# An Efficient Approach of Block Nested Loop Algorithm based on Rate of Block Transfer

Deepak Shukla Dr. Deepak Arora Rakesh Kr. Pandey K. K. Agrawal

Department of Computer Science and Engineering,
Amity University
Uttar Pradesh, Lucknow, India.

## ABSTRACT
In this paper, an approach has been proposed that makes the processing of join operation in database systems more efficient. In join operation processing relations that take part in the join process are required to be transferred to the main memory (RAM) from hard disk. In join operation processing when block nested loop algorithm is used to perform join between relations and multiple blocks of the relations that take part in the joining process are transferred from hard disk to main memory than in this case the main memory buffer allotted to the blocks of relation. Using this approach, multiple blocks are transferred for the relations that participates in the join operation processing, instead of transferring blocks one by one for each relation (or multiple blocks for one relation) without worrying about the large and small databases size. When this new approach is applied, the rate of block transfer during join operation processing using block nested loop algorithm get minimizes and join query processing become efficient, without loosing the level of complexity of the previous algorithms of block nested loop join(BNLJ).

## General Terms
Algorithm for join operation processing based on rate of block transfer.

## Keywords
Databases, Query Processing, Block Nested-Loop Join (BNLJ), RAM, and Hard Disk.

## 1 INTRODUCTION
Join operation combines tuples from two relations to produce the join result. Join processing records of the relations are on the whole expensive operations occurred in a database management system [1]. So, all query optimization algorithms firstly deal with joins [2]. In the previous years, many join techniques have been designed for better processing of join operation [3] and [4]. Join operation processing is the most expensive operation in database system and there are many ways proposed in the past for processing join query [5]. In database management systems it has been assumed that all data reside on main memory (RAM). In the past years the join algorithms analyzed has been firstly consisted of reckoning the number of disk pages transferred during the join operations [10].

In this paper the proposed approaches minimizes the number of block transfer during the join operation. When two relations (are placed on disk) are transferred from disk to main memory on the basis of multiple blocks transfer for both of the two relations. During this operation the complexity of the proposed algorithm is also maintained. Relations are stored on the disk in the form of files and these files are distributed on the disk's segments. When any join query is fired the operation performed is to transfer the relation from main memory to disk either one by one tuples are transferred for the relation to be joined or multiple tuples are transferred.

In join operation processing when block nested loop algorithm is used to perform join between relations and multiple blocks of the relations that take part in the joining process are transferred from hard disk to main memory than in this case the main memory buffer allotted to the blocks of relation at outer loop of the join algorithm is not less than that of five blocks and $(Z-5)$ is the maximum space allotted to the relation at inner loop of the join algorithm. Because at the point when relation at outer loop gets less space in main memory than blocks of relation at inner loop shows the worst scenario so, to make it efficient many database systems changes the positions of the relations from outer to inner loop and vice- versa. For this reason, the proposed block nested loop $(Z-5)$ algorithm is the one that gives best results when rate of block transfer is the comparison factor between the join algorithms. It means that rate of block transfer minimizes when block nested loop $(Z-5)$ algorithm is used in joining of two relations.

In this paper (section 2), discussion has been done on the basis of the algorithm previously developed and analysis of effectiveness is done as well. In section 3, an approach has been discussed to minimize the rate of block transfers during join operation processing without losing the level of complexities of previous join algorithms. An algorithm also has been proposed for it. In section 4, tables has been presented that compares the seek time, complexity, rate of block transfer and effectiveness of algorithms previously developed with the proposed algorithm. In section 4, an analysis of expected result is presented and some experimental results are also reported. In the final section, conclusion as well as future work has been presented.

## 2 BACKGROUND
In database management systems, join algorithms are used for the combination of the tuples from the relations for producing join result based on a join condition. There are many different types of Join algorithms [12]. Some times it is necessary to work with multiple relations as they were contains data of same entity. Then a single SQL query can manipulate data from all the relations. Join are used to achieve this. Relations are joining on attributes that have the same data type and width in the relations [17]. There are many join algorithms for joining relations based on join

condition. And, these algorithms scans the relations participated in the join operation to produce the joined relation as a result in the buffer of main memory. [12].

In this paper, an effective approach to join query processing is being proposed. So, for this it is proposed in this paper that how to minimize the rate of block transfers during join operation processing. Algorithms that support the proposed work for evaluating join are [7].

Nested Loops Join (NLJ).

Block Nested Loops Join (BNLJ –one block at a time)

Block Nested Loops Join (BNLJ –multiple block at a time)

## 2.1 Nested Loops Join (NLJ)

In Nested loop join algorithm, pairs of nested for loops are used to perform join operation. Relation 'r' is called outer relation and relation 's' is the inner relation of the join, since the outer loop for relation 'r' contains the inner loop for relation 's'. The algorithm uses the notation $tr \cdot ts$, where $tr$ and $ts$ are tuples; $tr \cdot ts$ denotes the tuples constructed by concatenating the attribute values of tuples $tr$ and $ts$ [15].

In Nested Loop Join, it is very necessary to find out which relation is scanned by the outer loop and which is scanned by the inner loop of the join algorithm as the relations are stored in the form of file on the hard disk. The percentage of records in the relation in database that will be joined with records in the other relation of the same [16].

Let us assume that, number of tuples in relation 'r' = $nr$, number of tuples in relation 's' = $ns$, number of blocks of relation 'r' = $bR$, number of blocks of relation 's' = $bS$, and number of blocks fits in main memory at once = Z.

Algorithm: 1 Nested Loop Join (Tuple-at-a-time) [15]

for each tuple $tr$ in r

{tuples of relation 'r' are scanned one by one.

for each tuple $ts$ in s

{tuples of relation 's' are scanned one by one.

If join condition is true for $(tr, ts)$

add $tr \cdot ts$ to the result.

}

}

Algorithm for join applied on the block which resides on the main memory. This algorithm ensures that how the join performed in main memory. In the scanning of each tuple of relation 'r', it should be clear that the tuples of relation 's' is scanned nr times, resulting $(nr * ns)$ scanning for total tuples during join operation processing. In the scanning of one tuple for relation 'r', $bS$ blocks of relation 's' has been scanned. In the scanning of nr tuples for relation 'r', $(bS * nr)$ scan needed. Total scans are equal to $(nr * bS + bR)$. nr seeks needed to scan relation 'r' and $bR$ seeks needed to scan relation 's' [15]. Total seeks are equal to $(nr + bR)$. Complexity = $O(nr * ns)$, where, nr and ns number of tuples contained in relation 'r' and relation 's' respectively.

## 2.2 Block Nested Loop Join (BNLJ-One Block at A Time)

In Block Nested Loop Join, when relations 'r' and relation 's' has to be joined, the outer loop is for reading the blocks of relation 'r' and inner loop is reading the blocks of relation 's'. If relation 'r' and relation 's' are small enough to fit into the main memory than the join operation is performed more effectively [6].

In Block nested loop join, before performing the join operation the relations to be joined are first placed into the main memory [12]. In Block Nested Loop Join algorithm, the number of disk accesses consists of two operations – one is to read the blocks of relation 'r' and other to access the disk for reading the blocks of relation 's'. [8].

Algorithm: 2 Block Nested Loop Join (Block-at-a-time) [9]

for each block $bR$ of 'r' do

{blocks of relation 'r' are scanned one by one.

for each block $bS$ of s do

{blocks of relation 'r' are scanned one by one.

Compute $bR \bowtie bS$ in memory

}

}

The Block Nested Loop Join algorithm is an advanced algorithm of the nested loop join algorithm which is used for transfer of blocks efficiently rather than transferring the tuples of the participating relations in the join operation. The block nested loop joins algorithm works by reading a block of tuples, from the outer and inner relation [10]. In BNLJ (one block at a time) chunks of each relation is transferred from hard disk to main memory where join operations is performed [9].

Block nested loop join algorithms break the outer relation 'r' into blocks that can fit into the main memory input buffer pages and then scanning of all the inner relation 's' for each block of the outer relation is performed [12]. Key on outer relation, for each block of relation 'r' is scanned, the $bS$ blocks of relation 's' are scanned and for $bR$ blocks of relation 'r', the $(bS * bR)$ times the blocks of relation 's' are scanned. The total block transfer is $((bR * bS) + bR)$, seek Time is $(2 * bR)$ and access complexity is $O(n^4)$ [15].

## 2.3 Block Nested Loop Join $(Z-2)$ (BNLJ-Multiple Block Transfer)

Block Nested Loop Join (Multiple-Block-Transfer) divides memory into two parts. $Zr$ blocks are used for relation 'r' and $Zs$ blocks of main memory are used for relation 's' [13]. If $(Z-2)$ blocks of relation 'r' are transferred to main memory and at $(Z-1)th$ location in the main memory one block of relation 's' is placed then the tuples of block of relation 's' are compared with tuples in the $(Z-2)$ blocks of relation 'r'. After satisfying the join condition, join result is produced as a joined relation.

In Block Nested Loop Join $(Z-2)$, the number of blocks transferred for relation 'r' is $(bR/(Z-2))$ and the number of blocks transferred for relation 's' is $(bR/(Z-2))*bS$.

Algorithm: 3 Block Nested Loop Join $(Z-2)$ [15]

for each block $bR$ $(Z-2)$ of relation 'r' do

{//block from relation 's'

for each block $bS$ of relation 's' do

{// tuples are scanned from $bR$

for each tuple $tr$ in $bR$ do

{//tuples are scanned from $bS$

f or each tuple $ts$ in $bS$

{Test pair $(tr,ts)$ to see if they satisfy the join condition

add $tr \cdot ts$ to the result

}

}

}

}

Block Nested Loop Join $(Z-2)$ [15]:-

Use the largest size that can fit in main memory, considering that space for the inner relation's buffer and the output also.

If memory has Z blocks, read $(Z-2)$ blocks of the outer relation at a time and also read a block of the inner relation to join it with all the $(Z-2)$ blocks of the outer relation.

Total Block transfer = $((bR/(Z-2))*bS)+bR$
Total seeks = $2(bR/(Z-2))$
Complexity = $O(n^4)$

## 2.4 Block Nested Loop Join $(Z-3)$ (BNLJ-Multiple Block Transfer) [14]

It has been assumed that total number of blocks that fits in the main memory is Z, total number of blocks in which relation 'r' fits on hard disk is $bR$, and total number of blocks in which relation 's' fits on disk is $bS$. If $(Z-3)$ blocks of relation 'r' are transferred from disk to main memory and two blocks of relation 's' are transferred from hard disk to main memory in one scan than the comparison is done and after satisfying the join condition the tuples are joined and transferred to output buffer at $Zth$ location of main memory. Then, the total number of block transfers decreases as compared to when transfer of $(Z-2)$ blocks and one block of relation 'r' and relation 's' has been done respectively. If $bR$ is less than Z or $bR$ equals to Z, then number of blocks transfers for relation 'r' is $(bR/(Z-3))$, and number of blocks transfers for relation 's' is $((bR/(Z-3))*bS)/2$. Then, the total block transfers during r join s is the addition of number of block transfers from relation 'r' and number of block transfers from relation 's' which is $(bR/(Z-3))+((bR/(Z-3))*bS)/2$.

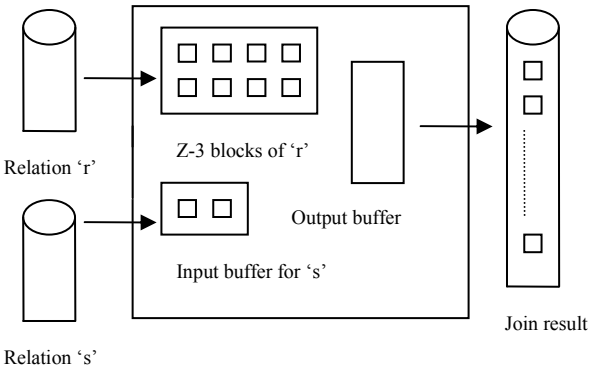


**Fig 1: Transfer of Blocks from relation 'r' and relation 's' to main memory in block nested loop $(Z-3)$ algorithm.**

Let us assume that,
$bR$ = total number of blocks on disk in which relation 'r' fits.
$bS$ = total number of blocks on disk which relation 's' fits.
$xr$ = Points to first tuple of first block of relation 'r' that is transferred to main memory.
$xs$ = Points to first tuple (first) of first block of relation 's' in main memory.
$bpr$ = Block Pointer i.e., it points to the block which is to be accessed.
$bps$ = Block Pointer of blocks of relation 's' in main memory. i.e., it points to the block which is under consideration.
Z = Total number of blocks that can fit in main memory.

Let us assume that, $(Z-3)$ blocks are transferred for inner loop from disk to main memory at $(Z-2)th$ and $(Z-1)th$ location transfer two blocks of relation 's' in main memory. $Zth$ location is left and is used as an output buffer for join result. Assuming relation 'r' is smaller in size in comparison to relation 's'.

Algorithm: 4 Block Nested Loop Join $(Z-3)$

$i = (bR/(Z-3))$
$j = ((bR/(Z-3))*bS)/2$
While $(i \neq 0)$ //blocks of relation 'r' outer loop

{Move $(Z-3)$ blocks of relation 'r' in main memory.

$bpr$ points to first block of relation 'r' in main memory

While $(j \neq 0)$ //blocks of relation 'r' outer loop

{Move two blocks of relation 's' at $(Z-2)th$ and $(Z-1)th$ location of main memory.

$bps$ points to first block of relation 's' in main memory

While ( $xr$ reaches end of block pointed by $bpr$ )

{Check whether all tuples of $bpr$ are exhausted or not.

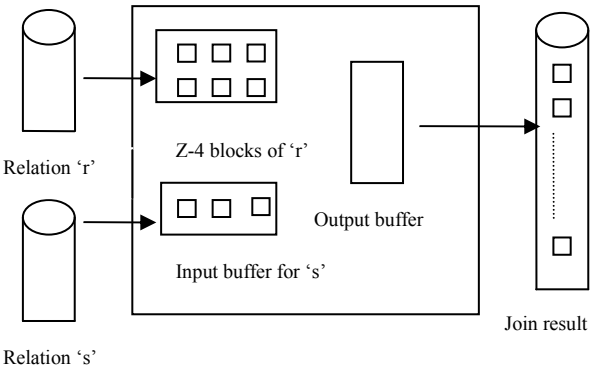


**Fig 2: Transfer of Blocks from relation 'r' and relation's' to main memory in block nested loop $(Z-4)$ algorithm.**

While ( $xs$ reaches end of block pointed by $bps$ )

{ Check whether bps are exhausted or not.

Compare tuples $xr$ and $xs$ and check the join condition, if it satisfies then adds $(xr \cdot xs)$ to output buffer.

$xs++;$

} $xr++;$

} $bps++;$ $j--;$

} $bpr++;$ $i--;$

}

# 3 PROPOSED WORK

## 3.1 Block Nested Loop Join $(Z-4)$ (Author's Proposed Algorithm)

It has been assumed that total number of blocks that fits in the main memory is Z, total number of blocks in which relation 'r' fits on hard disk is $bR$, and total number of blocks in which relation 's' fits on disk is $bS$. If $(Z-4)$ blocks of relation 'r' are transferred from disk to main memory and three blocks of relation 's' are transferred from hard disk to main memory in one scan than the comparison is done and after satisfying the join condition the tuples are joined and transferred to output buffer at $Zth$ location of main memory. Then, the total number of block transfers decreases as compared to when transfer of $(Z-3)$ blocks and two block of relation 'r' and relation 's' has been done respectively. If $bR$ is less than Z or $bR$ equals to Z, then number of blocks transfers for relation 'r' is $(bR/(Z-4))$, and number of blocks transfers for relation 's' is $((bR/(Z-4))*bS)/3$. Then, the total block transfers during r join s is the addition of number of block transfers from relation 'r' and number of block transfers from relation 's' which is $(bR/(Z-4))+((bR/(Z-4))*bS)/3$.

Let us assume that,

$bR$ = total number of blocks on disk in which relation 'r' fits.

$bS$ = total number of blocks on disk which relation 's' fits

$xr$ = Points to first tuple of first block of relation 'r' that is transferred to main memory.

$xs$ = Points to first tuple (first) of first block of relation 's' in main memory.

$bpr$ = Block Pointer i.e., it points to the block which is to be accessed.

$bps$ = Block Pointer of blocks of relation 's' in main memory. i.e., it points to the block which is under consideration.

Z = Total number of blocks that can fit in main memory.

Let us assume that, three blocks are transferred for inner loop from disk to main memory at $(Z-3)th$, $(Z-2)th$ and $(Z-1)th$ location transfer two blocks of relation 's' in main memory. $Zth$ location is left and is used as an output buffer for join result. Assuming relation 'r' is smaller in size in comparison to relation 's'.

Algorithm: 5 Block Nested Loop Join $(Z-4)$ (Author's Proposed Algorithm)

$i=(bR/(Z-4))$

$j=((bR/(Z-4))*bS)/3$

While $(i \neq 0)$ //blocks of relation 'r' outer loop

{Move $(Z-4)$ blocks of relation 'r' in main memory.

$bpr$ points to first block of relation 'r' in main memory

While $(j \neq 0)$ //blocks of relation 'r' outer loop

{Move three blocks of relation 's' at $(Z-3)th$ , $(Z-2)th$ and $(Z-1)th$ location of main memory.

$bps$ points to first block of relation 's' in main memory

While ( $xr$ reaches end of block pointed by $bpr$ )

{Check whether all tuples of $bpr$ are exhausted or not.

While ( $xs$ reaches end of block pointed by bps)

{Check whether bps are exhausted or not

Compare tuples $xr$ and $xs$ and check the join condition, if it satisfies then adds $(xr \cdot xs)$ to output buffer.

$xs++;$

} $xr++;$

} $bps++;$ $j--;$

} $bpr++;$ $i--;$

}

### 3.2 Block Nested Loop Join $(Z-5)$ (Author's Proposed Algorithm)

It has been assumed that total number of blocks that fits in the main memory is Z, total number of blocks in which relation 'r'

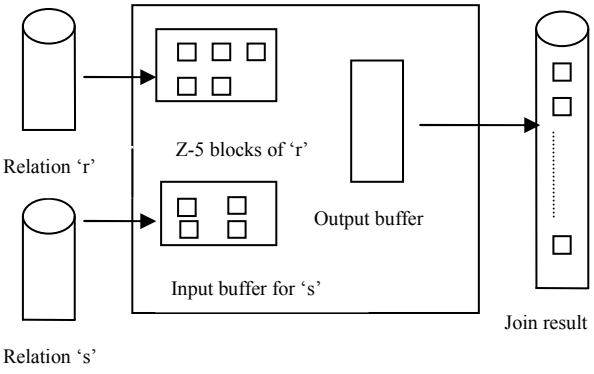


**Fig 3: Transfer of Blocks from relation 'r' and relation's' to main memory in block nested loop $(Z-5)$ algorithm**

fits on hard disk is $bR$, and total number of blocks in which relation 's' fits on disk is $bS$. If $(Z-5)$ blocks of relation 'r' are transferred from disk to main memory and four blocks of relation 's' are transferred from hard disk to main memory in one scan than the comparison is done and after satisfying the join condition the tuples are joined and transferred to output buffer at $Zth$ location of main memory. Then, the total number of block transfers decreases as compared to when transfer of $(Z-4)$ blocks and three block of relation 'r' and relation 's' has been done respectively. If $bR$ is less than Z or $bR$ equals to Z, then number of blocks transfers for relation 'r' is $(bR/(Z-5))$, and number of blocks transfers for relation 's' is $((bR/(Z-5))*bS)/4$. Then, the total block transfers during r join s is the addition of number of block transfers from relation 'r' and number of block transfers from relation 's' which is $(bR/(Z-5))+((bR/(Z-5))*bS)/4$.

Let us assume that,

$bR$ = total number of blocks on disk in which relation 'r' fits.
$bS$ = total number of blocks on disk which relation 's' fits
$xr$ = Points to first tuple of first block of relation 'r' that is transferred to main memory.
$xs$ = Points to first tuple of the first block of relation 's' in main memory.
$bpr$ = Block Pointer i.e., it points to the block to be accessed.
$bps$ = Block Pointer of blocks of relation 's' in main memory. i.e., it points to the block which is under consideration.
Z = Total number of blocks that can fit in main memory.

Algorithm: 6 Block Nested Loop Join $(Z-5)$ (Author's Proposed Algorithm)

$i = (bR/(Z-5))$
$j = ((bR/(Z-5))*bS)/4$

While $(i \neq 0)$ //blocks of relation 'r' outer loop
{Move $(Z-5)$ blocks of relation 'r' in main memory.
$bpr$ points to first block of relation 'r' in main memory
While $(j \neq 0)$ //blocks of relation 'r' outer loop
{Move four blocks of relation 's' at $(Z-4)th$, $(Z-3)th$, $(Z-2)th$ and $(Z-1)th$ location of main memory.
$bps$ points to first block of relation 's' in main memory
While ( $xr$ reaches end of block pointed by $bpr$ )
{Check whether all tuples of $bpr$ are exhausted or not.
While ( $xs$ reaches end of block pointed by bps)
{Check whether $bps$ are exhausted or not
Compare tuples $xr$ and $xs$ and check the join condition, if it satisfies then adds $(xr \cdot xs)$ to output buffer.
$xs++;$
} $xr++;$
$bps++;$ $j--;$
} $bpr++;$ $i--;$
}

## 4 RESULT & DISCUSSION

Complexity of the proposed algorithm that works in transfer of $(Z-4)$ and $(Z-5)$ blocks is not increased but the rate of block transfers and time efficiently decreases. It is shown in the table 1 that the complexities of the proposed algorithms named block nested loop $(Z-4)$ and block nested loop $(Z-5)$ algorithm are not increased in comparison to the previous join algorithms. So, from the complexity point of views the algorithms proposed in this paper are considerably good as they are good when rate of block transfer is the comparing factor between the block nested loop algorithms. It is also shown in the table 1 that form block nested loop $(Z-2)$ to block nested loop $(Z-5)$ the complexity is $O(n\text{^}4)$.

**Table 1. Complexities Comparison**

| Algorithms | Access Complexities |
|---|---|
| NLJ | $O(nr \cdot ns)$ |
| BNLJ(one block at a time) | $O(n\text{^}4)$ |
| BNLJ(Z-2) | $O(n\text{^}4)$ |
| BNLJ(Z-3) | $O(n\text{^}4)$ |
| BNLJ(Z-4) (Author's Proposed) | $O(n\text{^}4)$ |
| BNLJ(Z-5) (Author's Proposed) | $O(n\text{^}4)$ |

This is the table that shows the comparisons of previously developed algorithm with the algorithm that has been proposed in this paper. Seek time for of the proposed algorithms named block nested loop $(Z-4)$ and block nested loop $(Z-5)$ algorithm are calculated as $2*(bR/(Z-4))$ and $2*(bR/(Z-5))$ respectively. Where, $bR$ is the number of blocks of relation 'r' and Z is the size in pages of main memory.

**Table 2. Seek Time Comparison**

| Algorithms | Seek Time |
|---|---|
| NLJ | $bR+bS$ |
| BNLJ(one block at a time) | $2*bR$ |
| BNLJ(Z-2) | $2*(bR/(Z-2))$ |
| BNLJ(Z-3) | $2*(bR/(Z-3))$ |
| BNLJ(Z-4) (Author's Proposed) | $2*(bR/(Z-4))$ |
| BNLJ(Z-5) (Author's Proposed) | $2*(bR/(Z-5))$ |

In Nested Loop Join algorithm, the number of tuples transferred for relation 'r' is $(nr*bS)$ and the number of tuples transferred for relation 's' is $bR$, where nr is the number of tuples in relation 'r'. And, the total number of tuples transferred is presented in the table 3. In Block Nested Loop algorithm for one by one block transfer, the number of blocks transfer for relation 'r' is $bR$ and the number of blocks transferred for relation 's' is $(bR*bS)$. The total numbers of blocks transferred during block nested loop join for one by one transfer of blocks is presented in the table 3. In Block Nested Loop $(Z-2)$ algorithm, the number of blocks transferred for relation 'r' is $(\mathrm{bR}/(Z-2))$ and the number of blocks transferred for relation 's' is $(bR/(Z-2))*bS$. The total number of blocks transferred during block nested loop join $(Z-2)$ is presented in the table 3. In Block Nested Loop $(Z-3)$ algorithm, the number of blocks transferred for relation 'r' is $(bR/(Z-3))$ and number of blocks transferred for relation 's' is $((bR/(Z-3))*bS)/2$. The total number of blocks transferred during block nested loop join $(Z-3)$ is presented in the table 3. Rate of block transfer is minimized for this algorithm by ½ for transfer of blocks of relation 's'. In Block Nested Loop $(Z-4)$ algorithm, the number of blocks transferred for relation 'r' is $(bR/(Z-4))$ and number of blocks transferred for relation 's' is $((bR/(Z-4))*bS)/3$. The total number of blocks transferred during block nested loop join $(Z-4)$ that is proposed in this paper is presented in the table 3. Rate of block transfer is minimized for this algorithm by 1/3 for transfer of blocks of relation 's'.

**Table 3. Rate of Block Transfer**

| Algorithms | No. of blocks transferred (in total) |
|---|---|
| NLJ(tuples) | $(nr*bS+bR)$ |
| BNLJ (one block) | $((bR*bS)+bR)$ |
| BNLJ (Z-2) | $(bR/(Z-2))+((br/(Z-2))*bS)$ |
| BNLJ (Z-3) | $(bR/(Z-3))+((bR/(Z-3))*bS)/2$ |
| BNLJ (Z-4) (Author's Proposed) | $(bR/(Z-4))+((bR/(Z-4))*bS)/3$ |
| BNLJ (Z-5) (Author's Proposed) | $(bR/(Z-5))+((bR/(Z-5))*bS)/4$ |

In Block Nested Loop $(Z-5)$ algorithm, the number of blocks transferred for relation 'r' is $(bR/(Z-5))$ and number of blocks transferred for relation 's' is $((bR/(Z-5))*bS)/4$. The total number of blocks transferred during block nested loop join $(Z-5)$ that is proposed in this paper is presented in the table 3. Rate of block transfer is minimized for this algorithm by 1/4 for transfer of blocks of relation 's'.

# 5 CONCLUSION AND FUTURE WORK

In this paper an approach has been presented that minimizes the rate of block transfer during join operation processing. In this paper the proposed approach extended the algorithm of block nested loop join that is an effective approach to the join operation processing. However, changing the number of block transfers from disk to main memory is sufficient to minimize the rate of block transfers during join relation 'r' and relation 's'. Some of the main ideas of this paper are: (1) Avoid the fetching of one block transfer for relation 's', (2) For relation at inner loop, the rate of block transfer is decreased by 1/5, (3) The complexity level does not increase, and (4) Minimizes the number of scans. Comparison of the proposed algorithm on the basis of access complexity; seek time and the rate of block transfer has been proposed. A number of issues are there as the future work for this paper, it includes the implementation of the proposed algorithm and previous ones also and comparing the test results of the entire join algorithms for finding the best one on the basis of different factors as comparing factors like seek time, cost and join processing time etc.

# 6 ACKNOWLEDGMENTS

The authors are very thankful to their respected Mr. Aseem Chauhan, Chairman, Amity University, Lucknow, Maj. Gen. K.K. Ohri, AVSM (Retd.), Director General, Amity University, Lucknow, India, for providing excellent computation facilities in the University campus. Authors also pay their regards to Prof. S.T.H. Abidi, Director and Brig. U.K. Chopra, Deputy Director, Amity School of Engineering & Technology, Amity University, and Lucknow for giving their moral support and help to carry out this research work.

# 7 REFERENCES

[1] Noh, S.Y. and S.K. Gadia, 2008. Benchmarking temporal database models with interval based and temporal element-based time-stamping. Journal of System software, 81:1931-1943.

[2] Sinha, M. and S.V. Chande, 2010. Query optimization using genetic algorithms. Research Journal of Information Technology, 2:139-144.

[3] Gracfe, G., 1993, Query evaluation techniques for large database. ACM Computer Surveys, 25:73-170.

[4] Soo, M.D., R.T.Snodyrass and C.S. Jensen, 1994. Efficient evaluation of the valid – time natural join. Proceedings of the 10th International conference on data engineering, Feb. 14-18, Washington, DC, USA, PP: 282-292.

[5] Masaya NAKAYAMA, Masaru KITSUREGAWA and Mikio TAKAGI, 1998. Hash Partitioned Join Method Using Dynamic Destaging Strategy. Proceeding of the 14th VLDB Conference Los Angeles, California.

[6] Robert B. Hagmann, 1986. An Observation on Database Buffering Performance Metrices. Proceedings of the 12th International conference on very large Databases, Kyoto.

[7] M. H. Saheb, Oct. 2010. Efficient Algorithm for overlap – join,. Information Technology Journal, 10:201206.

[8] David J. Dewitt, Jeffrey. F. Naughton and Donovan A. Schneider, September, 1991. An Evaluation of Non-Equijoin Algorithms. Proceedings of the 17th International Conference on Very Large Databases, Barcelona.

[9] Philip W. Frey, Romulo Gonalves, Martin Kersten and Jens Teubner, 2009. Cyclo Join: A Join that Spins without Getting Dizzy. In the proceedings of ACM.

[10] Evan P. Haris and Kotagiri Ramamohanarao, 1996. Join Algorithm Costs Revisited Technical Report 93/5.

[11] Laura M. Haas, Michael J. Carey and Miron Livny, 1993. SEEKing the Truth about Ad Hoc Join Costs, May.Technical Report #1148.

[12] Seo-young Noh and Shashi K. Gadia, 2005. Efficient Self Join Algorithm in Interval-based Temporal Data Models. Technical Report, Department of Computer Science, Iowa State University, Ames, Iowa, USA. http://archives.cs.iastate.edu/documents/disk0/00/00/03/86/index.html.

[13] Michael L. Rupley, Jr. 2008. Introduction to Query Processing and Optimization. http://www.cs.iusb.edu/ technical_repots / TR- 20080105-1.pdf

[14] Deepak Shukla, Rakesh Kumar Pandey, Deepak Arora and Ajai Kumar Yadav, 2011. An effective approach for join operation processing. 2nd National conference on Global Trends and Innovations in Computer Application and Informatics, April 9-10th, Meerut, India.

[15] Abraham Silberschatz, Henry F. Korth and S.Sudarshan, 2006. Database System Concepts, 5th Edition, Mc Graw Hill Higher Education. [ISBN: 007-124476-X].

[16] Elmasri. R. and S.B. Navathe, 2009. Fundamentals of Database Systems. 5th Edition, Pearson Education. [ISBN: 978-81-317-1625-0].

[17] Ivan Bayross, 2008. SQL, PL/SQL The Programming Language of Oracle. 3rd Revised Edition, BPB Publications. [ISBN: 81-7656-964-X].