Pippal and Kushwaha *Journal of Cloud Computing: Advances, Systems and Applications* 2013, **2**:5
http://www.journalofcloudcomputing.com/content/2/1/5

Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

A simple, adaptable and efficient heterogeneous multi-tenant database architecture for ad hoc cloud

Sanjeev Kumar Pippal* and Dharmender Singh Kushwaha

Abstract

Data management and sharing is the challenge being faced by all the IT majors today. Adds over it, is the challenge faced by the cloud service providers in terms of multi-tenancy of data and its efficient retrieval. It becomes more complex in a heterogeneous computing environment to provide cloud services. A simple, robust, query efficient, scalable and space saving multi-tenant database architecture is proposed along with an ad hoc cloud architecture where organizations can collaborate to create a cloud, that doesnt harm their existence or profitability. An ad hoc cloud fits very well to the scenario where one wants to venture into remote areas for providing education services using a cloud. The results of the proposed multi-tenant database show 20% to 230% improvement for insertion, deletion and updation-queries. The response of the proposed approach is stable as compared to other system which degrades in terms of response time by 384% for increased number of attributes up to 50. The proposed approach is also space efficient by almost 86%. Dynamically changing cloud configurations requires adaptable database and mechanism to persist and manage data and exploit heterogeneous resources. The proposed ad hoc cloud handles heterogeneity of the involved nodes and deals with node specific granularity while decomposing workloads for efficient utilization of resources.

Keywords: Ad hoc Cloud, Heterogeneity, Multi-Tenant database

Introduction

Cloud computing is a computing paradigm where services and data reside in common space in scalable data centers, which are accessible via authentication. Cloud has three delivery models namely IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a service). Cloud computing [1] services can form a strong infrastructural and service foundation framework to provide any kind of service oriented computing environment. Ad hoc clouds [2,3] enable existing infrastructure as cloud compliant and the available resources in the system are utilized non-intrusively. An Ad hoc cloud is very efficient solution to problems faced by organizations to venture into remote areas for their IT infrastructure and support needs. Ad hoc cloud proliferate stakeholders to provide competitive services in a collaborative way.

*Correspondence: sanpippalin@gmail.com
Department of Computer Science and Engineering, MNNIT Allahabad, Allahabad, India

Most existing business entities refrain from spreading their reach to remote geographical regions because of concern of huge upfront investment and profitability. The same is true for establishing institutes in these areas where the admission seekers exist but the location is so remote or hostile that many of the professionals would be reluctant to work. For these scenarios, Ad hoc cloud holds huge potential and promise in starting these ventures. The resources of parent or fixed education cloud initially act as a feeder for this new establishment. Once the system is working and matures with time, this ad hoc infrastructure gradually translates into persistent setup. Ad hoc education-cloud, where a cloud computing framework is harnessed to manage information system of an educational institution would be highly efficient in terms of accessibility, manageability, scalability and availability. An ad hoc cloud would enable us to harness services offered by fixed education-cloud[4] and services created and composed within an ad hoc cloud.

Multi-tenancy implies that a single instance of application satisfies the requests of multiple clients. Each individual educational organization is considered as a tenant and all such organizations collaborate to create and participate in data-store building process. We propose a multi-tenant database for such a scenario where more than one tenants (Educational Institutions) collaborate to build the distributed database and use it by authorization [5]. In this scenario the tenants are free to join or leave. Providing dynamically adaptable multi-tenant database [6] with transactional level guarantee for the distributed data base to be used as a data store in the cloud formed with heterogeneous resources is our concern.

The major goal of this work is to implement multi-tenant database for an ad hoc cloud at remote location and provide the following sub goals to:

1. Provide an architecture that supports multi-tenancy in shared database shared schema scenario.
2. Find the best granularity level at which the work decomposition is to be done for heterogeneous environment.
3. Manage heterogeneity in terms of varying attributes, database technology and resources.
4. Optimize scheduling criteria and also provide load balancing.
5. Manage scalability and performance.

In order to meet the above goals we have developed a simple architecture that supports multi-tenancy and which work at optimum granularity with support for scalability and data management.

The rest of the paper is organized as follows. Section 'Related work' explains the related work done earlier. The details about the proposed architecture are elaborated in Section 'Proposed approach'. Section 'Results' presents the results obtained for the proposed architecture and section 'Conclusion & future work' provides the future scope of the work and concluding remarks.

Related work

Heterogeneity related work

Heterogeneity in terms of the cloud resources implies differences or variations in computing power of resources that could create further issues of performance and reliability. Some of the significant related work concerning Heterogeneity, Granularity, Replication, Load balancing and Scalability are:

A significant amount of work on load-balancing has emphasized on cluster based distributed systems. Condor [7] and Mosix [8] depend on check pointing and process migration to do load balancing in a cluster based distributed system. The heterogeneity of the cluster of workstations is managed by dynamically collecting load information and migrating active processes between cluster nodes to balance the load. This kind of load balancing techniques can be complementary to our work allocation techniques that focus on initial allocation of tasks according to capabilities of a node. Clusters of workstations have also been employed to host Web and Internet servers. A large amount of work on such cluster-based network servers has focused on request distribution as a means for handling the load imbalance in the cluster. Load-aware request distribution [9,10] use content-based request distribution which considers the locality of data and the load on the cluster nodes. Aron et al. [11] emphasizes on request isolation and resource management on cluster based distributed systems while [12] proposes cluster load balancing policies for fine grain network services. Load sharing in heterogeneous systems has been widely researched. [13] Evaluates and compare different load sharing algorithms for heterogeneous multicomputer systems. Goswami et al. [14] propose dynamic load sharing heuristics which manage workload in a distributed system by judging the resource requirements of processes. The author in [15] uses a proactive load sharing scheme for distributed systems which prevents the occurrence of load imbalance by collecting load and task execution behavior information in advance.

Karatza et al. [16] analyze load sharing policies for heterogeneous distributed systems to study the effect of load sharing on different classes of jobs. Berman et al. [17] explain an application specific scheduling approach for scheduling data parallel applications on a heterogeneous distributed system. Nieuwpoort et al. [18] elaborates load balancing strategies specifically for divide and conquer applications on a hierarchically organized distributed system. Kondo et al. [19] take into consideration a similar system model as ours and propose techniques for resource selection for short-lived applications on enterprise desktop Grids with the aim of minimizing the overall execution elapsed time of a single application. We consider a similar scenario but propose algorithms and heuristics for deciding the decomposition of tasks in order to load balance in a heterogeneous set of computation resources. Such scheduling algorithms have also been an active area of research in the field of divisible load scheduling. [20] Provides an overview of the research done in this field for master/worker architectures. Many approaches for scalability and data management services have been proposed like big table [21] and dynamo [22], but lacks in providing transactional level guaranty.

Multi-tenancy related work

Various approaches for multi-tenancy have been proposed depending on the degree of isolation. Three broad approaches are:

1. Separate database: In this approach, a separate database is used to store the data of an individual tenant.
2. Shared database, separate schema: This approach requires multiple tenants to be accommodated into a single database.
3. Shared database, shared schema: This approach involves same database and schema to be shared by all tenants.

Some of significant related work for providing multi-tenancy is as follows:

Universal table layout

A universal table [23] contains pre-specified number of fields. It consists of a Tenant_id column, a table column and all the data columns. Tenant_id is used to uniquely identify the data of a tenant whereas the table column refers to the id of the table for that tenant. This approach has been originated from Universal Relation where a table holds all the columns from all the tables. This approach is relatively easy to implement and queries are applied directly to the table.

Chunk folding

Chunk folding is a technique discussed in [24]. It vertically divides the logical tables into chunks and those are folded together into various physical tenants and are joined as needed. One table is used to store the base account information and other table is used to hold the extensions. This approach works by containing the heavily used parts of the schema into base tables and the rest part is mapped into the extensions.

Extension tables

The concept of extension tables came into picture after the development of decomposed storage model described in [24]. It divides a table of n-columns into n 2-column tables that are merged together. One problem with this approach is how to partition the table so that after joining these tables no extra information is generated.

Pivot tables

In this approach, a pivot table is created for a single column [25]. This table is shared by various tenant's tables. Each pivot table consists of a tenant column, Table column, a col column and a row column. Tenant column refers to the particular tenant. Table refers to the particular table for that tenant.

Multi-tenant shared table

In this approach, common contents from tenant information are separated as in [25]. This technique introduces the concept of tenants at database layer so that database engine can select an appropriate area for storage of data for that tenant.

An approach that deals with scalability issue is discussed in [26]. Two main problems are resolved; one is to resolve the sparseness of the universal table approach and second is to provide an indexing scheme for multi-tenant database. Three different approaches shared machine, shared process and shared table are discussed by Jacobs in [27]. In [28], a simulation study is done which analyzes the performance of different approaches to implement the multi-tenant databases. An approach for multi-tenant architecture supporting the SaaS model is discussed in [29]. The authors have proposed a cloudio software platform that is concerned with the flexibility of data model and managing the large data sets in the database.

Different challenges in multi-tenant applications are discussed in [27] such as scalability, security, performance, zero downtime and replication in [30].

Proposed approach

An ad hoc cloud is proposed with data persistence model along with task allocation and load balancing system, which works at best granularity. An efficient multi-tenant data base is also proposed. The load allocation system supports node specific granularity calculation for optimum allocation of resources in the environment. Ad hoc cloud architecture scenario is shown in Figure 1 along with its data center model which includes an efficient multi-tenant database.

An Ad hoc cloud derives data and cloud services from fixed cloud, further they are connected using an ad hoc link (V-SAT). The S, P and V nodes in the ad hoc data center represents Super-node (Permanent node at remote location with ad hoc connectivity with the fixed cloud to facilitate cloud formation at remote site), Persistent-node (organizations hosting cloud and data services) and Volunteer-nodes (other participating nodes within an organization). The S nodes promote the stake holders to establish their own collaborative dispersed data center. The P nodes within a data center provide reliability and availability through replication of services and data. The V nodes can voluntarily cache data and provide availability and performance in the absence of persistent node and large number of requests. The nodes participating in data center can be heterogeneous in terms of computing resources, database technology. All nodes participating in the data center are logically hierarchically organized and communication between them is encrypted with key shared and provided by hierarchically common parent node.

Data persistency

An ad hoc Data-center is proposed having some Super (S) nodes, some Persistent (P) nodes and other Volunteer (V) nodes. S nodes are permanent; P nodes are persistent node that store data on ad hoc basis and V

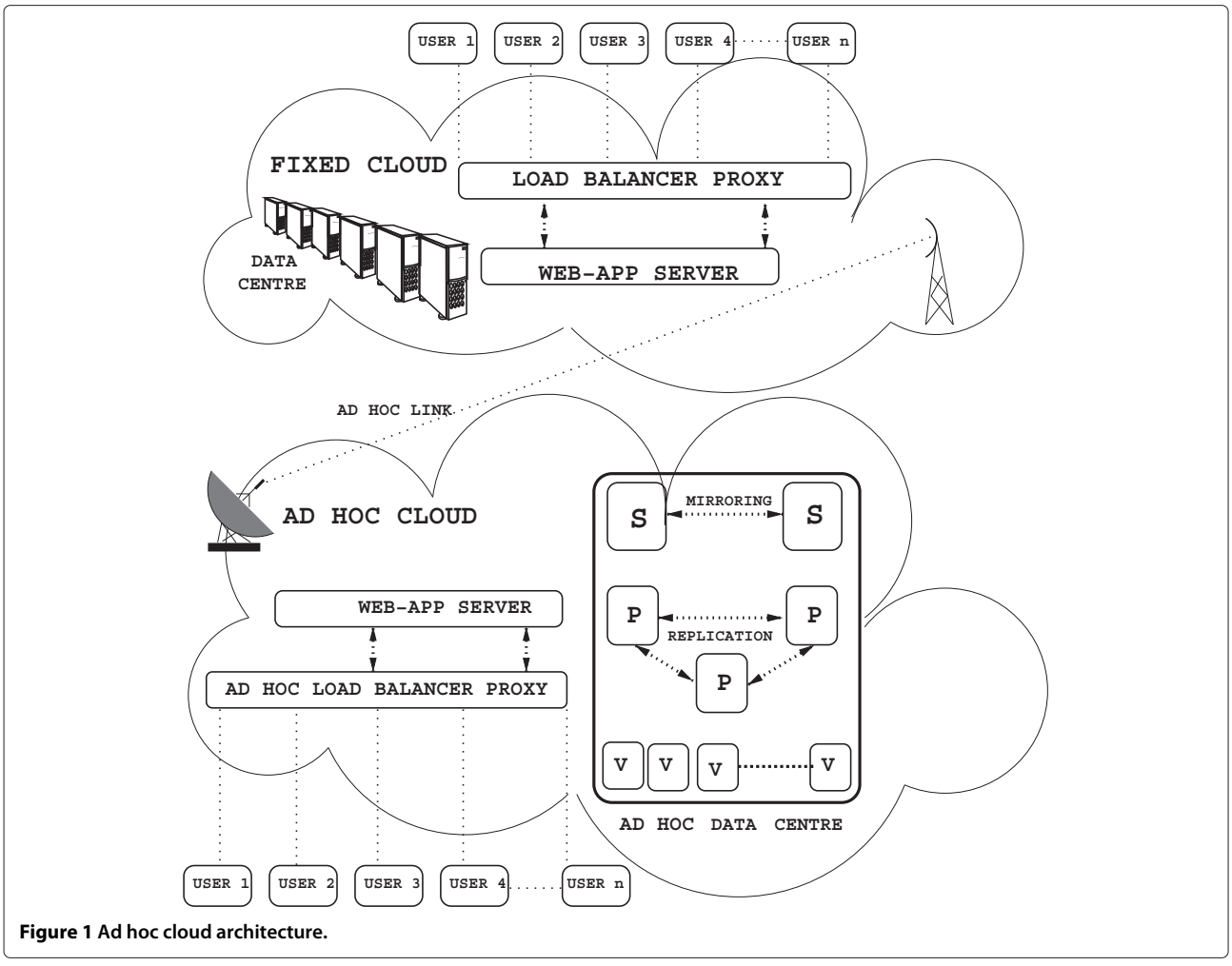

Figure 1 Ad hoc cloud architecture.

nodes voluntarily participate in Data-center. Mirroring is performed between S nodes to provide reliability, replication is performed between P nodes to increase availability and improve reliability further V nodes acts as new data sources or cache data for performance as shown in Figure 2. Data consistency is maintained for replicas using eager update protocol for frequent updates and lazy protocol is used for infrequent updates.

As shown in Figure 2 the OLTMs (Organizational Level Transaction Managers) are resource manager application pertaining to specific organizations. OLTM manage transaction within organizations whereas cross organization transactions are managed with help of HLTMs (High Level Transaction Managers). Each and every node participating in the data center is logically hierarchically organized with S node taking the root of the tree position with mirror support, P nodes as intermediate nodes in the tree and V nodes taking the leaf levels. Various issues arising out in data persistency like replication, granularity, failure handling and data domain are explained in the following sections.

Replication strategy, schema and data usage

The replication approach vary with the types of nodes and their characteristics as shown in Table 1. A nonvolunteer user does not replicate or store anything, it just uses the system whereas a volunteer node always replicate on demand and while quiting submits all at site updations to the hierarchical parent node or nearest neighbor node.

The persistent nodes always replicate to increase availability maintaining data consistency. The super nodes between themselves implement mirroring periodically so as to provide reliability. The S, P, V, nodes always downloads the schema in first use. The data population if V node is done on demand basis, whereas P and S nodes always update their data as consistency requirements.

Data domain and replication granularity

The data semantics of every type of user is bounded by boundaries as shown in the Table 2.

The V nodes data requirements are user need specific, P nodes data requirements are organizational specific whereas the S node data requirements are administration,

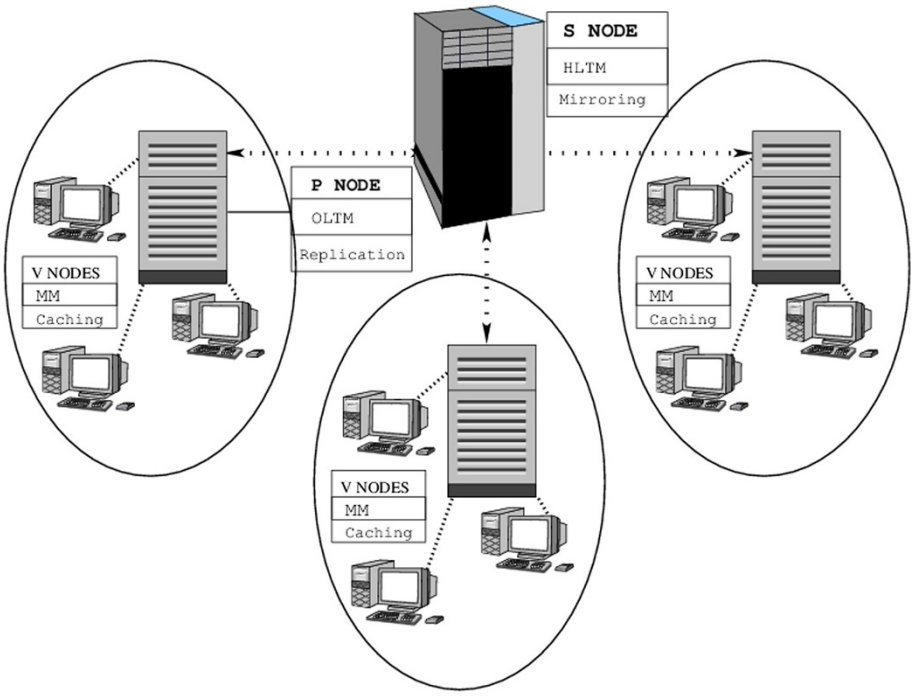

Figure 2 Ad hoc dispersed data center.

system support and management specific. The replication granularity for V node is record level dump task specific, for P node first complete dump is copied and further differential dump is used based on check pointing and for S nodes complete copy of the database dump is used among them.

We propose a simple application level check pointing approach for finding the differential dump that is records modified after a time stamp.

Tables 3 and 4 are used to manage the process of finding record level modifications. Table 3 is shown having R_id as primary key in all related tables along with other attributes and CKP_F being a flag field to denote the modification of records. Initially the field is set to F (false) indicating no modification, it is set to T (true) when there is a modification by itself or a replication update from some other node is received. Table 4 stores modified record ids and time stamp of recording the checkpoint in a table called checkpoint table.

Table 1 Replication approach, schema and data use

Type of user	User characteristics	Schema & data population
Non Volunteer	User have no contribution in data center	No & never
Volunteer	User allows sharing but can exit anytime	Yes & when in use
Persistent	User are stake holders	Yes & always
Super	Super users	Yes & always

Failure handling strategy

For the various classification of nodes the specific failure handling strategy in case of node failure is given. A proactive approach is used by the failed nodes, after recovering from failure the V nodes and S nodes populate themselves using a differential dump.

Application level Check-pointing is used to calculate difference. Further as shown in Table 5. S nodes use complete dump from mirror sites.

Load balancing

At primary level load balancing is done by the DNS using a round robin scheduling among the p nodes available in the working set. For load balancing in a heterogeneous environment the important decision parameters are granularity of sub-task, application requirements, computation, communication resources available and task dependency. In our solution to this problem we develop a heuristic

Table 2 Data domain and replication granularity

Type of user	Data domain	Replication granularity
Non Volunteer	Only surfing	Not required
Volunteer	Need Specific	Difference dump (record level)
Persistent	Organizational specific	First Complete then Difference dump
Super	Administration specific	Complete dump (database dump)

Table 3 Tenant table with check-pointing

R_id	.....	CKP_F
1	—–	F
2	—–	T
3	—–	T
4	—–	F
5	—–	F

that works upon the above mentioned parameters and also takes into account the total number of nodes available in the working set and the total size of the task. Generic granularity for a task (participation in data center) can be calculated as follows.

$$Generic_{Granularity} = Total_{Tasksize}/Total_{Numberofnodes} \quad (1)$$

We improve the granularity calculation by inculcating computation and communication resources available respectively. We redefine this generic granularity to be as specific heuristic granularity w.r.t. specific nodes. We take into account the last successful subtask execution and also history of such executions to calculate granularity.

For the entire network, Min_granularity is defined as per Min_bandwidth , Min_memory and Min_CPU available. Similarly Max_granularity is defined.

Algorithm 1 WORKLOAD DECOMPOSITION($Task_{size}$, $History_{decomp}$)
Require: $Task_{size}$ *and* $History_{decomp}$.
Ensure: *Optimal granularity selection.*

1: $Current_{decomp} \leftarrow History_{decomp}$
2: $Old_{granularity} \leftarrow Task_{size}/History_{decomp}$
3: **while** TRUE **do**
4: $Current_{decomp} \leftarrow (Current_{decomp} + Hi_{decomp})/2$
5: $Current_{decomp} \leftarrow (Current_{decomp} + Hi_{decomp})/2$
6: **if** *Task executed at* $New_{granularity}$ **then**
7: $History_{decomp} \leftarrow Current_{decomp}$
8: $Best_{size} \leftarrow New_{granularity}$
9: **return** WORKLOADDECOMPOSITION ($Best_{size}, History_{decomp}$)
10: **else**
11: **return** $Best_{size}$
12: **end if**
13: **end while**

Table 4 Checkpoint table

CKP_id	TS (Time Stamp)	R_id
1	1.0	2,3
2	1.1	1,3
3	1.2	0

Table 5 Failure handling approach

Type of user	Replication approach	Failure handling
Non Volunteer	Not required	Not required
Volunteer	Submits before it switches off	Populate from previous timestamp
Persistent	Replicate always	Copy nearest replica (record level)
Super	Mirror always	Copy attached mirror (database dump)

The execution capability of a system is subjective and depends upon factors like $CPU_{available}$, $Memory_{available}$, $Network_{bandwidth}$, $Disk_{available}$. Therefore to objectively decide the specific node for execution for specific task, a heuristic is needed to assign task to a specific node. To do this the resources like CPU, Memory and Network bandwidth are graded from 0 to 1. Any request for execution is mapped to best fit node as per required resources for execution. The nearest match to request is allocated and the task is scheduled to execute on the matched node. The normalized node profile table for 10 nodes is shown in Table 6. Where each resources is graded between 0 to 1. The grades are decided as per the min and max unit of the resource present in the environment and, min is assigned 0 and max is assigned 1 and all intermediate nodes are graded accordingly.

The Table 6 shows resource statistics for a partial working set among the nodes, Table 7 grades and normalizes them between 0 to 1 for all resource instances.

Further Table 8 is calculated from Table 7 to provide a grid that enables us to take decision regarding task scheduling linked with execution node id. The task is allocated to a node which is just almost capable for execution. The Assign Exec. flag in the table indicates the node being assigned for task execution of size equal to or less than its RAM size.

Table 6 Node resource profile

NODE ID	RAM (MB)	CPU (MHz)	N/W (Kbps)	HDD (MB)
N1	400	600	60	6000
N2	700	1800	70	5000
N3	100	600	50	4000
N4	1000	400	70	1000
N5	100	1600	90	5000
N6	700	2000	50	4000
N7	100	1400	40	2000
N8	100	1800	30	2000
N9	200	1400	40	2000
N10	600	1800	40	3000

Table 7 Node profile normalized data

NODE ID	RAM	CPU	N/W	HDD
N1	0.4	0.6	0.6	0.6
N2	0.7	0.9	0.7	0.5
N3	0.1	0.3	0.5	0.4
N4	1.0	0.2	0.7	0.1
N5	0.1	0.8	0.9	0.5
N6	0.7	1.0	0.5	0.4
N7	0.1	0.7	0.4	0.2
N8	0.1	0.9	0.3	0.2
N9	0.2	0.7	0.4	0.2
N10	0.6	0.9	0.4	0.3

The following rules are used while making decision:

1. JOB_{size} is equal to RAM_{size}.
2. Free $Disk_{space}$ is ten times of RAM_{size}.
3. If RAM_{size} vs $Disk_{space}$ ratio is less than 1 : 10. THEN alter RAM_{size} by $DISK_{size}/10$ in the Table 6.
4. If $N/W_{bandwidth} < (1/6)$ of RAM_{size} implies discard node for participation in data center.
5. If $CPU_{available}$ <2 times of RAM_{size} implies discard node for participation in data center.
6. If cumulative sum of the normalized grades is less that 1 unit discard the node for participation in data center.

Following rules are used to assign values in Table 8 and decide about task assignment:

NODE ID: Node id of nodes involved.
RAM vs HDD: if Ratio of $RAM_{size}VsHDD_{size}$ more than 1:10 then TRUE else FALSE.
N/W:Thresh hold >: $Task_{size}/6$ kbps implies TRUE.
CPU: If available CPU is greater than equal to $2*Task_{size}$ implies TRUE.
TOTAL: Total sum of grades < 1 implies no assignment.
ASSIGN EXEC.: $*$ implies RAM_{size} Altered.

The heuristics were developed after many iterations of execution with different values of the proportion factor and finally these values were experimentally determined and found to produce reasonable results.

Scalability and data management

We propose a light weight data store capable of providing transactional level guaranty. Our data store would have Organizational level transaction manager (OLTM) and Higher level transaction manager (HLTM) as shown in Figure 3. The transactions within an organization would be handled by OLTM and between organizations would be handled by HLTM. Elasticity at data store level is important as it would not limit upper layers for scalability. The Meta-data Manager (MM) implementation provides decoupling of database and transaction manager and it also provides mapping of distributed database partitions into OLTM. Synchronous replication of MM is required for fault tolerance. Storage layer takes care of replication of data and fault tolerance. Slower nodes can use meta data caching for improved performance. Since HTLM are stateless therefore to improve performance during scalability spawning a new HTLM is easy. Further data base migration between data-store or in cloud can be done as discussed in Albatross [31].

Maintaining a working set

In ad hoc cloud nodes can be joining and leaving randomly, so it is important to formulate a mechanism to find out live donation based or volunteer resources, which can be exploited for task execution. To solve this issue we maintain a working or live set of processors. Table 9. below show different scenarios which different types of node may exhibit. The hierarchically parent node always keeps track of live and volunteering to donate resources and keeps propagating this information up in the hierarchy. As soon as a node quits it is immediately removed from the working set of processors. A node may also be removed due to node or communication failure. The node resuming after failure initiates for updating of its local database. When a (P) persistent node joins the working set for the first time it downloads the schema, and data is replicated in entirety within domain, only if a threshold number of requests are received. If a node rejoins it calculates difference using check-pointing and does a record level differential replication.

Before quitting it either submits to hierarchical node or replicates to nearest neighbor. In case of a (S) super node, complete backup of database is replicated to the new node. A super node never shuts down randomly or

Table 8 Decision grid for task assignment

NODE ID	RAM vs HDD	N/W	CPU	TOTAL	ASSIGN EXEC.
N1	T	T	T	1.9	TRUE
N2	F	T	T	2.8	*
N3	T	T	T	1.3	
N4	T	T	F	2.0	FALSE
N5	T	T	T	2.3	TRUE
N6	F	T	T	2.6	*
N7	T	T	T	1.4	TRUE
N8	T	F	T	1.5	FALSE
N9	T	T	T	1.5	
N10	F	T	T	2.2	*TRUE

*in Table 8 implies representation of RAM_{size} is altered.

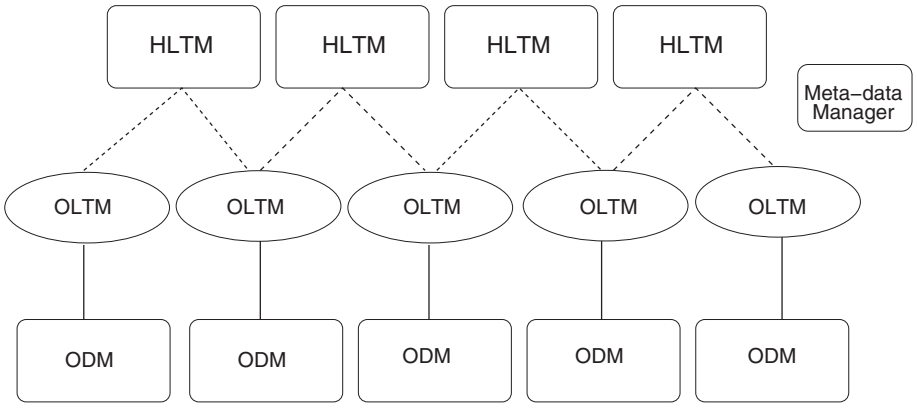

Figure 3 Decoupling of database and transaction managers.

frequently, for maintenance related shut down, differential backups may be used for consistency requirements. The (V) Volunteer node rarely gets a replicated copy, if there are no persistent node nearby and for temporal requirement a volunteer node may also act as a replica, but as soon as the requirements are satisfied it submits back, or if the persistent nodes are up, the persistent nodes bully the volunteer nodes and all further requests are served by the persistent node after handover.

Multi-tenant architecture

The Proposed approach for multi-tenant database is designed over the shared database shared schema technique. We decided for shared database and shared schema approach as it is suitable for Large number of tenants with lesser data and hence entries (as required by our application) and thus minimizing cost and leverage benefit of using same h/w, s/w, database, schema and table for all tenants and at the same time guaranteeing them isolation and security. In case we go for shared database approach we would be limited by the number of instances of database supported by the DB server. So adding more tenants will add more cost. In case, if we go for separate schema approach then in case of failure, schema restore from backup will be forced on other users also with different schema on the same data base (if no replica for the same schema is present), also (which is a time consuming task). Our proposed approach makes use of extension table.

Table 9 Events and associated actions of a participating user

Events	Action
New user joins	Schema download
User re-joins	If require schema update (schema populated with exported XML) check pointing used to find difference
Daemon user	Regularly schema and data exports updated
User quits	Submit, replicate, mirror as applicable

Extension table approach

In the universal table model it is a big challenge to decide the number of custom fields (columns in table). Providing less number of columns might restrict the tenants who wish to use a large number of custom fields. A large number of such fields may result in large number of NULL values in the database table. Second problem is of differing data types of these columns [32].

In recent times, the use of multi-tenant database systems increased multi-fold. In multi-tenant database a data center is hosted by a service provider and the tenants subscribe to the services provided by the service provider [26,33]. Figure 4 shows three tables used in the basic approach that makes use of extension table. The primary table keeps the Tenant_id and record_id and some other fields. record_id field uniquely identifies the transaction made by a particular tenant. By extracting the value of record_id field, one can extract the values from the extension table. For a single record_id, there are number of rows in the extension table.

The number of rows for a particular record_id is the number of columns in the logical table of that tenant. The Meta Data table tells about the data types of these fields.

Whenever a tenant inserts data into its table, Meta Data table is accessed to match the given values against the data types of the Meta data table. An Extn_ID in extension table is associated with an Extn_ID field of Meta Data table. This extension-id is unique for each column and is used to know the data type and external label of that field in the logical table for that tenant. An extension table contains the actual data for all the tenants. In case of universal table structure, columns, which are not used by a particular tenant, contain the NULL value, this results in wastage

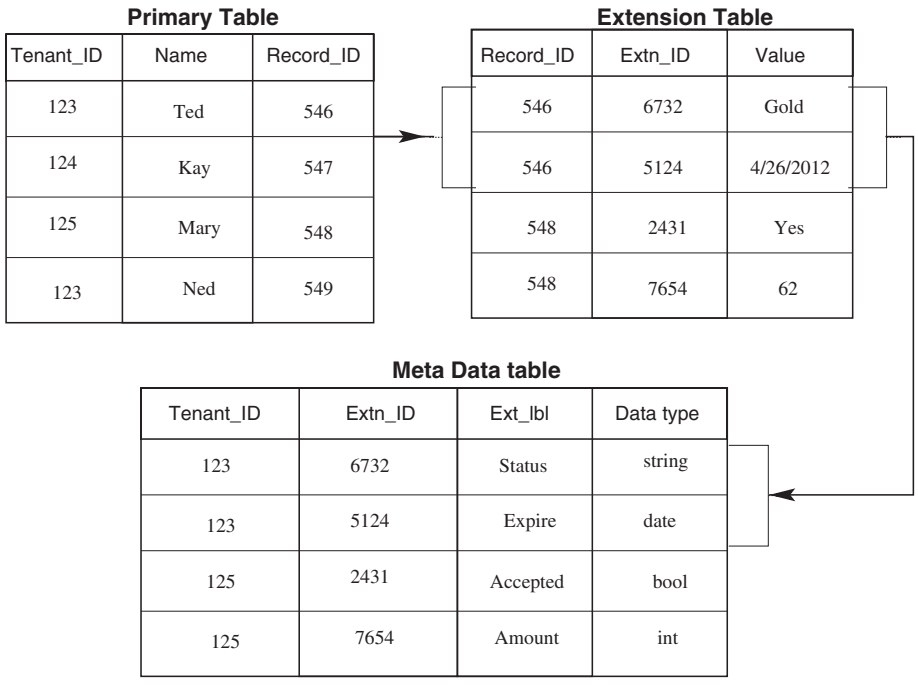

Figure 4 Use of extension table.

of space. Extension table concept overcomes this problem. Figures 4 and 5 shows an extension table and a Meta Data table.

In the basic approach of extension table discussed above, following drawbacks can be observed:

1. Extension table contains a lot of information for Meta Data i.e. for a single row of table of a tenant that consists of four columns, The Record_ID and Extn_ID are repeated four times this information introduces a kind of redundancy.
2. Whenever a query for insertion, deletion or update is performed three tables are accessed which increases the query processing time. In our proposed approach following concepts are introduced and implemented:

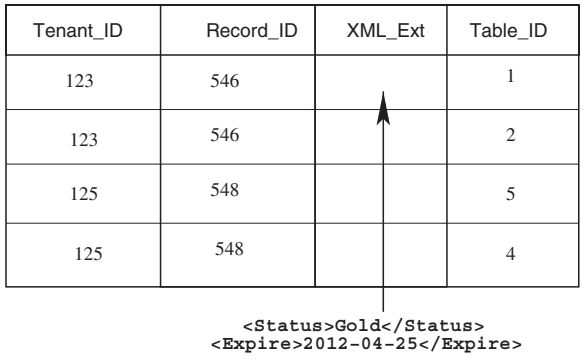

Figure 5 Modified extension table.

3. Concept of XML object into a database is used that helps to reduce the size of extension table as well as eliminates the need of a primary table.
4. An approach that achieves multiple table creation for a tenant is proposed and successfully implemented. Figure 5 shows the proposed approach where extension table consists of a Tenant_id, a Record_id, an XML attribute and a Table_id. Tenant_id and Record_id uniquely identify a particular record. A Record_id is used to associate each transaction with a unique record number. XML object contains the data for an entire row of a Tenants logical table. Tags in a single XML object refers to the name of a particular field in the corresponding table. Table_id field represents the id of the table in which a particular record is inserted for the specified tenant. The tenant specifies the name of the table and our proposed system generates unique id for that table for that tenant.

A table that maintains the information about all the tables of all tenants is created. This table maps the Table_id field of the extension table to the name of the table which a tenant is referring to.

Creation of Table

A tenant is free to use any number of custom fields assuming that service provider has created sufficient number of fields in the main database schema. A tenant is free

Table 10 A metadata table

$Tenant_ID$	Tbl_Name	$Table_ID$
123	Employee	1
123	Purchase	2
125	Customer	5
125	Funds	4

to create any number of tables and use any data type (supported by that DBMS) for its fields.

Whenever a tenant specifies a new table name, this name is stored in the table_meta_data table. Table 10 shows the structure of table_meta_data table.

Insertion in the Table

A tenant specifies the name of the table and supplies the values. Our proposed architecture follows a sequence of steps to insert the values in the main extension table as follows:

1. The table id of a particular table for that tenant is extracted from the table_meta_data table.
2. Meta_data table is accessed to know the data types of the fields.
3. A Record_id, identifying this particular transaction, is generated and is inserted into the Record_ID column.
4. An XML document with inserted values is created whose tags are the column names in the table.
5. This XML document is inserted into XML_Ext column of the extension table.
6. Table_Id extracted from the table_meta_data is inserted into the Table_Id column of extension table.

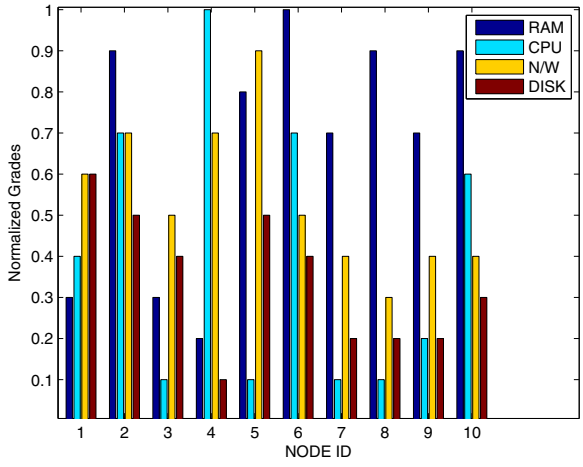

Figure 6 Sample testbed configuration.

Updating the information in the database

The table name and the name of the field is specified, and following steps are followed:

1. It accesses the table_meta_data table from where it retrieves the id of the table for that tenant
2. In extension table, it finds out the rows corresponding to that table id and tenant.
3. From the XML documents, which are related to the Table_ID, it makes a Xquery that gives field names.
4. It modifies the value in the XML document and stores it back in the extension table with the same record id.

Deleting records The name of the table along with indicative key is for a specific record is provided. Similar to the update process, the table_meta_data is accessed to know the Table_ID of the table. Later extension table is accessed to know the rows corresponding to that Table_ID. The entire row, containing The XML document in which the specified value for the given field is found, is deleted.

Results

To implement the proposed approach, MySQL database in Ubuntu has been used. Ubuntu is installed over VMware and all involved nodes (computers) are configured with heterogeneity. Factors of heterogeneity are allocated CPU power, allocated RAM and disk space and network bandwidth. To test and generate report python is used as scripting language. The test bed comprise of 90 nodes for the distributed multi-tenant data base. The processing capability ranges from 500 MHz to 2.4 GHz for processor, 500MB to 1500MB for RAM and 10-30 GB of free disk space as shown in Table 4.

The proposed approach has been successfully implemented and queries like selection, insertion and deletion have been experimented. For more added attributes in a table the performance is slightly better, and saves a lot of space as compared to the extension table approach. This benefit comes from the use of XML in the attributes.

Test bed Configurations: A sampled 10 nodes configuration is shown in the Table 4. and depicted in the following graph in Figure 6. The graph for test bed configuration shows heterogeneity among processors in terms of speed, among RAM in terms of size of primary memory, among HDD in terms of size and space available for secondary

Table 11 No. of Queries executed per sec

Type of query	Proposed approach	Extension table approach
Insertion	804	665
Deletion	804	238
Updation	662	197
Selection	747	1026

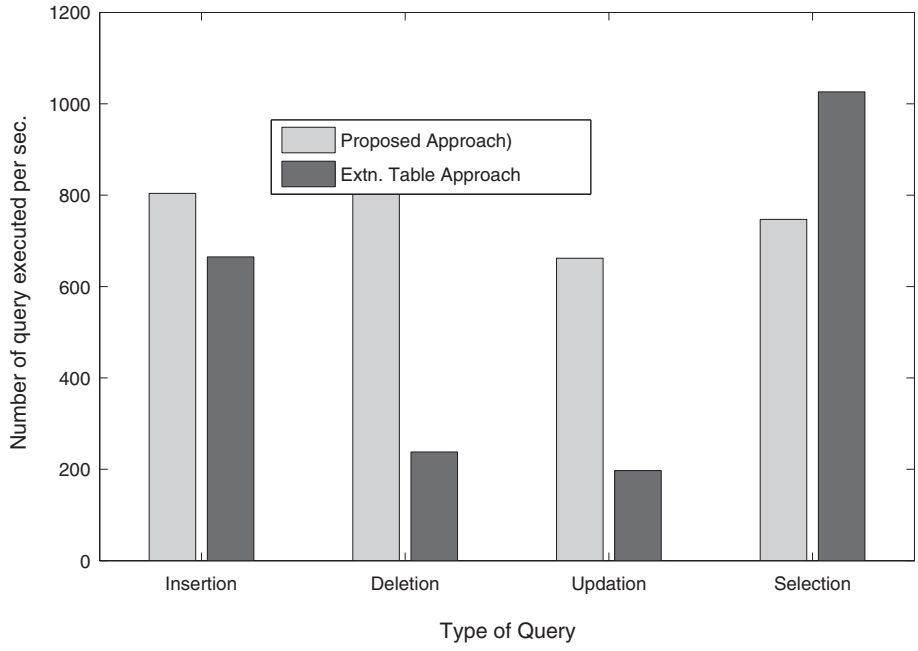

Figure 7 Showing no. of queries executed per second.

storage, among network bandwidth in terms of data rate available.

Comparison with extension table approach

The number of rows in the original extension table depends upon the number of fields in a tenants table. But in our approach it contains only a single entry for a row. Therefore a lot of space savings and also solves the NULL value problem with the extension table approach.

In Table 11 and Figure 7 it can be seen that except for select query all other query outperform the extension table approach. As shown in Figure 7 Statistically for

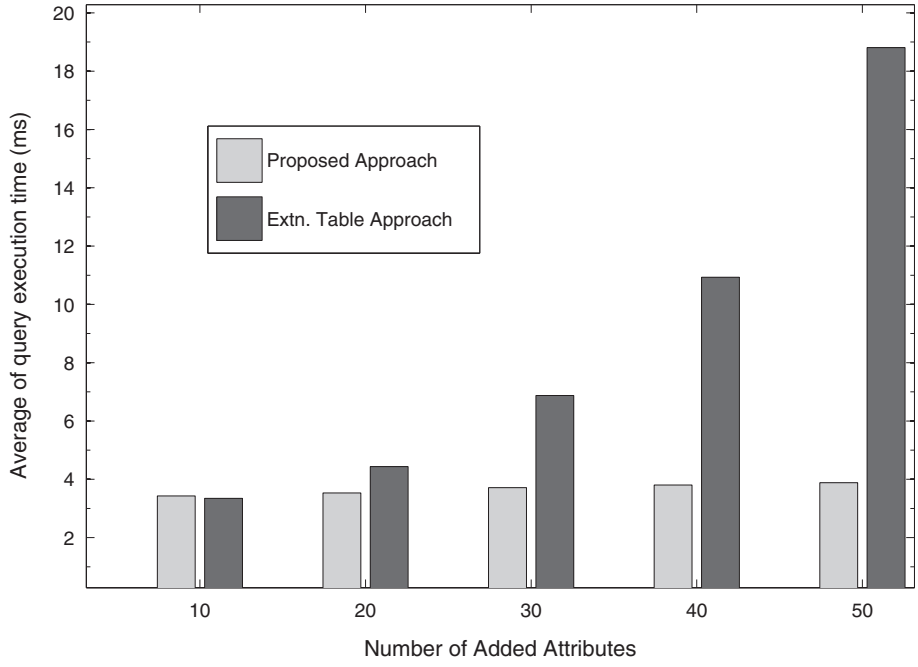

Figure 8 Average performance for added attributes.

Table 12 Avg. performance (time taken) for added attributes

Number of added attributes	Proposed approach (ms)	Extension table approach (ms)
10	3.427	3.345
20	3.532	4.435
30	3.712	6.874
40	3.804	10.932
50	3.884	18.809

Table 13 Comparison for space requirements (bytes) for added attributes

Number of added attributes	Proposed approach (ms)	Extension table approach (ms)
0	2048	15360
2	2048	16384
4	2560	17408
8	3072	19456
16	4096	23552

insertion, deletion and updation query there is a gain in number of query execution per second of 20%, 230% and 236% respectively and there is drop by 27% for selection query, which is slow due to parsing of XML file. The gain is due to less number of joins involved in the schema.

The average response time for query execution for added attributes is almost constant for our approach as shown in Figure 8 and Table 12, but for extension table approach it increases exponentially. Figure 8 shows there is exponential rise in the response time with increase in number of attributes in the extension table approach where as our approach yields a constant response time approximately. The response time in the extension table approach degrades varying from 2.45% to 384.5% in five consecutive increases in step size of 10 attributes. The increase in attributes involves creation of new tables therefore more number of joins are required to satisfy a query. Therefore in the extension table approach the response time increases with added attributes. Since in the proposed approach due to the use of XML attribute the number of tables created would be less therefore resulting in lesser number of joins required for query execution.

The better performance in response time in our approach is due to the use of XML filed, which accommodates and adjusts extra added attributes in the XML filed.

Figure 9 show the increase in space requirements in the extension table approach, with the increase in number of attributes, whereas in our approach the space requirement increases linearly. This is again due to the adjustment of group of attributes into one field in for of XML file.

In Table 13 we consider in the implementation model that a total of 10 common attributes are present in the multi-tenant database along with this 20 tenant specific attributes are there. The size of a field is 512 bytes. Maximum allowed attributes in an XML file is 4. The efficiency gained in space on an average is 86%. It could be greater

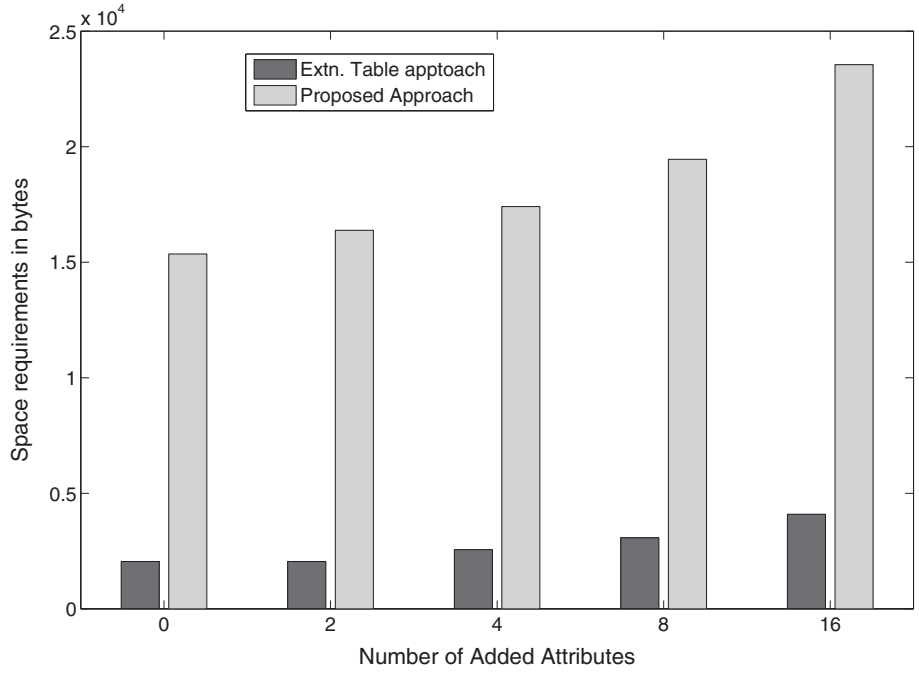

Figure 9 Space requirements with increase in attributes.

Table 14 Comparison for concurrent queries

No. of concurrent queries	Proposed approach (time in ms)	Extension table approach (time in ms)
10	11.44	11.05
20	24.55	23.04
50	66.98	62.98
100	140.86	132.45
200	292.37	262.34
500	604.42	535.67

if more than 4 attributes could be allowed in a field. In our approach we are only limited by the size of XML file that can fit in field as constraint by the native database technology used.

Table 14 lists the response time for concurrent queries as the number of concurrent queries increases the graph in Figure 10 depicts that after the number of concurrent queries crosses the 200 mark the performance of our approach slightly degrades but does not affect the applications response substantially. The response time for concurrent query execution is within comparable range with the extension table approach.

Conclusion & future work

In this paper, an attempt has been made to implement the Multi-tenant database for an ad hoc cloud that offers operational advantages over the existing ones. It fits very well in scenarios where SaaS cloud services are to be delivered between multiple clients (institutions). The proposed multi-tenant database accommodates larger number of tenants because a single database instance is used to store the data of multiple tenants. Another advantage of the proposed work is that the tenants are allowed to create multiple tables which add flexibility in terms of having varied set of attributes as specifically required for its application. It is evident from the result that our approach performs much better in terms of space saving in terms of solving the NULL value problem as compared to other multi-tenant approaches. With increase the number of attribute in the table the query performance drops with the extension table approach as compared to our approach, which is due to more number of attribute and more number of joins required to execute query. The multi-tenant database architecture proposed is highly efficient in terms of query execution, space saving and change in number of attributes. The performance is moderate and comparable with extension table approach for concurrent requests. The results of the proposed work show 20% to 230% improvement for insertion, deletion and updation-queries. The response of the proposed approach is stable as compared to other system which degrades in terms of response time by 384% for increased number of attributes up to 50. The proposed approach is also space efficient by average of 86% for 2 to 16 more added attributes. Further the work decomposition algorithm proposed optimally calculates the node specific granularity, which helps in performance and better resource utilization by optimizing

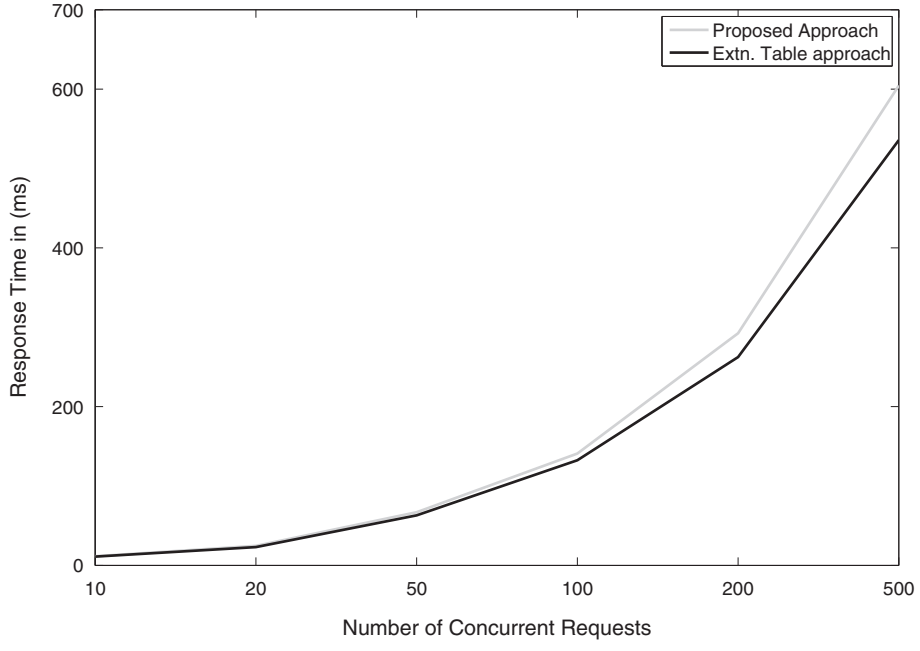

Figure 10 Performance comparison for concurrent.

the resource allocation policy. The data management and scalability approach discussed is simple to implement and proves to be practically efficient due to the concept used for decoupling the database manager and the transaction manager. The replication scheme discussed uses a simple approach to calculate change in database state. It further improves availability and strengthens reliability and uses a simple application level check-pointing to find differential updates. Further work can be done to remove the limitation imposed by database attribute field so as to include a sufficiently large XML file.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
SKP and DSK designed the research (project conception, development of overall research plan and study oversight). SKP conducted research (hands-on conduct of the experiments and data collection). SKP and DSK analyzed data or performed statistical analysis and wrote paper. SKP had primary responsibility for final content. Both authors read and approved the final manuscript.

Acknowledgements
A special thank you goes to those who contributed to this paper: Mr. Manu Vardhan for his valuable comments and sharing his knowledge. Prof. Saurabh Raina for proof reading the paper. The Lab staff, JRE Group of Institution and MNNIT Allahabad for hosting the research.

Received: 14 May 2012 Accepted: 22 January 2013
Published: 4 March 2013

References

1. Mell P, Grance T (2009) The NIST definition of cloud computing, version 15, NIST. Retrieved January 2010. http://www.nist.gov/itl/csd/cloud-102511.cfm
2. Kirby G, Dearle A, Macdonald A, Fernandes A (2010) An approach to ad hoc cloud computing. In: DBLP: CoRR, Volume abs 1002.4738
3. Chandra A, Weissman J (2009) Nebulas: Using distributed voluntary resources to build clouds. In: Proceedings of the 2009 conference on Hot topics in cloud computing. ACM id 1855535, USENIX Association
4. Pippal S, Kushwaha DS (2012) Architectural design of education cloud framework extendible to ad hoc clouds. In: IEEE 2nd International Conference on Recent Advances in Information Technology (RAIT)
5. Chapin PC, Skalka C, SeanWang X (2008) Authorization in trust management: features and foundations. Comput Surv 40(3): 1–48
6. Das S, Agrawal D, El Abbadi A (2009) ElasTraS: An elastic transactional data store in the cloud. In: Proceedings of the conference on Hot topics in cloud computing (HotCloud'09)
7. Litzkow MJ, Livny M, Mutka MW (1988) Condor : A hunter of idle workstations. In: 8th International Conference on Distributed Computing Systems. IEEE Computer Society Press, Washington, pp 104–111
8. Barak A, Guday S, Wheeler RG (1993) The MOSIX Distributed Operating System, Load Balancing for UNIX. In: Lecture Notes in Computer Science, vol 672. Springer-Verlag, Berlin; New York
9. Pai VS, Aron M, Banga G, Svendsen M, Druschel P, Zwaenepoel W, Nahum E (1998) Locality-aware request distribution in cluster-based network servers. SIGOPS Oper Syst Rev 32(5): 205–216
10. Aron M, Sanders D, Druschel P, Zwaenepoel W (2000) Scalable content aware request distribution in cluster-based network servers. In: Proceedings of the USENIX 2000 Annual Technical Conference, San Diego
11. Aron M, Druschel P, Zwaenepoel W (2000) Cluster reserves: a mechanism for resource management in cluster-based network servers. In: Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems
12. Shen K, Yang T, Chu L (2002) Cluster load balancing for fine-grain network services. In: Proceedings of the 16th International Parallel and Distributed Processing Symposium, (IPDPS'02), Fort Lauderdale FL
13. Banawan SA, Zeidat NM (1992) A comparative study of load sharing in heterogeneous multicomputer systems. In: Proceedings of the 25th Annual symposium on Simulation
14. Goswami KK, Devarakonda M, Iyer RK (1993) Prediction based Dynamic Load-Sharing Heuristics. In: IEEE Transactions on Parallel and Distributed Systems
15. Anane R, Anthony RJ (2003) Implementation of a Proactive Load Sharing Scheme. In: Proceedings of the 2003 ACM symposium on Applied computing
16. Berman F, Wolski R, Figueira S, Schopf J, Shao G (1996) Application Level Scheduling on Distributed Heterogeneous Networks. In: Proceedings of the 1996 ACM/IEEE conference on Supercomputing
17. Karatza HD, Hilzer RC (2002) Load Sharing in Heterogeneous Distributed Systems. In: Proceedings of the 2002 Winter Simulation Conference
18. van Nieuwpoort RV, Kielmann T, Bal HE (2001) Efficient load balancing for wide-area divide and conquer applications. In: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming
19. Kondo D, Chien AA, Casanova H (2004) Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids. In: Proceedings of the 2004 ACM/IEEE conference in Supercomputing
20. Beaumont O, Casanova H, Legrand A, Robert Y, Yang Y (2005) Scheduling Divisible Loads on Star and Tree Networks:Results and Open Problems. IEEE Trans Parallel Distributed Syst (TPDS) 16(3): 207–218
21. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2006) Bigtable: a distributed storage system for structured data. In: OSDI. pp 205–218
22. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon's highly available key-value store. In: SOSP. pp 205–220
23. Maier D, Ullman JD (1983) Maximal objects and the semantics of universal relation databases. In: ACM Trans. Database System. p 114
24. Copeland GP, Khoshafian SN (1985) A decomposition storage model. In: The Proc. of 1985 ACM SIGMOD International conference on Management of Data. ACM Press, pp 268–279
25. Grund M, Schapranow M, Kruege J, Schaffner J, Bog A (2008) IEEE Symposium on Advanced Management of Information for Globalized Enterprises. pp 1–5
26. Hui M, Jiang D, Li G, Zhou Y (2009) Supporting Database Applications as a service. In: IEEE 25thInternational Conference on Data Engineering. pp 832–843
27. Jacobs D, Aulbach S (2007) Ruminations on Multi-Tenant Databases, Datenbanksysteme. In: Bro, Technik undWissenschaft (German Database Conference) BTW. pp 514–521
28. Wang ZH, Guo CJ, Gao B, Sun W, Zhang Z, Hao W (2008) Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing. In: IEEE International Conference on e-Business Engineering. pp 94–101
29. Domingo EJ, Nino JT, Lemos AL, Lemos ML, Palacios RC, Berbis JMG (2010) A cloud computing-oriented multi-tenant architecture for business information systems. In: IEEE 3rd International Conference on Cloud Computing. pp 532–533
30. Vardhan M, Verma S, Bhatnagar P, Kushwaha DS (2012) Eager computation and lazy propagation of modifications for reducing synchronization overhead in file replication system. In: IEEE 3rd International Conference on Computer and Communication Technology (ICCCT-2012)
31. Das S, Nishimura S, Agrawal D, El Abbadi A (2011) Albatross: Lightweight Elasticity in Shared Storage Databases for the Cloud using Live Data Migration. In: 37th International Conference on Very Large Data Bases (VLDB)
32. Aulbach S, Grust T, Jacobs D, Kemper A, Rittinger J (2008) Multi-tenant databases for software as a service:schema-mapping techniques. In: The proc. of International Conference on Management of Data - SIGMOD. pp 1195–1206
33. Hacigumus H, Iyer B, Mehrotra S (2002) Providing database as a service. In: 18th International Conference on Data Engineering. pp 29–38

doi:10.1186/2192-113X-2-5
Cite this article as: Pippal and Kushwaha: **A simple, adaptable and efficient heterogeneous multi-tenant database architecture for ad hoc cloud.** *Journal of Cloud Computing: Advances, Systems and Applications* 2013 **2**:5.