World of Computer Science and Information Technology Journal (WCSIT)
ISSN: 2221-0741
Vol. 5, No. 8, 135-141, 2015

Free Scale and SOLR Architectures for Load Handling of Social Networking Sites

Nirmal Dhara
Department of Computer Science
Christ University
Bangalore, India

Shashank K
Department of Computer Science
Christ University
Bangalore, India

Abstract— The social networking sites are adding enormous load to the servers. Balancing such an increasing load for faster response is really challenging. We propose a cost effective, loosely coupled and highly scalable middleware architecture, Free Scale Architecture (FSA), which helps the sites to handle increasing load with minimal changes to its fundamental architecture by dividing the load among several servers. FSA uses Classification Data Engine (CDE), Tiny Intelligent Agent (TIA), Logical Router (LR), Dynamic Bottleneck Tracker (DBT) and SOLR as part of the architecture which helps fetch data and classify, into clusters, shrink Master and Slave data, balance the increasing load, help find the reason of any bottleneck with balancing and routing the load among servers, and cache the most frequently used data for faster retrieval, respectively. Further, CDE requires starting a separate thread to copy the clusters into a local domain, and this is known as parallel process. The parallel process saves nearly 80% of the time. We also discuss the implementation of FSA and SOLR architecture for social networking sites, and how social networking sites can use the FSA and SOLR architecture to increase scalability and handle the unexpected load.

Keywords-Free Scale Architecture (FSA); Logical Router (LR); Classification Data Engine (CDE); Social Networks (SN); Failure Management (FM); Dynamic Bottleneck Tracker (DBT); Simple Failure Handling (SFH); Permanent Failure (PF); Temporary Failure (TF); Critical Failure (CF); Tiny Intelligent Agent (TIA).

I. INTRODUCTION

Social networking sites are the most common sites and tools for many, especially, the younger generation. These sites, sometimes, also act as marketing tools. The number of transactions per second of some social networking sites is much higher than traditional web applications. There is still an enormous potential for growth among these social networking sites in terms of users and data. Social networking sites handle loads of data about the users and their usage patterns. They also handle a lot of personal information data. The major challenge for social networking sites is to balance the ever growing data load due to the exponential usage of those sites. The ever growing increase in the number of registered users, in many cases, has failed the servers in providing performance and reliability.

User performance is the key to the social networking sites' better performance. Social networking sites are generally deployed as a cluster of systems. It becomes difficult to handle the rapid growth of data if there is no architecture to support a high performance. There are available solutions to handle the rapidly growing load by means of increasing hardware resources. But the approach is expensive. The total number of monthly active users of Facebook as of January 2015 is 1.35 billion. The sudden increase by 1382% of Twitter users in 2009 was a difficult task to handle. Twitter changed their architecture several times mostly to manage the increase in active users. Many social networking sites are obsolete as they were not able to handle the sudden load of increasing users. These examples project scalability issues. The main aim of the current work is to increase throughput and data load.

The main purpose of increasing the scalability is to accommodate the changes in the data growth. Scaling an existing system will be more difficult than a new system. The easiest way of scaling an existing system is by adding more hardware. Social networking sites, as big as Facebook require few hundreds of terabytes of memory across thousands of servers. To overcome this problem, adding an extra layer on Free Scale Architecture (FSA) will cache the data from the main system for local users. FSA is a middleware which will divide the load among several servers in a cost effective manner. Caching the data will be periodically done for reading and writing the data to the main system. Extra data processing will be available in this FSA middleware.

LR in both the local and the main systems play the vital role to reduce the media load. These logical routers is configured in such a way that the routers are auto-generated as the load increases. Every logical router is asynchronous in nature. The social networks are different than the other networks since the other networks may not have the increasing

load issues. Online social network users visualize the complexity of relations and interaction. There will be a TIA which would run periodically and replicates the database to a local server. It is important to know the relationship among these tables. A few algorithms which take into account the community structures in social networking sites have been proposed [1]. This TIA will be intelligent enough such that it can copy the data to suitable local system, and there should not be any duplication of data among local systems. TIA will be multi-threaded and will not run in the main server or in the local system. There will be a separate system with multi-tasking, which will have the address of all local systems, and it will start large number of threads to complete the copy process. Scaling for sites like Facebook and Twitter is really difficult due to the unknown nature of increasing load and the status update in Twitter will be very high than any other popular SN.

Facebook requires hundreds of Terabytes of memory across thousands of machines and will cost more. As the number of users increase it will increase the internet traffic, memory and CPU usage. So the whole system becomes slow. Performance of the response and request depends on the number of users, CPU and memory.

II. Social Networking Model

A social network is a combination of user, links, groups and likes. The user should register to participate in a social network. Without registration, no user will be allowed to access any data from the database. A registered user may create new groups, may like the content of other users' post and will also be allowed to create several others links. Suppose any user is sharing a link for marketing purpose, then the visiting permission for that link should be set ahead of time. A user is not allowed to see any private content of the other users. Users' read permission will be divided as public, private, friends and friends of friends. They are the four permission of data reading. If any user's data is public, it will be visible by all registered users and non-registered users, as well. And the data will be available in any search engine. Private content is visible to existing users or can be accessed by other users who are authorized to access such data. For example, if the content security level is set to friends then only friends of the user will be allowed to access the data. Also if the security level of the content is set to friends of friends then all the friends of a friend of the user is allowed to access the data. Social networking users often share the content like uploading the video and music. Most of the social networking sites like Facebook and Twitter have similar functionalities. Sometimes the social networking sites will allow the user to send the free Short Message Service (SMS) and chat with other users. Considering social networking as a graph, then graph G = (V, E) Where V will be visited users and E is the load. SN graphs are highly dynamic as users join frequently, and it increases the density.

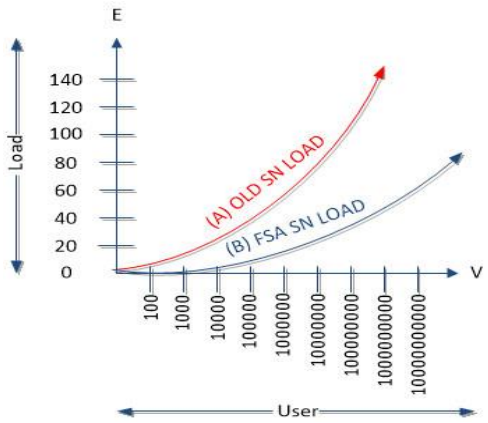

Figure 1. (A) SN without FSA load increase rapidly on the basis of increasing users, (B) SN used FSA, load increase slowly on the basis of increasing users.

III. Contributions In FSA

The main contribution of FSA is design and implementation. Dividing the main architecture into several middleware's will help the social networking sites to reduce the network traffic and accelerate the performance.

IV. Function Of FSA

FSA reduces the new social networking site architecture designer's time. Designers can concentrate on designing light weight User Interface and modules that can be added on the new social networking sites. Achieve faster response with low cost at quick turnaround time to solve the bottlenecks of existing social networks; there is no need to change the entire architecture. Instead, FSA needs to combine two extra middleware, TIA and LR. FSA is a new design pattern that reduces heavy load and traffic in web applications. FSA will be good practice for a new developer or an architect to start their new social networking sites. FSA will be easy and clearly understandable for any users. It will not only reduce the traffic over social networks, but it also passes the data in a secure way. It ensures data sequence and the relation among the objects during data snap shot. Clustering is another added benefit. Many new social networking sites were unable to handle the load; due to the poor load balancing, due to which many users left using the particular social networking site and moved to other.

V. Nature OF FSA

FSA solves the bottlenecks for any online web applications. It creates two extra layers called TIA and LR. The response time of the servers is slow, though all the SN has multiple numbers of servers for every country. This kind of issue occurs due to poor network routing. Most of the time routers are overloaded and may not be able to handle the load, some routers are free and some routers are overloaded. FSA uses Classification Data Engine (CDE), Tiny Intelligent Agent (TIA), Logical Router (LR), Dynamic Bottleneck Tracker (DBT) and SOLR as part of the architecture which helps fetch

data and classify, into clusters, shrink Master and Slave data, balance the increasing load, help find the reason of any bottleneck with balancing and routing the load among servers, and cache the most frequently used data for faster retrieval, respectively. Further, CDE requires starting a separate thread to copy the clusters into a local domain, and this is known as parallel process. The parallel process saves nearly 80% of the time.

A. *Logical Router (LR)*

LR transfers the load to another router whenever it is overloaded. All of the LRs have intelligent method to check the load; it always checks if the existing LR is overloaded. If the existing LR is overloaded then the method looks for nearby LR and transfer the load to another LR. If all of the LRs are overloaded, it calls for create method in order to create a new LR. However, all LRs being overloaded are a rare situation.

LR is self-created, hence the developer or architect does not have to spend time over this. It is difficult to predict which LR will take the responsibility for which request, and it will ask for which server to handle the request; hence all of these systems work independently. There will be another process which will check if more LRs are free or not utilized for a long time then it will kill the LR. Internally LR routing uses the shortest path algorithm. Each time a physical slave (node) is added to an FSA, immediately it will calculate the distance among other connected nodes. Hence, it will keep the record as long as the nodes are alive. This process helps us to reach to the destination quickly.

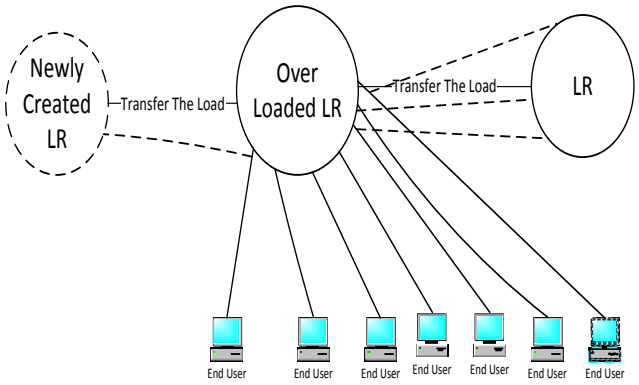

Figure 2. Shows logical router, creates new LR and transfer the load to LR.

B. *Tiny Intelligent Agent*

TIA is a dynamic backend process which helps to respond quickly. The main goal of TIA is shrinking the master and slave servers. There will be an intelligent data divider, which will take care of the logical linking among the objects when dividing the data. There will be several processes to copy the data from the master server to the slave server and vice versa. After coping, it resumes linking the data and the problem does not arise during data shrinking request. This shrinking process will happen live and to take the snap shot of the data, there will be no need to restart the server. So, the uptime of the server will be close to 100%.

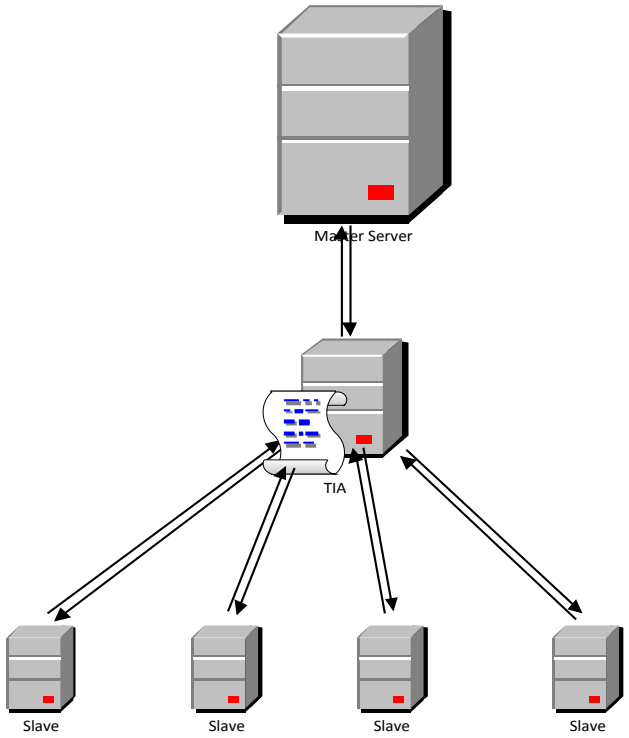

Figure 3. TIA working process using threads.

If there is a power failure for any slave servers, then TIA will work for remaining servers. It will retain the shrinking history. So the loss of data and duplication is avoided. There will be a minimum of one TIA for each slave. If the work is more and requires data copy, then automatically a new TIA will be created. There will be another type of TIA which will write the data in the master server. TIA maintains a hash table at the time of linking. It matches the hash number if all the data has been linked properly.

C. *Data Partitioning (DP)*

Data is stored in the form of a stream. Before copying the data, DP partitions the data into several clusters. Before dividing into several clusters, data relationship should be taken into consideration.

FSA uses graph partitioning to reduce the edges. Data should be in the same bucket. Whenever the data is divided into clusters, a number is assigned for every chunk and including the location. Hence, the data can be read easily. TIA may read or write the data cluster, and it will put the data in a proper place. Once the entire cluster is copied, then it will create the link among the entire data. If there are any errors during copying, TIA will register the time of linking. By this process, TIA will reinitiate the process. Hence, the performance of the process is drastically increased. Existing social network users partitioning the data for backend cleanup host the disjoint data into several servers [2].

D. *Minimize Interactions between Master and Slave Servers*

TIA always keeps the record about the master and slave. It reduces the transaction between both the servers. If there are

any updates for master server, then transaction will happen between the master and slave servers, else transaction will happen between slave and client servers. If master server is busy and one slave is writing the data into the master server, then all other servers need not wait for the writing. They will write the data into persistent data. The entire master has auto refresh functionality. Whenever TIA finds any persistent content, it will refresh the data and save it into a database.

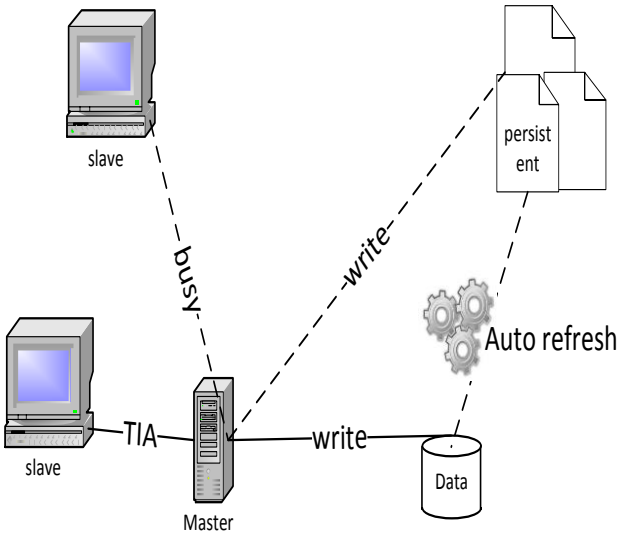

Figure 4. Master Slave data refresh using TIA.

E. *Dynamic Bottleneck Tracker (DBT)*

During runtime, DBT will always check for performance bottlenecks. If it finds any performance issue, it immediately will note the situation and send the notification to the developer about the issue. In addition to this, if the DBT finds any runtime error or situation where changes are required, it will report to the concerned team.

VI. IMPLEMENTATION OF FSA

Firstly, the requirements that FSA need to address is described. Next, we discuss regarding the problem solved by FSA. Finally, we discuss why existing social partitioning solutions will be inadequate to meet our set of requirements.

A. *Requirements*

Maintain local cache with securities: At the time of local caching, security is of primary concern. All related data should be copied from the slave server. If the nearest slave is down or there is no connectivity, it does not notify any messages. Instead, it looks for the nearest slave server. If all the slave servers are down or upon not finding any slave server, it will directly take the data from the master server. All sensitive data fields will be encrypted so the other applications would not be able to use the data though it is available locally. Encryption is the key for every application and will be unique, that is, dynamically generated for each application. There will be one auto validator which will check for the sequence of data. If all the data is copied in the correct sequence, then it is in ideal state or else, destroy mechanism will be activated and that will delete the particular portion of the data which is not copied. Hence, backup mechanism for the deleted portion is initiated.

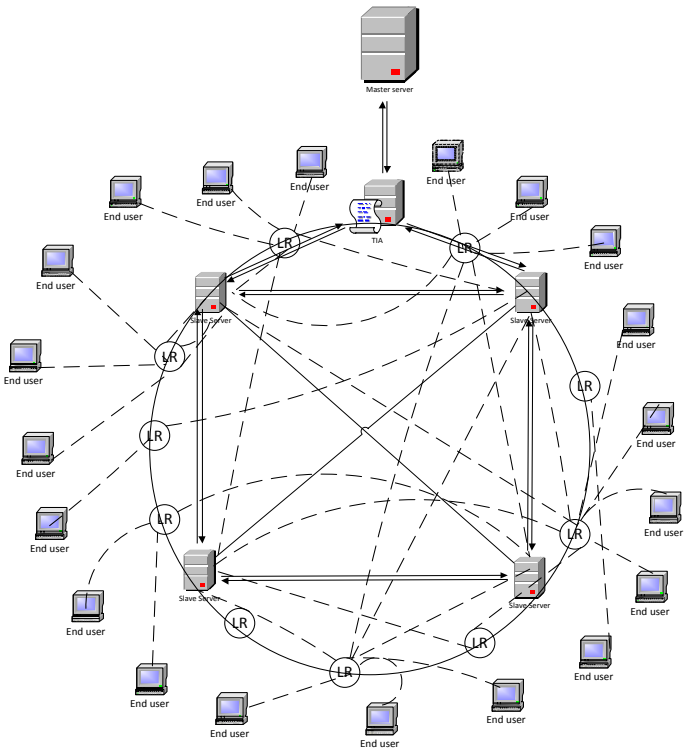

Figure 5. FSA Architecture.

B. *Replica Matrix (RM)*

RM will be a two dimensional array and maintains a matrix at the time of writing the data between slave and master. Based on operation level success, RM updates the matrix accordingly. Initially, the matrix will be set to false. If all values in the matrix is set to true, then it will stop copying the data from the master server. Always TIA looks for the matrix value in order to avoid duplicate operation. RM saves the time by avoiding duplicate operations.

C. *Load Balancing (LB)*

LB becomes a major issue in social networking sites. There will be huge requests of data operation on which load increases. To handle the increasing load, the system should divide the load among several servers. Logical router will play a vital role here. Internally it will use the shortest path algorithm to route the load among many servers. The entire server may not have the updated data, if system routes the request to another server due to huge load and that server does not have the updated data, it will again route to another server until system finds the updated data or finished searching among all the servers. Finally, if the server is not available, then it will do the update from existing server, and it will send the response with the updated data.

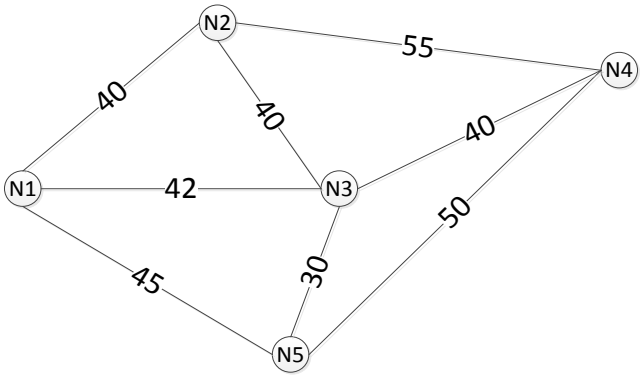

Figure 6. Shortest path used in LR.

D. Stability

Operations are dynamic and ensure that all operation should be stable which helps the social networking site to increase stability. There should not be any failure of reading and writing the data, if they exist, then system should automatically report the issue. System will consider the nodes as a graph connected with each other. At the time of routing, stability should be considered. Each time FSA partitions the graph, or connect the note, system should calculate the runtime and give the access for the calculated time. Operations like read, write and server uptime is considered during stability. All servers are not restarted during the process. If any server restarted automatically the entire load will be transferred to another server. So our server uptime is always 100%. Next is persistent data. This is a separate database, which is not modified frequently. Whenever the database is busy, and not able to take the load, it writes the data into persistence. Hence, the read and write of data will never fail. Stability provides information about performance, whenever performance goes down immediately. When performance crosses the limit, system takes the decision to transfer the load handled by LR.

At the time of copying the data, system handles performance and stability. If some request comes to a server and data copying is going on that server, then LR will transfer the request to another server. Data copy is not simultaneous for all servers. FSA do not keep the higher number of replica as it increases the network traffic. FSA removes the replica automatically.

E. Minimize the duplication

To calculate the duplicate data FSA uses matrix. If any matrix value is false then only FSA copies the data, or else, it does not copy it. This process saves time to find the duplicate data. Instead of comparing all the data, it compares only the matrix. FSA should keep in mind that there are no duplicate data; if there are any duplicates then they are no longer in use, that data should be cleaned. Before cleaning the data, FSA must check if there are any dependencies. FSA should take care the LR and dynamic slaves. Any unused LR and dynamic slaves should kill the process to free the memory.

F. Adding new slave node

Adding new node would be simple for plug and play. No need to restart the FSA or master server. Whenever any node is added, it will install the required software and inform the master. The master will immediately assign the LR, and then it will increase the number of nodes. Next, master will refresh the FSA and the entire node will be active and will be able to handle the response.

G. Remove a slave node

Slave removal process is followed to remove a slave. Node should not be removed directly. Data loss can happen if nodes are removed directly. FSA should inform the master regarding the slave which will be removed. Once FSA starts that process, the master will update any dependency of slave and will transfer the request to the nearest slave. Next, master will give message to remove the node from FSA and will remove the server logically and physically. If a processing node is removed suddenly from FSA, then request will transfer from LR to other slave servers at a certain time.

H. UI Evaluation

FSA recommends optimization on User Interface level. All the request and response should be Ajax based so that the page loading time can be reduced. All the frequently used pages need to cache in the local system, so there is no need to load the page again and again. Only data needs to be sent to the browser and server.

I. Relational Data Classification

To increase the performance of reading the data, FSA has taken a new approach. There will be a CDE. This CDE will run periodically in the backend. The main task of the CDE is classifying the data into many branches. Data in social networks are not similar. There will be data for “like”, “share”, “personal details”, or “friend details”. This will help us to copy the data to a local server; it will reduce the chances of linking the data in a wrong way. Here FSA may take the input from the developer to classify the data. If no input from the developer then FSA will randomly classify the data into possible clusters. All the related data will be copied at a time. So it will be faster to copy the work. Not only classification will help us to copy the data without any mistake it will reduce the copy time. Suppose there is 20 GB of data and it takes 4 hours to copy, FSA divides the data into 8 related clusters. And they take 20, 50, 30, 20, 10, 40, 30, 40 minutes respectively. Now our idea is to start all the relational data and copy at the same time. If the entire task starts parallel, then it will take maximum 50 minutes to finish the task. So, FSA can save 190 minutes which is 79.16% of efficiency. Here calculation of data may involve some time due to backend process which is not considered by FSA.

J. Failure handling

For any application, there will be a failure situation. So there should be a system which will handle the failure. Similar kind of mechanism exists in FSA, and this is known as Simple Failure Handling (SFH). All the clusters are interrelated; it

maintains the hierarchy at the time of copying the data and it includes Failure Management (FM). In case of error, FM will try to fix the issue by calling some predefined functions. If FM fails to fix the issue, then a notification will be sent to the admin with the details. FM is divided the error into three types, namely Permanent Failure (PF), Temporary Failure (TF) and Critical Failure (CF). PF is treated as link failure, server removal and power cut. There will be some predefined method to automatically solve the issue. SFH defines the number for each failure. To handle the PF, SFH needs to do manual work. So SFH reports to the developer or admin that there is a failure in a particular server with the failure number. Then the admin will check the failure number and will find the exact error and will take the decision, so that it will be easy for them to solve the issue. TF is treated as a code dump, memory issue and late response. Bad logic results all kind of bottleneck, if there are any kinds of TF it will try to invoke the system call and will pick the appropriate method which will help us to solve the issue automatically. CF is very critical and it may stop the social network from working. SFH will report the admin with line number. If there are any issues in CF, CF will create red bullet ticket with admin details. If red bullet ticket is created, then the developer needs to solve the issue within one hour.

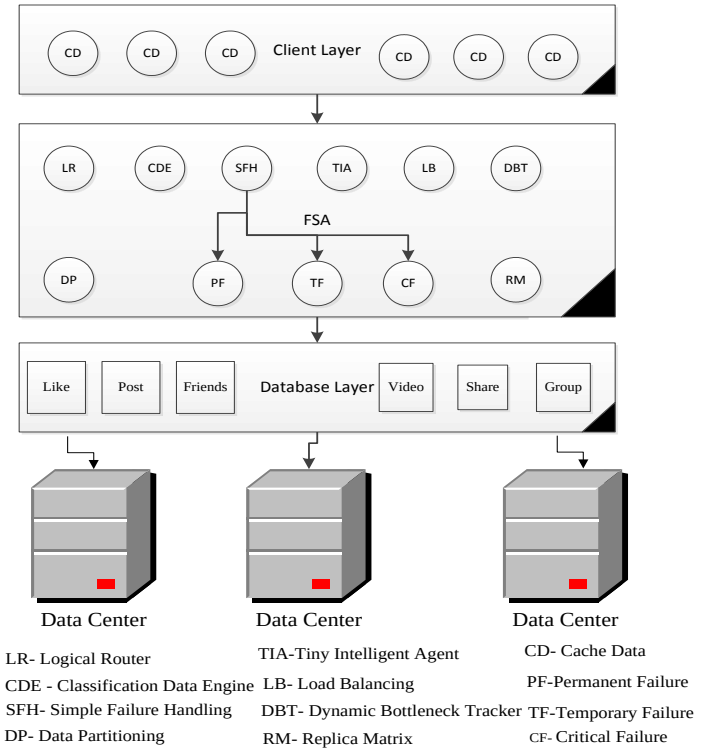

Figure 7. FSA Layers.

VII. SOLR ARCHITECTURE

SOLR is pronounced as "solar". It is an open source enterprise search platform from the Apache Lucene project. Its major features include full-text search, hit highlighting, faceted search, dynamic clustering, database integration and rich document handling.

A. *Administration User Interface for SOLR*

The SOLR web based admin interface provides various views that includes file configuration, query submission, settings to view log files, settings to control java environment and managing distributed configurations.

B. *SOLR Fields and Schema Design*

SOLR organizes the data for indexing and explains how a SOLR schema defines the fields and field types. SOLR uses the data to organize within the document files.

C. *SOLR Analyzers, Tokenizes and Filters*

SOLR prepares text for indexing and searching. Analyzers parse text and produce a stream of tokens, lexical units used for indexing and searching. Tokenizes break field data down into tokens. Filters perform other transformational or selective work on token streams.

D. *SOLR Indexing and Data Operations*

Commit, optimize and rollback describes the indexing process and data operations.

E. *Searching*

Searching presents an overview of the search process in SOLR. It describes the main components used in searches, including request handlers, query parsers and response writers. It lists the query parameters that can be passed to SOLR, and it describes features such as boosting and faceting, which can be used to fine-tune search results [3].

F. *Well-Configured SOLR Instance*

SOLR instance discusses performance tuning for SOLR. It begins with an overview of the solrconfig.xml file, then tells you how to configure cores with solr.xml, how to configure the Lucene index writer and more [3].

G. *SOLR Management*

SOLR Management discusses important topics for running and monitoring SOLR. It describes running SOLR in the Apache Tomcat servlet runner and Web server. Other topics include how to back up a SOLR instance, and how to run SOLR with Java Management Extensions (JMX) [3].

H. *SOLR Cloud*

SOLR Cloud describes the most exciting and newest features of SOLR. SOLR Cloud also provides comprehensive distributed capabilities.

I. *SOLR Distribution*

SOLR Distribution describes how to grow and divide a large index into sections called shards, which are then distributed across multiple servers or by replicating a single index across multiple services.

J. Client APIs

Client APIs describes how to access SOLR through various client APIs, including JavaScript, JSON and Ruby [3].

K. SOLR Database Evaluation

SOLR uses MYSQL and Oracle as the most popular RDBMS. Developer can be found easily and most of the developers understand these RDBMS. SOLR provides the minimum loading facility. Initially SOLR will load the latest 25 rows for the entire query and gives the functionality to get the older data in a mouse scroll. 10 rows will be appended to the page on every scroll. Most frequently, data will be indexed. Indexing is done to provide the quick search results and separate program which creates the indexing every 20 minutes. Whenever search command is fired, SOLR searches in the index table first, if not found, then immediately it will run the indexing program. If it is still not found, then it will return the message as not found. For SQL join special care taken by SOLR. If any SQL query takes more time to execute, then SOLR stores that query into a table and sends the notification to the developer. This way, SOLR reports the entire bottleneck.

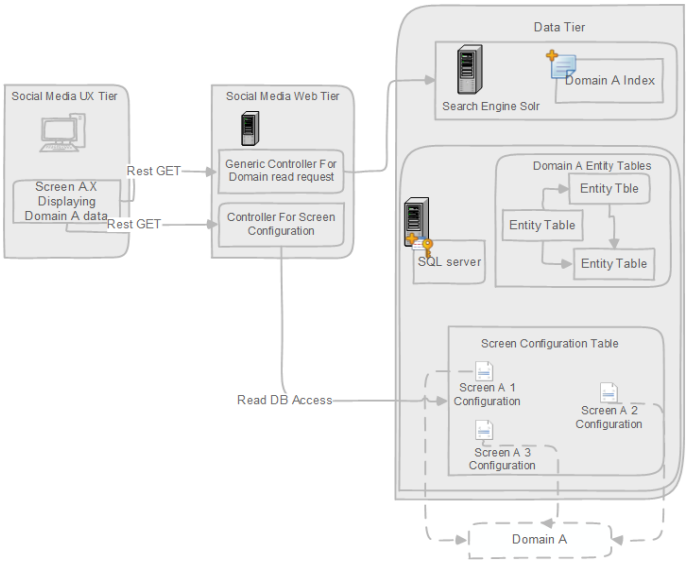

Figure 8. SOLR Architecture.

VIII. Conclusion

The proposed architecture focuses more on increasing the scalability of social networking sites by dividing the network load and route them among servers using LR. LR is 50% faster than normal routing; hence reduces the time for routing and increases response time. TIA is used to buffer the data on local machine and write the data to the master server automatically. During the write operation thread concept will be used to increase the usage of CPU. CDE is thread based and operates in parallel, which reduces the time at least by 80%. This paper discusses comparison of data and analysis using SFH. SFH will handle the issues and ensures more stable social networking sites. This paper also discusses full-text search, hit highlighting, faceted search, dynamic clustering, database integration and rich document handling using SOLR.

Acknowledgment

We thank Chandra J who is our lecturer who helped us in formatting the paper and suggesting the journal.

References

[1] M. E. J. Newman. Modularity and community structure in networks. Proc. Natl. Acad. Sci. USA 103, 8577–8582 (2006).

[2] J. Leskovec, K. J. Lang, A. Dasgupta and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters.

[3] CoRR, abs/0810.1355, 2008.

[4] Apache Solr Reference Guide. https://cwiki.apache.org/confluence/display/solr/.

[5] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In ACM IMC ’07.

[6] Apache Solr from the Solr website.

[7] http://lucene.apache.org/solr/.

[8] B. Carrasco, Y. Lu, and J. M. F. da Trindade. Partitioning social networks for time-dependent queries. SNS 2011, pp. 2:1–2:6.

[9] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In Proc. of IMC’09, pages 49–62, New York, NY, USA, 2009. ACM

[10] F. Schneider, A. Feldmann, B. Krishnamurthy and W.Willinger. Understanding online social network usage from a network perspective. In IMC ’09.

[11] J. Gray, P. Helland, P. E. O'Neil, D. Shasha. The dangers of replication and a solution. SIGMOD Conf. 1996: pp.173-182 MSR-TR-96-17.

[12] J. Leskovec, J. Kleinberg and C. Faloutsos. Graph evolution: Densification and shrinking diameters. ACM Transactions on KDD, 1:1, 2007.

[13] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra and P. Rodriguez. The little engine(s) that could: scaling online social networks. SIGCOMM 2010, pp.375–386.

[14] M. E. J. Newman and Juyong Park. Why social networks are different from other types of networks. Phys. Rev. E 68, 036122 (2003).

[15] Notes from scaling mysql up or out. http://venublog.com/2008/04/16/notes-from-scaling-mysql-up-or-out/.

[16] Rightscale. http://www.rightscale.com.

[17] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. J. ACM, vol. 56, no. 2, pp. 1–37, 2009.

[18] Status net. http://status.net.

[19] Tsung: Distributed load testing tool.

[20] http://tsung.erlang-projects.org/.

[21] Twitter architecture

[22] http://highscalability.com/scaling-twitter-making-twitter-10000-percent-faster.