Novelty Journals

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 2, Issue 1, pp: (1-16), Month: January - April 2015, Available at: www.noveltyjournals.com

Database Tree Groups

[1]Indu Sekhar A, [2]Dr.G.Narsimha

[1]MCA, M.Tech,, Associate Professor, Dept. of Computer Science and Engineering JNTUH, Kukatpally, Hydrabad, Telangana-500085. India
[2]M.Tech, Ph.D, Associate Professor, Dept. of Computer Science and Engineering JNTUH, Kukatpally, Hydrabad, Telangana-500085. India

Abstract: **Due to overwhelming development of Consumer Centric Cloud applications, the cloud databases also are facing the challenges. The most important among these challenges include on fly scalability. When an application is sassified, data from 'n' entities start residing into the cloud database. The underlying cloud database should be in a position to accommodate the data from 'n' entities. Here comes the need for scalability and security. This paper mainly focuses on addressing these two issues. As it is evident that cloud computing has evolved as one of the most powerful and important paradigm for internet or web based applications. Scalability, dynamic growth, metering and economics of scale from large operators are the major causes for the growth of the cloud infrastructure. Most of the web based or cloud applications are data driven. The underlying database forms a critical component in the cloud deployment schema. In this paper a new architecture is proposed. This architecture is an extension to the architecture proposed in the paper "Mechanism for Mounting Multiple Databases over a Universally Shared Instance". This paper proposes a new Architecture in which database trees are grouped as database tree groups. The remainder of this paper is devoted in developing the proposed architecture for cloud databases.**

Keywords**: Dynamically Resizable Shared Universal Area, (DRSUA), Root Node Data Cache (RNDC), Root Node Common Cache (RNCC), Root Node Compilation Cache (RCC), Information Storage Cache, Root Node Log Cache (RNLC), Miscellaneous Cache (MC), Background Processes (B/P), Root Node Monitor Process (RNMP), Child Database Writer Process (CDBWP), Child Database Log Writer (CDBLGWR), Archiver (ARCH), Checkpoint Process (CKPT), Root Node Instance (RNI), Database (DB), Tree T.**

1. INTRODUCTION

Before entering into the Architecture proposed in this paper, a brief introduction is given on Database Trees. The concept of Database Trees is given in the paper *"Mechanism for Mounting Multiple Databases Over a Universally Shared Instance"*. A Database Tree is a binary tree with at least two child node databases. A database tree is like a normal binary tree and starts with a root node. A root node in database tree is a collection of memory structures called instance not associated with any database. A root node in database tree architecture is classified in two parts. They are

i) Universally sharable and dynamically resizable instance; and

ii) Partition table.

This partition table is again divided into two parts. They are:

Unique Identifier, and

i) Record Area.

The below diagram depicts a root node in database tree architecture.

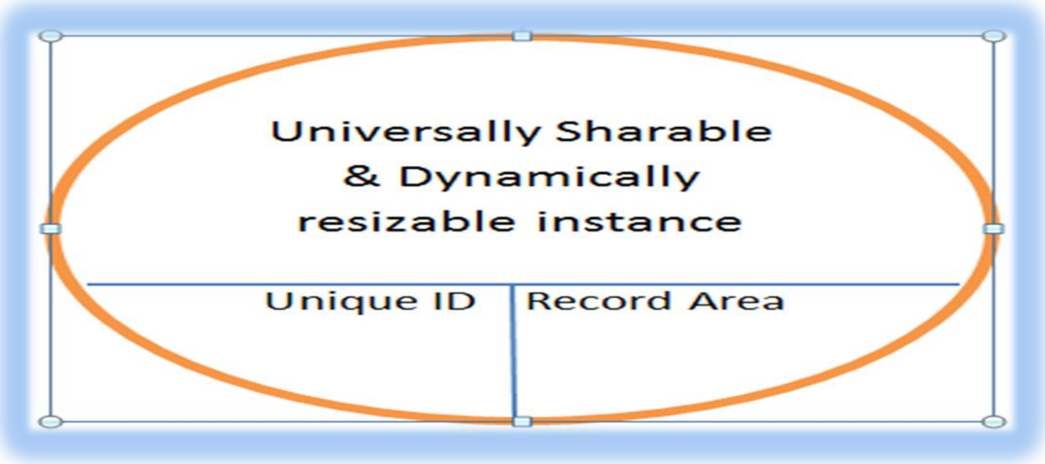


Figure 1: A Root Node in database tree Architecture

Unique Identifier is used to show which database is mounted on the root instance. Record area holds the complete information about all the databases in the tree along with their initialization parameter information. Initialization parameters define the sizes of the instances for each child node databases. These initialization parameters are unique for each database. To know more about a Database Tree and it's logical Architecture please refer the paper *Mechanism for Mounting Multiple Databases Over a Universally Shared Instance"*, published in the IJSRES, December, 2014 edition. The below sections introduces the new Architecture which groups database trees together. It also describes the rules and conditions for grouping the database trees. Below sections explain the logical architecture of database tree.

2. ARCHITECTURAL COMPONENTS

This Architecture can be viewed as databases formed as Trees. The below sections details the Tree and various logical components.

2.1 Tree

A tree in the current Architecture is a binary tree with at least two child node databases. In the current Architecture tree is like a normal binary tree and starts with a root node. A root node in the current Architecture is a collection of memory structures called instance not associated with any database. A root node in the current Architecture is classified in two parts. They are universally sharable and dynamically resizable instance, and Partition table. This partition table is again divided into two parts. They are, Unique Identifier, and Record Area. Unique Identifier is used to show which database is mounted on the root instance. Record area holds the complete information about all the databases in the tree along with their initialization parameter information. Initialization parameters define the sizes of the instances for each child node databases. These initialization parameters are unique for each database.

2.2 Logical Division

This Architecture can be divided logically and physically. The advantage of this division is that the physical storage can be changed without changing the logical structure. Logically the current Architecture can be divided into root node and instance. At any given instance a node can have only two nodes because it is binary.

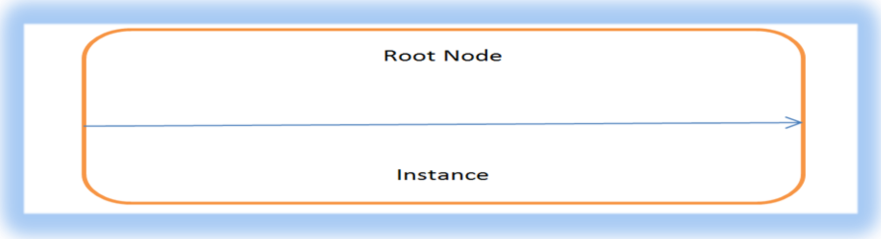


Figure 2 :- Logical Division

2.2.1 Root Node

A root node is logically divided into instance and a partition table.

2.2.2 Instance Architecture

An instance in the current Architecture can be defined as a collection of memory structures. Instance can again be divided into two components.

1. Dynamically resizable shared universal area, also called DRSUA, and
2. Background processes which monitor and control the dynamically resizable shared universal area, also called DRSUA.

The various components in the current Architecture are discussed in the below sections.

2.2.2.1 Dynamically Resizable Shared Global Area (DRSGA)

Dynamically resizable shared global area is again divided into the following components.

a) Root Node Data Cache,

b) Root Node Common Cache,

c) Root Node Log Cache, and

d) Miscellaneous Cache.

2.2.2.2 Root Node Data Cache (RNDC)

A Root Node Data Cache in the current Architecture can again be divided into two parts. The First part stores the database node information and the second part contains the data blocks retrieved from the child database. The same information is also replicated into the Record Area of the partition table.

2.2.2.3 Root Node Common Cache (RNCC)

A Root node common cache can be divided into

i. Root Node Compilation Cache and

ii. Information Storage Cache.

2.2.2.3.i Root Node Compilation Cache (RCC)

The Root node compilation cache consists of the Programming Languages concerned like SQL / PLSQL area. Execution plan is generated in this Area. Execution plan holds actual physical address of the underlying storage like LUNS (Logical Unit Numbers). This can be used to retrieve the actual data blocks of a particular database into root node data cache.

2.2.2.3.ii Information Storage Cache

1. Information Storage Cache contains the following information
2. Meta data of all the nodes in the tree.
3. It holds entire information of each node. Like, where the physical files of each database are located, their initialization parameters details, and corresponding instance details.
4. Meta data of the tree. Meta data of the tree refers to the information related to the physical address details of the root node instance, its starting address.

2.2.2.4 Root Node Log Cache (RNLC)

The Root Node Log Cache stores redo entries and undo entries. This is a log of changes made to the root node database. The redo and undo entries stored in the root node caches are written to an online log file of the child node database.

2.2.2.5 Miscellaneous Cache (MC)

This pool is dynamically used to configure the developmental languages related information or for parallel or for stream processing.

2.2.3 Background Processes (B/P)

The relationships between the child nodes physical and memory structures existing inside the root nodes are maintained and enforced by the Background Processes. These are root nodes own background processes that may vary in the number depending on the child nodes in the tree.

The trace files act as log files to the background processes. This means, each background process will have an associated trace file created in the respective LUNs of the Database.

The following subsections describe the backup processes.

2.2.3.1 Root Node Monitor Process (RNMP)

Root Node Monitor Process is a background process that performs root node instance recovery at the start of any node database. In a tree environment RNMP can perform instance recovery of any instances that have failed. RNMP also cleans up any instance stains that are no longer in use and recovers dead transactions skipped during crash and instance recovery because of file-read or offline errors. This background process also cleans up failed user process. RNPM is responsible for releasing the locks i.e. cleaning up the cache and freeing resources that the process was using. Its effect can be seen when a process holding a lock is killed.

2.2.3.2 Child Database Writer Process (CDBWP)

The Child database writer process background process is responsible for managing the contents of the data block buffer cache and dictionary cache. CDBWP performs batch writes of changed block. This architecture uses write-ahead logging. CDBWP does not need to write blocks when a transaction is committed. In the most common case, CDBWP writes only when more data needs to be read into the system global area and too few database buffers are free. The least recently used data is written to the data files first. Although there is only one System Monitor and one CDBPMON process running per database instance, once can have multiple CDBDBWR processes running at the same time.

2.2.3.2 Child Database Log Writer (CDBLGWR)

The Child database log writer (CDBLGWR) background process manages the writing of the contents of the redo log buffer to the online redo log files. CDBLGWR writes the log entries in the batches form. The Redo log buffers entries always contain the most up-to-date status of the database. Note CDBLGWR is the only one process that writes to the online redo log files and the only one that directly reads the redo log buffer during to the normal database operation.

For every client process there exists a Process Global Area generated corresponding to it. In an the current Architecture architecture unlike in a normal RDBMS and ORDBMS it also contains a unique identifier which identifies a node database in the tree. Basing on the unique identifier, the corresponding database will be mounted on the instance. In the current Architecture a universal and sharable dynamic instance is present in the root node of the tree. The instance parameters are initialized using a parameter file which is dynamically configurable just like in any other Relational Database Management Systems. After the database ID is recognized, a search is made in the partition table of the root node. This partition table provides the exact location of the child database node. Once the child database node is recognized, the corresponding database is mounted on the instance in the root node.

3. CONDITIONS FOR TREE DATABASE FORMATION

In this architecture each entity can store its data in its own database. The following figure depicts the formation of the tree in the current Architecture.

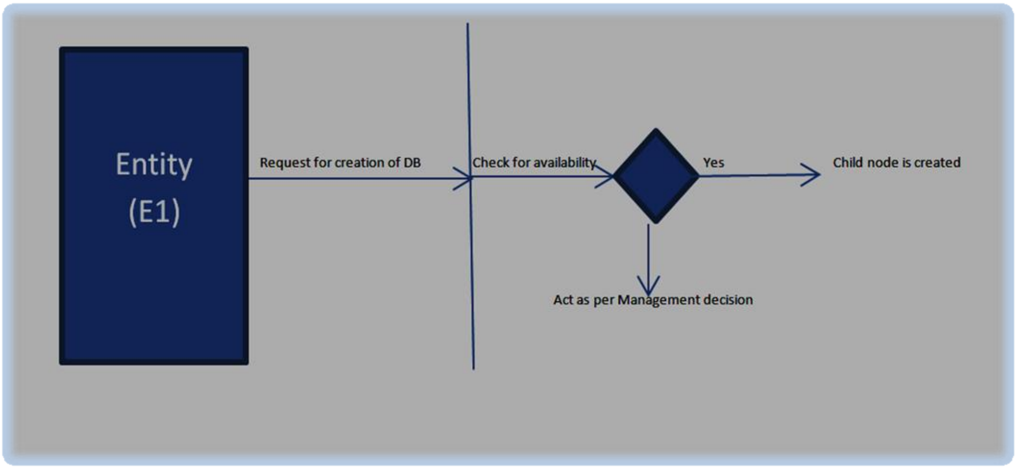


Fig 3.1 Condition for Tree Formation in the current Architecture

Before allotting a child node, a check is made in the Partition Table also called, the Tree Dictionary of the root node instance. It checks whether the requested instance size can be allotted to the requesting entity or not. Allotment can only be made if the following condition is met.

"*If requesting instance size is* < *root instance size*".

Then allotment can be made.

"*Else if, requesting instance size is* > *root instance size*".

Then Allotment cannot be made.

4. DATABASE TREE GROUPS

A database tree groups in the current architecture may be defined as a group of trees which have their own instances. A database tree groups, with one tree is considered as a tree. In a database tree groups, there may exist more than one tree. The following figure depicts a database tree groups architecture.

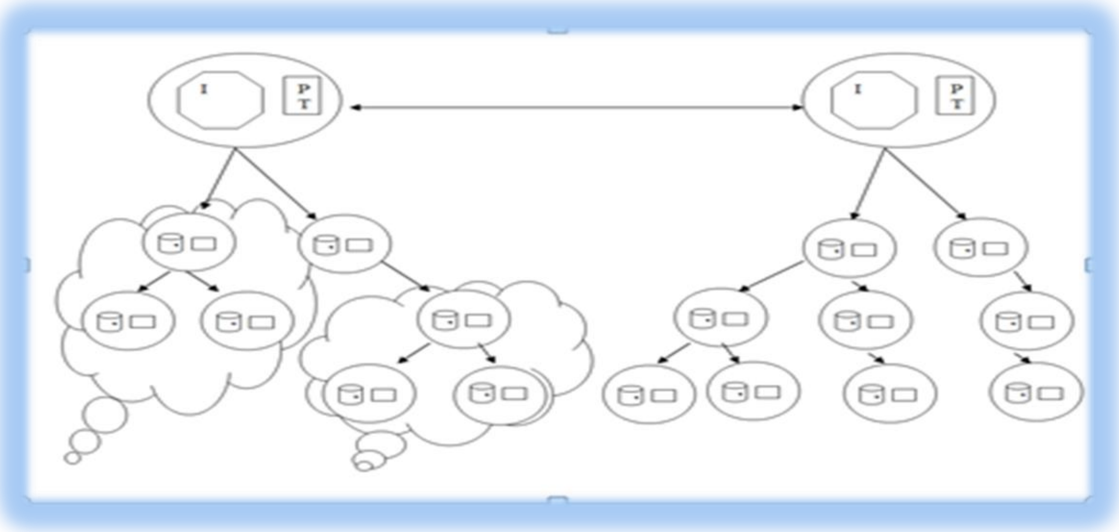


Figure 4.1:- Database Trees Groups

The above figure depicts a forest with more than one tree inside. In the above example, two trees exist inside the forest.

The following rules apply for the forests in current.

i) Inter nodal relations can span across trees in the forest. This means, database from one tree can be mounted on another tree root node instance.

ii) An current architecture which has one or more than one tree is considered as a forest.

iii) Trees and nodes in an current architecture are independent of the underlying operating system.

iv) If there exists ‘n' nodes in a tree then a forest with 2^n-1 trees can be formed.

v) All the trees in the forest should follow the same naming convention.

4.1 Root Node Architecture in a Forest

A root node in current architecture is classified into three parts. They are :

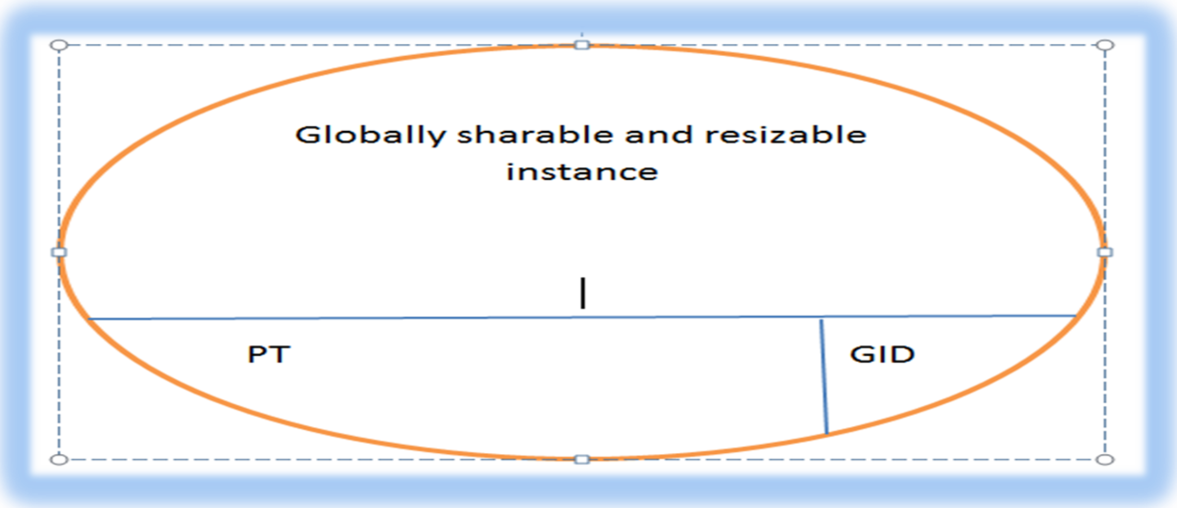
i) Universally sharable and dynamically resizable instance, where Max (Child Node instance size) < Root node instance size,

ii) Local Partition table, and

iii) Global instance dictionary.

A new component is added to the root instance of a tree in a forest called “Global Instance Dictionary”. This Global instance dictionary holds the complete information of all the trees in the forest along with the respective child node details.

The following figure depicts a root instance of the tree in a forest.



PT: Partition Table

GID: Global Instance Directory

Figure 4.2: Figure depicting the root instance architecture in a Forest

When a forest is formed, with more than one tree the following checks are introduced in the trees.

i) A new component called “Global instance dictionary” is created to hold the complete information of all the trees in the forest.

ii) A Child node database from one tree can be mounted on another tree root instance.

4.2 Formation Process

Generally, forests are formed when the below two conditions are met.

i) When the size of the child node’s instance is greater than the root node’s instance size.

ii) When a request comes from the customer to create a new tree, within the same domain.

The following figure depicts the forest formation process.

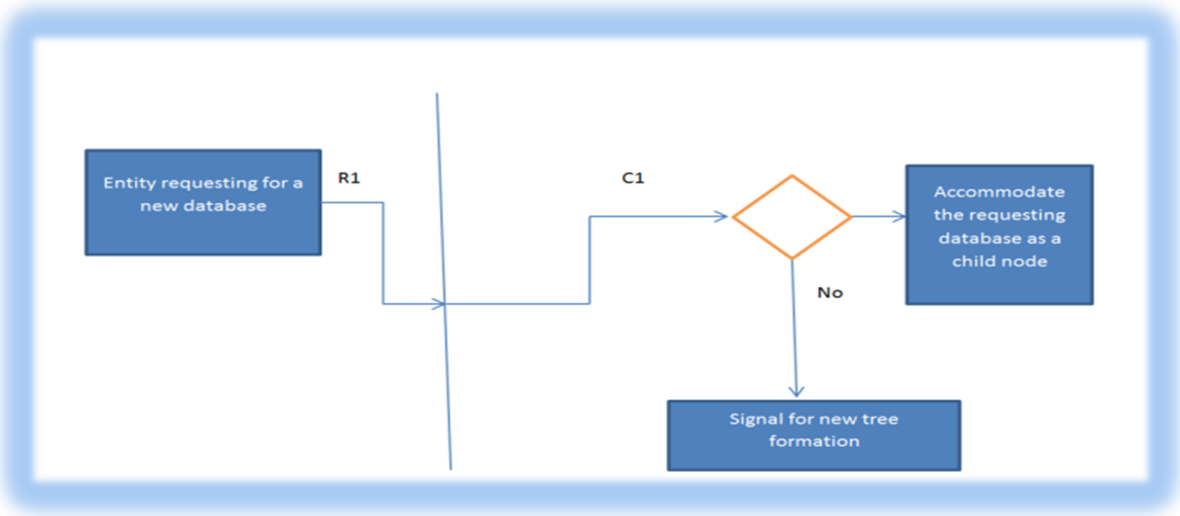

Entity requesting for a new database
R1
C1
Accommodate the requesting database as a child node
No
Signal for new tree formation

Figure 4.3: Figure depicting formation of Forest

R1: Request to create a new DB from the entity E1.

C1: A check is made between the total size of the requesting database instance with the root node instance.

Condition - 1

If the size of the Requesting instance < size of the root instance then accommodate the requesting DB as one of the child node, Or, generate a signal for the creation of a forest.

This signal sends a mail to the management group concerned, with a Y or N parameter values.

The management thus after checking the feasibility conditions sends Y or N values. Depending on the Y or N a forest is formed.

Condition – 2

When a request comes from the customer to create a new tree, within the same domain.

The following figure depicts tree situation when a new forest request is made.

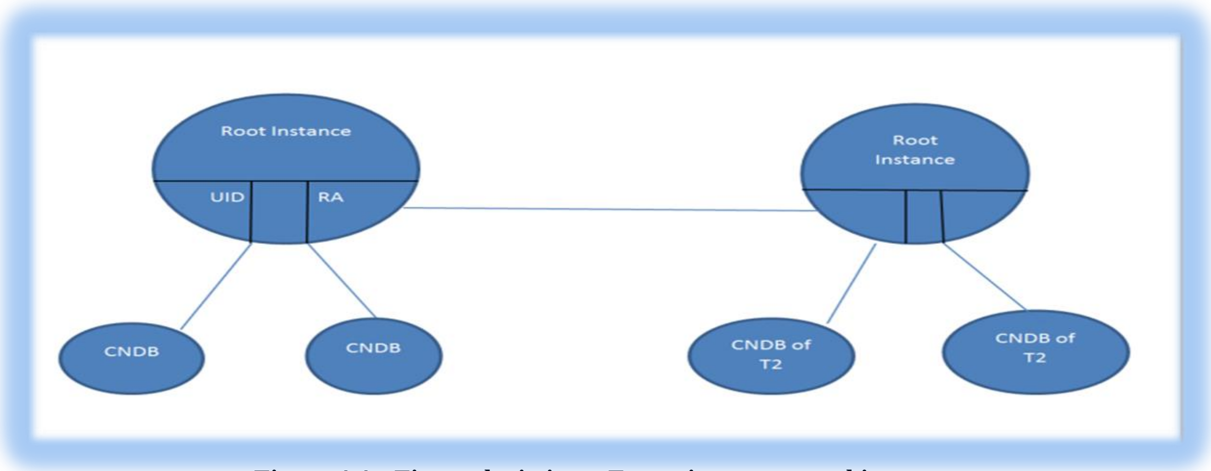


Figure 4.4:- Figure depicting a Forest in current architecture

UID: Unique Identifier

RA: Record Area

Thus a forest can be used, in the following situations:

When a database has to be mounted and if already a database is present in the root instance, then the corresponding database can be mounted on any of the available root instances in the forest.

Database Tree Group Formation

Let us consider the following scenario, where 6 requests have come to the Cloud provider.

Following are the details of the database instance sizes

i. DB1 = 1 GB

ii. DB2 = 0.5 GB

iii. DB3 = 2 GB

iv. DB4 = 4 GB

v. DB5 = 6 GB

vi. DB6 = 18 GB

vii. DB7 = 12 GB

Details of the Root Node details

i. First Root Node = 10 GB

ii. Second Root Node = 20GB

Database Group Trees are formed in the below fashion:

Step-1 The size of the first Database instance creation request is taken into account. The size of the first DB's instance is compared with the first root node instance (RNI). Obviously, first RNI can accommodate the first database creation request (DBCR). At the first stage the below tree is formed

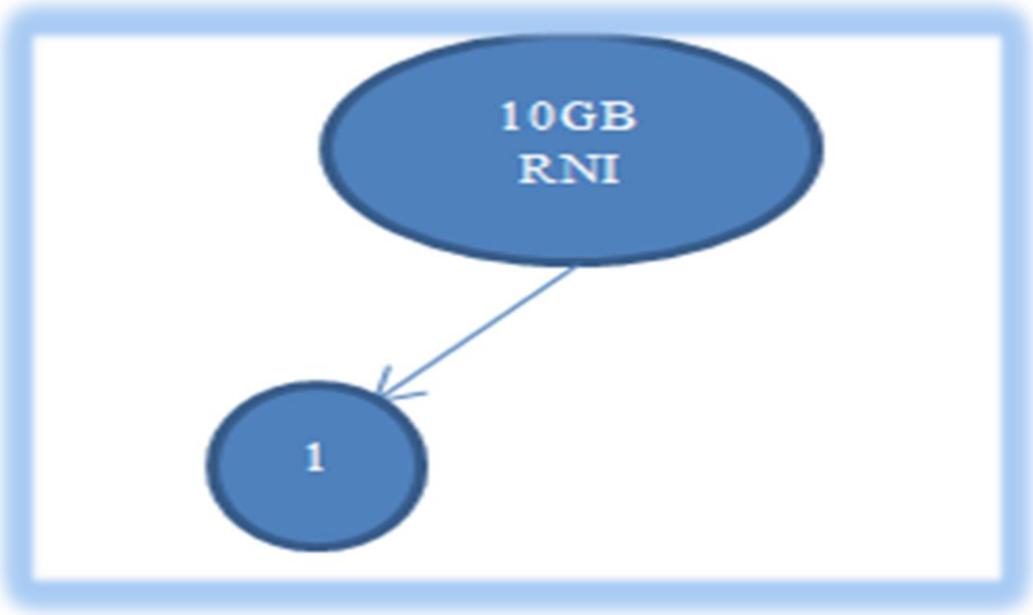


Figure 4.5: Stage 1 in Database Tree Group Formation

Step-2 In the second stage, the second DBI is compared with RNI. Since first RNI > Second DBI, the second DB is accommodated in the existing tree, T1. Then the T1 is extended in the below fashion.

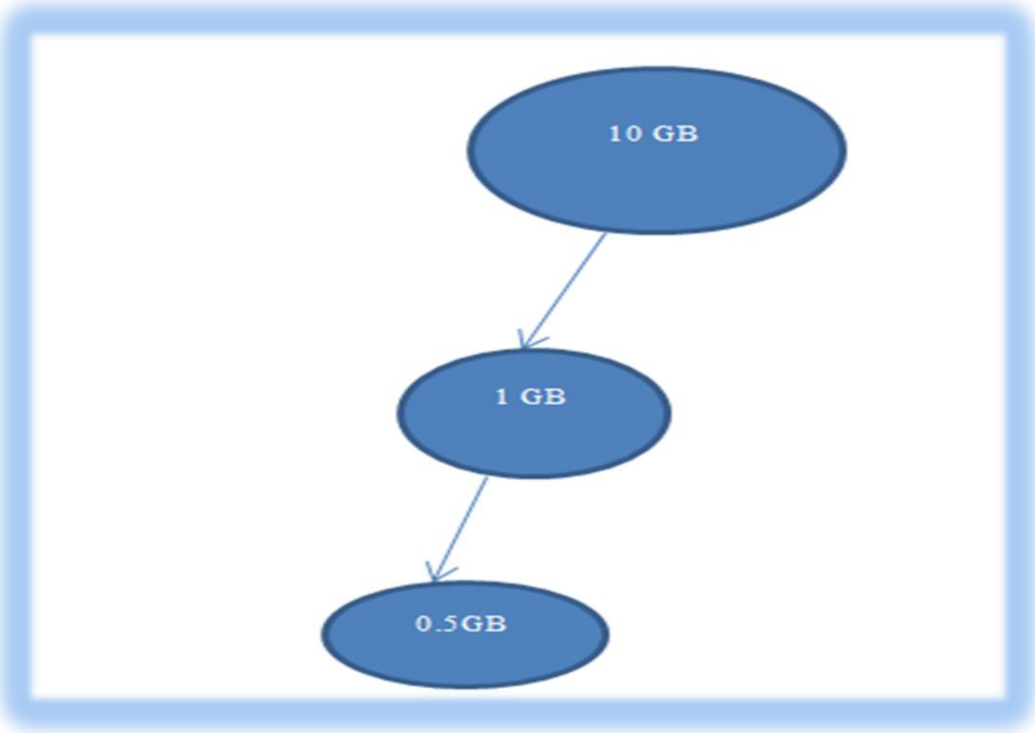


Figure 4.6: Stage 2 in Database Tree Group Formation

Step-3 In the Third stage, the third DBI is compared with the existing RNI. Since 2 GB < 10 GB, the third DBI can be accommodated in the existing T1.

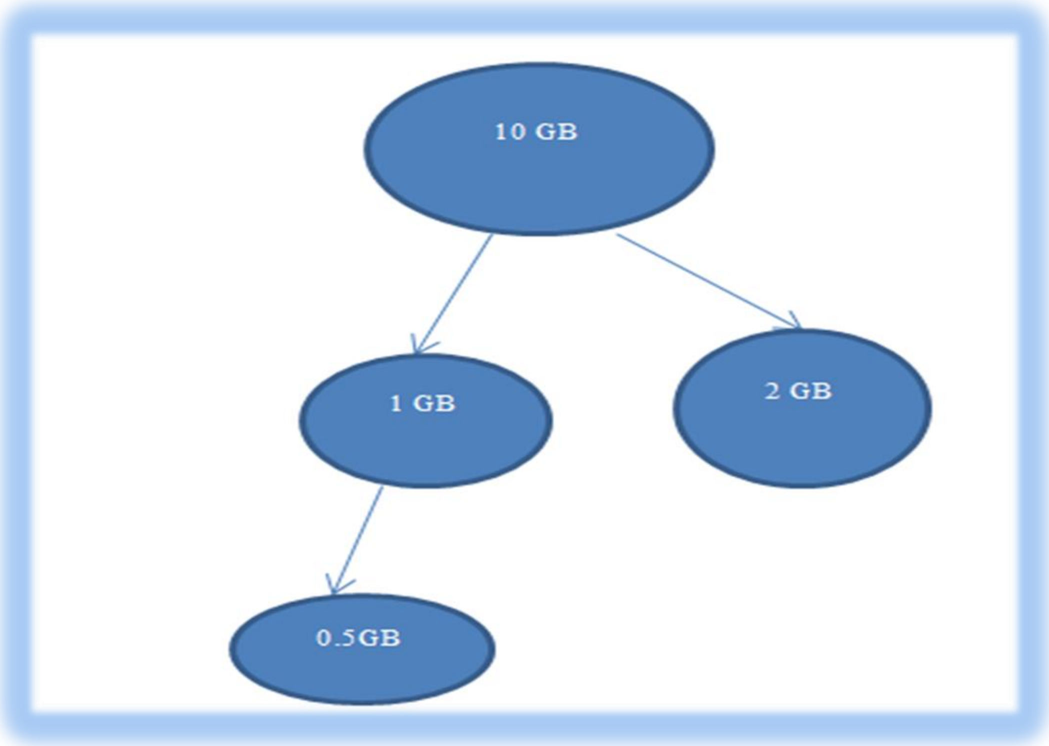


Figure 4.7: Stage 3 in Database Tree Group Formation

Step–4 In the fourth stage, the fourth DBI is compared with the existing RNI. Since 4 GB < 10 GB, the fourth DBI can be accommodated in the existing T1.

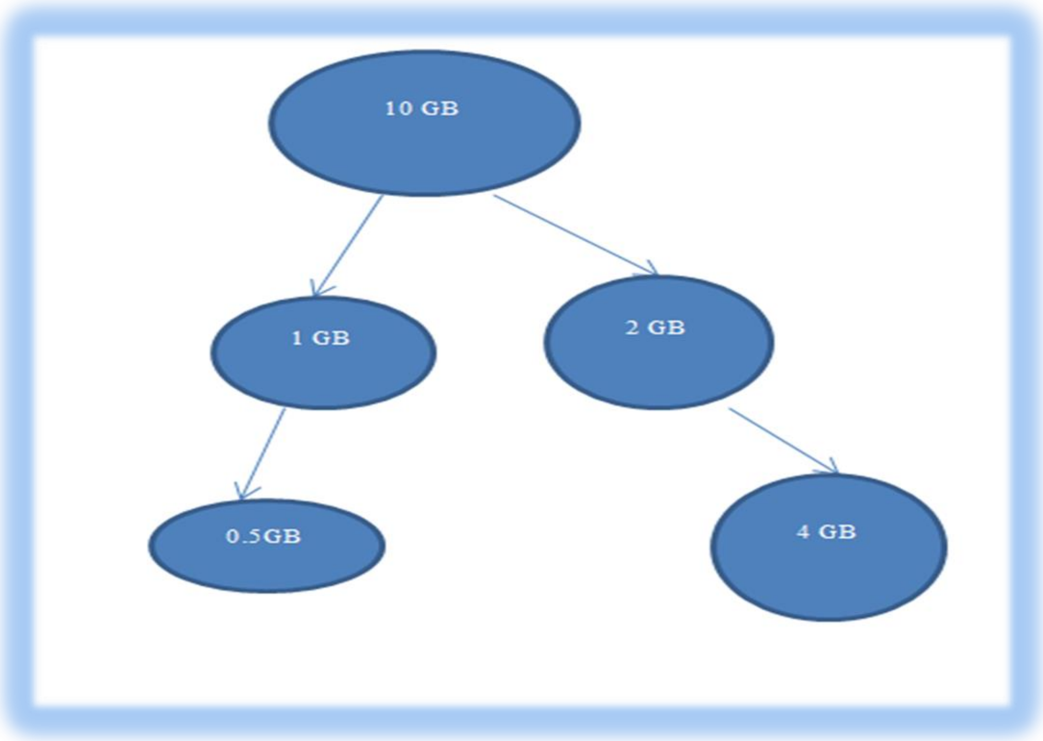


Figure 4.8: Stage 4 in Database Tree Group Formation

Step–5 In the fifth stage, the fifth DBI is compared with the existing RNI. Since 6 GB < 10 GB, the fifth DBI can be accommodated in the existing T1.

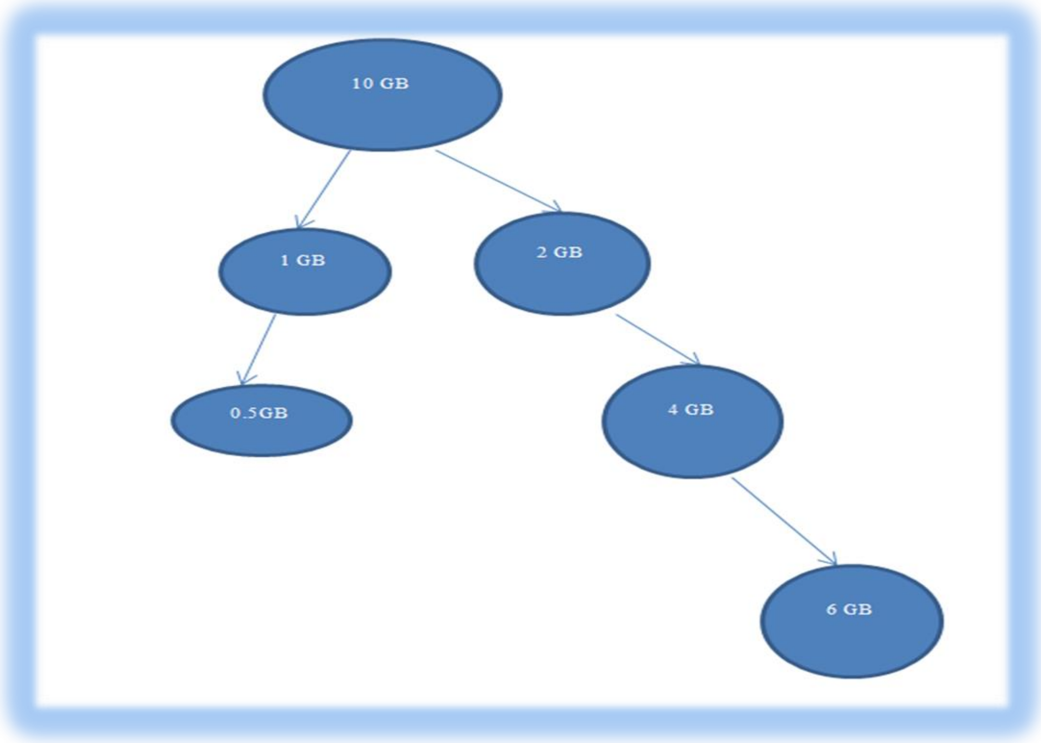


Figure 4.9: Stage 5 in the Database Tree Group Formation

Step–6 In the sixth stage, the sixth DBI is compared with the existing RNI. Since 18 GB > 10 GB, the sixth DBI cannot be accommodated in the existing T1. Then an interrupt is generated for the formation of the DTG. DTG is created in the below manner:

i. First the PT of the first RNI is updated with the address of the second RNI. This address can be the Internet Protocol number.

ii. Next the T2 is created in the same fashion as any Tn in the DBT mechanism.

Thus, the DBTG is created in the below manner.

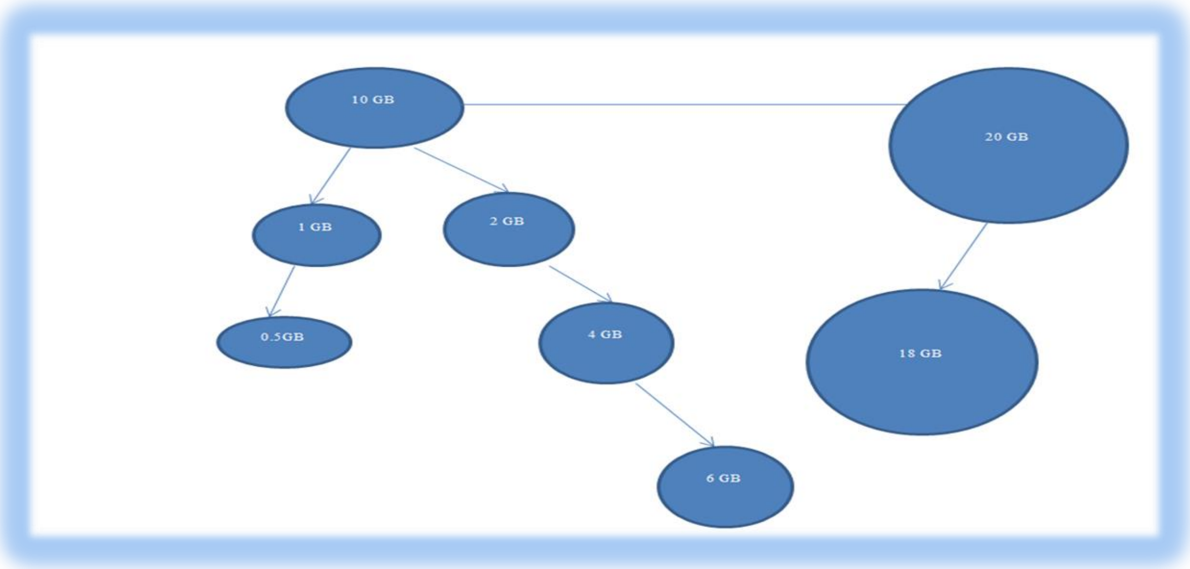


Figure 9: Stage 6 in Database Tree Group Formation

Step-7 In the seventh stage, the seventh DBI is compared with the existing RNI. Since 15 GB > 10 GB, the seventh DBI cannot be accommodated in the existing T1. Immediately, the second RNI instance address is searched in the first RNI's PT. After obtaining the second RNI address, the size of the requesting DBI is compared with the second RNI. Since seventh DBI < second RNI, a new node is attached to the T2 in the below fashion.

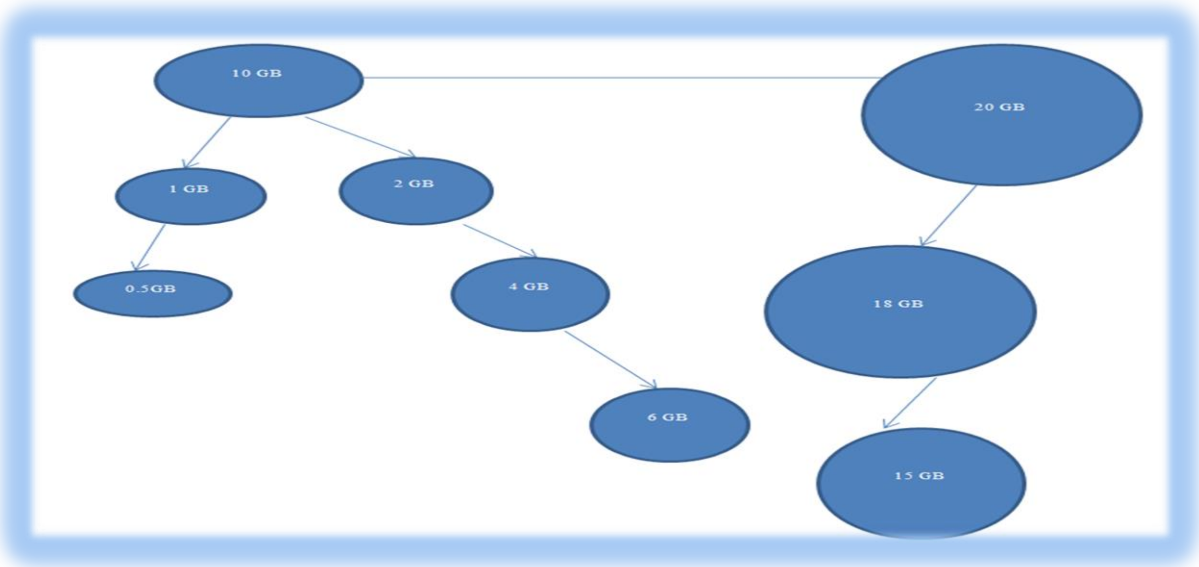


Figure 10: Stage 7 in Database Tree Group Formation

Testing the Implementation

In order to test the current architecture the following steps are followed

1. Since Oracle has similar memory structures, it is taken as reference database.
2. 2 Separate RAC 200 series machines were taken, with 10GB and 20 GB RAMS respectively.
3. RHEL 6, with ext4 files system is installed.
4. First machine with 10 GB RAM is considered as first RNI and second machine with 20 GB RAM is considered as second RNI.
5. 7 DBs with the following instance sizes are considered.

Testing is done in the below fashion.

1. First DB results are considered.
2. The main change occurs with the DB6, since T2 starts from DB6.
3. DB7 results are collected and analysed**.**

Database Name	Instance Size in GBs
DB1	1
DB2	0.5
DB3	2
DB4	4
DB5	6

Testing Results for DB1

Work Load Table

DBName	DBID	Instance	Inst Number	Release	RAC
DB1	**7659**	**DB1**	**897654**	**10.2**	**10.2**

Shared Pool Statistics of DB1

Memory Usage at the Begin	Memory Usage at the End	%SQL with execution > 1 at the beginning	%SQL with execution > 1 at the end	%Memory for SQL w/exec >1 at the begining	%Memory for SQL w/exec >1 at the end
74.46	**75.21**	**78.96**	**82.21**	**77.82**	**80.24**

Table1: Snap Table

	Snap Id	Snap Time	Sessions	Cursors / Sessions
Begin Snap	7246	20-Nov-14	46	8.4
End Snap	7247	11/20/2014 4:00	45	7.2
Elapsed		62 (mins)		
DB Time		59.24 (mins)		

Average Wait Statistics

Event	Wait	Time(s)	Avg Wait (ms)	%Total Call Time	Wait Class
DB file sequential Read	1,110,787	4.627	6	144.8	User I/O
CPU Time	-	1,217		39.5	
Buffer exterminate	442	424	986	14.6	Other
Log file parallel write	2,541	192	76	4.1	System I/O
Control file parallel write	1,312	34	22	1.2	System I/O

Load Profile of DB1

	Per Second	Per Transaction
Redo size	**680242.36**	**4,600,168.00**
Logical reads	**134,812.33**	**1,147,447.92**
Block change	**8934.46**	**67,509.66**
Physical reads	**409.24**	**2,322.91**
Physical writes	**241.32**	**3,218.32**
User calls	**3.20**	**19.32**
parses	**2.82**	**30.00**
Sorts	**0.76**	**4.92**
Logons	**0.04**	**26.92**
Executes	**11.46**	**0.16**
Transaction	**0.13**	**88.46**

Instance Effeciency Percentage (Target 100%) of DB1

Buffer Nowait%	100.00
Buffer Hit%	**89.79**
Library Hit%	**86.47**
Execute to parse %	**65.48**
Parse CPU to Parse elapsed%	**33.42**
Redo Nowait%	**100.00**
In-memory sort%	**100.00**
Sort parse %	**73.46**
Latch hit %	**100.00**
%Non-Parse CPU	**98.99**

Testing Results of DB6

Shared Pool Statistics of DB6

DBName	DBID	Instance	Inst Number	Release	RAC
DB2	**7664**	**DB2**	**897655**	**10.2**	**10.2**
Memory Usage at the Begin	Memory Usage at the End	%SQL with execution > 1 at the beginning	%SQL with execution > 1 at the end	%Memory for SQL w/exec >1 at the begining	%Memory for SQL w/exec >1 at the end
74.64	**75.29**	**78.99**	**84.21**	**77.86**	**80.28**

Snap Table for DB6

	Snap Id	Snap Time	Sessions	Cursors / Sessions
Begin Snap	7246	20-Nov-14	46	8.9
End Snap	7247	11/20/2014 6:00	45	7.8
Elapsed		62 (mins)		
DB Time		59.24 (mins)		

Load Profile of DB6

	Per Second	Per Transaction
Redo size	**780272.36**	**4,600,168.00**
Logical reads	**334,814.53**	**1,147,447.92**
Block change	**9936.36**	**47,509.66**
Physical reads	**609.44**	**2,422.91**
Physical writes	**341.22**	**6,218.32**
User calls	**4.20**	**47.32**
parses	**2.82**	**40.00**
Sorts	**0.76**	**5.92**
Logons	**0.06**	**46.92**
Executes	**11.48**	**0.66**
Transaction	**0.16**	**89.46**

Instance Effeciency Percentage (Target 100%) of DB6

Buffer Nowait%	100.00
Buffer Hit%	**89.79**
Library Hit%	**86.47**
Execute to parse %	**65.48**
Parse CPU to Parse elapsed%	**33.42**
Redo Nowait%	**100.00**
In-memory sort%	**100.00**
Sort parse %	**73.46**
Latch hit %	**100.00**
%Non-Parse CPU	**98.99**

Testing Results of DB7

DB7 Instance details

DBName	DBID	Instance	Inst Number	Release	RAC
DB3	**7665**	**DB7**	**897656**	**10.2**	**10.2**

Shared Pool Statistics of DB7

Memory Usage at the Begin	Memory Usage at the End	%SQL with execution > 1 at the beginning	%SQL with execution > 1 at the end	%Memory for SQL w/exec >1 at the begining	%Memory for SQL w/exec >1 at the end
94.43	**83.47**	**88.97**	**84.41**	**86.62**	**90.24**

Snap Table of DB7

	Snap Id	Snap Time	Sessions	Cursors / Sessions
Begin Snap	7246	20-Nov-14	46	8.9
End Snap	7247	11/20/2014 7:00	45	7.8
Elapsed		92 (mins)		
DB Time		99.54 (mins)		

Load Profile of DB7

	Per Second	Per Transaction
Redo size	**980282.38**	**6,696,168.00**
Logical reads	**754,614.43**	**5,347,456.62**
Block change	**9936.36**	**77,609.88**
Physical reads	**609.44**	**5,567.96**
Physical writes	**341.22**	**8,97.62**
User calls	**4.20**	**68.67**
parses	**2.82**	**67.00**
Sorts	**0.76**	**8.92**
Logons	**0.06**	**36.92**
Executes	**11.48**	**099**
Transaction	**0.16**	**99.49**

Instance Effeciency Percentage (Target 100%) of DB7

Buffer Nowait%	100.00
Buffer Hit%	**99.86**
Library Hit%	**86.48**
Execute to parse %	**45.48**
Parse CPU to Parse elapsed%	**23.42**
Redo Nowait%	**100.00**
In-memory sort%	**100.00**
Sort parse %	**43.46**
Latch hit %	**100.00**
%Non-Parse CPU	**58.99**

5. FINAL CONCLUSION

From the results below conclusions are drawn:

1. T1 creation has performance hiccups.
2. As the DB is mounted and unmounted, the time to mount the next DBs gradually increased (but no fixed rate of growth is observed).
3. The memory also increased.
4. Time to mount DB6 increased by 44%. This shows that there will be some latency at the time of new Tree creations.
5. Though there are performance hiccups, security can be guaranteed because actual data files reside in the customers domain.
6. This architecture required network lines with high bandwidth between the cloud provider and the customer domains**.**

ACKNOWLEDGEMENTS

I would like to thank Dr. G.Narasimha, for his excellent support and guidance in completing this paper successfully.

REFERENCES

[1] Indu Sekhar A, G. Narasimha. Mechanism for Mounting Multiple Databases Over Universally Sharable Instance, International Journal of Scientific Research and Engineering Studies (IJSRES) Volume 1 Issue 6, December 2014, ISSN: 2349-8862
Buyya Rajkumar, Broberg James & Goscinski. *Cloud Computing Principles and Paradigms*, John Wiley & Sons, Inc., Hoboken, New Jersey USA, 2011, ISBN: 978-0-470-88799-8

[2] Judith Hurwitz, Robin Bloor, Marcia Kaufman, and Dr. Fern Halper. *Cloud Computing for Dummies*, Wiley Publishing, Inc. USA, 2010, ISBN: 978-0-470-48470-8

[3] John W. Rittinghouse, James F. Ransome, *Cloud Computing Implementation*, Management, and Security, Taylor and Francis Group, LLC, 2010, ISBN: 978-1-4398-0680-7

[4] McAfee, Database Security in Virtualization and Cloud Computing Environments, 2011, White Paper

[5] Elena Ferrari, Database as a Service: Challenges and Solutions for Privacy and Security, 2009 IEEE APSCC

[6] Oracle, Database as a Service Reference Archtiecture – An Overview, October 2011, White Paper

[7] NIST, The NIST definition of cloud computing, September 2011, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf