International Journal of Engineering and Management Research e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
www.ijemr.net https://doi.org/10.31033/ijemr.9.1.1

Simulation of BRKSS Architecture for Data Warehouse Employing Shared Nothing Clustering

Bikramjit Pal[1] and Mallika De[2]
[1]Assistant Professor, Department of Computer Science and Engineering, JIS University, Kolkata, West Bengal, INDIA
[2]Retired SSO (Professor Rank), Department of Engineering and Technological Studies, University of Kalyani, Kalyani, West Bengal, INDIA

[1]Corresponding Author: biku_paul@rediffmail.com

ABSTRACT

The BRKSS Architecture 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 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. This architecture uses the concept of load balancing by moving the transaction among various nodes within the clusters so that the overhead of a particular node can be minimised. In this paper we have simulated the working of BRKSS architecture using JDK 1.7 with Net beans 8.0.2. We compared the result of performance parameters sch as turnaround time, throughput and waiting time with existing hierarchical clustering model.

Keywords-- BRKSS Architecture, Shared Nothing Clustering, Buffer, Cloud, Router, Switch, Hubs, Load Balancing, Databases, Turnaround Time, Throughput, Waiting Time, Ports

I. INTRODUCTION TO BRKSS ARCHITECTURE

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 [1] is simulated, 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 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).

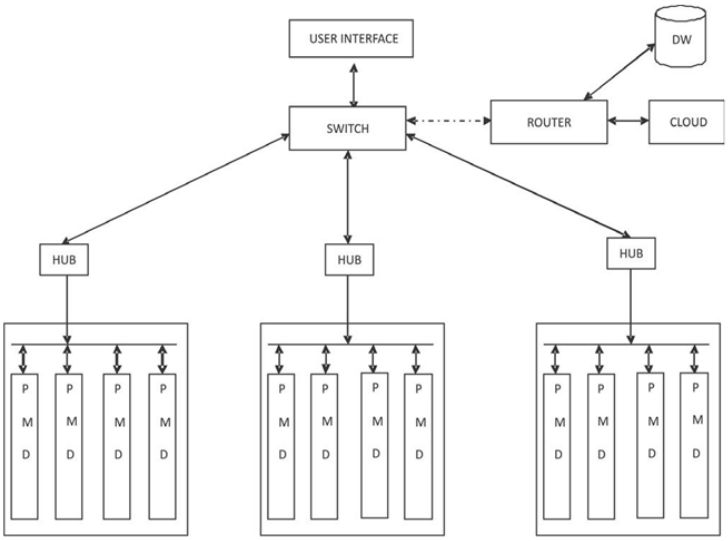


Fig. 1. BRKSS Architecture

This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1

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*

II. 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 $rem_n = max_n – max_q$ 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.

This work is licensed under Creative Commons Attribution 4.0 International License.

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 $rem_n = max_n / 2$ in all the nodes, then situation is stable, continue with the execution and move to Step–9.
- If $rem_n > max_n / 2$ in all the nodes, then give them more time to reach the stable situation and then move to Step–9.
- If $rem_n = max_n / 2$ in half of the nodes and $rem_n > max_n / 2$ in other half, then give some time for execution so that most of the nodes would either reach to $rem_n < max_n / 2$ or $rem_n = max_n / 2$. Then move to Step–9.
- If in most of the nodes rem_n is much less than $max_n / 2$ and in a few nodes $rem_n = max_n / 2$, then continue with the execution and after that move to Step–9.
- If rem_n is much less than $max_n / 2$ in maximum nodes and in some nodes $rem_n > max_n / 2$, 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: $rem_n <= max_n / 2$
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.
$rem_c = max_c – max_e$
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 $rem_c = max_c / 2$ in all the active clusters, then situation is stable, continue with the execution and wait for condition 11 (*a*) (*v*) to occur.
- If $rem_c > max_c / 2$ 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 $rem_c = max_c / 2$ in half of the active clusters and $rem_c > max_c / 2$ in other half, then give some time for execution so that most of the clusters will either reach to $rem_c < max_c / 2$ or $rem_c = max_c / 2$ and wait for condition 11 (*a*) (*v*) to occur.
- If in most of the active clusters rem_c is much less than $max_c / 2$ and in a few active cluster $rem_c = max_c / 2$, then continue with the execution and wait for condition 11(*a*) (*v*) to occur.
- If rem_c is much less than $max_c / 2$ 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 $rem_c <= max_c / 2$ 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

This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1

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.

This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1

Load Balancing is given in the table below :

Nodes	$d1$	$d2$	$d3$	$d4$
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 $d1$, then final table will be as given below:

Nodes	$d1$	$d2$	$d3$	$d4$
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 $d2$, then the final table will be as given below:

Nodes	$d1$	$d2$	$d3$	$d4$
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	$c1$	$c2$	$c3$
Executed Transactions (max_e)	0	0	0
Remaining Transactions (rem_c)	60,000	60,000	60,000
After t_y1 interval,			
Clusters	$c1$	$c2$	$c3$
Executed Transactions (max_e)	20,000	20,000	30,000
Remaining Transactions (rem_c)	40,000	40,000	30,000

This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1

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	$c1$	$c2$	$c3$
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 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

III. SIMULATION OF THE ALGORITHM AND RESULT ANALYSIS

The BRKSS algorithm has been simulated by using JDK 1.7 with Netbeans 8.0.2 and the database has been maintained by MySQL. The algorithm takes the following user inputs: number of cluster, number of nodes, user queries which may be numerous at a particular time period. User queries are the transaction that determines the performance of a DW. The output is obtained for Turnaround Time, Waiting Time and Throughput for a given set of inputs and the result is compared with existing pseudo mesh schema.

I have discussed the results for three cases, which are shown in Table 3.1, Table 3.2 and Table 3.3. Also, the comparative result analysis of the proposed and existing hierarchical clustering model is displayed graphically.

This work is licensed under Creative Commons Attribution 4.0 International License.

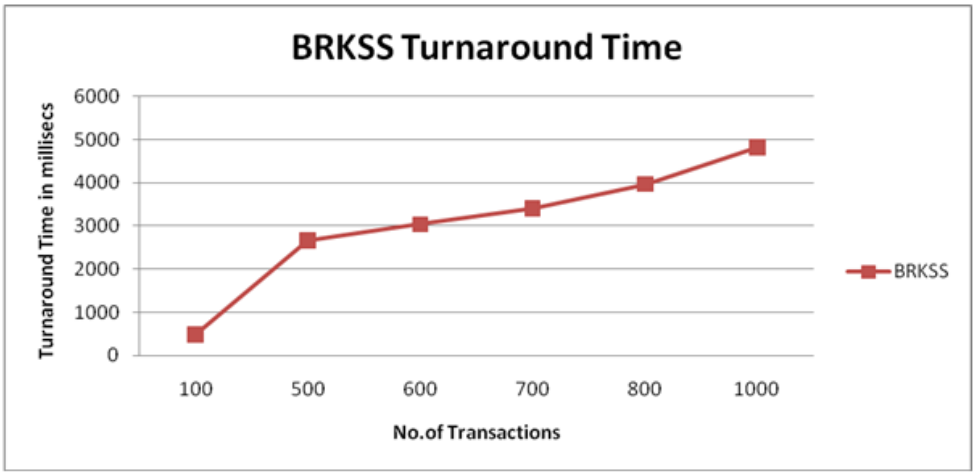
e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1

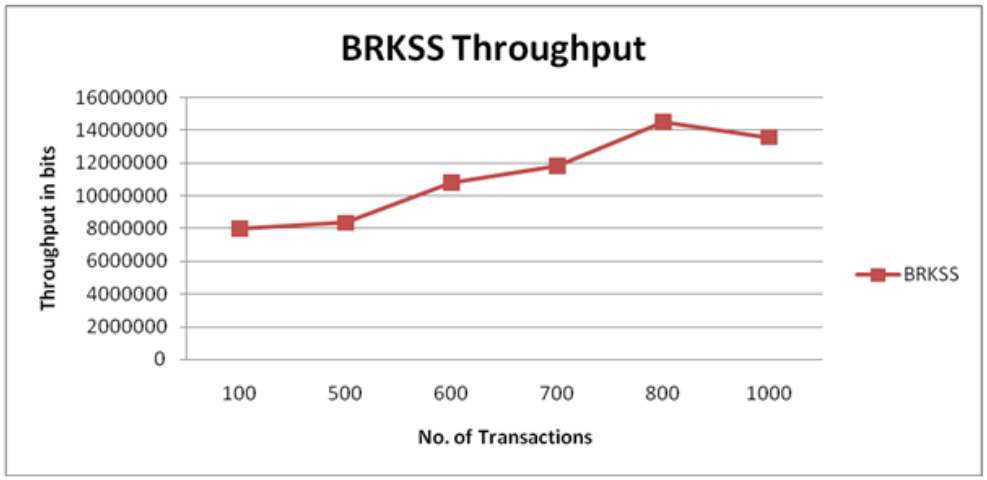
Simulation Results of BRKSS algorithm for 10 nodes and 2 clusters

Table 3.1

No. of transactions	Type of architecture	Turnaround time	Throughput	Waiting time
100	BRKSS	486	7993368	486
	Existing	1049	6391464	649
500	BRKSS	2674	8358360	2674
	Existing	3506	6673616	3105
600	BRKSS	3041	10801256	3041
	Existing	4160	8127600	3759
700	BRKSS	3406	11850384	3406
	Existing	5033	10582072	4600
800	BRKSS	3971	14539848	3970
	Existing	5330	12751888	4945
1000	BRKSS	4825	13598344	4809
	Existing	6598	8127904	6212

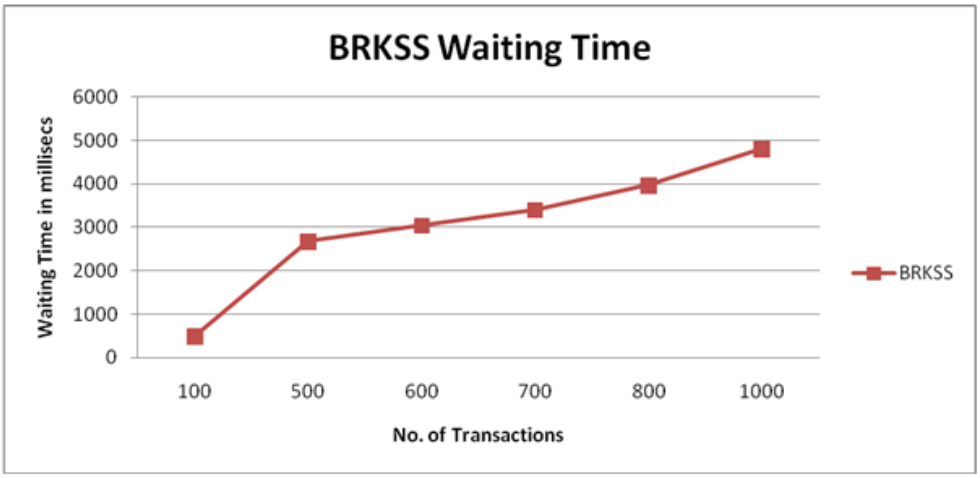
Graphical Representation



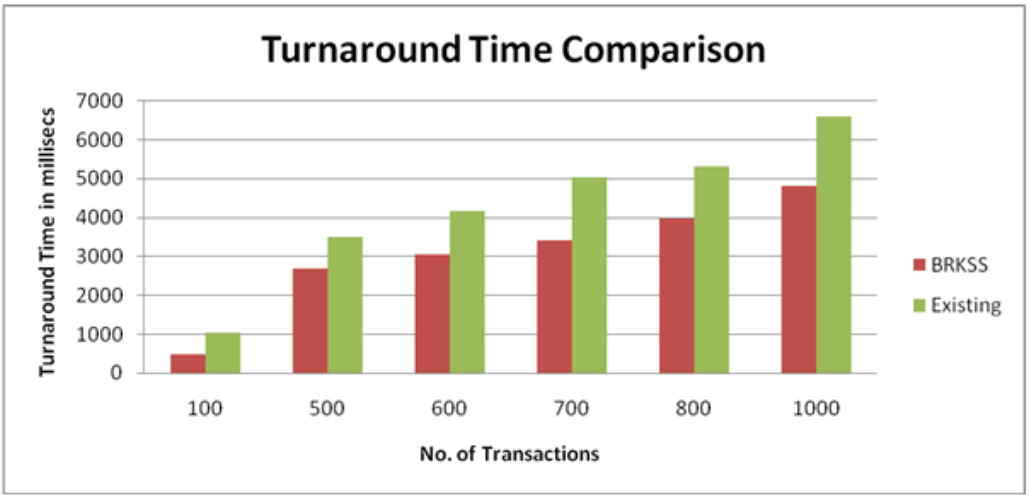


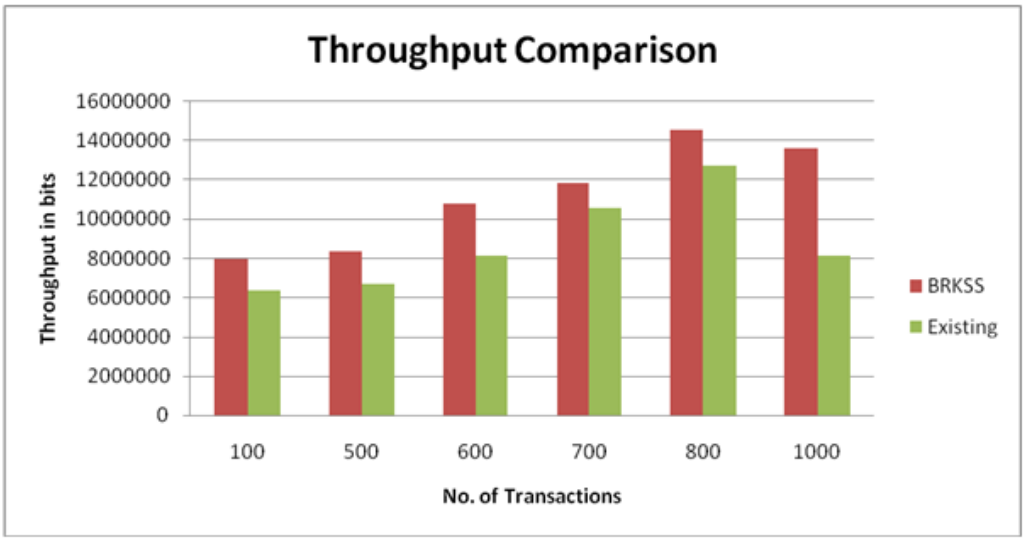
This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1



Comparison with Existing Model





This work is licensed under Creative Commons Attribution 4.0 International License.

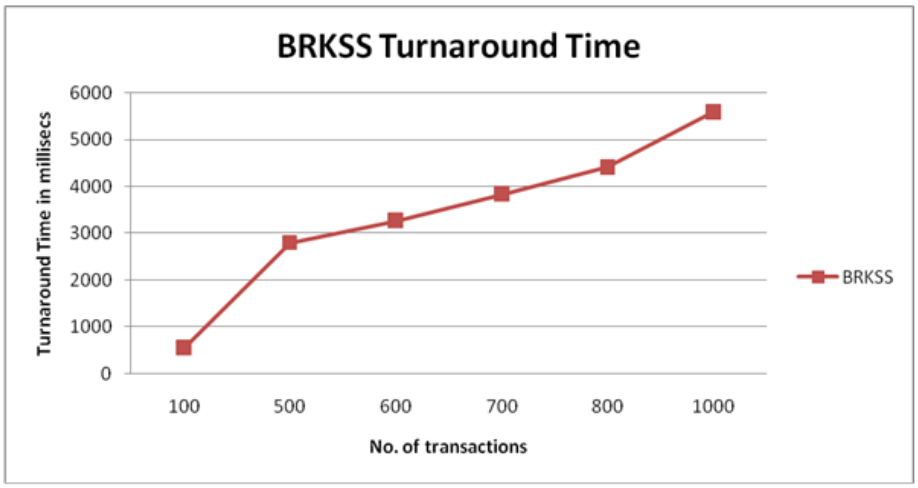
e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1

Simulation Results of BRKSS algorithm for 20 nodes and 5 clusters

Table 3.2

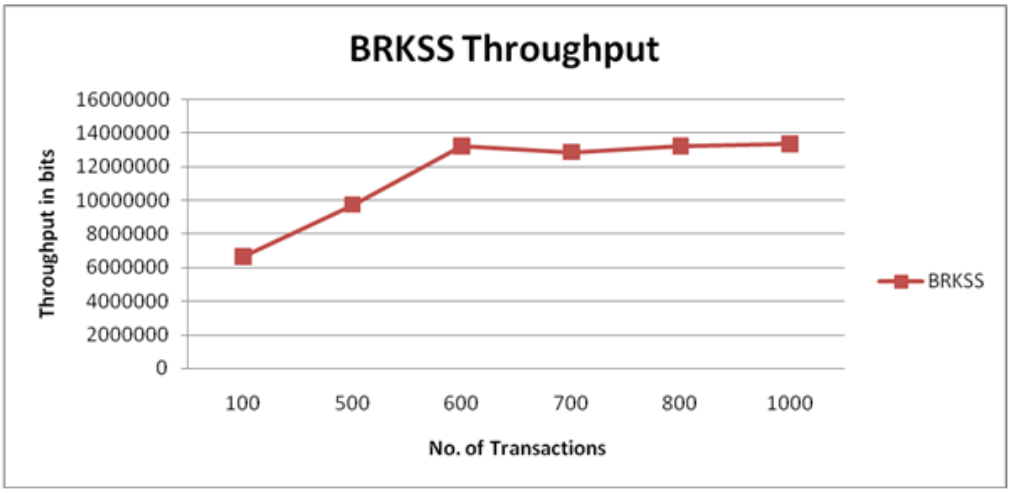
No. of transactions	Type of architecture	Turnaround time	Throughput	Waiting time
100	BRKSS	547	6643584	532
	Existing	1138	6174360	755
500	BRKSS	2795	9754000	2795
	Existing	3670	6629392	3223
600	BRKSS	3267	13243136	3267
	Existing	4177	8125200	3776
700	BRKSS	3837	12896984	3837
	Existing	4593	12287472	4208
800	BRKSS	4408	13232920	4408
	Existing	5427	12731096	5026
1000	BRKSS	5595	13390704	5593
	Existing	6661	8109856	6276

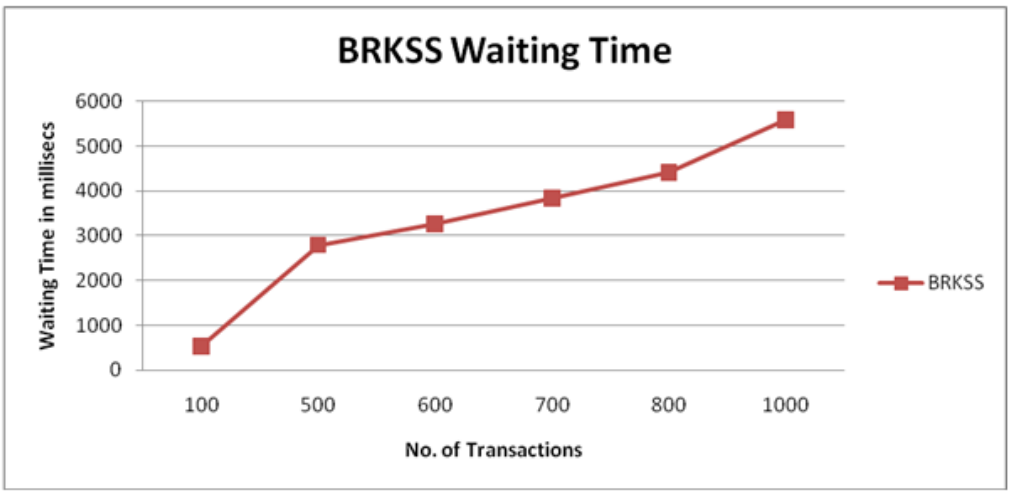
Graphical Representation



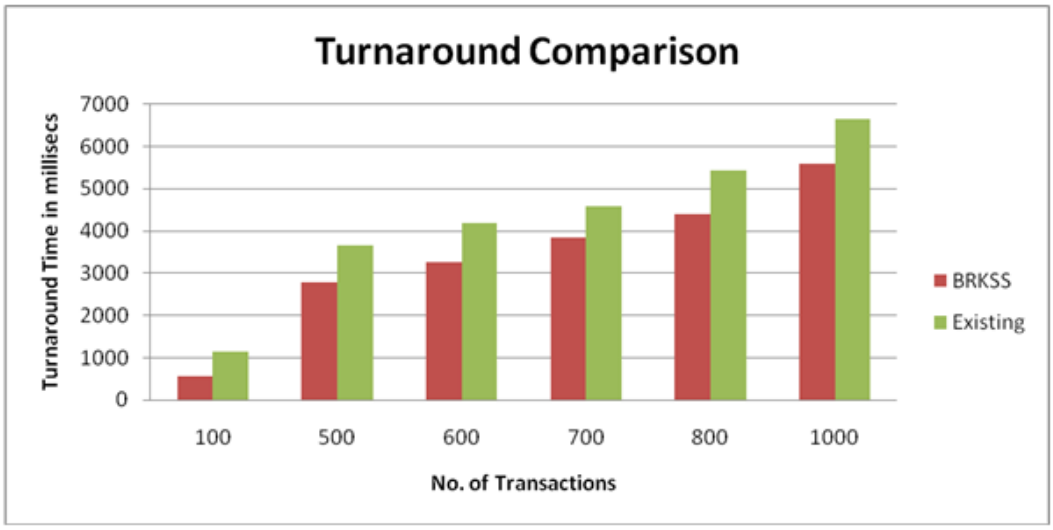
This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1



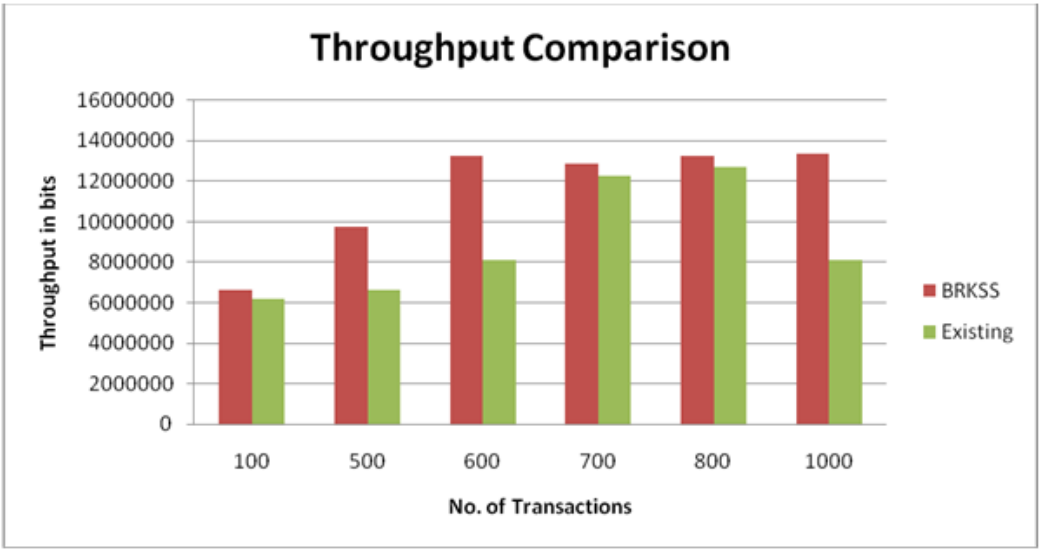


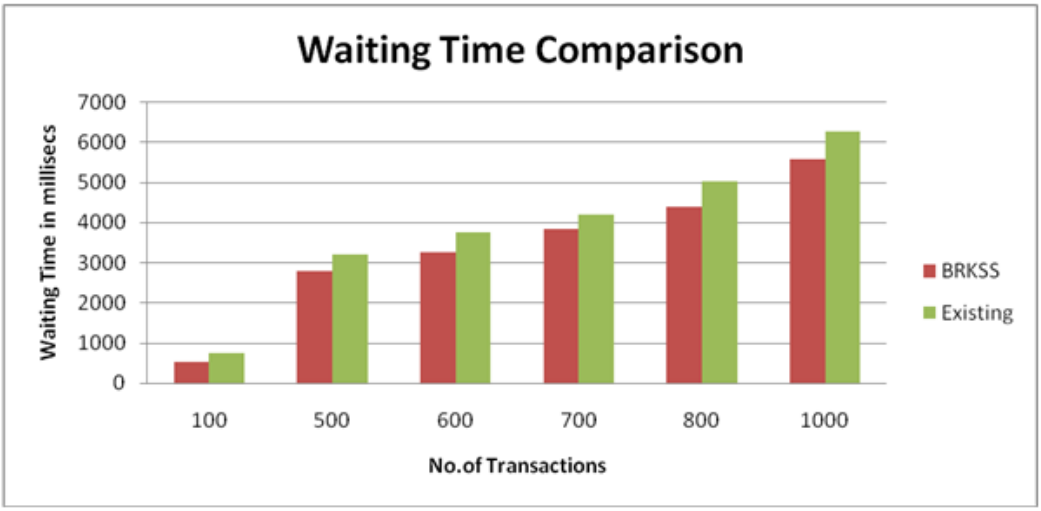
Comparison with Existing Model



This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1





Simulation Results of BRKSS algorithm for 12 nodes and 3 clusters

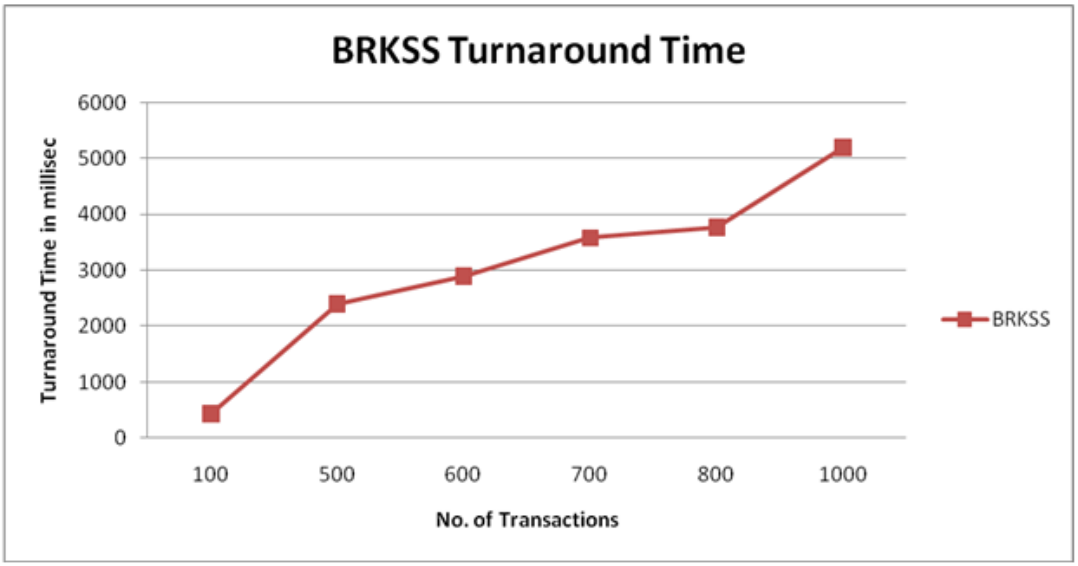
Table 3.3

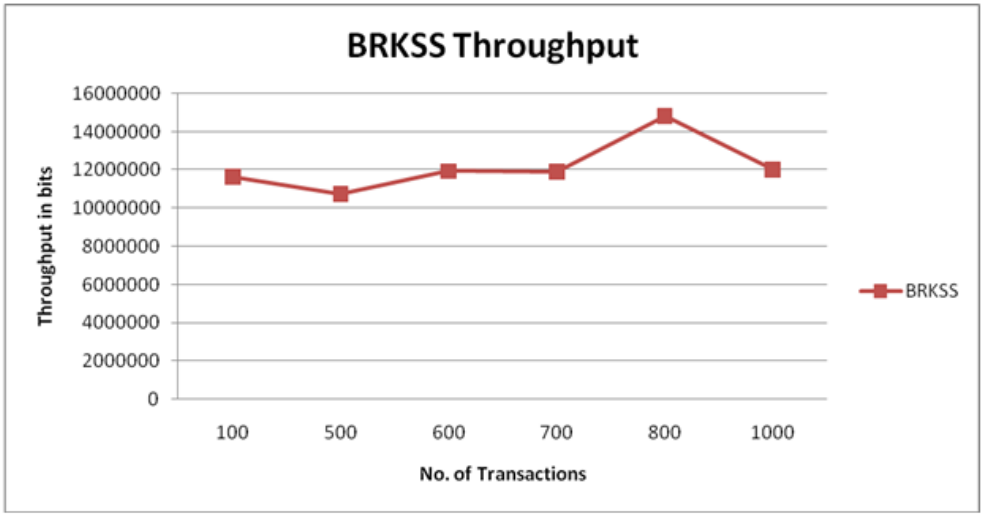
No. of transactions	Type of architecture	Turnaround time	Throughput	Waiting time
100	BRKSS	432	11655368	432
	Existing	1052	5887560	648
500	BRKSS	2402	10735200	2402
	Existing	3646	9174184	3176
600	BRKSS	2890	11939640	2890
	Existing	4093	8126504	3707
700	BRKSS	3581	11901064	3581
	Existing	4895	10673280	4509
800	BRKSS	3772	14839744	3772
	Existing	5195	12756376	4794
1000	BRKSS	5199	12014528	5199
	Existing	6813	8152160	6412

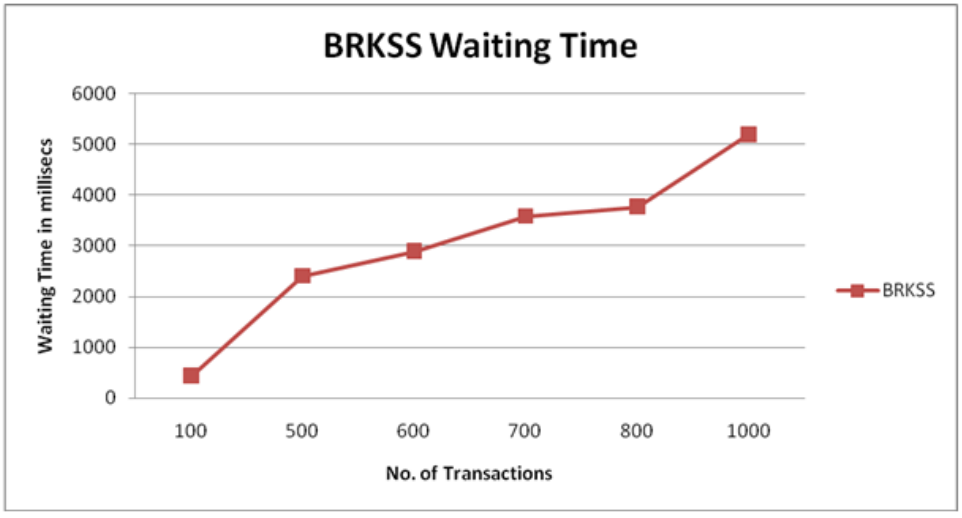
This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1

Graphical Representation



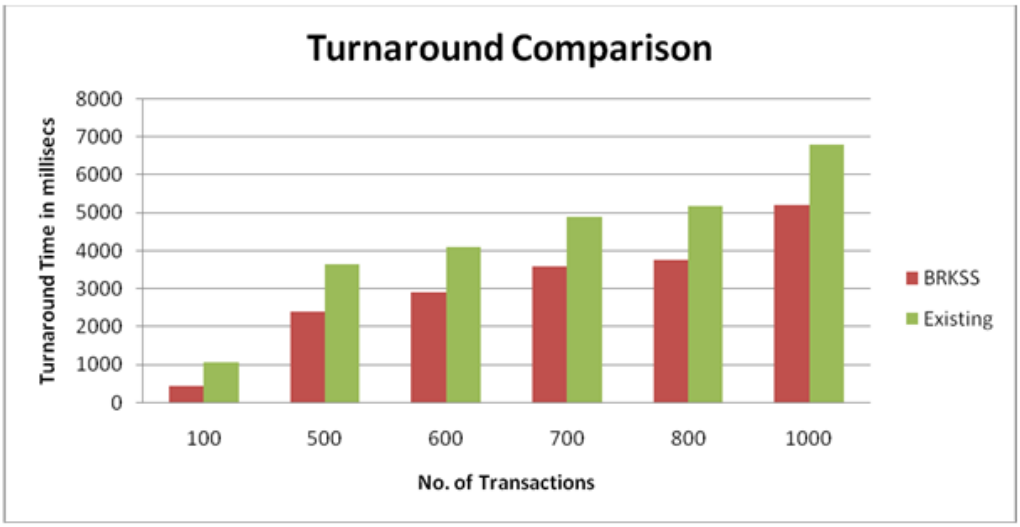


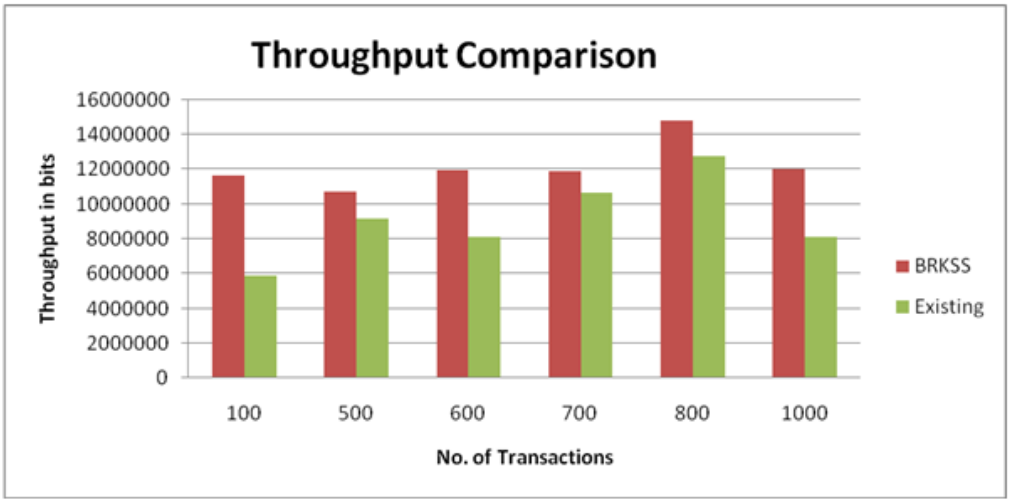


This work is licensed under Creative Commons Attribution 4.0 International License.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume- 9, Issue- 1, (February 2019)
https://doi.org/10.31033/ijemr.9.1.1

Comparison with Existing Model





This work is licensed under Creative Commons Attribution 4.0 International License.

As discussed, this architecture is based upon shared nothing clustering that can scale-up to a large number of computers, increase their speed and maintain the workload. To support it the proposed algorithm has been simulated and the results shown that the performance of BRKSS is better than the existing algorithm.

IV. CONCLUSION

The simulation of BRKSS algorithm has given positive results in its favour when it is compared with existing hierarchical clustering algorithm in terms of turnaround time, throughput and waiting time. Also, the results are consistent for different permutations and combinations of nodes and clusters.

REFERENCES

[1] Pal Bikramjit, Chowdhury Rajdeep, Verma Kumar Gaurav, Dasgupta Saswata, & Dutta Shubham. (2016). Proposed BRKSS architecture for performance enhancement of data warehouse employing shared nothing clustering. *International Science Press, 9*(21), 111 – 121.

[2] Lee, S. (2011). Shared-nothing vs. shared-disk cloud database architecture. *International Journal of Energy, Information and Communications, 2*(4), 211-216.

[3] Popeanga, J. (2014). Shared-nothing' cloud data warehouse architecture. *Database Systems Journal, V*(4), 3-11.

[4] Müseler, T. (2012). *A survey of shared-nothing parallel database management systems [Comparison between teradata, greenplum and netezza implementation]*. Available at: https://pdfs.semanticscholar.org/0365/e61fbf06872334f8063ff03fbe1a260f210c.pdf?_ga=2.38538275.1267149839.1549360061-719258772.1545998512.

[5] De Witt, D., J., & Gray, J. (1991). *Parallel database systems: The future of database processing or a passing fad?*. Available at: https://jimgray.azurewebsites.net/papers/CacmParallelDB.pdf.

[6] Furtado, P. (2009). A survey on parallel and distributed data warehouses. *IGI Publishing, 5*(2), 57-77.

[7] Minhas, U., F., Lomet, D., & Thekkath, C., A. (2011). Chimera: Data sharing flexibility, shared nothing simplicity. *IDEAS*, Springer Verlag.

[8] Datta, A., Moon, B., & Thomas, H. (1998). A case for parallelism in data warehousing and OLAP. *Proceedings of the 9th International Conference on Database and Expert Systems Applications*.

[9] DeWitt, D., J., & Gray, J. (1992). Parallel database systems: The future of high performance database systems. *Communication of the ACM, 35*(6), 85–98.

[10] Ezeife, C., I. & Barker, K. (1995). A comprehensive approach to horizontal class fragmentation in a distributed object based system. *Distributed and Parallel Databases, 1*, 247–272.

[11] Ezeife, C. I. (1998). A partition-selection scheme for warehouse aggregate views. *International Conference of Computing and Information, Manitoba, Canada*.

[12] Jurgens, M. & Lenz, H–J. (1999). *Tree based indexes vs. bitmap indexes: A performance study*. Available at: https://www.researchgate.net/publication/220841960_Tree_Based_Indexes_vs_Bitmap_Indexes_A_Performance_Study.

[13] Kimball, R. (1996). *The data warehouse toolkit*. (3rd ed.). Wiley and Sons, Inc. Available at: http://www.essai.rnu.tn/Ebook/Informatique/The%20Data%20Warehouse%20Toolkit,%203rd%20Edition.pdf.

[14] Patel, A., & Patel, J. M. (2102). Data modeling techniques for data warehouse. *International Journal of Multidisciplinary Research, 2*(2), 240–246.

[15] Farhan, M., S., Marie, M., E., El-Fangary, L., M., & Helmy, Y., K. (2011). An integrated conceptual model for temporal data warehouse security. *Computer and Information Science, 4*(4), 46–57.

[16] Eder, J. & Koncilia, C. (2001). Changes of dimension data in temporal data warehouses. *Proceedings of Third International Conference on Data Warehousing and Knowledge Discovery, Munich, Germany, LNCS, Springer*, 284–293.

[17] Golfarelli, M., Maio, D., & Rizzi, S. (1998). The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems, 7*(2-3), 215–247.

[18] Golfarelli, M. & Rizzi, S. (1998). A methodological framework for data warehouse design. *Proceedings of ACM First International Workshop on Data Warehousing and OLAP, DOLAP, Washington*, 3–9.

[19] Bernardino, J. & Madeira, H. (2019). *Data warehousing and OLAP: Improving query performance using distributed computing*. Available at: https://www.academia.edu/4241918/Data_Warehousing_and_OLAP_Improving_Query_Performance_Using_Distributed_Computing.

[20] Albrecht, J., Gunzel, H., & Lehner, W. (1998). An architecture for distributed OLAP. *International Conference on Parallel and Distributed Processing Techniques and Applications*. Available at: https://www.tib.eu/en/search/id/TIBKAT%3A316251313/Proceedings-of-the-International-Conference-on/.

[21] Comer, D. (1979). The ubiquitous B-tree. *ACM Computing Surveys, 11*(2), 121–137.

This work is licensed under Creative Commons Attribution 4.0 International License.