Greedy Based Privacy Preserving in Data Mining using NoSQL

Nancy Bhardwah[#1], Gurvinder Kaur[*2]
[2]GGS College of Modern Technology, Kharar

***Abstract* - Database management system (DBMS) has been an integral part of the development of internet and other large scale systems. Databases of a size that have never existed before emerged, and systems that can support such databases had to be developed. As in a context management platform the number of context publications always keeps rising, further data partitioning schemes will also be researched, to better handle the always-increasing data load. The NoSQL ecosystem, unlike relational databases, is headed towards specialization, so different solutions are headed in different directions, leaving the door open for new players to emerge, and making the ecosystem an exciting and ever-evolving field. Retrieving an Item in NoSql on a 1000KB Item database takes on average over 35 ms as compare to that of 50 KB in 30 ns.**

I. INTRODUCTION

Database management system (DBMS) has been an integral part of the development of internet and other large scale systems. The traditional approach of relational and object oriented DBMS has experienced its limitation as it faces the problem of flexible scaling required for modern systems. Databases of a size that has never existed before emerged, and systems that can support such databases had to be developed. These huge databases could no longer be contained in one physical system, but should be run on a distributed system. Accommodating for scalability and performance meant giving up on some important aspects related to the reliability of the database transactions.

Non relational databases is a broad class of database management systems identified by non-adherence to the widely used relational database management system model. Non relational databases are not built primarily on tables, and generally do not use SQL for data manipulation. Non relational database systems are often highly optimized for retrieval and appending operations and often offer little functionality beyond record storage. The reduced run-time flexibility compared to full SQL systems is compensated by marked gains in scalability and performance for certain data models.

Non relational database management systems are useful when working with a huge quantity of data when the data's nature does not require a relational model. The data can be structured, but Non relational database is used when what really matters is the ability to store and retrieve great quantities of data, not the relationships between the elements. Usage examples might be to store millions of key–value pairs in one or a few associative arrays or to store millions of data records.

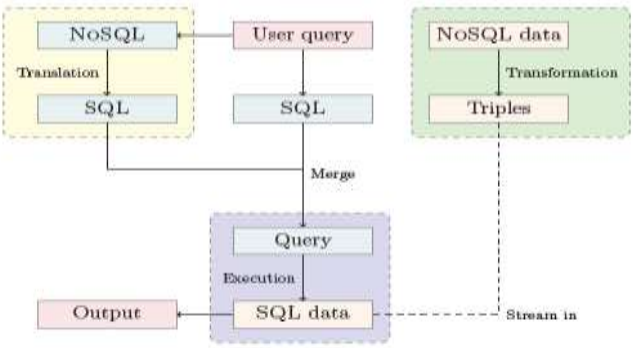


Fig -1 Architectural workflow of Query

II. NOSQL DATABASE

Hadoop: Apache Hadoop is an open-source software framework used for distributed storage and processing of very large data sets. It consists of computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are a common occurrence and should be automatically handled by the framework. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data parallely. This approach takes advantage of data locality[3] – nodes manipulating the data they have access to – to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking. The base Apache Hadoop framework is composed of the following modules:

Hadoop Common – contains libraries and utilities needed by other Hadoop modules

Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.

Hadoop YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications.

Hadoop MapReduce – an implementation of the MapReduce programming model for large scale data processing.

The term Hadoop has come to refer not just to the base modules above, but also to the ecosystem, or collection of additional software packages that can be installed on top of or alongside Hadoop, such as Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, Apache Spark, Apache ZooKeeper, Cloudera Impala, Apache Flume, Apache Sqoop, Apache Oozie, Apache Storm.
Apache Hadoop's MapReduce and HDFS components were inspired by Google papers on their MapReduce and Google File System.
The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell scripts. Though MapReduce Java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program.

HBase: HBase is an open source, non-relational, distributed database modeled after Google's BigTable and is written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System), providing BigTable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data. HBase features compression, in-memory operation, and Bloom filters on a per-column basis as outlined in the original BigTable paper.[1] Tables in HBase can serve as the input and output for MapReduce jobs run in Hadoop, and may be accessed through the Java API but also through REST, Avro or Thrift gateway APIs. HBase is a column-oriented key-value data store and has been idolized widely because of its lineage with Hadoop and HDFS. HBase runs on top of HDFS and is well-suited for faster read and write operations on large datasets with high throughput and low input/output latency.

HBase is not a direct replacement for a classic SQL database, however Apache Phoenix project provides an SQL layer for HBase as well as JDBC driver that can be integrated with various analytics and business intelligence applications. The Apache Trafodion project provides a SQL query engine with ODBC and JDBC drivers and distributed ACID transaction protection across multiple statements, tables and rows that uses HBase as a storage engine.
HBase is now serving several data-driven websites, including Facebook's Messaging Platform. Unlike relational and traditional databases, HBase does not support SQL scripting; instead the equivalent is written in Java, employing similarity with a MapReduce application.

Advantages of NoSQL

1: Elastic scaling: For years, database administrators have relied on scale up — buying bigger servers as database load increases — rather than scale out — distributing the database across multiple hosts as load increases. However, as transaction rates and availability requirements increase, and as databases move into the cloud or onto virtualized environments, the economic advantages of scaling out on commodity hardware become irresistible. RDBMS might not scale out easily on commodity clusters, but the new breed of NoSQL databases are designed to expand transparently to take advantage of new nodes, and they're usually designed with low-cost commodity hardware in mind.

2: Big data: Just as transaction rates have grown out of recognition over the last decade, the volumes of data that are being stored also have increased massively. RDBMS capacity has been growing to match these increases, but as with transaction rates, the constraints of data volumes that can be practically managed by a single RDBMS are becoming intolerable for some enterprises. Today, the volumes of "big data" that can be handled by NoSQL systems, such as Hadoop, outstrip what can be handled by the biggest RDBMS.

3: Goodbye DBAs - Despite the many manageability improvements claimed by RDBMS vendors over the years, high-end RDBMS systems can be maintained only with the assistance of expensive, highly trained DBAs. DBAs are intimately involved in the design, installation, and ongoing tuning of high-end RDBMS systems. NoSQL databases are generally designed from the ground up to require less management: automatic repair, data distribution, and simpler data models lead to lower administration and tuning requirements.

4: Economics: NoSQL databases typically use clusters of cheap commodity servers to manage the exploding data and transaction volumes, while RDBMS tends to rely on expensive proprietary servers and storage systems. The result is that the cost per gigabyte or transaction/second for NoSQL can be many times less than the cost for RDBMS, allowing you to store and process more data at a much lower price point.

5: Flexible data models: Change management is a big headache for large produc|tion RDBMS. Even minor changes to the data model of an RDBMS have to be carefully managed and may necessitate downtime or reduced service levels. NoSQL databases have far more relaxed — or even nonexistent — data model restrictions. NoSQL Key Value stores and document databases allow the application to store virtually any structure it wants in a data element. Even the more rigidly defined BigTable-based NoSQL databases (Cassandra, HBase) typically allow new columns to be created without too much fuss. The result is that application changes and database schema changes do not have to be managed as one complicated change unit. this will allow applications to iterate faster, though,clearly, there can be undesirable side effects if the application fails to manage data integrity

III. LITERATURE SURVEY

DBSCAN: Density Based Spatial Clustering of Applications with Noise. In this section, we present the algorithm DBSCAN (Density Based Spatial Clustering of Applications with Noise) which is designed to discover the clusters and the noise in a spatial database. Ideally, we would have to know the appropriate parameters Eps and MinPts of each cluster and at least one point from the respective cluster. Then, we could retrieve all points that are density-reachable from the given point using the correct parameters. But there is no easy way to get this information in advance for all clusters of the database.

However, there is a simple and effective heuristic to determine the parameters Eps and MinPts of the "thinnest", i.e. least dense, cluster in the database. Therefore, DBSCAN uses global values for Eps and MinPts, i.e. the same values for all clusters. The density parameters of the "thinnest" cluster are good candidates for these global parameter values specifying the lowest density which is not considered to be noise.

The idea of it was:

1. ε-neighbor: the neighbors in ε semi diameter of an object
2. Kernel object: certain number (MinP) of neighbors in ε semi diameter.
3. To a object set D, if object p is the ε-neighbor of q, and q is kernel object, then p can get "direct density reachable" from q.
4. To a ε, p can get "direct density reachable" from q; D contains Minp objects; if a series object p1, p2,….. pn, p1= q. pn = q. then pi+1 can get "direct density reachable" from pi.,

Pi D ,1≤ i≤ n.

5. To ε and MinP, if there exist an object o(o<D) p and q can get "direct density reachable" from o,p and q are density connected. Density Reachability and Density Connectivity: Density reachability is the first building block in dbscan. It defines whether two distance close points belong to the same cluster. Points p1 is density reachable from p2 if two conditions are satisfied: (i) the points are close enough to each other: distance (p1, p2) <e, (ii) there are enough of points in is neighborhood: |{r: distance(r, p2)}|>m, where r is a database point.

Density connectivity is the last building step of dbscan. Points p0 and pn are density connected, if there is a sequence of density reachable points p1,i2,...,i(n-1) from p0 to pn such that p(i+1) is density reachable from pi. A dbscan cluster is a set of all density connected points.

Explanation of DBSCAN Steps

DBScan requires two parameters: epsilon (eps) and minimum points (minPts). It starts with an arbitrary starting point that has not been visited. It then finds all the neighbor points within distance eps of the starting point.

If the number of neighbors is greater than or equal to minPts, a cluster is formed. The starting point and its neighbors are added to this cluster and the starting point is marked as visited. The algorithm then repeats the evaluation process for all the neighbours recursively.

If the number of neighbors is less than minPts, the point is marked as noise.

If a cluster is fully expanded (all points within reach are visited) then the algorithm proceeds to iterate through the remaining unvisited points in the dataset.

Advantages of DBSCAN

DBScan requires two parameters: epsilon (eps) and minimum points (minPts). It starts with an arbitrary starting point that has not been visited. It then finds all the neighbor points within distance eps of the starting point.

If the number of neighbors is greater than or equal to minPts, a cluster is formed. The starting point and its neighbors are added to this cluster and the starting point is marked as visited. The algorithm then repeats the evaluation process for all the neighbours recursively.

If the number of neighbors is less than minPts, the point is marked as noise.

If a cluster is fully expanded (all points within reach are visited) then the algorithm proceeds to iterate through the remaining unvisited points in the dataset. Disadvantages of DBSCAN

DBScan requires two parameters: epsilon (eps) and minimum points (minPts). It starts with an arbitrary starting point that has not been visited. It then finds all the neighbor points within distance eps of the starting point.

IV. EXISTING TECHNIQUES

Context aware architectures provides richer, target services to end users also taking care of privacy of users. In this we are using XMPP for its communication protocol and publish context information in a context broker. This context information is provided by Context-Agents, such as mobile terminals, sensor networks and social networks. Due to the nature of the XMPP protocol, the context information is provided in XML form.

XCoA is a Broker-based Context Architecture proposal, by D. Gomes, built around the XMPP protocol, more specifically XMPP Publish-Subscrib. It built on top of the results of C-Cast, but using the XMPP Publish-Subscribe protocol for communication instead of Web-Services.

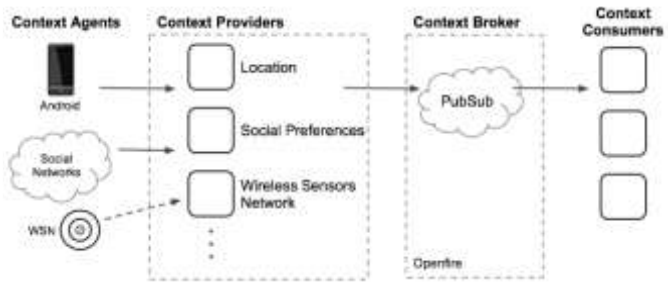


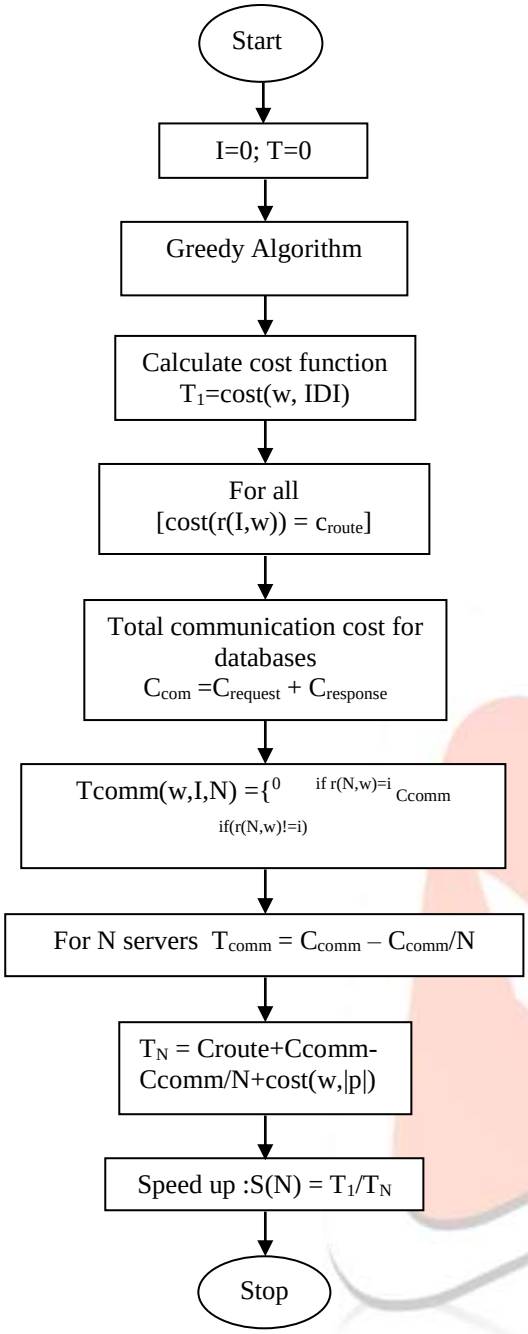
Fig 2: Architecture of XCOA

In this architecture, context information is first collected by Context Agents (CA) such as mobile terminals, social networks, or wireless sensor networks, and aggregated by CPs dealing with specific context scopes such as Location, or Social Preferences. Each CP is responsible to publish the aggregated context information on a CB Publish/Subscribe service which stores this information (History), and sends it to CCs as requested using the PubSub model. All the context information between the CPs, the CB and the CCs is exchanged in XMPP Publish-Subscribe messages. CCs can subscribe to specific Context data, such as

Location, and receive notifications when the Providers publish information in the Broker. Due to the nature of the XMPP protocol, all context information is published in XML form, inside XMPP PubSub Items [11]. CPs can be organized as PubSub Nodes, and context information is published in the respective nodes CCs can then subscribe specific nodes, for specific context types, receiving only the desired information. Most XMPP servers use a relational databases for persistent storage, and as such store Items as an XML string in the database. Such was also the case with XCoA platform, which used Openfire as an XMPP server and PostgreSQL as a database.

V. GREEDY BASED PRIVACY PRESERVING SCHEME

A lot of work has been done on data allocation and vertical partitioning but to the best of our knowledge, no work solves the exact same problem as the present paper: distributing both transactions and attributes to a set of sites, allowing attribute replication, preserving single-sitedness for read queries and prioritizing load balancing vs. total cost minimization. We therefore order the references below by increasing estimated problem similarity and do not mention work dedicated on vertical partitioning of OLAP databases. The full version of this paper describes some additional related work. Cornell and Yu generated a non-remote, disjoint partitioning minimizing the amount of disk access by recursively applying a binary partitioning. The partitioning decisions were based on an integer program and with strong assumptions on a System-R like architecture when estimating the amount of disk access. Agrawal also constructed a disjoint partitioning with non-remote partition placement. They used a two-phase strategy where the first phase generated all relevant attribute groups using association rules considering only one query at a time, and the second phase merged the attribute groups that were useful across queries. Son and Kim presented an algorithm for generating disjoint partitioning by either minimizing costs or by ensuring that exactly k vertical fragments were produced. Inter site transfer costs were not considered. The partitioning was produced using a bottom-up strategy, iteratively merging two selected partitions with the best “merge profit” until only one large super-partition existed. The k-way partitioning was found at the iteration having exactly k partitions and the lowest-cost partitioning was found at the iteration with the lowest cost. Chu and Ieong minimized the amount of disk access by constructing a non-remote and non-disjoint vertical partitioning. Two binary partitioning algorithms based on the branch and-bound method were presented with varying complexity and accuracy. The partitioning were formed by recursively applying the binary partitioning algorithms on the set of “reasonable cuts”. Navathe et al. considered the vertical partitioning problem for three different environments: a) single site with one memory level, b) single site with several memory levels, and c) multiple sites. The partitions could be both disjoint and non-disjoint. A clustering algorithm grouped attributes with high affinity by using an attribute affinity matrix together with a bond energy algorithm. Three basic algorithms for generating partitions were presented which, depending on the desired environment, used different prioritization of four access and transfer cost classes.



Flow Chart

I is counter
T is time
T_i is time at ith second
W is weight of database
R is routing function i.e. route through which the data is communicating
C_{comm} is cost of communication
C_{route} is cost of route finding and attaining
i = 0;
t_i =0;
Total time taken for m/c we'll define a cost function
Call Greedy Algorithm
T_i = cost(w,|D|)
For all[cost(r(I,w)) = c_{route}
Total communication cost for databases
C_{com} = $C_{request}$ + $C_{response}$
Communication Time Tcomm(w,I,N) ={0 if r(N,w)=i Ccomm if(r(N,w)!=i)
For N servers $T_{comm} = C_{comm} – C_{comm}/N$

Total time for N servers
T_N = Croute+Ccomm- Ccomm/N+cost(w,|p|)
Speed up is
$S(N) = T_1/T_N$

VI. METRICS USED

An essential aspect to data structures is *algorithms*. Data structures are implemented using algorithms. An algorithm is a procedure that you can write as a C function or program, or any other language. An algorithm states explicitly how the data will be manipulated.

Some algorithms are more efficient than others. We would prefer to chose an efficient algorithm, so it would be nice to have metrics for comparing algorithm efficiency.

The *complexity* of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process. Usually there are natural units for the domain and range of this function. There are two main complexity measures of the efficiency of an algorithm:

- *Time complexity* is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm. "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take. We try to keep this idea of time separate from "wall clock" time, since many factors unrelated to the algorithm itself can affect the real time (like the language used, type of computing hardware, proficiency of the programmer, optimization in the compiler, etc.). It turns out that, if we chose the units wisely, all of the other stuff doesn't matter and we can get an independent measure of the efficiency of the algorithm.
- *Space complexity* is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm. We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this. We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit. Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.

VII. RESULTS AND DISCUSSION

The usage of a NoSQL vertical partitioning storage system in an XMPP-based Context Architecture provides important performance advantages, and allows for higher availability and reliability, while also partitioning vertically. The NoSQL ecosystem, unlike relational databases, is headed towards specialization, so different solutions are headed in different directions, leaving the door open for new players to emerge, and making the ecosystem an exciting and ever-evolving field.

Using the vertical partitioning schemes in NoSql database performance of our system is increased as compare to previous one in terms of load, communication time with respect to data load and number of nodes. As shown in graph 1 and 2 communication time of the system to generate values from database is improved in which we are assuming our results with respect to data load and number of nodes. As the graphs show the performance degradation of Item retrieval when the database size increases in the number of Items. Retrieving an Item in Hadoop on a at 50KB data size it takes 40ms unit of time. There is a little degradation in performance of proposed scheme as we increase the time duration. performance degradation of in terms of load as the number of nodes increased gradually. when the No of nodes time taken to search a particular item in the database also increased.

No of Users / Technique	**1**	**2**	**3**	**4**	**5**	**6**
Privacy Preserving	1.1	1.15	1.2	1.3	1.35	1.4
Greedy Based Privacy Preserving	0.7	0.8	0.85	0.9	0.9	0.95

Table 1: Space Complexity

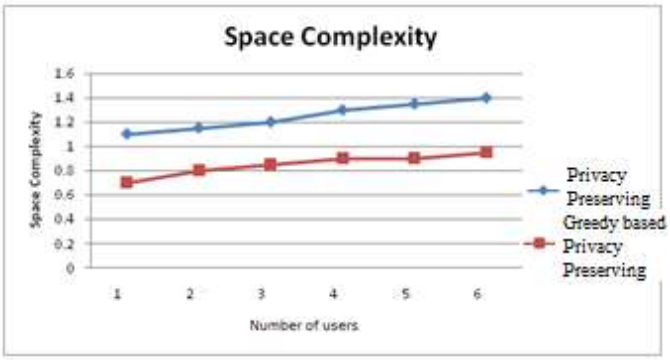


Fig 3: Space Complexity

Fig 3 is the representation of space complexity of existing and proposed system. In the existing one the space complexity is approx 1.4 where as in case of proposed it is approx 1 which is quite less than that of existing system.

No of Users / Technique	1	2	3	4	5	6
Privacy Preserving	1.2	1.3	1.5	1.7	1.8	2
Greedy Based Privacy Preserving	1	1.1	1.12	1.3	1.4	1.5

Table 2: Time Complexity

Fig 4: Time Complexity

Fig 4 is the representation of execution time of existing and proposed system. In the existing one the execution time is approx 2 sec where as in case of proposed it is approx 1.5 sec which is quite less than that of existing system.

No of Users / Technique	1	2	3	4	5	6
Privacy Preserving	96	95.7	95.4	95.1	94.7	94.3
Greedy Based Privacy Preserving	100	99.6	99.4	99.3	99.1	98.8

Table 3: Purity

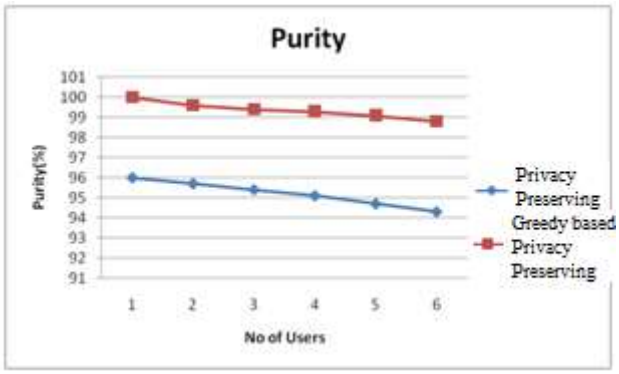


Fig 5: Purity

Fig 5 is the representation of purity of data elements in existing and proposed system. In the existing one the average purity is approx 95.2% sec where as in case of proposed it is approx 99.4% which is quite better than that of existing system.

No of Users / Technique	1	2	3	4	5	6
Privacy Preserving	97	96.7	96.5	96.3	96.1	95.4
Greedy Based Privacy Preserving	100	99.3	99.1	98.5	98.3	98.2

Table 4: Accuracy

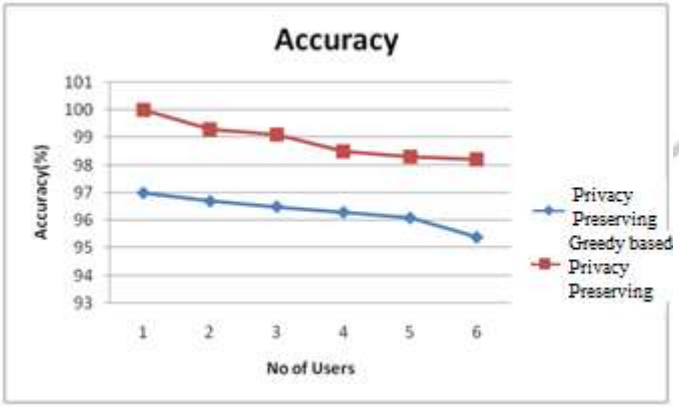


Fig 6: Accuracy

Fig 6 is the representation of purity of data elements in existing and proposed system. In the existing one the average purity is approx 96.3% sec where as in case of proposed it is approx 99.1% which is quite better than that of existing system.

VIII. CONCLUSION

As in a context management platform the number of context publications always keeps rising, further data partitioning schemes will also be researched, to better handle the always-increasing data load. The NoSQL ecosystem, unlike relational databases, is headed towards specialization, so different solutions are headed in different directions, leaving the door open for new players to emerge, and making the ecosystem an exciting and ever-evolving field. For the future scope we can implement the distributed database on all nosql databases. When the No of nodes time taken to search a particular item in the database also increased. Retrieving an Item in NoSql on a 1000KB Item database takes on average over 35 ms as compare to that of 50 KB in 30 ns. There is a little degradation in performance of proposed scheme as we increase the data size.

IX. REFERENCES

[1]. Christof Strauch,"NoSQL databases" Stuttgart Media University Feb2011.
[2]. Vatika Sharma, Meenu Dave, "Comparison of SQL and NoSQL Databases", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 8, August 2012 .
[3]. Aleksandar Milanovic, Miroslav Mijajlovic, "A Survey of Post-relational Data Management and NOSQL movement", Communications of the ACM,December 2011.
[4]. "Oracle NoSQL Database", An Oracle White paper, September 2011.

[5]. Nuno Santos, Oscar M. Pereira, Diogo Gomes, "Context Storage Using NoSQL", Conferência sobre Redes de Computadores , Coimbra, Portugal,Volume 2, Nov 2011.
[6]. Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, "Dynamo: Amazon's Highly Available Key-value Store",In proceeding of symposium on operating systems principles, October 2010.
[7]. V. Srinivasan, Brian Bulkowski, "Citrusleaf: A Real-Time NoSQL DB which Preserves ACID" ,In Proceedings of the VLDB Endowment,Volume 4, September 2011.
[8]. Rick Cattell, "Scalable SQL and NoSQL Data Stores", Communications of the ACM, December 2011.
[9]. Loannis Konstantinou, Evangelos Angelou, Christina Boumpouka, Dimitrios Tsoumakos, Nectarios Koziris,"On the Elasticity of NoSQL Databases over Cloud Management Platforms", Proceedings of the ACM international conference on Information and knowledge management, October 2011.
[10]. Nitin R. Chopde, Prof. Pritish A. Tijare,"A Survey- Review on Load balancing Algorithms", International Journal of Advanced Research in Computer Science, Volume 3,No.1, Jan 2012.
[11]. Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears, "Benchmarking Cloud Serving Systems with YCSB", Communications of the ACM, June 2011.
[12]. Denis Nelubin, Ben Engber, "Ultra-High Performance NoSQL Benchmarking",In Thumbtack Technology, Janurary 2013.
[13]. Nuno Santos, Oscar M. Pereira, Diogo Gomes, "Context Storage Using NoSQL", Conferência sobre Redes de Computadores , Coimbra, Portugal,Volume 2, Nov 2011.
[14]. "Oracle NoSQL Database", An Oracle White paper, September 2011.
[15]. oracle.com/technetwork/database/nosqldb/learnmore/nosql-database-498041.pdf
[16]. Raghav Mehra, Nirmal Lodhi, Ram Babu, "Column Based NoSQL Database, Scope and Future", International Journal of Research and Analytical Reviews, E ISSN 2348 –1269, PRINT ISSN 2349-5138, Volume 2 Issue: 4, OCT–DEC 2015, pp: 105-113
[17]. Rajat Aghi, Sumeet Mehta, Rahul Chauhan, Siddhant Chaudhary, Navdeep Bohra, "A comprehensive comparison of SQL and MongoDB databases", International Journal of Scientific and Research Publications, ISSN 2250-3153, Volume 5, Issue 2, February 2015, pp: 1-3
[18]. Rick Cattell, "Scalable SQL and NoSQL Data Stores", Communications of the ACM, December 2011.
[19]. Supriya S. Pore, Swalaya B. Pawar, "Comparative Study of SQL & NoSQL Databases", International Journal of Advanced Research in Computer Engineering & Technology, ISSN: 2278 – 1323, Volume 4 Issue 5, May 2015, pp: 1747-1753
[20]. Vatika Sharma, Meenu Dave, "Comparison of SQL and NoSQL Databases", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 8, August 2012.
[21]. V. Srinivasan, Brian Bulkowski, "Citrusleaf: A Real-Time NoSQL DB which Preserves ACID", In Proceedings of the VLDB Endowment,Volume 4, September 2011.