IJCTA, 9(21), 2016, pp. 111-121
© International Science Press

Proposed BRKSS Architecture for Performance Enhancement of Data Warehouse Employing Shared Nothing Clustering

Bikramjit Pal* Rajdeep Chowdhury Kumar Gaurav Verma*** Saswata Dasgupta**** Subham Dutta*******

Abstract : Parallel database systems have three main architectures where either scalability or load sharing is maintained. Study elucidates that no single architecture itself can provide both the facilities of good scalability and load sharing. In contrast to the above studies, a hybrid architecture which is mainly based upon shared nothing clustering that maintains both scalability and also tries to achieve load balancing is presented and implemented. Here the concept of parallel databases has been implemented as they are the key to high performance. So the objective is to develop a hybrid system which is cost effective and also performs well while dealing with large amount of data. Performance enhancement has always been a main issue in case of large data warehouses, so in the paper it has been ensured to resolve it with the notion of parallel processing by proposing two algorithms. One is node based algorithm which is mainly based upon Multilevel Feedback Queue Scheduling (MLFQ) and another is cluster based load balancing algorithm. Both the algorithms are based upon push migration

Keywords : Parallel Data, Scalability, MLFQ, Push Migration, Load Sharing, Shared Nothing Algorithm, Cluster, Node Based Algorithm.

1. INTRODUCTION

In this era of internet and cloud computing, increased amount of data has been enforced to make significant changes in the commercial database field. Parallel database processing has become a new trend which is being adopted widely due to its technology-driven and application-driven features [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Clouds and super servers nowadays use parallel database processing with scalable high speed interconnection network, so that it can perform large amount of inexpensive processing. Typically these systems utilize the capacity of multiple numbers of locally distributed processing nodes inter-connected by means of high speed network. The proposed architecture consists of a single switch and multiple hubs (based upon the number of ports in the switch) along with multiple shared nothing clusters and a dedicated output device that acts as an interface for end users to achieve high cost effectiveness compared to the mainframe-based configurations.

* Research Scholar, Department of Engineering and Technological Studies, University of Kalyani, Kalyani, Nadia – 741235, West Bengal, India,

** Research Scholar, Department of Engineering and Technological Studies, University of Kalyani, Kalyani, Nadia – 741235, West Bengal, India Corresponding Author Email – dujon18@yahoo.co.in

*** Student, Department of Computer Science and Engineering, JIS College of Engineering, Kalyani, Nadia – 741235, West Bengal, India, Student,

**** Department of Computer Science and Engineering, JIS College of Engineering, Kalyani, Nadia – 741235, West Bengal, India, Student,

***** Department of Computer Application, JIS College of Engineering, Kalyani, Nadia – 741235, West Bengal, India

Parallel database systems provide both high throughputs for Online Transaction Processing (OLTP) as well as short response times for complex ad-hoc queries [18, 19, 20]. Primary aim is to implement the same in the hybrid model [24]. So, it has been tried to compose multiple cluster computer system comprising of independent computer nodes connected by a network.

The interconnection is done with some commercial product having higher bandwidth and lower latency. The main objective behind selecting cluster system is that they have low price-performance ratio compared to the massive parallel processing computers.

2. LITERATURE SURVEY

Following papers have been studied as part of Literature Review for ease in reference :

Sunguk Lee, "Shared-Nothing vs. Shared-Disk Cloud Database Architecture," Published at Research Institute of Industrial Science and Technology, Pohang, Korea [1]

Cloud protocol, architecture, implementation and services specifications are still in its immature stage. The characteristic, advantages and disadvantages of both shared disk and shared nothing database architectures for cloud computing has been discussed here.

Janina Popeanga, "Shared-Nothing' Cloud Data Warehouse Architecture," Published at Database Systems Journal, Volume–V, Number–4, 2014 [2]

Three widely used parallel data warehouse architectures have been defined and their clarification has been given regarding the most suitable architecture to develop a data warehouse in the cloud. In the end the tables are transposed into "shared-nothing" architecture, for analyzing the query performance.

Thomas Müseler, "A Survey of Shared-Nothing Parallel Database Management Systems" [Comparison between Teradata, Greenplum and Netezza Implementations], Published at IRCSE 2012, Mälardalen University, Sweden [3]

The main contribution of the paper is the presentation of the current technology in the shared-nothing database sector. The concepts of the manufacturers Teradata, Green plum and Netezza has been discussed for data warehouse requirements. Furthermore it has been discussed that whether shared-nothing architecture can be adapted to other application fields for future implementation or not.

David J. DeWitt, Jim Gray, "Parallel Database Systems: The Future of Database Processing or a Passing Fad?" [4]

Parallel database machine architectures have evolved from the use of exotic hardware to a software parallel dataflow architecture based on conventional shared-nothing hardware. These new designs provide impressive speedup and scale-up while processing relational database queries. This paper reviews the techniques used by such systems and surveys current commercial and research systems.

Umar Farooq Minhas, David Lomet, Chandramohan A. Thekkath. "Chimera: Data Sharing Flexibility, Shared Nothing Simplicity," Published at IDEAS, Springer Verlag, September, 2011 [6]

The current database market is fairly evenly split between shared nothing and data sharing systems. While shared nothing systems are easier to build and scale, data sharing systems have advantages in load balancing. The approach of this paper isolates the data sharing functionality from the rest of the system and relies on well-studied, robust techniques to provide the data sharing extension. This reduces the difficulty in providing data sharing functionality, yet provides much of the flexibility of a data sharing system. The design and implementation of Chimera – a hybrid database system, targeted at load balancing for many workloads and scale-out for read-mostly workloads is presented here.

3. PROPOSED WORK

A substantial amount of work has been done to enhance the performance of data warehouses in many different ways. In this paper, an architecture named as 'BRKSS Architecture' is proposed, which is based upon shared nothing clustering that can scale-up to a large number of computers, increase their speed and maintain the work load. The proposed architecture comprises of a console along with a CPU that also acts as a buffer and stores information based on the processing of transactions, when a batch enters into the system. This console is connected to a switch (p-ports) which is again connected to the c-number of clusters through their respective hubs. The architecture can be used for personal databases and for online databases like cloud through router. As shown in Figure–1, the BRKSS Architecture comprises of multiple nodes connected by a high speed LAN. Apiece node has its own Processor (P), Memory (M) and Disk (D).

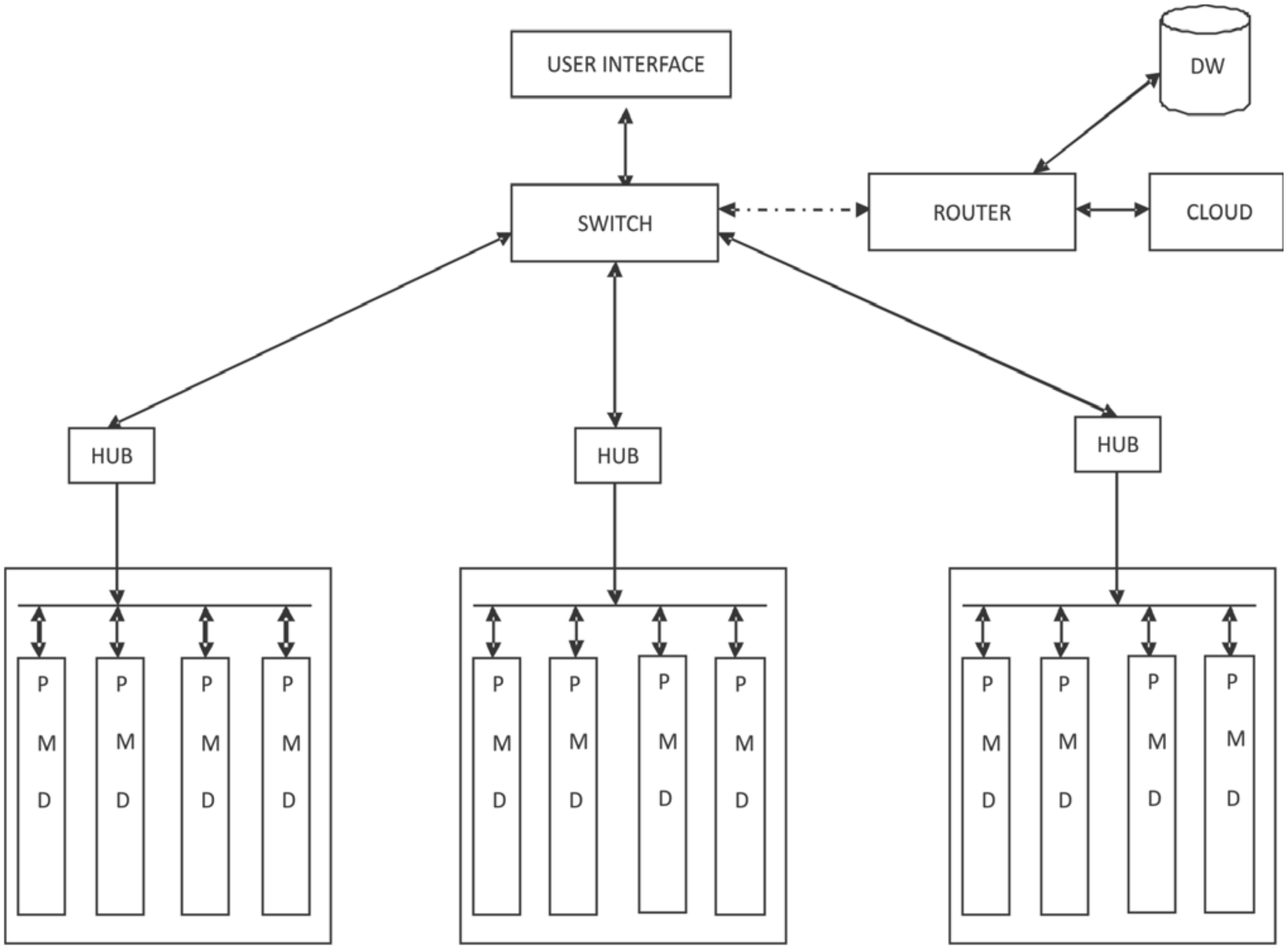


Fig. 1. BRKSS Architecture

MAXIMUM POSSIBILITY OF CLUSTERS AND NODES

Here, the number of clusters formed and the number of nodes depend upon the number of ports in the switch. Two ports of the switch will be used for connecting with the console and the router. Suppose that'd' is the number of nodes in each cluster. In Table–1, the table gives an idea about the maximum number of clusters that could be formed. Here, up to 64 port switch have been shown which could be increased based on how much large is the data warehouse.

Table 1.

Number of Switch Ports	8	16	32	64
Number of Hubs	6	14	30	62
Number of Clusters	6	14	30	62
Number of Node	$6*d$	$14*d$	$30*d$	$62*d$

4. PROPOSED ALGORITHM

To overcome the limitations of load balancing in shared nothing clustering Inter-query Parallelism has been implemented in the proposed algorithm where many diverse queries or transactions are executed in parallel with one another on many processors. This will not only increase the throughput but will also scale up the system.

The steps of the algorithm are stated below :

Step–1 : Consider the number of transactions entering into the system in a batch mode.
[Suppose 'm' numbers of transactions are there in a batch]

Step–2: Check the number of clusters.
[Suppose 'c' be the number of clusters]

Step–3: Calculate the maximum value for each cluster ($\max_c$) and node ($\max_n$).

$$\max_c = m/c$$
$$\max_n = \max_c/d$$

Where, $\max_c = 0$ and $\max_n = 0$ initially and d is the number of nodes in a cluster.

Step–4 : Distribute all the transactions evenly in the cluster based upon the $\max_c$ value and in the nodes based upon $\max_n$ value.

Node Based

Step–5 : Now, calculate $\max_q = \max_n/10$
Where, $\max_q$ is the number of transactions that will enter into the MLFQ apiece time for execution and also calculate $\text{rem}_n = \max_n - \max_q$ for apiece node
Where, rem_n is the remaining number of transactions of a node.

Step–6 : Now for Node based Load Balancing, perform MLFQ Scheduling in apiece node.

Step–6 (*a*) : Allocate a ready queue to the processor of all the nodes and split the ready queue into 'q' number of queues.

Step–6 (*b*) : Put highest priority to q_0 as q_0 is the first queue and lowest priority to q_n as q_n is the last queue.

Step–6 (*c*) : Perform Round Robin Scheduling from q_0 to q_{n-1} and FCFS in q_n.

Step–6 (*d*) : Follow the MLFQ rules while performing the scheduling.
Considering two jobs A and B entering into the queue, apply the following rules:

Rule–1 : If Priority (A) > Priority (B), A will run (B doesn't).

Rule–2 : If Priority (A) = Priority (B), A and B both run in RRS.

Rule–3 : When a job enters the system, it is placed at the highest priority, that is, the topmost queue.

Rule–4 : Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced, that is, it moves down one queue. This is called the Gaming Tolerance.

Rule–5 : After some time period S, move all the jobs in the system to the topmost queue. This is also known as Priority Boost.

The above rules are applicable for a transaction or a query as well.

Step–6 (*e*) : At the end of apiece transaction, take up a new one from rem_n.

Step–6 (*f*) : After time interval $t_{z,}$ status regarding the number of executed transactions and the remaining transactions will be send to the buffer from apiece node of a cluster.

Step–7 : If value of rem_n does not become 0 within time $t_{z,}$ perform Node Based Load Balancing through Push Migration approach.

Step–7 (*a*) : After receiving the status, check in the buffer.

- If $\text{rem}_n = \text{max}_n / 2$ in all the nodes, then situation is stable, continue with the execution and move to Step–9.
- If $\text{rem}_n > \text{max}_n / 2$ in all the nodes, then give them more time to reach the stable situation and then move to Step–9.
- If $\text{rem}_n = \text{max}_n / 2$ in half of the nodes and $\text{rem}_n > \text{max}_n / 2$ in other half, then give some time for execution so that most of the nodes would either reach to $\text{rem}_n < \text{max}_n / 2$ or $\text{rem}_n = \text{max}_n / 2$. Then move to Step–9.
- If in most of the nodes rem_n is much less than $\text{max}_n / 2$ and in a few nodes $\text{rem}_n = \text{max}_n / 2$, then continue with the execution and after that move to Step–9.
- If rem_n is much less than $\text{max}_n / 2$ in maximum nodes and in some nodes $\text{rem}_n > \text{max}_n / 2$, then start performing load balancing.

Step–7 (*b*) : When condition 7 (*a*) (*v*) occurs in the node(*s*), then send a signal to the console through switch.

Step–7 (*c*) : Console in return will send an instruction to the node(*s*) to submit the remaining transactions rem_n.

Step–7 (*d*) : Redistribute rem_n into other nodes, depending upon the condition: $\text{rem}_n < = \text{max}_n / 2$

Step–8 : Continue Step–7(*a*) to Step–7 (*d*) until max_c gets executed.

Step–9 : With the end of all the transactions, again a new max_c will enter and repeat the above steps.

Cluster Based

If after t_y time, the console does not get any information regarding a particular cluster, then it will assume that a fail over has occurred in the cluster. Then the console will perform cluster based load balancing to shift the load of the fail over cluster to the rest of the active clusters.

Step–10 : After time interval 't_y', console will check the executed transactions max_e and the remaining transactions rem_c for apiece cluster and a copy of rem_c transaction will be send to the buffer.

$$\text{rem}_c = \text{max}_c - \text{max}_e$$

Step–11 : Perform Cluster Based Load Balancing through Push Migration approach when cluster fail over will take place.

Step–11 (*a*) : After time $t_{y,}$ check in the buffer.

- If $\text{rem}_c = \text{max}_c / 2$ in all the active clusters, then situation is stable, continue with the execution and wait for condition 11 (*a*) (*v*) to occur.
- If $\text{rem}_c > \text{max}_c / 2$ in all the active clusters, then give them more time to reach the stable situation and wait for condition 11 (*a*) (*v*) to occur.
- If $\text{rem}_c = \text{max}_c / 2$ in half of the active clusters and $\text{rem}_c > \text{max}_c / 2$ in other half, then give some time for execution so that most of the clusters will either reach to $\text{rem}_c < \text{max}_c / 2$ or $\text{rem}_c = \text{max}_c / 2$ and wait for condition 11 (*a*) (*v*) to occur.
- If in most of the active clusters rem_c is much less than $\text{max}_c / 2$ and in a few active cluster $\text{rem}_c = \text{max}_c / 2$, then continue with the execution and wait for condition 11(*a*) (*v*) to occur.
- If rem_c is much less than $\text{max}_c / 2$ in all the active clusters, then performs load balancing.

Step–11 (*b*): Redistribute rem_c of the fail over cluster into the other active clusters that would satisfy the condition $\text{rem}_c < = \text{max}_c / 2$ in the active clusters.

Step–12 : Continue Step–11 (*a*) and Step–11 (*b*) until m gets executed.

Step–13 : At the end of all the transactions, again a new batch will enter and repeat the above steps.

EXAMPLE : Suppose the number of transactions (m) in one batch is 1, 80, 000, number of clusters (c) = 3 and number of nodes in apiece cluster (d) = 4.

Then,
$$\max_c = (1,80,000/3) = 60,000$$
$$\max_n = (60,000/4) = 15,000$$

The stable condition for apiece node is given by:
$$\max_n/2 = 7500$$
$$\max_q = \max_n/10 = 1500$$

Node Based Load Balancing

Number of transactions entering into the MLFQ will be either max$_q$ or multiplicand of max$_q$ like :

$$1500 * 1 = 1500$$
$$1500 * 2 = 3000$$
$$1500 * 3 = 4500$$
$$1500 * 4 = 6000$$

At apiece t_z interval, a status about the nodes will be send to the console. The console will get information about the remaining transactions of apiece node (rem$_{n)}$ and will decide whether continuous execution or load balancing is required or not.

Initially,

Nodes	*d*1	*d*2	*d*3	*d*4
Executed Transactions (max$_q$)	0	0	0	0
Remaining Transactions (rem$_n$)	15,000	15,000	15,000	15,000

After t_z1 Interval,

Nodes	*d*1	*d*2	*d*3	*d*4
Executed Transactions (max$_q$)	1,500	1,500	1,500	1,500
Remaining Transactions (rem$_n$)	13,500	13,500	13,500	13,500

After t_z2 Interval,

Nodes	*d*1	*d*2	*d*3	*d*4
Executed Transactions (max$_q$)	6,000	6,000	3,000	4,500
Remaining Transactions (rem$_n$)	7,500	7,500	10,500	9,000

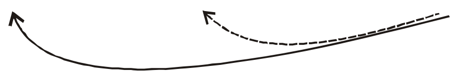
After t_z3 Interval,

Nodes	*d*1	*d*2	*d*3	*d*4
Executed Transactions(max$_q$)	6,000	6,000	1,500	1,500
Remaining Transactions(rem$_n$)	1,500	1,500	9,000	7,500

After third Iteration in *d*1 and *d*2, rem$_n$ is much less than their max$_n$ / 2 and in *d*4, rem$_n$ is stable, but in *d*3, rem$_n$ > max$_n$ / 2, so, Node Based Load Balancing is performed. Here, 1,500 transactions would be taken away from *d*3, making it stable and then putting that load into either *d*1 or *d*2.

Load Balancing is given in the table below :

Nodes	*d*1	*d*2	*d*3	*d*4
Executed Transactions ($\max_q$)	6,000	6,000	1,500	1,500
Remaining Transactions (rem_n)	1,500	1,500	9,000	7,500



New table after Load Balancing:

If remaining transactions are transferred to *d*1, then final table will be as given below:

Nodes	*d*1	*d*2	*d*3	*d*4
Executed Transactions ($\max_q$)	6,000	6,000	1,500	1,500
Remaining Transactions (rem_n)	3,000	1,500	7,500	7,500

If remaining transactions are transferred to *d*2, then the final table will be as given below:

Nodes	*d*1	*d*2	*d*3	*d*4
Executed Transactions ($\max_q$)	6,000	6,000	1,500	1,500
Remaining Transactions (rem_n)	1,500	3,000	7,500	7,500

While performing the above Iterations, a status about all the nodes and their clusters would go to the console and it will get updated on a regular basis.

Cluster Based Load Balancing

The console will get information in the time interval t_y about the executed number of transactions, that is, $\max_e$ and a copy of all the remaining transactions rem_c. So, when fail over of any cluster occurs, then the console will send the unexecuted copy of transactions of the fail over cluster to the other clusters.

Initially,

Clusters	*c*1	*c*2	*c*3
Executed Transactions ($\max_e$)	0	0	0
Remaining Transactions (rem_c)	60,000	60,000	60,000

After t_y1 interval,

Clusters	*c*1	*c*2	*c*3
Executed Transactions ($\max_e$)	20,000	20,000	30,000
Remaining Transactions (rem_c)	40,000	40,000	30,000

At the end of t_y1 interval, console will have the status of $\max_e$ and a copy of rem_c within it, till t_y2 execution ends successfully. After that it will hold a copy of rem_c and status of $\max_e$ till t_y3 execution ends successfully.

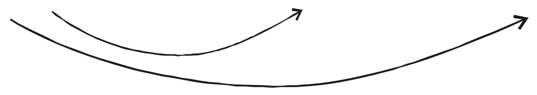
After t_y2 interval,

Clusters	*c*1	*c*2	*c*3
Executed Transactions ($\max_e$)	FAIL OVER	10,000	10,000
Remaining Transactions(rem_c)	40,000	30,000	20,000

In t_y2, a fail over occurs and the console that is holding the value of rem_c from t_y1 interval will distribute it to the other active clusters until they themselves come to a value much less than $\text{max}_c / 2$.

After t_y3 interval,

Clusters	$c1$	$c2$	$c3$
Executed Transactions (max_e)	FAIL OVER	15,000	15,000
Remaining Transactions (rem_c)	40,000	15,000	5,000



Load Balancing :

Clusters	$c1$	$c2$	$c3$
Executed Transactions (max_e)	FAIL OVER	15,000	15,000
Remaining Transactions (rem_c)	0	30,000	30,000

5. PARAMETER ANALYSIS

SPEED UP

Speed Up is the extent to which more hardware can perform the same task in less time than the original system. With added hardware, speed up holds the task constant and measures time savings. In this case, speed up will be linear as shown in Figure–2, as the number of transactions is increasing with the increase in number of CPU.

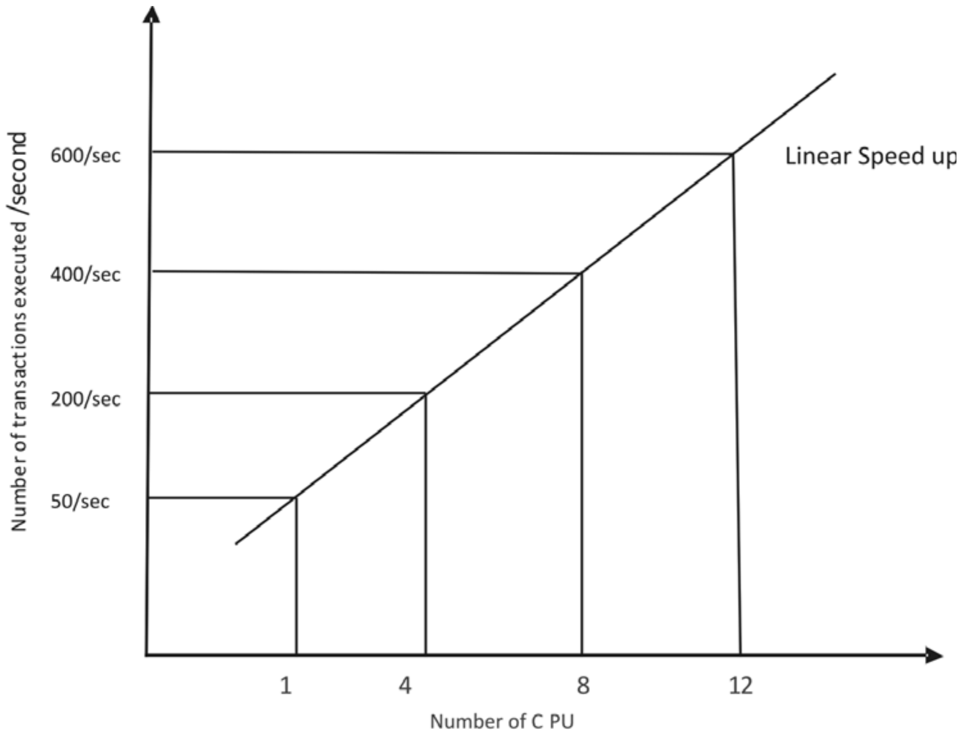


Fig. 2. Linear Speed Up.

Sub Linear Speed Up can occur when the rate of execution of apiece CPU is diverse, that is, either high or low, but their overall execution is high. In Figure–2, execution of apiece CPU is 50, so with the increase in number of CPU, overall execution is also increasing (in multiple of 50). But in Figure–3, four CPUs are performing 50 transactions each, whereas eight CPUs are performing approximately 37 transactions each and twelve CPUs are performing approximately 58 transactions each, but their overall execution is high. In this case the Speed Up will be sub linear.

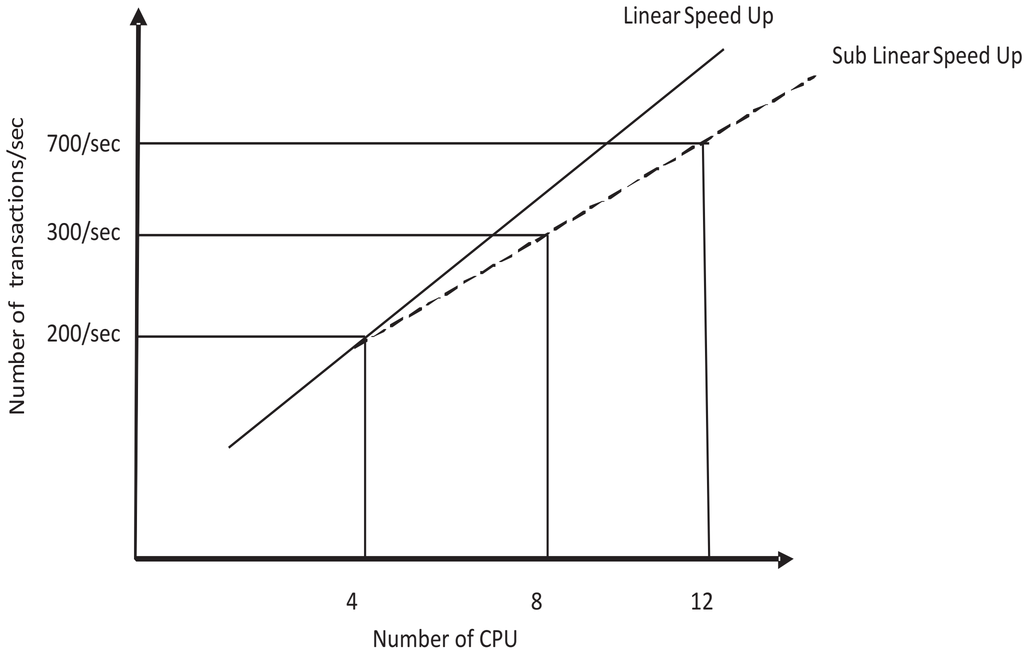


Fig. 3. Sub Linear Speed Up

SCALE-UP

Scale-up is the ability to keep the same performance level (Response Time) when both workload (Transaction) and resources (CPU, Memory) increase proportionally. In Figure–4, number of transactions per second is increasing proportionately with the increase in number of CPU. So, the Scale-up is linear in this case.

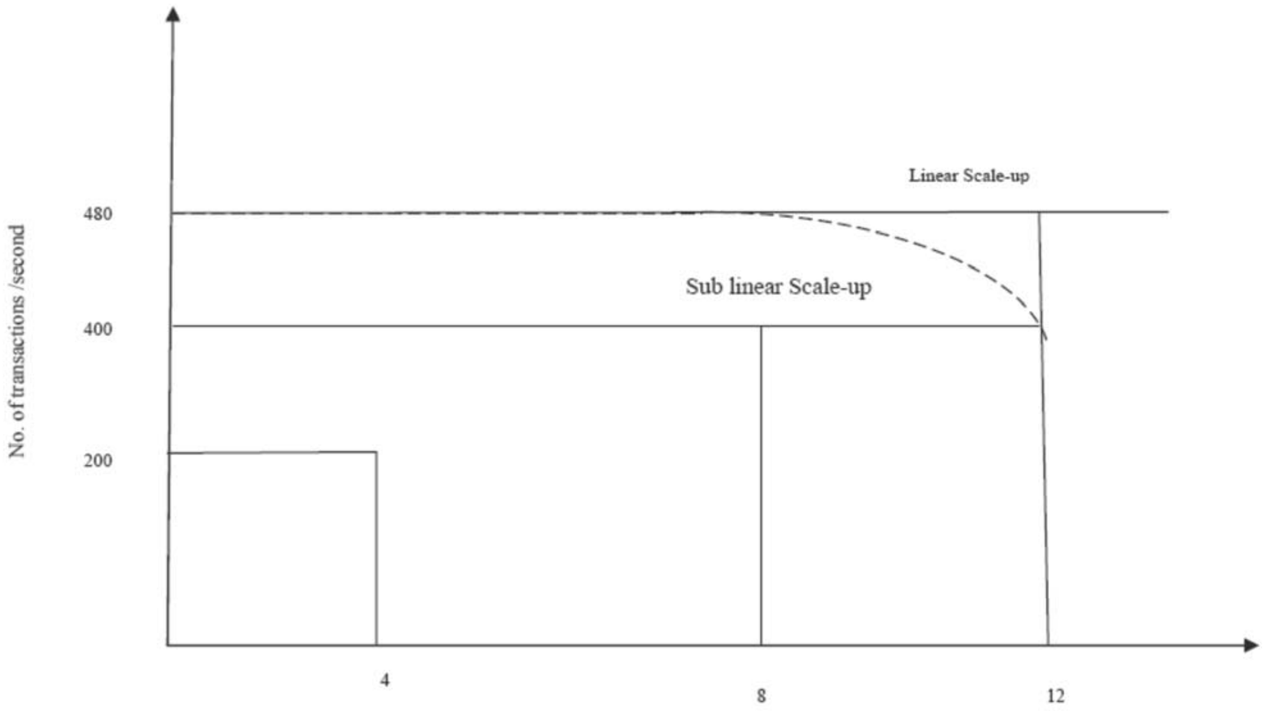


Fig. 4. Scale Up

Just like Sub Linear Speed Up, Scale Up can be also sub linear. If the rate of the transaction decreases per node, then sub linear scale-up will occur. In this case, the overall execution of transaction will always be high as shown in Figure–5.

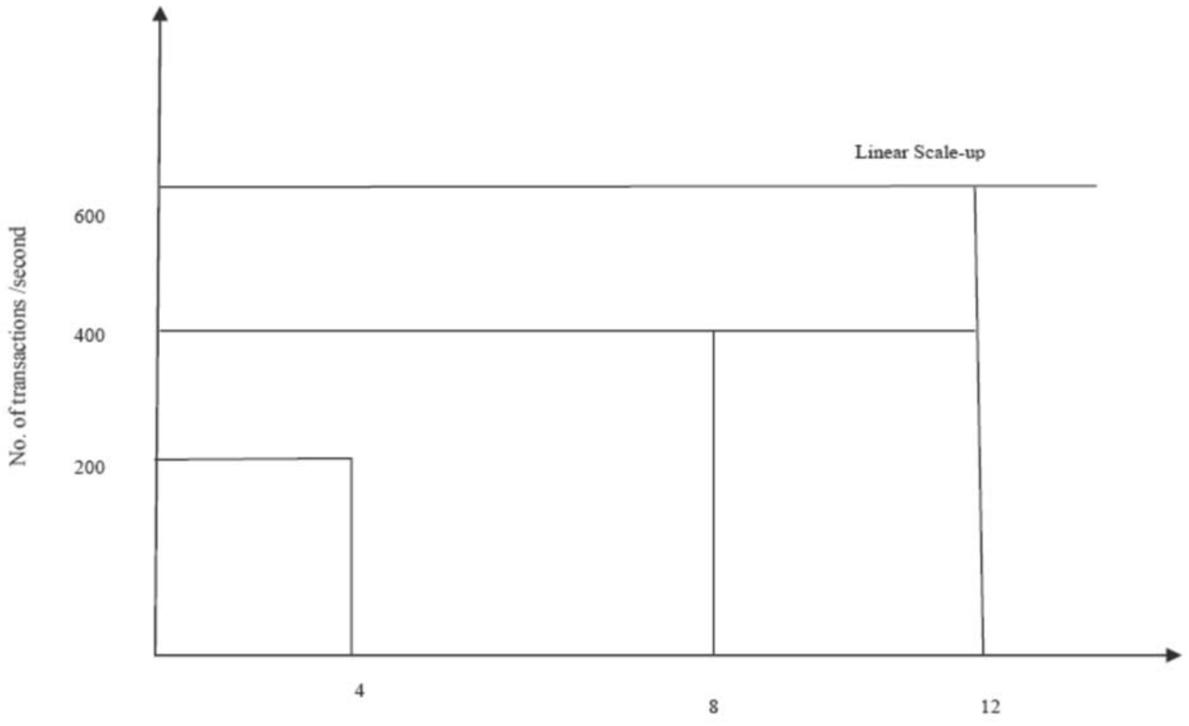


Fig. 5. Sub Linear Scale Up.

6. CONCLUSION

The problems of shared nothing architecture where scalability and speed up is high but load balancing is an issue have been instantiated and studied. From the extensive study, it has transpired that the problem could be solved by introducing the completely new and novel BRKSS Architecture that supports shared nothing clustering. The designing of the architecture is done in such a way that it can be implemented easily in large systems for performing huge amount of task and load can be balanced based upon the proposed algorithm that would work in the console. The load balancing strategy follows two ways, that is, with respect to the nodes and with respect to the fail over of any cluster. Thus, the performance does not get affected too much.

7. REFERENCES

1. Lee, S., "Shared-Nothing vs. Shared-Disk Cloud Database Architecture**",** International Journal of Energy, Information and Communications, 2 (4), November, 2011
2. Popeanga, J., "Shared-Nothing' Cloud Data Warehouse Architecture", Database Systems Journal, V (4), 2014
3. Müseler, T., "A survey of Shared-Nothing Parallel Database Management Systems" [Comparison between Teradata, Greenplum and Netezza Implementation], IRCSE 2012, Mälardalen University, Sweden
4. DeWitt, D., J., Gray, J., **"**Parallel Database Systems: The Future of Database Processing or a Passing Fad?", The Research was partially supported by the Defence Advanced Research Projects Agency under contract N00039-86-C-0578 by the National Science Foundation under grant DCR-8512862 and by Research Grants from Intel Scientific Computers, Tandem Computers and Digital Equipment Corporation
5. Furtado, P., "A Survey on Parallel and Distributed Data Warehouses", IGI Publishing, 5 (2), April–June, 2009
6. Minhas, U., F., Lomet, D., Thekkath, C., A., "Chimera: Data Sharing Flexibility, Shared Nothing Simplicity", IDEAS, Springer Verlag, September, 2011
7. Datta, A., Moon, B., Thomas, H., "A Case for Parallelism in Data Warehousing and OLAP", Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA'98), September, 1998
8. DeWitt, D., J., Gray, J., "Parallel Database Systems: The Future of High Performance Database Systems", Communication of the ACM, 35 (6), June, 1992, pp.85–98

9. Ezeife, C., I., Barker, K., "A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System", Distributed and Parallel Databases, 1: 247–272, 1995

10. Ezeife, C., I., "A Partition-Selection Scheme for Warehouse Aggregate Views", International Conference of Computing and Information, Manitoba, Canada, June, 1998

11. "HP Intelligent Warehouse", Hewlett Packard White Paper, 1997

12. Jurgens, M., Lenz, H–J., "Tree Based Indexes vs. Bitmap Indexes: A Performance Study", International Workshop, DMDW' 99, Heidelberg, Germany, 1999

13. The Data Warehouse Toolkit, Kimball, R., Ed. J., Wiley and Sons, Inc., 1996

14. Patel, A., Patel, J., M., "Data Modeling Techniques for Data Warehouse", International Journal of Multidisciplinary Research, 2 (2), 2012, pp. 240–246

15. Farhan, M., S., Marie, M., E., El-Fangary, L., M., Helmy, Y., K., "An Integrated Conceptual Model for Temporal Data Warehouse Security", Computer and Information Science, 4 (4), 2011, pp. 46–57

16. Eder, J., Koncilia, C., "Changes of Dimension Data in Temporal Data Warehouses", Proceedings of Third International Conference on Data Warehousing and Knowledge Discovery, DaWaK' 01, Munich, Germany, LNCS, Springer, 2001, pp. 284–293

17. Golfarelli, M., Maio, D., Rizzi, S., "The Dimensional Fact Model: A Conceptual Model for Data Warehouses", International Journal of Cooperative Information Systems, 7 (2-3), (1998), pp. 215–247

18. Golfarelli, M., Rizzi, S., "A Methodological Framework for Data Warehouse Design", Proceedings of ACM First International Workshop on Data Warehousing and OLAP, DOLAP, Washington, 1998, pp. 3–9

19. Bernardino, J., Madeira, H., "Data Warehousing and OLAP: Improving Query Performance Using Distributed Computing"

20. Albrecht, J., Gunzel, H., Lehner, W., "An Architecture for Distributed OLAP", International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA' 98, 1998

21. Comer, D., "The Ubiquitous B-tree", ACM Computing Surveys, 11(2): 121–137, 1979

22. Chowdhury, R., Datta, S., Dasgupta, S., De, M., "Implementation of Central Dogma Based Cryptographic Algorithm in Data Warehouse for Performance Enhancement", International Journal of Advanced Computer Science and Applications, 6 (11), November, 2015, pp. 29–34.

23. Chowdhury, R., Pal, B., Ghosh, A., De, M., "A Data Warehouse Architectural Design Using Proposed Pseudo Mesh Schema", Proceedings of First International Conference on Intelligent Infrastructure: 47th Annual National Convention of Computer Society of India, Science City Auditorium, Kolkata, December, 2012, pp. 138–141, ISBN (13)–978-1-25-906170-7 & ISBN (10)–978-1-25-906170-1

24. Chowdhury, R., Pal, B., "Proposed Hybrid Data Warehouse Architecture Based on Data Model", International Journal of Computer Science and Communication, 1 (2), July–December. 2010, pp. 211–213