extended BlobSeer client will read the objects based on the meta-information of each object from the object catalogue or meta-information mapped to the existing aggregations scheme [4]. BlobSeer will then read all requested objects and return them to the DDAS. Finally the DDAS will perform any reduction operations that are included in the aggregation scheme. The list of objects will then be transmitted to the client application in the same format as it was first stored.

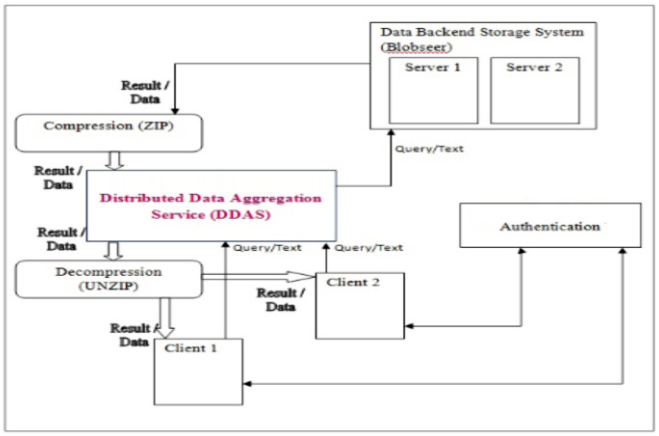


Figure 1. DDAS system architecture [4]

III. Algorithm

J2EE frameworks have relentlessly moved into application frameworks, which target to provide coherent programming in all tiers and thereby amalgamate the application stack. The Spring Framework is the foremost product in this space, with embracing comparable to that of Hibernate. Spring provides an alternative to EJB in many applications—for example, delivering declarative transaction management to any POJO. The Spring approach has proven to deliver excellent results in many kinds of projects, from small Web applications to large enterprise applications. Collectively, these products are referred to as lightweight containers to distinguish them from traditional J2EE approaches. By decoupling a POJO model from J2EE APIs, which are hard to stub at test time, lightweight containers greatly simplify unit testing. It's possible to unit test in a plain JUnit environment, without any need to deploy code to an application server or to simulate an application server environment. Given the increased and deserved popularity of test-driven development, this has been a major factor in lightweight frameworks' popularity.

Using J2EE we have designed banking analysis application which works as follows:-

Step 1: Customer creates account in multiple bank applications that maintains its transaction details.

Step 2: The same customer registers itself in banking analysis application.

Step 3: The customer adds his/her different bank account details that has to be aggregated into the banking analysis application.

Step 4: If the number of bank accounts added by the customer is greater than 0 then the bank analysis application verifies each and every detail with its corresponding banks and validates it in the system.

Step 5: The application maintains a separate table in the database for each customer registered.

Step 6: Using DDAS the application retrieves JSON objects from the corresponding banks and produces it to the user.

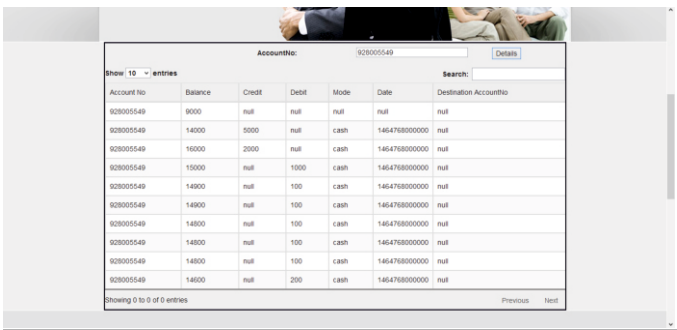
Step 7: Every time the user logs into the banking analysis application the credentials are authenticated by the server.

Step 8: The report is generated in a format that can be directly copy pasted into excel sheets for performing complex arithmetic operations.

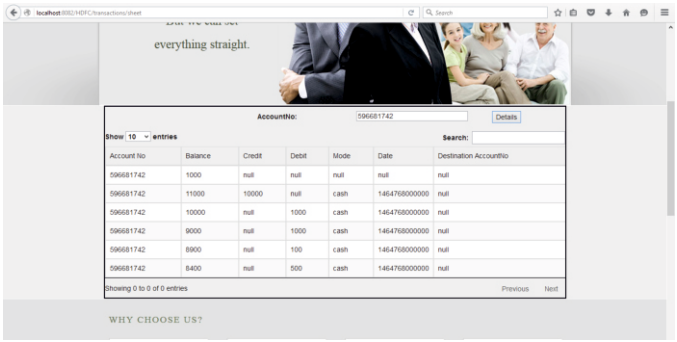
Step 9: In this way the user is able to get an idea about his/her expenses and can control them accordingly.

IV. Results

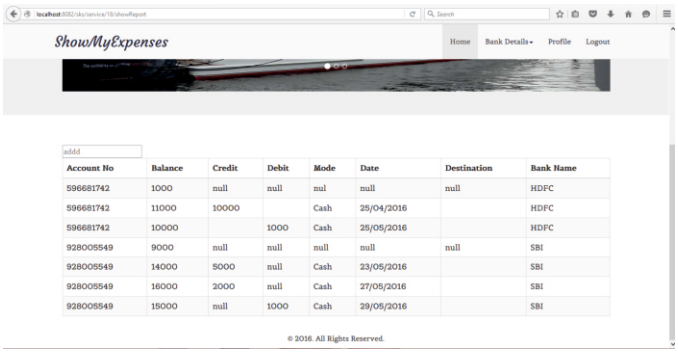
1. The first screenshot suggests user's first bank account details.



2. The second screenshot suggests user's second bank account details.



3. The third screenshot shows the aggregated data of both the user accounts from different banks.



V. Conclusion

So as we see there is a vast need for data management and aggregation over the globe, based upon various domain and researches. Moreover the requirements have become more specific and complex to be considered hence a solution is required which yields high performance for large scale distributed data and various data centric applications. So to overcome this we have Distributed Data Aggregation Service (DDAS) relying on BlobSeer. DDAS ensures a high level of performance in all aspects such as data storage and aggregation solution, which has been explained based on the model.

So this system aims to provide a secure distributed data aggregation using DDAS, this helps to check all the transactions at a go thereby saving a lot of time. This also gives us an idea about where and how we are spending our money and we can accordingly manage our transactions. Single aggregated approach is more compatible than the previous systems and easy to use, the main issue is to secure this system using various security algorithms. Later on it generates a report and gives you the details about the credited and debited transaction about different banks of different accounts.

VI. REFERENCES

1. S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of data grids for distributed data sharing, management, and processing," ACM Comput. Surv., vol. 38, June 2006.
2. T. Glatard, J. Montagnat, and X. Pennec, "Efficient services composition for grid-enabled data-intensive applications," in *Proceedings of the IEEE International Symposium on High Performance and Distributed Computing*, Jun. 2006, pp. 333–334.
3. B. Nicolae, G. Antoniu, L. Boug´e, D. Moise, and A. Carpen-Amarie, "Blobseer: Next-generation data management for large scale infrastructures," *J. Parallel Distrib. Comput.*, vol. 71, pp. 169–184, February 2011.
4. Florin Pop, Gabriel Antoniu, Vlad Serbanescu, Valentin Cristea, "Architecture of Distributed Data Aggregation Service" in *Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications*, IEEE Computer Society, 2014.
5. G. Antoniu, L. Boug´e, D. Moise, A. Carpen-Amarie, and B. Nicolae, "Blobseer: Next-generation data management for large scale infrastructures," Author manuscript, published in "Journal of Parallel and DistributedComputing"71,2(2011).
6. A. Brampton, A. MacQuire, I. A. Rai, N. J. P. Race, and L. Mathy, "Stealth distributed hash table: a robust and flexible super-peered dht," in *Proceedings of the 2006 ACM CoNEXT conference*, ser. CoNEXT '06. New York, NY, USA: ACM, 2006, pp. 19:1–19:12.
7. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, pp. 4:1–4:26, June 2008.
8. M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?" in *Proceedings of the 2008 international workshop on Data-aware distributed computing*, ser. DADC '08. New York, NY, USA: ACM, 2008, pp. 55–64.
9. W. Hummer, P. Leitner, and S. Dustdar, "Ws-aggregation: distributed aggregation of web services data," in *Proceedings of the 2011 ACM Symposium on Applied Comput-*

ing, ser. SAC'11. New York, NY, USA: ACM, 2011, pp. 1590–1597.

10. J. Chen, S. Sehrish, W.-K. Liao, A. Choudhary, and K. Schuchardt, "Improving the average response time in collective i/o," in *Recent Advances in the Message Passing Interface*, ser. LNCS 6090, 2011, pp. 71–73.

11. Ashok Vemuri, Merlyn Mitra, Anand Bhushan. "Case Study: NextGen Client Data Aggregation and Reporting."