# SaaS Level based Middleware Database Integrator Platform

Sanjkta Pal

***Abstract*—In purpose of data searching acceleration, the fastest data response is the major concern for latest cloud environment. Regarding this, the intellectual decision is to enrich the SaaS level applications. Amongst the SaaS based applications, service level database integration is the recent trend to provide the integrated view of the heterogeneous cloud databases through shared services using DBaaS. But the generic limitations interacted during the database integration are dynamic adaptability of multiple databases structure, dynamic data location identification in the concern databases, data response using the data commonality. Data migration technique and single query approach are the two individual solutions for the proposed limitations. But the side effects during data migration technique are extra space utilisation and excess time consumption. Again, the single query approach suffers from worst case time complexity for data connectivity, data aggregation and query evaluation. So, to find a suitable data response solution by eliminating these combined major issues, a graph based Middleware Database Integrator Platform or MDIP model has been proposed. This integrator platform is actually the flexible metadata representation technique for the concerned heterogeneous cloud databases. The associativity and commonality among components of multiple databases would be further helpful for efficient data searching in an integrated way. For the incorporation within the service level but not in the services, MDIP is considered as the different platform. It is applicable over any service based database integration in purpose of data response efficiency. Finally, the quality assessment using evaluated query time compared with already proposed SLDI shows better data access quality. Thus, its expertise dedication in data response can overcome summarised challenges like data adaptation flexibility, dynamic identification of data location, wastage of data storage, data accessing within minimal time span and optimised cost in presence of data consistency, data partitioning and user side scalability.**

***Keywords*—Database integration; Integrator platform; Multi-Level graph; Subset of vertices; First class edge; Concrete edge; Connectivity edge***

## I. INTRODUCTION

In cloud computing environment, huge amount of data sets are handled through services. The reason is the opaque nature of the services, for which it can typically hide the implementation details from the service consumers and is able to provide facility of returning information in a request-reply form through shared service environment. In the cloud storage, generally data are of varied types and incremental in nature. For this reason, the relational databases are not sufficient to store that heterogeneous type huge amount of data following the schematic structure. Remembering these issues, NoSQL databases are used to store huge amount of cloud data following the schema on read operation. But in course of data accessing, the automation is needed at cloud provider side. That causes accelerated consumer based service provisioning and data instance management. To reach towards the prescribed goal, DBaaS assistance is needed [9] [10]. Because, using the data service support at SaaS service model, DBaaS can deliver high quality of data to a large number of users. That satisfies multi-tenant scenario [16].

In purpose of data handling in cloud environment, there may be multiple numbers of heterogeneous cloud databases to store large scale data items. So, for cloud data handling, database integration concept comes. This can handle different types of data from multiple cloud databases in an integrated fashion. This concept leads towards database integration. But, in the database integration subtitle, one of the most challenging approaches is the deliverability of integrated view of different data sets which are situated in distributed heterogeneous cloud databases. If the database integration is done over the services, then that service based database integration [12] [13] must be focused as more effective approach than IaaS or PaaS based database integration. The reason is the services' ability to extract dynamic view of multiple cloud databases. But in sense of robustness of any mechanism, every mechanism suffers from some incompleteness as well as some challenges. Similarly this service based database integration technique also suffers from flexible adaptability of the structure of multiple databases and also lacks in dynamic identification of data location in the concern database or databases against users data request using their commonality. Depending on these challenges, some solutions have been found. Those are, data migration technique and single query approach. In data migration technique the data transformation form relational database to NoSQL database has been focused [1] [2]. But this technique suffers from extra space utilisation for storing duplicate data, and excess time for data migration. In single query approach, data collection is possible from relational as well as from NoSQL database just using a single query [3] [4] [5] [6]. In the context, the single query approach also suffers from the worst case time complexity for data connectivity, maximised time for data aggregation and maximised time for query evaluation.

So, surveying all the possible techniques for multiple database handling, it can be concluded that database integration through SaaS is the effective approach rather than others. Because service based database integration can deliver integrated view of data within minimum data accessing cost as well as minimum implementation cost, in presence of consistency, service partitioning and service share-ability. But

The author is presently not affiliated to any Institution/Organisation.

for the above challenges during service based database integration, some modification is needed over it. So, to overcome the issues, a different platform in the service level in needed which can act as the integrator of multiple heterogeneous databases maintaining the flexible adaptation of another new database.

Remembering all the issues and its possible solutions, a Middleware Database Integrator Platform (in abbreviation it is termed as MDIP) has been proposed. This platform would act as the database integrator and would able to provide integrated view of distributed heterogeneous cloud databases after adapting those multiple databases structure. The applicable area of this MDIP is SaaS service model. This middleware architecture does not ensure formalisation in services or in composition of services. Rather this middleware architecture ensures a different platform concept in between Application service and Data service, in which multiple number of heterogeneous cloud databases can store their database details in combined fashion for further integrated data deliverability. This mechanism is applicable in any service based database integration for the optimised time consumption during data response. So, in purpose of implementation of MDIP concept, a multi-level graphical approach has been considered. The concept can easily map the cloud databases and their components in different levels to form the metadata by maintaining the data instance inter relationship and commonality. This causes reduced time and fast data retrieval during users query response. At last, a comparison on query evaluation time has been done within already existing Service Level Database Integration mechanism [12] and proposed mechanism after incorporating it in SLDI. The comparison focuses on the quality assurance in sense of better data availability and optimised time for data management. Thus the approach can overcome the prerequisite challenges like data model adaptation flexibility, dynamic identification of data location, wastage of data storage, data accessing with minimal implementation cost as well as minimum time in presence of data consistency, data partitioning and maximum scalability. Summarising all the characteristics and solved issues of the proposed approach, it can be concluded that the MDIP approach would be supportive for further accelerated efficient data retrieval in the latest cloud environment.

## II. Related Work

Till now, many approaches have been proposed to provide dynamic integrity of cloud databases for the deliverability of the integrated view of heterogeneous data instances. Those are briefly discussed in below.

In [1], to support advance database architecture, relational as well as NoSQL databases would be involved in data adapter system through three different approaches. To simplify the query evaluation, data adapter system integrates and handles the transformation from SQL to NoSQL approach is accessed. In [2], a framework is introduced to support migration from relational database to NOSQL database. The framework is modularised into two parts. The first is migration module, which enables seamless migration in between databases and the second is mapping module is used to translate and execute the requests in any database management system for returning the integrated view. In [3], the Triple fetch query language on the platform for integrating relational and NoSQL databases claims to provide applications to leverage the benefits of the relational as well as NoSQL databases using the single relational database query. This query may produce results from relational database and from NoSQL database rather than single output within minimal cost. In [4], a generalised query interface is designed for unity of both relational and NoSQL databases. In the scenario of unity allows SQL queries to automatically translate and execute with the help of underlying API of the relational and NoSQL data storages. As a whole virtualise system is applied to join data and query from both relational and NoSQL databases using a single SQL query. In [5], to provide the concrete benefit of NoSQL databases with relational database, a dual fetch query language system has been proposed. The platform is introducing a query syntax. This helps to provide combined data from separate databases in a single application. In [6], a framework has been evolved for integrating relational as well as NoSQL databases. The efficiency of the framework is the answering the queries after collecting them from integrated data sources. The framework offers optimised query translation within minimal cost for integrating MySQL (as relational database) and Mongo DB (as NOSQL database) through an aggregated cost. In [7], comparison in between NoSQL and relation databases has been magnified and also specifies the limitations during real world applications. Here the mechanism proposes the solution to solve the limitations using through integrated data sources for yielding better data responses through simple or complex queries. In [8], due to absence of proper tool for migration from relational database to NoSQL, a conversion has been proposed. This helps data migration from relational database (SQL) to NoSQL database (Mongo DB) using query. The common structure of the proposed query processing language can handle NoSQL data and relational data together.

## III. Framework for MDIP

Considering all the summarised generic challenges, a mechanism has been proposed to resolve the mentioned summarised issues. Regarding those issues, a Middleware Database Integrator Platform or in abbreviation MDIP approach is considered, where an individual platform rather than services would be engaged to provide the integrated view of heterogeneous cloud databases. Even for easier data availability, the integrator platform takes the responsibility as dynamic metadata representation after accepting new database and its model in a flexible way. Here, the target is to formalise the flexible metadata representation after collecting the data models from multiple cloud databases showing the interconnectivity and commonality among database instances. In this way, the formalisation can provide the draft for attached cloud databases using their interconnectivity and their commonality. This would be further helpful for users' data response by follow the strict navigation in reverse direction.

### A. Graphical Representation of MDIP Framework:

A formal representation has been diagrammed using a graphical approach. Then MDIP can be realised using multi-level digraph $M$ $(G: (V, E), L)$ which can be extendable unto

multiple levels, *L*. In the graphical scenario, the components of the cloud databases are considered as the vertices of the multi-level graph, which are denoted as *V*. The set of directed edges of the graph are defied by the interconnection in between pair of vertices are formally denoted as $E \subseteq (V \times V)$, where $(V \times V)$ is the representation of the pair of consecutive vertices within a layer or in between layers.

For deploying the multi-level graph *M* (*G, L)*, some issues can be evolved over the components of MDIP graphical framework. So, more formal description of those components are defined below.

*1) Vertices:* In the MDIP graphical scenario, multiple cloud databases and their intermediate components are considered as the vertices of the graph. For this graphical design, the numbers of databases are themself considered as the vertices, which are the residence in the top level of the graph, in a single plane. The intermediate components of those databases can be realised as the subordinate consecutive lower level vertices. Whenever the same type of database components would be allowed to reside in same level, then those database components must be declared as co-planar vertices of the graph. Here in the graphical scenario, every vertices *V* are denoted by the combination of subset of vertices and level notation, like $S_{pj}(L_i)$, where, $S_{pj}$ is the notation of $j^{th}$ number vertex in the $p^{th}$ subset of vertices at level $L_i$.

For this approach, all the black circles are considered as vertices and are denoted by V.

*2) Subset-of-vertices:* In the graphical scenario, the total number of co-planar vertices can be clustered into some number of subset of vertices. Those subset of vertices are denoted as $S_p$ at a particular level *L* of the multi-level graph *M (G, L)*. Here the arbitrary number *p* must range in between 1 to *n* or formally it will be denoted as $1 \leq p \leq n$. So, the formal representation of the subset of vertices at a particular level *L* of the graph can be represented as,

$$S_1(L) / S_2(L) / S_3(L) / \ldots\ldots / S_p(L) / \ldots\ldots\ldots / S_n(L) \subset S(L)$$

Or,

$$S(L) = S_1(L) \cup S_2(L) \cup S_3(L) \cup \ldots. \cup S_p(L) \cup \ldots\ldots \cup S_n(L).$$

This means, the combination of all subsets of vertices in a particular level must form a complete level.

For the presence of multi-level concept, if $L_i$ represents the $i^{th}$ level in the multi-level graph *M* (*G, L)*, then for non-co-planar subsets of vertices, any lower level subset of vertices must be considered as the subset of a particular subset of vertices in its consecutive upper level. Or in formal it would be represented as,

$$S(L_0) \supset S(L_1) \supset S(L_2) \supset \ldots\ldots\ldots\ldots \supset S(L_i)$$

For example, in a particular level of the MDIP graph, the cluster of similar components at a particular level and can be decomposable into finite number of subsets. In vice-versa, the union of those subsets of vertices must form a complete level of the graph.

For this approach, all the triangular solid shapes in the upper part of any level are considered as subset of vertices, but in the lower part, the lightly shaded areas containing vertices are considered as the subset of vertices in elaborate fashion.

According to MDIP graphical concept, cloud databases must exist at the top level of the graph. Then their subordinate components would be placed in its lower level maintaining the proper sequence. Those subsets of vertices must exist at a particular level in a clustered way. Form the concept of subset of vertices it is declarable that any top level sub set of vertices is the superset of its subordinate level's subset of vertices.

*3) Levels:* In the graphical representation, cloud databases and their subordinate components must be non-co-planar. Maintaining the consequent placement of different non-co-planar database components at different stages will discuss the level concept in the graph.

For multi-level graph *M (G, L)*, levels $L_i$ can be defined by the non-co-planar sets of vertices and their connectivity using edges. At a particular level, all the placed vertices or database components are considered as co-planar. If $V_i$ denotes the set of co-planar vertices at a particular plane or level $L_i$ and the set of vertices $V_j$ are denoting the set of another co-planar vertices at a particular plane or level $L_j$, then the two different co-planar sets of vertices must exist at different plane or formally $L_i \neq L_j$. Then, as per definition of non-co-planar sets of vertices, different planes of the graph must be regarded as levels.

Using the concept of multiple levels, any level $L_j$ will be said as consecutive of level $L_i$, if level $L_j$ must maintain the provided relation: i.e. $L_j = L_{i+1} / L_{i-1.}$ Here, the number of levels must range up to some positive finite number. Because for any cloud databases, attributes are the granular components and those attributes cannot be further decomposable. But for the level concept, those levels always maintain the connectivity, which can be represented through the edge notation denoted by set *E*.

TABLE I. DATABASE COMPONENTS AND LEVELS ASSOCIATIVITY IN MULTI-LEVEL MDIP GRAPH

| | |
|---|---|
| MySQL Database, Mongo DB | Level 0/ top level |
| Schemas of databases | Level 1/ intermediate level |
| Attributes of Databases | Level 2/ lower level |

In the graphical scenario, for the simplicity of the graphical framework, at the top level of the graph, numbers of cloud databases are placed. So, for this reason, the number of cloud databases would be regarded as the co-planar graph, contained at same level. In the next level of the graph, the subordinate components of those cloud databases (like collection of schemas) would be placed in its proper graphical level maintaining the planarity of the vertices. Similarly, multi-level graph would be formed by placing those different database components at different levels in a proper sequence, which are also non-co-planar in nature. Here for the MDIP graphical scenario, the database components and their assumed levels are provided in Table 1.

*4) Count ability of the Subset of Vertices:* In the graphical scenario, if the sub set of vertices are represented by $S_p$, at particular level $L_i$. Then, formally the total number of sub-sets *b* in a particular level $L_i$ can be represented as,

$$c\ (L_i) = c(\sum_{p=0}^{n}(S_p(L_i)) = |b|.$$

Here, each level of the multi-level graph has possibility to be decomposable into multiple numbers of sub sets of vertices maintaining the requirements. These subsets of vertices are regarded as the sub-graph in a level in the graph. Continuing in this way, the multi-level graph will contain total number of sub-graphs same as the total number of subset of vertices used in different levels in the whole graph. Continuing in this way, in the multi-level graph *M (G, L)*, the total count of the sub-graph must be the sum of total number of sub-graphs in every levels. Then the formal representation of the total count of the sub-graphs must be,

$$C = |\sum_{i=0}^{m} c\ (Li)\ |,$$

Whenever the maximum number of levels is *m*

According to MDIP graphical concept in Figure 1, here two cloud databases are used in the graph. This indicates the single set of vertices at top level containing the databases. In the next level, if their schemas are defined, then two different sets of vertices (here schemas) for two different cloud databases would be represented. For the two sets of vertices in the second level, the cardinality of the sub-graph in the second level must be declared as two. And at the lowest level, there exists five different subsets of vertices depending on this concept.

Then using the count ability of the subset of vertices, the total count of sub-graphs in the whole graph would be,

$$C = (|\sum_{i=0}^{m} c(Li)|) = c\ (L_0) + c\ (L_1) + c\ (L_2) = 1+2+5 = 8$$

*5) Edges:* In MDIP, whenever a cloud database gradually can be decomposed into multiple number of subordinate components (i.e. cloud databases, schemas, attributes etc.), then non-co-planar database components must be mapped into different levels in the multi-level graph. Continuing this process, the components of the cloud databases (denoted as the vertices) of same level or different levels must be connected some other consicutive components maintaining their physical connectivity.

So, the set of edges can be categorised into two different types. Those are,

*a) Intra level connectivity edges:* These set of edges are responsible for connecting a pair of co-planar vertices situated in a particular level. For this category of edges, the situation of the end vertices may be in a single subset of vertices or may be in different subset of vertices. Depending on this, these set of edges may be categorised into two types. Those are,

- *Intra subset connectivity edges:* These set of edges are responsible for connecting a pair of vertices situated in a subset of vertices. If *Fi* denotes the set of Intra connectivity edges for connecting any two vertices $v_i$ and $v_j$, situated at same sub set of vertices $S_p$ at level $L_i$, then the formal representation can be defined as,

$$Fi \subset (S_{pi}\ (L_i) \times S_{pj}\ (L_i))$$

where, $S_{pi}\ (L_i)$ denotes vertex $V_i$ and $S_{pj}\ (L_i)$ denotes vertex $V_j$. The solid arrow headed solid lines represent these intra connectivity edges.

- *Inter subset connectivity edges:* These set of edges are responsible for connecting a pair of vertices situated in two different subsets of vertices in a particular level. If *Di* denotes the intra level connectivity edges for connecting any two edges $v_i$ *and* $v_j$, situated at different sub set of vertices named as $S_p$ and $S_q$ at a particular level $L_i$,, then its formal representation can be defined as,

$$Di \subset (S_{pi}(L_i) \times S_{qj}(L_i))$$

where, $S_{pi}\ (L_i)$ denotes vertex $V_i$ and $S_{qj}\ (L_i)$ denotes vertex $V_j$. Solid arrow headed dashed lines represent these inter subset connectivity edges.

*b) Inter level Connectivity edges:* These set of edges are responsible for connecting a pair of vertices situated in two different subsets of vertices in two consecutive levels. If *Pi* denotes the inter level connectivity edges then its formal representation can be defined as

$$Pi \subset (S_p(L_i) \times S_q(L_{i+1}))/\ (S_q\ (L_{i+1}) \times S_p(L_i)),$$

Where, $S_p$ and $S_q$ are denoting two different sub sets of vertices accordingly at the levels $L_i$ and $L_{i+1}$. In the inter-level edge representation, two different types of edges are defined. Those are,

- *Upward directed edges:* In this set of edge representation, edges are directed towards upward. In the given scenario, the edge direction is from lower level components (i.e. like attributes) towards upper level components (finally the used database). Following these upward directed edges in a proper sequence, a user can find her requested data from the concerned cloud databases. So, the consecutive sequential usage of upward directed edges can form a complete request path.

- *Downward directed edges:* In the second set of edge representation, edges are directed towards downwards. Where, the edge direction is from upper level components (i.e. like the used database) towards lower level components (finally attributes). Following these downward directed edges in a proper sequence, the data can be stored in the cloud database. So, the consecutive sequential usage of downward directed edges can form a complete data storage path.

The blank arrow headed solid line represents these inter level connectivity edges. If the edges are directed towards upper level then the edges are upward directed edges. If the edges are directed towards lower level then the edges are downward directed edges.

So, the formal representation of the set of edges can be defined as,

$$Ei = Fi \cup Di \cup Pi$$

*6) Dissection of a single level, its necessity and advantage:* For the graphical simplicity, every level has been dissected into two different parts. Among them, the lower part must contain the clustered vertices and their connectivity details and the upper part must contain only the number of subset of vertices.

Within a level, any subset of vertices in the upper part would be connected with its lower level components using a single edge. The reason is to avoid multiple edges connectivity complication. For this scenario, this single edge connectivity in between subset of vertices and its vertices is actually the summarised consideration of multiple inter connectivity edges.

So, formally the representation would be,

$$Ri \subset (S_p(L_i) \times S_{pi}(L_i))$$
$$Or$$
$$Ri \subset (S_p(L_i) \times \{S_{p1}(L_i), S_{p2}(L_i), \ldots\ldots, S_{pr}(L_i)\})$$

Where $S_p(L_i)$ is the representation of the $p^{th}$ subset of vertices situated at the upper part of the level $i$, and $S_{pi}(L_i)$ is the representation of the $i^{th}$ vertex in the $p^{th}$ subset-of-vertices at lower part of the $i^{th}$ level. This connectivity must explain the total count of edges equals with the number of vertices situated in $S_p(L_i)$ subset-of-vertices. If the $S_p(L_i)$ subset-of-vertices contains $r$ number of vertices in the set, then for interconnectivity $r$ number of edges must exist. Here the vertices to subset-of-vertices functional connectivity will deliver the common edge in place of $r$ number of inter connectivity edges.

For the concise characteristics, any two co-planar subsets of vertices connectivity in the upper part of a level can explain the abstract relationship. But its lower part can explain the absolute relationship within the vertices in a single subset of vertices or within multiple co-planar subsets of vertices for its detail description.

Similarly for the inter level connectivity discussion, any two vertices for two consecutive levels must be connected with the single edge for avoiding multiple edges to connect all of its nearer suordinates.

So, formally the representation would be,

$$Pi \subset (S_{pi}(L_i) \times S_q(L_{i+1}))$$
$$Or$$
$$Pi \subset (S_{pi}(L_i) \times \{S_{q1}(L_{i+1}), S_{q2}(L_{i+1}), \ldots\ldots, S_{qr}(L_{i+1})\})$$

Where $S_{pi}(L_i)$ is the representation of the $i^{th}$ vertices situated at $p^{th}$ subset-of-vertices at level $i$, and $S_q(L_{i+1})$ is the representation of the $q^{th}$ subset-of-vertices at consecutive $i+1^{th}$ level. This connectivity must explain the number of edges equals with the number of vertices situated in $S_q(L_{i+1})$ subset-of-vertices. Here also, if the $S_q(L_{i+1})$ subset-of-vertices contains $r$ number of vertices in the set, then for interconnectivity, single edge would be placed as the substitute of $r$ number of edges.

Graphically inter level connectivity edges are the detail explanation of this type of connectivity.

### B. Presentation of MDIP graph and its detail description:

Figure 1 shows a simple scenario through the proposed MDIP graphical model. In the graph, three levels have been used, those are $S(L_0)$, $S(L_1)$ and $S(L_2)$. Among these $S(L_2)$ represents lower level and the highest level is represented by $S(L_0)$. In the highest level, two vertices are situated. They are noted as $S_{11}(L_0)$ and $S_{12}(L_0)$. In real concept these two nodes are denoting the used two different cloud databases, i.e. DB1 as $S_{11}$ and DB2 as $S_{12}$. Here the upper part of the level $L_0$ denotes the subset of vertices $S_1(L_0)$, which contains the discussed two vertices. In this level the interconnectivity within two databases lacks the concreteness in explanation. So, that connectivity edge is the first class edge.

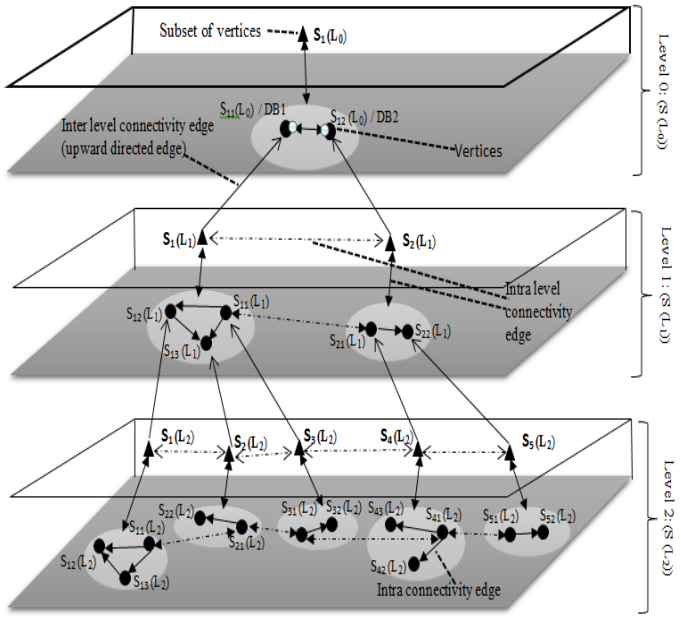


Fig. 1. Graphical representation of MDIP using multi-level digraph

TABLE II. SUMMARISED GRAPHICAL NOTATION FOR MDIP GRAPHICAL NOTATION

| Formal notation | Description of notation | | Graphical notation |
|---|---|---|---|
| V | Set of vertices | | ● |
| S | Subset of vertices | | ▲ |
| E | Set of intra level connectivity edges | Intra subset connectivity edges | → |
| | | Inter subset connectivity edges | ←- - -→ |
| | Set of inter level connectivity edges | Upward directed edges | Blank headed arrows towards upward direction |
| | | Downward directed edges | Blank headed arrows towards upward direction |
| | Set of edges for connecting the components of upper part and lower part with in level | | <- - -> |

In the next level, means at level 1, the clusters of schemas of the proposed databases has been magnified. For two different databases, the set of schemas have been presented into two subsets of vertices. The first subset is under vertex $S_{11}$ $(L_0)$ and the second subset is under vertex $S_{12}$ $(L_0)$ of level $S$ $(L_0)$ and those vertex subsets are denoted as $S_1$ $(L_1)$, $S_2$ $(L_1)$. So, the upper part of level $S$ $(L_1)$ contains these two subsets of vertices, means $S_1$ $(L_1)$ and $S_2$ $(L_1)$. Here, the inter connectivity edges responsible for connecting the set of vertices $<S_{11}$ $(L_0)$, $S_1$ $(L_1)>$ and $<S_{12}$ $(L_0)$, $S_2$ $(L_1)>$ can discuss the connectivity with all vertices, which are situated in the lower part of the level. In the lower part of the level, first subset containing three vertices $S_{11}$ $(L_1)$, $S_{12}$ $(L_1)$ and $S_{13}$ $(L_1)$. In the second subset, numbers of vertices are two and they are denoted as $S_{21}$ $(L_1)$ and $S_{22}$ $(L_1)$. Here, inter level connectivity edges responsible for connecting the set of vertices $<S_{11}$ $(L_0)$, $S_1$ $(L_1)>$ and $<S_{12}$ $(L_0)$, $S_2$ $(L_1)>$. These edges can discuss the connectivity with all vertices, which are situated in its lower part of the level.

For the next lower level (here the last level) means at level 2, the set of attributes are used. At level 2 five set of vertices have been used for discussing five schemas in the upper part of the level. Here the used sets of vertices are denoted as $S_1$ $(L_2)$, $S_2$ $(L_2)$, $S_3$ $(L_2)$, $S_4$ $(L_2)$, and $S_5$ $(L_2)$ and these are the vertices of the upper part of the level. The connectivity of these subsets of vertices can't clear the concrete connectivity. So, for the concrete view, those subsets are decomposable into lower part showing its concrete connectivity. Among them, the first subset containing three vertices $S_{11}$ $(L_2)$, $S_{12}$ $(L_2)$ *and* $S_{13}$ $(L_2)$, the second subset contains two vertices and they are denoted as $S_{21}$ $(L_2)$ and $S_{22,}$ $(L_2)$, the third subset contains another two vertices, which are denoted as $S_{31}$ $(L_2)$ and $S_{32}$ $(L_2)$. For the fourth set, the numbers of vertices are three and are denoted as $S_{41}$ $(L_2)$, $S_{42}$ $(L_2)$ and $S_{43}$ $(L_2)$, and finally in the fifth subset, the numbers of vertices are two and are denoted as $S_{51}$ $(L_2)$ and $S_{52}$ $(L_2)$. Because of the attribute declaration in the level 2, this level is unable for further decomposed into next level, because attribute components always maintain the granularity feature in its provider databases.

In this proposed graphical scenario, the used components of the databases in a single level easily be declared as co-planar. But for the whole graph concept, databases components situated at different levels may be declared as non-co-planar.

### C. Decomposibility of the Levels:

In this multi-level graph concept, there is a possibility to decompose a particular level of the graph into another level using some characteristics. But this decomposition process may be continued up to a finite range. Because, the assumed last level components may not be further decomposable using the proposed characteristics. In reality, any cloud database can be decomposable unto its attributes. This situation is for the granularity of the attributes in every database. Then, in the graph, the first used level must be considered as the parent level or highest level of the graph, and the assumed last level must be considered as the leaf level or lower level of the graph.

Continuing in this way, the proposed MDIP graphical framework may be decomposable into multiple levels. But this decomposability scenario must follow some characteristics associated with the graph. Those are,

*1) First class edge and concrete edge:* In the concept of individual level (i.e. any particular level of cloud databases), there may be multiple sets of vertices. All these vertices must maintain the planarity and may have intra level connectivity among them. In this graphical concept, every level has been decomposed into two parts. The lower part (in Figure 1) shows the coplanar vertices in their defined section, means subset of vertices. The upper part (in Figure 1) of the level only shows the number of subsets of vertices (means subgraph) used in the level. This explains that the lower part of the level is the elaborate dissection of the upper part of the level. In the scenario, whenever the connectivity has been shown in between the two components in the upper part of any particular graphical level, then the concreteness of that edge can be discussed into the lower level vertex connectivity. So, in the upper part of the level shows the abstract connectivity of two sets of vertices using first class edges [11], and then at lower level, the vertices connectivity will explain the concrete edges.

*2) Scalability during decomposition of first class edge:* During the explanation of total graphical concept, if the vertices connectivity within the upper part of the level shows the abstract connectivity using first class edge, then the upper part of the discussed level must be decomposable into consecutive lower part to provide the concrete connectivity of vertices. Whenever the vertex connectivity within the lower part of the level are may not be further decomposable for the atomic nature of the vertices, that level must be considered as the extreme lower level or leaf level $L_i$. But the absence of concrete decomposability, permits the lower part of the level to be further decomposed into consecutive lower level.

In reality, for this MDIP graphical approach, cloud databases are considered as the top level vertices. Let, those databases are further decomposable into cluster of schemas in the next level, and then those schemas must be regarded as the vertices in the next level. But in reality, the database schemas are not child level components. So, the schemas must be further decomposable into attribute details. Then in the consecutive lower level, those database attributes must be arranged. In the database detailing, the attributes may not be further decomposable into subordinate components. So, the graphical level with attribute detail must be declared as the extreme lower level in the multi-level graph. In this case, different cluster of schemas of different databases, different attribute detail of different schemas must form individual subsets-of-vertices maintaining their planarity. During the graphical formalisation, the upper part of the level must show only the number of subsets-of-vertices.

*3) Algorithm to accomplish the complete decomposition in the multi-level graph:* To decompose a particular level of the graph into its lower level, anyone must follow the decomposability into defined steps. Those are,

Step1: take any edge, which connects a pair of co-planar vertices.

Step 1a: if the edge is intra subset connectivity edge, then pair of vertices will reside in a single subset-of vertices. Then check step 2 cases.

Step 1b: if the edge is inter subset connectivity edge, then the pair of vertices will reside in different subset-of vertices. Then check step 2 cases.

Step 2: Check the database components equivalent with those vertices.

Step 2a: If both the vertices represent child level database components, then go to step 4.

Step 2b: if both the vertices represent intermediate level database components, then go to step 3.

Step 2c: if one vertex represent child level database component and the other vertex represent intermediate level database components, then go to step 3.

Step 3: Decompose those vertices into further lower level components or vertices.

Go to step 1 (Continue the process until it find Step 2a case to end the decomposition).

Step 4: Stop further decomposition.

End process.

## IV. Illustration of the Proposed MDIP Framework

To illustrate the Middleware Database Integrator platform or MDIP, the real life example on healthcare data storage has been taken. Here for the presence of relational data as well as semi-structured data for remote health care, two different types of databases are used. Those are, MySQL database, used for storing relational data and Mongo DB database used for storing semi-structured data.

In the illustration, MySQL database is taken to store the patients' demographic data, doctors' demographic data and doctor's schedule. For storing those data in a structured schematic way, a database named 'HEALTHCARE' has been declared in the MySQL database, in which the three tables are designed [12].

For storing the prescription details, which poses the ever increased volume data with respect to a particular patient, Mongo DB database has been used. In Mongo DB, the declared database name is 'RHC' [12]. In this RHC database, here also three collections (means table) has been declared. Those are PATIENT, EPRESCRIPTION and PRESCRIPTION DETAILS. The characteristics like the declaration of different types of documents (tuples or rows) in different collections (tables), there is no need to specify the attributes data type under which the data would be inserted in the Mongo DB database. But for declaring the inter-connection within the tables or intra-connection in between tables in the database, some common attributes have been declared in the tables. Here, Table 3 shows the table details of MySQL database as well as of Mongo DB database.

For data collection in an integrated way from the multiple number of tables or collections within a single database or multiple number of databases, the correlation among tables or collections or within databases are mandatory. MySQL supports the foreign key concept for interconnection within the tables in the single database for the above reason. So, in MySQL database, DOC_ID is assigned as a foreign key in PATIENT table and also in DOCSCHEDULE table for searching the doctor's details (i.e. doctor's name, specialisation as well as doctor's schedule) by any patient. But Mongo DB does not support any foreign key concept within the collections. So, for collection's inter-connection in Mongo DB, reference concept has been used. This referencing concept may not be validated throughout the whole collection. Only the referenced documents of a collection can be referred by the concerned particular documents situated at other collection. The referencing syntax is like,

```
>db.eprescription.insert({name:"ramesh",pid:db.patient.find()[1]._id,docid:db.doctor.find()[1]._id,age:"41",disease:"fever",bp:"110/79",pulse:"92",medicine:"paracitamol 650"})
```

Here, this particular document of collection EPRESCRIPTION taking reference from the PATIENT collection's document. So, using database details, and the inter-connections or intra-connections among them, the use case diagram of the MDIP graph is given below.

In the use case diagram sketched in Figure 2, MySQL database and Mongo DB database are two different states at top level. The associativity among those databases or states can explain the relationship among their internal components. So, the two databases can provide schemas through the generalised view in the next level UML. Here, in the UML the clusters of schemas of two databases are grouped into two different packages. Then the single generalisation indicator can illustrate the database relationship with all of its schemas contained in the schema group. Continuing in this way, in the next level, all the attribute detail of the schemas and their associativity has been shown. But for the regarded case, attributes are further do not decomposable into next level. So, the attribute details are regarded as the last level of MDIP.

Here different components of different level discussed in the use case diagram of the multi-level MDIP Graph have been provided with their identifying ID in Table 4.

To discuss the provided tables or collections inter or intra connection among them, it is important to provide the attribute details with their commonality. To illustrate the graphical dissection of the vertices of the complete graph, a simplified example on healthcare data has been used. From the above table (Table 4), two different sets of tables or collections have been diagrammed. In Figure 2, *MS1, MS2* and *MS3* are associated with MySQL database and they actually are the representation of the three tables of the MySQL database, named PATIENT, DOCTOR and DOCSCHEDULE. But the three collections *MD1, MD2* and *MD3* are associated with Mongo Database and they actually are the representation of the three collections of the concerned database, named PRESCRIPTIONDETAILS, PATIENT, and EPRESCRIPTION.

TABLE III. DATABASE DESCRIPTION OF MYSQL AND MONGO DB DATABASES

| **Database 1: MySQL Database name: HEALTHCARE** | | | |
|---|---|---|---|
| Table name | Attribute name | Primary key | Foreign key |
| PATIENT | P_ID, NAME, ADDRESS, AGE, PHNO, DOC_ID | P_ID | DOC_ID |
| DOCTOR | DOC_ID, DNAME, SPECIALISATION, PHNO, ADDRESS | DOC_ID | |
| DOCSCHEDULE | DOC_ID, DNAME, VISITING DAY, VISITING HOUR | DOC_ID, DNAME | DOC_ID |
| **Database 2: Mongo DB Database name: RHC** | | | |
| Table name | PATIENT, EPRESCRIPTION, PRESCRIPTIONDETAILS | | |

One remarkable thing in the Mongo DB database is the declaration of attribute. Because, in Mongo DB database attribute declaration is not mandatory for storing data. But for database to database interconnectivity, some basic attributes in the collections of Mongo DB have been declared. These attribute declaration is supportive for further metadata representation.

In MySQL's PATIENT table, being the primary key, P_ID maintains functional dependency relationship with other attributes. And the NAME attribute also maintains functional dependency relationship with AGE attribute. Again, as the foreign key of the PATIENT table, DOC_ID manages inter-table relationship with other tables and manages the efficient data collection. Following the same process, in Mongo DB database, common attributes P_ID have been declared in PATIENT collection as $T4_1$, in PRESCRIPTIONDETAILS collection as $T5_1$ and in EPRESCRIPTION collection as $T6_1$. In the complete scenario, P_ID of MySQL database as well as Mongo DB Database will maintain the Inter-database connectivity. The unique P_ID usage indirectly maintains database to database connectivity, which helps to collect patient details by a doctor.

Finally, the metadata representation can illustrate the above use cases using their attributes having interconnectivity among them.

Using the root structure in the JSON format given in Figure 3, it is easy find the common attributes by finding the leaf nodes. The relationship provided by the commonality in the leaf level or child level can help to investigate the suitable data or sets of data in a single fashion or in integrated fashion by interrogating their concern schemas and their proper databases and.

This schematic presentation is applicable in between two types of used services. I.e. the platform is suitable to reside within the Application service and Data service. It helps to interrogate the requested data after placing user request at the Application service side. Because, for data interrogation using metadata representation would be further helpful for collecting data form the concerned databases within minimal effort through the Data services.

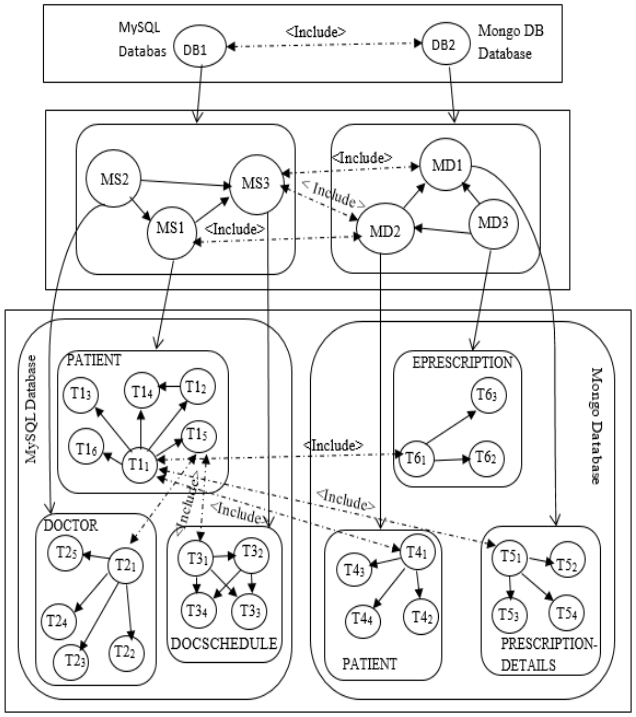


Fig. 2. Use case diagram for storing health care data in MySQL and Mongo DB database against the concept of MDIP graphical approach

TABLE IV. DIFFERENT COMPONENTS OF USED CLOUD DATABASES WITH IDS FOR ILLUSTRATION OF MDIP

| **MySQL Database: DB1** | | | |
|---|---|---|---|
| Schema name | ID | Attribute name | ID |
| PATIENT | MS1 | P_ID | $T1_1$ |
| | | NAME | $T1_2$ |
| | | ADDRESS | $T1_3$ |
| | | AGE | $T1_4$ |
| | | DOC_ID | $T1_5$ |
| | | PHNO | $T1_6$ |
| DOCTOR | MS2 | DOC_ID | $T2_1$ |
| | | DNAME | $T2_2$ |
| | | SPECIALISATION | $T2_3$ |
| | | PHNO | $T2_4$ |
| | | ADDRESS | $T2_5$ |
| DOCTORSCHEDULE | MS3 | DOC_ID | $T3_1$ |
| | | DNAME | $T3_2$ |
| | | VISITING DAY | $T3_3$ |
| | | VISITING HOUR | $T3_4$ |
| **Mongo DB Database: DB2** | | | |
| Schema name | ID | Attribute name | ID |
| PRESCRIPTIONDETAILS | MD1 | P_ID | $T5_1$ |
| | | P_NAME | $T5_2$ |
| | | P_REPORT | $T5_3$ |
| | | MEDICINE_LIST | $T5_4$ |
| PATIENT | MD2 | P_ID | $T4_1$ |
| | | P_NAME | $T4_2$ |
| | | AGE | $T4_3$ |
| | | PHNO | $T4_4$ |
| EPRESCRIPTION | MD3 | P_ID | $T6_1$ |
| | | P_REPORT_DATE | $T6_2$ |
| | | DNAME | $T6_3$ |

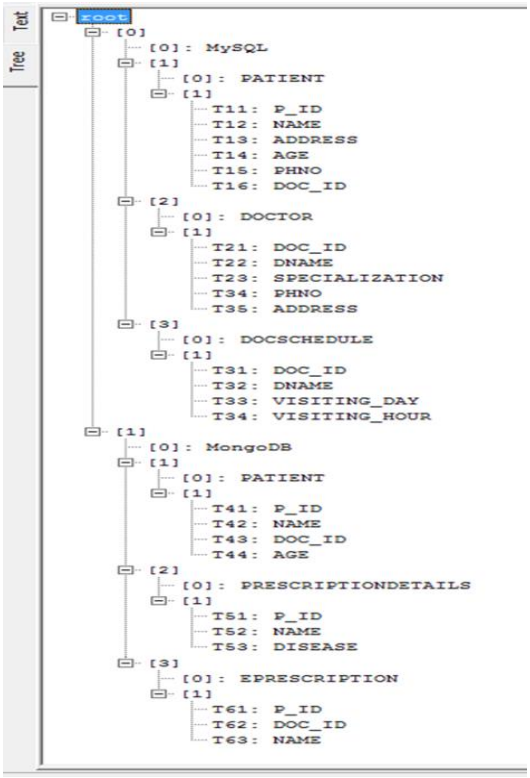


Fig. 3. Tree structure for different databases metadata representation

TABLE V. SET OF QUERIES OF MYSQL AND MONGO DB FOR QUERY EVALUATION TIME

| | *MySQL query MongoDB query* | *MongoDB query* |
|---|---|---|
| 1 | `Select * from patient where p id= ?;` Select | patient prescription details |
| 2 | `Select * from doctor where doc id= ?;` | Selects patient details |
| 3 | `Select count (*) from patient where doc id =? group by p id;` | Selects patients details |
| 4 | `Select a .dname, a . specialisation, a . address, b . name, b . p_id from doctor a, patient b where a .doc_id = b .doc_id;` | Select doctor details |
| 5 | `Select a.dname, a . visitingday, a . visitinghour, b . name, b . p_id from docschedule a, patient b where a . doc_id = b . doc id;` | Select doctor details |

TABLE VI. QUERY EVALUATION TIME MEASURED IN MICROSECONDS

| | Q1 | Q2 | Q3 | Q4 | Q5 |
|---|---|---|---|---|---|
| SLDI Case 5 | 2,781 | 3,221 | 3,971 | 4,719 | 9,156 |
| Using MDIP | 1466 | 1398 | 1753 | 1871 | 2527 |

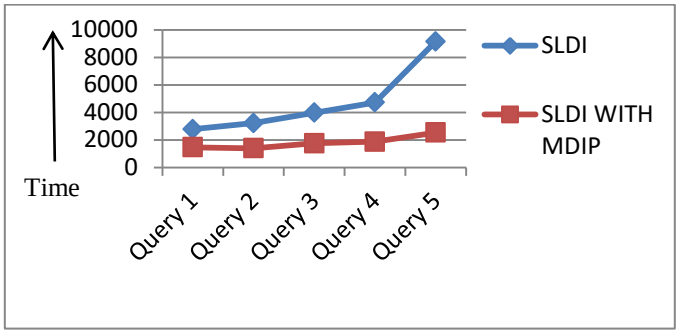


Fig. 4. Comporison chart of SLDI and SLDI with MDIP for quality assessment

## V. QUALITY ASSESSMENT THROUGH COMPARISON ANALYSIS BY QUERY EVALUATION

In concern to evaluate the quality assessment, incorporating the proposed MDIP with SLDI [12], query evaluation time has been measured. To get the integrated result, the same set of queries have been evaluated which were used in SLDI [12] are given in Table 5. The given sets of quires are able to collect data from the concern databases in an individual way. But to collect the individual results in an integrated way, multiple database function calling can be done under a single loop, like,

```
If (p_id= ?) search
{
MySQL function();
MongoDB function():
}
```

For showing the better quality evaluation for the proposed mechanism, the implementable experimental query evaluation time has been compared with existing query evaluation time done in SLDI paper [12].

Form the existing SLDI approach, the case number 5 (having 2 different databases, 2 different data services for connecting those databases individually and single Application service) has been selected. For the accurate comparison result, the proposed MDIP mechanism has been implemented over the same set of queries to get the evaluated time during integrated data retrieval. Here the respond time has been measured in microseconds and the measured time is given in Table 6. After plotting the query evaluation time of SLDI case 5 and SLDI case 5 using MDIP in the comparison chart plotted in Figure 4, the measured growth rate shows the better quality for MDIP incorporated SLDI. Because, during simple query evaluation the difference within query evaluation time of two different mechanism are lower. But whenever, the type of query becomes more complex, the difference within query evaluation time becomes greater. That shows the better performance during MDIP usage in SLDI mechanism. So, the usage of MDIP causes lower time consumption in a drastic way during complex query evaluation. That shows the better performance during MDIP usage over SLDI mechanism.

## VI. Solved Issues by MDIP Framework

The presence of individual platform in the service level for monitoring the database integration concept, this MDIP approach has been proposed. Unlike the existing models which act as the database integrator in different way, this proposed MDIP framework differs because of its ability to heal the unsolved functional as well as non-functional issues during its application. Here the summarised solved issues are given below. Those are,

### A. Database adaptation flexibility:

Unlike any other service based database integration mechanisms, the proposed MDIP mechanism over service based database integration doesn't suffer from database adaptation. Intellectually it supports flexibility to accept any newer database's model and is able to deliver the detail description of that database maintaining the commonality with another existing database description. For this type of resolution, the mechanism effects the database integration during users query evaluation.

### B. Database heterogeneity:

The graceful concern for database adaptation flexibility in MDIP scenario explains multiple databases support for cloud environment. This concept for multiple databases always doesn't ensure similar types of databases, but also it ensures the support of heterogeneous types of databases.

### C. Distribute support:

The applicability of MDIP in cloud environment along with its heterogeneous support indirectly explains the distributed support. Because, during heterogeneous databases support always does not ensure that the databases are the residence of a single location, rather it revels the positions of those databases in distributed location in the cloud environment.

### D. Dynamic Identification of data location:

The MDIP mechanism simplifies the deliverability of the integrated view of multiple databases through the data identification against a single query, either form a single database or from multiple numbers of databases using the commonality and relationship among databases. So, before searching the databases blindly to find the appropriate data after placing query, the mechanism identifies the exact data location. This causes helpful for further data response.

### E. Memory space utilisation:

The proposed MDIP mechanism efficiently response users query by providing the integrated view after individually capturing the data sets from multiple databases. So, in the mechanism there is no need to overwrite the data form one database to another to supply the integrated view of requested data instances. For this reason, MDIP avoids data redundancy and causes lower space utilisation.

### F. Cost efficiency:

As the developing platform of the MDIP mechanism is service level, the implementation details causes lower development cost. Again for service usage, this software based implementation also causes lower maintenance cost and its efficient data response also degrades the data availability cost.

### G. Data availability:

For data response after placing the users query, dynamic data location identification in the flexible metadata format for multiple databases decreases the data searching time in a remarkable way. This additive feature over service level database integration causes more effective data availability.

### H. Data consistency:

The concept of data consistency is to manage the successful incorporation of the latest updated data in the concern database during data handling. The proposed MDIP mechanism is suitable to accept eventually the final updated data model, which causes deliverability of the last updated data within a short time span. This concept explains the data consistency support.

### I. Data partitioning:

For any user query evaluation, the implementation of MDIP mechanism mandates to find out the concerned data sets from multiple numbers of databases using their relationship and commonalities. During data set searching mechanism, data location may be identified through the previously partitioned metadata structure of the databases, which shows the data partitioning. Eventually this concept supports the effective data response.

### J. User side scalability:

Because of the service level implementation, the integrator platform would reside in between Application services and Data services. Where, Application service is responsible for user interaction and the Data service is responsible for database interaction. For the attachment of these two types of services, the multi-tenancy [16] feature of the services would support the scalable numbers of end users in accordance with their needs.

### K. Overall efficiency:

This is a non-functional factor for checking the overall strength using the intended output. For the proposed MDIP mechanism, the ease of data response with the help of different impact factors discussed previously, explains overall efficiency.

## VII. Conclusion and Future Work

The proposed MDIP mechanism is the progressive approach over any service level database integration to ease the data response against maximised customers need. Beside the service level database integration, the MDIP mechanism can be implemented in an individual service level platform. This resides within Application service and Data service and applicable over any service based database integration. The working principle for the mechanism is to diagram all about for a single database using its multiple components relationship or for the multiple databases through the commonality of their components. The explanation is conducted through the multilevel graphical concept. Here multilevel concept explains the multiple stages of data components of the cloud databases. This platform does not

find the data physically from the data storage, rather it helps to find requested data or data sets from multiple heterogeneous databases using its tree structured metadata view against single user query that explains the data partitioning. Actually it helps to find the related data using the proper track. For this, the mechanism eventually causes better data response within optimised time span in presence of data consistency which has been shown through the query evaluation time comparison with existing SLDI approach over same set of queries. So, for the efficient data deliverability in the presence of data availability, data consistency and data partitioning, shows the CAP theorem [14] [15] support for the proposed MDIP mechanism. Like the ACID properties for relational databases, the desirable CAP properties support inversely explains the distributed heterogeneous database support for the proposed MDIP. Again for the service support, it can bear on scalable multi-tenant support as well as lower implementation cost and lower maintenance cost. So, the overall MDIP consideration mandates efficient data response avoiding any type of data duplication and complex query evaluation.

The future scope for MDIP would reveal some other quality matrices for the purpose of quality assessment of any other quality factors in comparison with existing service level database integration approaches. Again, the additional application 'dynamicity' over flexible database adaptation can forward the proposed MDIP mechanism towards database virtualisation. This may cause the attachment of additional cloud databases as per requirement basis and may effect with its more efficient data response.

## REFERENCES

[1] Liao, Y.T., Zhou, J., Lu, C.H., Chen, S.C., Hsu, C.H., Chen, W., Jiang, M.F. and Chung, Y.C., 2016. Data adapter for querying and transformation between SQL and NoSQL database. *Future Generation Computer Systems*, *65*, pp.111-121.

[2] Rocha, L., Vale, F., Cirilo, E., Barbosa, D. and Mourão, F., 2015. A Framework for Migrating Relational Datasets to NoSQL1. *Procedia Computer Science*, *51*, pp.2593-2602.

[3] Oluwafemi E. Ooju, Sahalu B. Junaidu and S.E. Abdullaahi, "TripleFetchQL: A Platform for Integrating relational and NoSQl Databases", International Journal of Applied Information System (IJAIS), Volume-10 No-5, February 2016, pp. 54-57.

[4] Lawrence, Ramon. "Integration and virtualization of relational SQL and NoSQL systems including MySQL and MongoDB." In Computational Science and Computational Intelligence (CSCI), 2014 International Conference on, vol. 1, pp. 285-290. IEEE, 2014.

[5] Thankgod S. Adeyi, Saleh E. Abdullahi, Sahalu. B Junaidu, "DualfetchQL System: A Platform for Integrating Relational and NoSQL Databases", International Journal of Engineering Research & Technology, Vol.2 - Issue 12 (December - 2013), pp.1973-1981

[6] Curé O, Hecht R, Le Duc C, Lamolle M., "Data integration over nosql stores using access path based mappings", International Conference on Database and Expert Systems Applications, Springer Berlin Heidelberg, 2011 Aug 29, pp. 481-495.

[7] Sangeeta Gupta, G.Narsimha, "CORRELATION AND COMPARISON OF NOSQL SPECIMEN WITH RELATIONAL DATA STORE", IJRET: International Journal of Research in Engineering and Technology, Volume: 04 Special Issue: 06, May-2015, pp.1-5.

[8] DikshaKoul, DevayaniPawar, RadhikaRanade, VishakhaPatil, "SQL2MongoDB", International Journal of Computer Science and Information Technology Research, Vol. 3, Issue 1, Month: January - March 2015, pp. 317-321.

[9] 'Database-As-A-Service Saves Money, Improves IT Productivity And Speeds Application Development'. A Forrester Consulting Thought Leadership Paper Commissioned By VMware, (October, 2012).

[10] An Oracle White Paper on Enterprise Architecture (September 2011) 'Database as a Service Reference Architecture – An Overview'.

[11] Anirban Sarkar, Narayan C Debnath, "Aspect Algebra: The Operational Semantics for Aspect Oriented Software", 9th International Conference on Information Technology: Next Generation (ITNG 2012) [IEEE], PP 139 – 144, Las Vegas, USA, April 2012.

[12] Trushna Parida, Sanjukta Pal, Anirban Sarkar, "SaaS level Database Integration in Cloud Environment through Database as a Service", International Journal of Services Technology and Management (Inderscience Publisher), in press, 2017. [ISSN print: 1460-6720]

[13] GhadaElSheikh, Mustafa Y. ElNainay, Saleh ElShehaby and Mohamed S. Abougabal, 'SODIM: Service Oriented Data Integration based on Map Reduce', Alexandria Engineering Journal, volume 52, Elsvier, 2013 pp 313-318.

[14] Seth Gilbert and Nancy A. Lynch. Perspectives on the CAP Theorem,http://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf, (Accessed 23rd july 2012)

[15] Seth Gilbert and Nancy Lynch. 'Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services', ACM SIGACT News, volume 33 Issue 2, (June 2002), pp.51-59.

[16] Sanjukta Pal, Amit K. Mandal, Anirban Sarkar, "Application Multi-Tenancy for Software as a Service", International Journal, ACM SIGSOFT, Software engineering notes, Volume 40, Issue 2, ACM, NY, March 2015, pp. 1-8.