Int. Journal of Computing and Optimization, Vol. 1, 2014, no. 1, 17 - 45
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ijco.2014.445

Processing of Heterogeneous Big Data in an Atrain Distributed System (ADS) Using the Heterogeneous Data Structure r-Atrain

Ranjit Biswas

Department of Computer Science
Jamia Hamdard University
Hamdard Nagar, New Delhi–110062, India

Abstract

The present day world dealing with big data (expanding very fast in 4Vs : Volume, Varity, Velocity and Veracity) needs **N**ew advanced kind of logical and physical storage structures, **N**ew advanced kind of heterogeneous data structures, **N**ew mathematical theories and **N**ew models : all these four together we call by 4Ns. For processing of the expanding big data in 4Vs, neither any existing data structure nor any existing pattern of distributed system is sufficient or appropriate. For a success, there is no other way but to develop a 'new data structure' and a 'new type of distributed system' having a tremendous extent of mutual compatibility between them, with a facility of unlimited scalability. With these views in mind, we design a special type of distributed system called by 'Atrain Distributed System' (ADS) which is very suitable for processing big data using the heterogeneous data structures r-atrain or the homogeneous data structure r-train. A simple 'Atrain Distributed System' is called an uni-tier ADS. The 'Multi-tier Atrain Distributed System' is an extension of the uni-tier ADS. The ADS is scalable upto any extent as many times as required. Two new type of network topologies are defined for ADS called by 'multi-horse cart' topology and 'cycle' topology which can support increasing volume of big data. Where r-atrain and r-train data structures are introduced for the processing of big data, the data structures 'heterogeneous data structure MA' and 'homogeneous data structure MT' are introduced for the processing of big data including <u>temporal</u> big data too.

Both MA and MT can be well implemented in multi-tier ADS. We define cyclic train and cyclic atrain, and then doubly linked train/atrain. A method is proposed on how to implement Solid Matrices, n-dimensional arrays, n-dimensional larrays etc. in a computer memory using the data structures MT and MA. The combination of r-atrain (r-train) with ADS and the combination of MA (MT) with multitier ADS can play a major role in a new direction to the present data-dependent giant galaxies of organizations, institutions and individuals to process any big data irrespective of the influence of 4Vs.

Mathematics Subject Classifications: 68P05, 68M14, 11C20, 15B36

Keywords: train, atrain, cyclic train/atrain, doubly linked train/atrain, MT, MA, hematrix, helatrix, pilot computer (PC), distributed computer (DC), multi-horse cart topology, cycle topology, atrain distributed system (ADS), twin address, doubly twin address, uni-tier ADS, multi-tier ADS

1 Introduction

The present universe of big data [3,4,6,7,13] is expanding very fast in 4Vs **:** Volume, Varity, Velocity and Veracity, and also in many more directions. How to deal with big data, how to process big data in an efficient way within limited resources, etc. are of major concern to the computer scientists now-a-days. In particular, the 'Velocity' at which the big data have been expanding (or, the 4Vs in which big data have been expanding very fast in the present day world) does not have a one-to-one matching with the 'Velocity' at which the new hardware or new software or new mathematical theories or new models are being developed by the scientists. We designate the following two sets by 4V-set and 4N-set [11] :
(i) 4V-set = { Volume, Varity, Velocity and Veracity}, and (ii) 4N-set = {New Theories, New Hardware, New Software, New Models}. It is obvious that big data can be efficiently processed by a faster development of the 4N-set only. If 4V-set continues its dominance over 4N-set with respect to time, then it will be difficult to the world to think of "BIG DATA : A Revolution That Will Transform How We Live, Work, and Think" [13].
This paper is organized to discuss four topics**:** (i) Development of a new 'Theory of Solid Matrices/Latrices'; (ii) Design of two new type of network topologies called by 'multi-horse cart' topology and 'cycle' topology; (iii) Design of a suitable and scalable distributed System with these two new types of network topologies for processing heterogeneous or homogeneous big data using the heterogeneous data structure r-atrain [8,9] and the homogeneous data structure r-train [8,9]; (iv) Design of a new type of distributed System for processing temporal big data using the heterogeneous data structure MA and the homogeneous data structure MT. We start with theory and methods for homogeneous data and then extend the same for heterogeneous data. We present

implementation strategy by suitable examples.

2 Homogeneous Data Structures 'MT' for Solid Matrix/Latrix

A solid matrix (n-SM) [11] is an n-dimensional logical hyper-matrix where n > 2 and the elements are objects from the region RR [10]. We say that it has n number of hyper layers. However, a solid matrix is a mathematical object and should not be confused with the data structure 'n-dimensional array' in computer science. For details about the n-dimensional array (multi-dimensional array) and its MATLAB implementation, one could see any good MATLAB book. The notion of Larray is introduced in [8,9]. A latrix [11] is a rectangular array of numbers (or, objects from the region RR [10] and ε elements [8,9] where the datatype of the ε elements depends upon the concerned residence larray. A solid latrix (n-SL) [11] is an n-dimensional hyper-latrix where n > 2.
The solid latrix/matrix [11] is useful if the big data is temporal big data. Otherwise, there is no need to choose for solid latrix/matrix. Because, a latrix/matrix can be scalable upto any extent by increasing the number of rows and/or the number of columns to store big data (Needless to mention that not all big data can be stored in the latrix/matrix model).
If there is no consideration of time stamp upon the data, the logical storage structure '2-D latrix' (2-D matrix as a particular case) is sufficient to store homogeneous big data as the number of rows/columns in a 2-D latrix (matrix) can be scalable upto any big extent. Our objective is to propose an appropriate data structure and a compatible distributed system to deal with big data if stored in a 2-D latrix (2-D matrix) of big order. In this section we propose a dynamic homogeneous data structure MT to deal with big data of homogeneous datatype which can be logically stored in a SL/SM. MT is the abbreviation for 'Multi Trains', as it is an extension of the homogeneous data structure 'Train' proposed by Biswas [8,9]. In the homogeneous data structure train, there are logically two layers **:** the pilot is the upper layer and the coaches are in the lower/inner layer. We extend the notion of Train by incorporating nil or one or more number of intermediate layers between the pilot (upper layer) and linked-coaches (lower layer) to develop a new homogeneous data structure 'MT'. Here MT is the abbreviation for 'Multi Trains'. The intermediate layers are usually Trains, but could be pilots, linked-coaches, or larrays [8,9] too. Type of the various layers, according to the construction-needs for the problems under study, are decided by the developers on behalf of the organization concerned. Thus train may be regarded as a special case of MT, where there is(are) no intermediate layer(s) between the upper layer and the lower layer. The total number of layers is called the height, and then height(Train) = 2, and height(MT) ≥ 2.

2.1.1 Implementation of a 3-SM (3-SL)

About details of implementation of a r-Train in a 8086 memory, one could see

[8,9] as a pre-requisite. For implementing a 3-SM (3-SL) of homogeneous data, we use MT of height 3 only. However, in general, for implementation of a higher dimensional n-SM (n-SL) of homogeneous data, we use MT of height n. Consider a 3-SM/SL S of size m×n×h given by S = < M_1, M_2, M_3, …, M_h > of homogeneous data. To implement this 3-SM (3-SL) S in computer memory we need actually one chief Pilot of h number of independent Trains (one for each layer of 3-SM or 3-SL), where each Train is having its own pilot and contains m number of linked coaches. All the h number of trains are independent, but for every train all its coaches are linked/shunted. Surely, we need to consider a MT M with height = 3, i.e. three layers in total. The meaning of the term 'layer' used in SM and also here in MT are to be carefully differentiated. The implementation method follows bottom-to-upward approach as shown below **:-**

Lower Layer L_1 of the MT M :
It is the chief pilot P = < M_1, M_2, M_3, ……, M_h > of the MT M. The 'START' of the MT M points at the chief pilot P. This chief pilot P is nothing but a larray [8,9] of h number of elements M_i. The element M_i is the address of the ith layer of the 3-SM, which is the 'START' S_i of the ith Train T_i in the MT M.

Middle Layer L_2 of the MT M :
It is the larray [8,9] of h number of pilots corresponding to h number of independent Trains (i.e. h number of independent r-Trains where r = n for the present case), given by < T_1, T_2, T_3, ……, T_h > where each T_i corresponds to m number of linked/shunted coaches given by :

$$T_i = < C^i_1, C^i_2, C^i_3, \ldots\ldots, C^i_m >, \quad i = 1, 2, 3, \ldots\ldots, h.$$

At the time of implementation, one has to take care of the 'status' of each coach.

Upper Layer L_3 of the MT M :
In this layer, corresponding to each i there m number of coaches, and consequently there are in total mh number of coaches C^i_j (i = 1, 2, 3, ……, h and j = 1, 2, 3, ……, m). For a given n-train T_i, each coach C^i_j (for j = 1, 2, 3, …, m) has n number of passengers [12], together with one more (the last one) which is one of the following :

(i) an address to the next coach if "$j < m$", or
(ii) address to the first coach of immediate higher layer if "$i < h$ and $j = m$", or
(iii) an invalid address **X** if "$i = h$ and $j = m$".

However, if the data are not all homogeneous, the developer has to go for using the heterogeneous data structure MA instead of MT, (as mentioned in details in Section-6 in this paper). Scalability is an open option to the developer, by increasing the number of middle layers.

2.1.2 Example

For the sake of simple presentation here we ignore big size 3-SM, but consider a small size 3-SM S = < M_1, M_2> of size 3×7×2 of homogeneous data, given by

$$M_1 = \begin{array}{|c|c|c|c|c|c|c|}\hline 2 & 9 & 5 & 8 & 7 & 6 & 6 \\ \hline 4 & 6 & 3 & 5 & 4 & 1 & 8 \\ \hline 9 & 4 & 0 & 4 & 6 & 2 & 0 \\ \hline \end{array} \quad \text{and} \quad M_2 = \begin{array}{|c|c|c|c|c|c|c|}\hline 5 & 8 & 2 & 4 & 0 & 8 & 1 \\ \hline 3 & 8 & 1 & 7 & 9 & 3 & 0 \\ \hline 8 & 6 & 2 & 7 & 1 & 8 & 5 \\ \hline \end{array}$$

Fig. 1. Two layers of a SM S of height 2

For implementing this 3-SM S, we need two 7-Trains T_1 and T_2 where each 7-Train will have three coaches, as below :-

$$T_1 = < C^1{}_1, C^1{}_2, C^1{}_3 > \quad \text{and} \quad T_2 = < C^2{}_1, C^2{}_2, C^2{}_3 >.$$

It is clear that status [8,9] of each of these six coaches is 0, as there is no ε element in this 3-SM. However, it is obvious that for 3-SL, status [8,9] of few coaches could be other than 0.

Table. 1. Layer-1 (bottom layer) latrix L_1 of a 3-SL

42	18	------- --	49
ε	25	------- --	08
49	ε	------- --	ε
02	00	------- --	05
ε	ε	------- --	50

Suppose that the 'START' of the 3-SM S is the address 1A12h. Also suppose that the 7-train T_1 is stored in 8086 memory at the address E74Bh and the 7-train T_2 is stored at address D310h. Then the following will be incorporated in the MT :-

Lower Layer L_1 of the MT M :

The START M will point to the chief Pilot P = $<M_1,M_2>$ = < E74Bh, D310h>.
Now, suppose that the address of the coach $C^1{}_1$ is 5008h, the address of the coach $C^1{}_2$ is A210h, and the address of the coach $C^1{}_3$ is 00AFh. Also suppose that the address of the coach $C^2{}_1$ is CA76h, the address of the coach $C^2{}_2$ is CC80h, and the address of the coach $C^2{}_3$ is BEBAh. Then the following will be incorporated in the MT :-

Middle Layer L_2 of the MT M :

It is the larray $< T_1, T_2 >$ of two pilots T_1 and T_2 which are the two 7-Trains given by

T_1 = < 5008h, A210h, 00AFh> and T_2 = < CA76h, CC80h, BEBAh >.

From the chief pilot, one can visit directly to any of these two 7-Trains. M_1 points at the 7-Train T_1 and M_2 points at the 7-Train T_2.

Upper Layer L_3 of the MT M :

In this layer, corresponding to each 7-Train T_1 and T_2, there 3 number of coaches, and consequently there are in total 3.2 = 6 number of coaches ($C^1{}_1$, $C^1{}_2$, $C^1{}_3$) and ($C^2{}_1$, $C^2{}_2$, $C^2{}_3$). Each coach $C^i{}_j$ has 7 number of passengers, together with one more (the last one) which is an address to the next coach if $j < 3$, address to the

first coach of immediate higher layer if "i = 1 and j = 3", but to an invalid address for "i = 2 and j = 3"). The status of each coach in this example is 0. From the 7-Train T_1, one can visit directly any of its coaches $C^1{}_1$, $C^1{}_2$ and $C^1{}_3$; and similarly from the 7-Train T_2 one can visit directly any of its coaches $C^2{}_1$, $C^2{}_2$ and $C^2{}_3$.
The Fig.2 below shows the implementation of the data structure MT and Table-2 shows how the 3-SM S is stored in 8086 Memory. In this example we consider a SM of small data for which the r-Trains with r = 7 have been used. During the implementation with the data structure MT, one can use as many Trains as required according to the size of big data. But for any MT the top most layer shall always consist of coaches only, not of any Train. In this case, for storing every coach the 'GETNODE' will always provide sixteen number of free consecutive bytes from the memory. In each of such nodes, the first fourteen bytes contain the information and the last two bytes contain an address as explained earlier. However, T_1 and T_2 being larrays will require six bytes each.

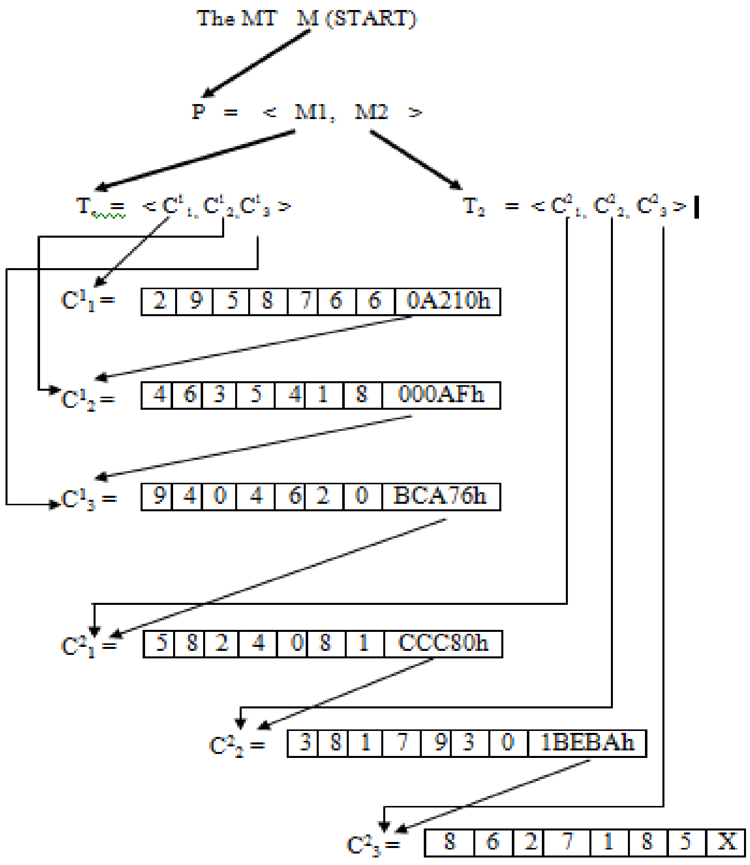
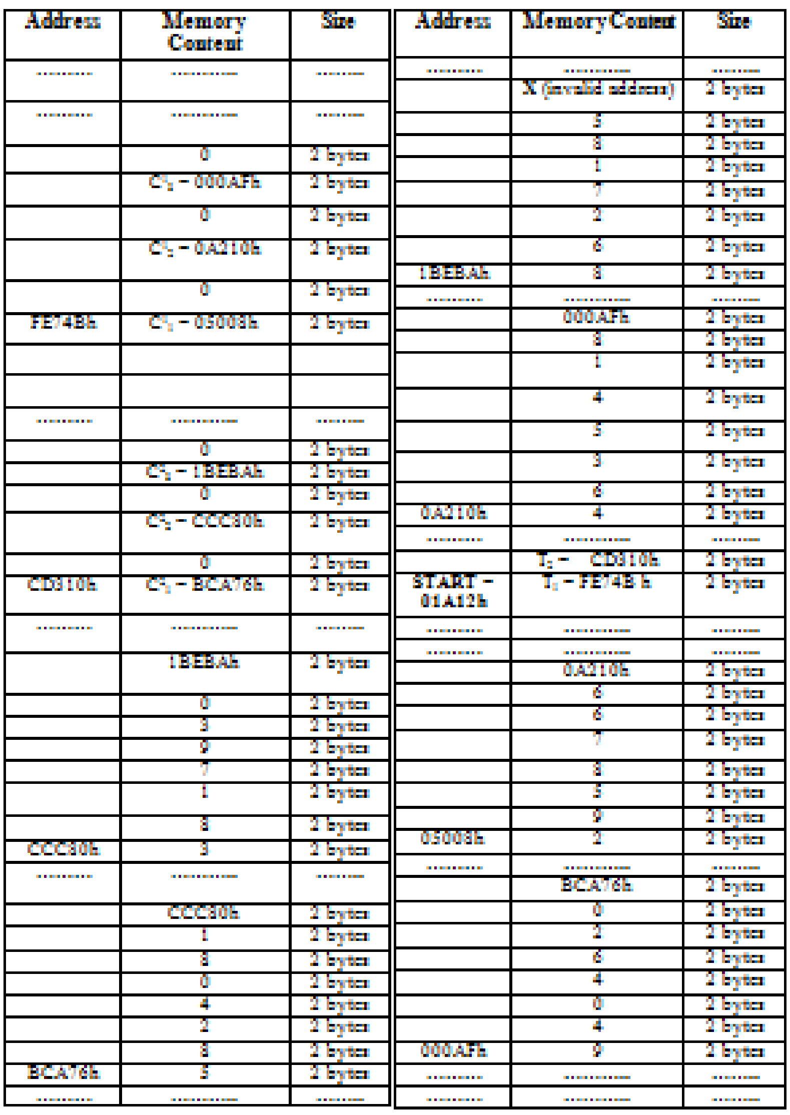

The MT M (START)
P = < M1, M2 >
T1 = < C11, C12, C13 >
T2 = < C21, C22, C23 >
C11 = 2 9 5 8 7 6 6 0A210h
C12 = 4 6 3 5 4 1 8 000AFh
C13 = 9 4 0 4 6 2 0 BCA76h
C21 = 5 8 2 4 0 8 1 CCC80h
C22 = 3 8 1 7 9 3 0 1BEBAh
C23 = 8 6 2 7 1 8 5 X

Fig. 2. Implementation of the data structure MT for the 3-SM S of height 2

The Table.2 shows how this 3-SM (3-SL) S is stored in 8086 Memory starting from START = 1A12h. In many real cases, big data can be viewed as an n-SM or an n-SL. For implementation of an n-SM (n-SL) S of size $m_1 \times m_2 \times \ldots\ldots \times m_n$ of homogeneous data, we need to use a MT of height n as shown below :

Upper Layer L_n of the MT M :
Middle Layer L_{n-1} of the MT M :
Middle Layer L_{n-2} of the MT M :
………………………………..
………………………………..
Middle Layer L_2 of the MT M :
Lower Layer L_1 of the MT M :

Table. 2. A 3-SM (3-SL) in 8086 Memory



Address	Memory Content	Size
………	…………	………
………	…………	………
	0	2 bytes
	C'_1 – 000AFh	2 bytes
	0	2 bytes
	C'_2 – 0A210h	2 bytes
	0	2 bytes
FE74Bh	C'_1 – 05008h	2 bytes
………	…………	………
	0	2 bytes
	C'_2 – 1BEBAh	2 bytes
	0	2 bytes
	C'_2 – CCC80h	2 bytes
	0	2 bytes
CD310h	C'_1 – BCA76h	2 bytes
………	…………	………
	1BEBAh	2 bytes
	0	2 bytes
	3	2 bytes
	9	2 bytes
	7	2 bytes
	1	2 bytes
	8	2 bytes
CCC80h	3	2 bytes
………	…………	………
	CCC80h	2 bytes
	1	2 bytes
	8	2 bytes
	0	2 bytes
	4	2 bytes
	2	2 bytes
	8	2 bytes
BCA76h	5	2 bytes
………	…………	………

Address	Memory Content	Size
………	…………	………
	X (invalid address)	2 bytes
	5	2 bytes
	8	2 bytes
	1	2 bytes
	7	2 bytes
	2	2 bytes
	6	2 bytes
1BEBAh	8	2 bytes
………	…………	………
	000AFh	2 bytes
	8	2 bytes
	1	2 bytes
	4	2 bytes
	5	2 bytes
	3	2 bytes
	6	2 bytes
0A210h	4	2 bytes
………	…………	………
	T_2 – CD310h	2 bytes
START – 01A12h	T_1 – FE74B h	2 bytes
………	…………	………
………	…………	………
	0A210h	2 bytes
	6	2 bytes
	6	2 bytes
	7	2 bytes
	8	2 bytes
	5	2 bytes
	9	2 bytes
05008h	2	2 bytes
………	…………	………
	BCA76h	2 bytes
	0	2 bytes
	2	2 bytes
	6	2 bytes
	4	2 bytes
	0	2 bytes
	4	2 bytes
000AFh	9	2 bytes
………	…………	………
………	…………	………

3 Hematrix & Helatrix : Storage Model for Big Data of Heterogeneous Datatype

A **Hematrix (HM)** is a rectangular logical array of objects of heterogeneous data types where the data are heterogeneous in different rows, but identical inside every row. Thus every row itself contains homogeneous data, but the property of heterogeneity incorporated in the different rows. Thus a hematrix H may have objects like **:** image, DOC file, PDF file, integer number, string of characters, etc. in different rows, but same type of objects inside a row. In the logical structure HM, for any given row all the cells will require equal amount of space in memory for storing their respective contents, but cells of different rows will require different amount of space in memory.

A **Helatrix (HL)** is similar to a hematrix, but it may contain ε elements. As a trivial case every hematrix is a helatrix but the converse is not true. An Example of a Helatrix H of order 5×7 is given below as a bottom layer of a 3-SHL :-

Table. 3. Layer-1 (bottom layer) Helatrix H_1 of a 3-SHL

h_{11}	h_{12}	-------- --	h_{17}
h_{21}	h_{22}	-------- --	h_{27}
h_{31}	h_{32}	-------- --	h_{37}
h_{41}	h_{42}	-------- --	h_{47}
h_{51}	h_{52}	-------- --	h_{57}

In the helatrix H above, h_{1i} are files or ε elements each of size close to (but less than) 1 MB, h_{2i} are integers or ε elements, h_{3i} are files or ε elements each of size close to (but less than) 25 MB, h_{4i} are strings or ε elements each of maximum 50 characters, h_{5i} are files or ε elements each of size close to (but less than) 10 MB, where i = 1, 2, 3….., 7. It may be seen in [8,9] that the ε elements introduced in such a way that they do not have any standard datatype but they occupy space according to the datatype of the concerned larrays [8,9]. A **solid hematrix (SHM)** is an n-dimensional hyper-hematrix where n > 2 and the elements are objects of heterogeneous data types, none being ε elements [8,9]. We say that it has n number of hyper layers.

A **solid helatrix (SHL)** is an n-dimensional logical hyper-helatrix where n > 2. The mathematical models HM, HL, SHM, SHL can easily support big data of heterogeneous datatype because of the fact that they are scalable upto any extent by extending the number of rows or columns or height, according to our requirement. The Solid Hematrix/Helatrix is useful if the big data is a temporal big data. Otherwise, there is no need to choose for solid Hematrix or solid Helatrix.

The logical storage structure ‘2-D Helatrix’ (2-D Hematrix as a particular case) is sufficient to store heterogeneous big data as the number of rows/columns in a 2-D helatrix can be scalable upto any big extent. Our objective is to propose an appropriate data structure to deal with big data if stored in a 2-D helatrix (2-D hematrix) of big order. Consequently, instead of MT or MA which are useful for temporal big data, the heterogeneous data structure r-Atrain (r-train for homogeneous big data) will be the appropriate data structure in a distributed system of appropriate architecture.

4 Atrain Distributed System (ADS) for Big Data

In this section we introduce a new type of distributed system for big data called by ‘Atrain Distributed System’ (ADS) with new type of network topologies called by ‘Multi-horse Cart Topology’ and ‘Cycle Topology’ which are explained below.

4.1 Atrain Distributed System (ADS)

A distributed system consists of a collection of autonomous computers (may be at different locations) connected through a network and distribution middleware which enables all the computers to work by sharing the resources of the system, so that users perceive the system as a single, integrated computing facility.
For the purpose of storing big data, we use a particular type of distributed system in which there is a **Pilot Computer** ‘PC’ connected to m number of computers called by **Distributed Computers** DC-1, DC-2, DC-3, …., DC-m**.** Additional connection of computers is allowed only with the distributed computers but sequentially at the end so as to be identified by DC-(m+1), DC-(m+2)…. etc., but not with the Pilot Computer PC which is one and only one with unique identity. From PC to every DC, there is a connectivity. But from DC-i to DC-j where $j = i+1$ and $j \neq m$, all such connections are either <u>unidirectional</u> (forward) or <u>bidirectional</u> (forward and backward both), but not both in one distributed system. Besides that, the developer (of the concerned organization) may choose for a connection from DC-m to DC-1 to make it <u>circular</u> for unidirectional, and both DC-m to DC-1 with vice-versa for bidirectional to make <u>circular</u>. For non-circular system, the last DC may be connected with an invalid address if it is unidirectional or in addition to that the first DC may have an invalid backward address for bidirectional system). Such type of distributed system is called an ‘**Atrain Distributed System’ (ADS)**. The name of this type of distributed system is called so because of the fact that it can support the powerful heterogeneous data structure r-atrain to process big data with any challenge from 4Vs.

4.2 ‘Multi-horse Cart’ Topology and ‘Cycle Topology’ of Network

An atrain distributed system (ADS) may look apparently to be in the network topology tree where the Pilot Computer is the root node (parent node), and distributed computers are the children nodes sequentially (horizontally) connected. But, as per definition of various types of network topologies [1], it is neither a tree topology nor a bus/ring/star/mesh/hybrid topology. If the last DC be connected with an invalid address (not to the DC-1), then the topology of the network is called by **'Multi-horse Cart'** Topology, as shown in Fig.3 and Fig.4 below. But if the last DC be connected to the DC-1 making it circular, then the topology of the network is called by **"Cycle topology",** because it looks like a ring (wheel) of a riding pedal cycle connected to the centre PC by spokes, as shown in Fig.5 and Fig.6 below (however, if the circular connection from DC-m to DC-1 is not incorporated, it will not be a cycle topology). But whatever be the topology, a DC can communicate with any other DC either directly or via DCs or via PC. In this sense, a multi-horse cart topology is also capable of providing the service like a cycle topology.

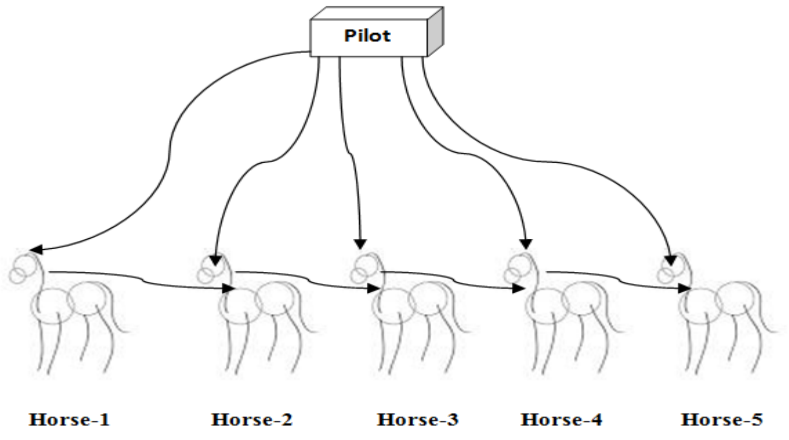


Fig. 3. A Multi-horse Cart Topology of Network (with twin address **e**)

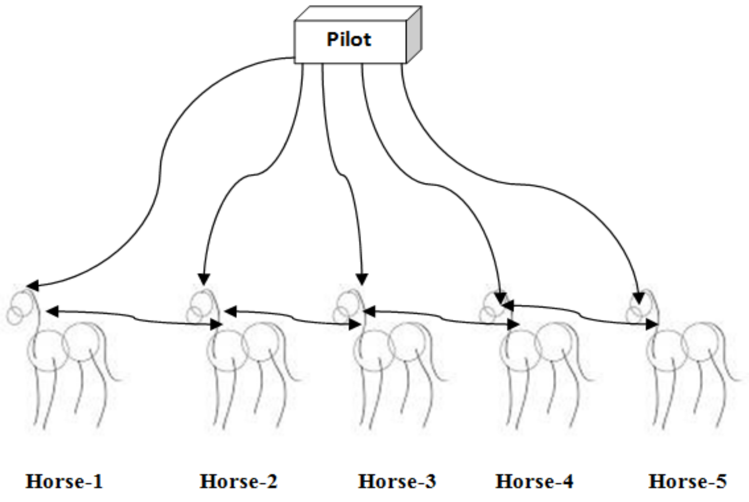


Fig. 4. A Multi-horse Cart Topology of Network (with doubly twin address **e**)

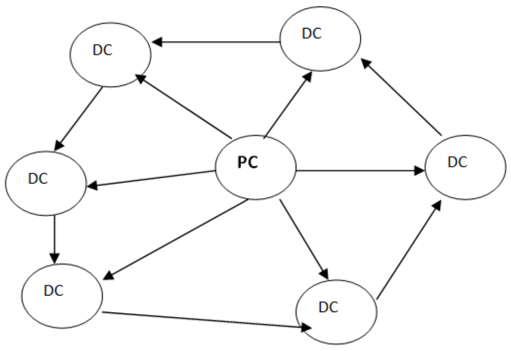


Fig. 5. A Cycle Topology of Network (with twin address **e**)

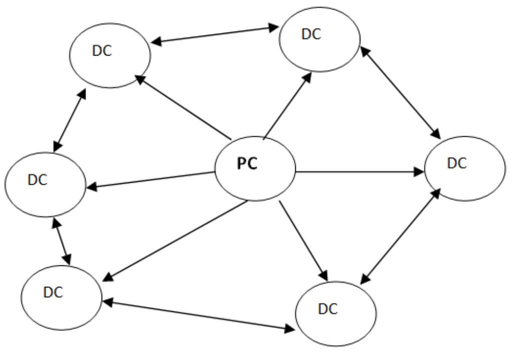


Fig. 6. A Cycle Topology of Network (with doubly twin address **e**)

This type of Atrain Distributed System (ADS) is in fact an **uni-tier ADS** as shown in Fig.7. below. However, Fig.6 shows a distributed system which is not an ADS, because the corresponding topology is not a cycle topology.

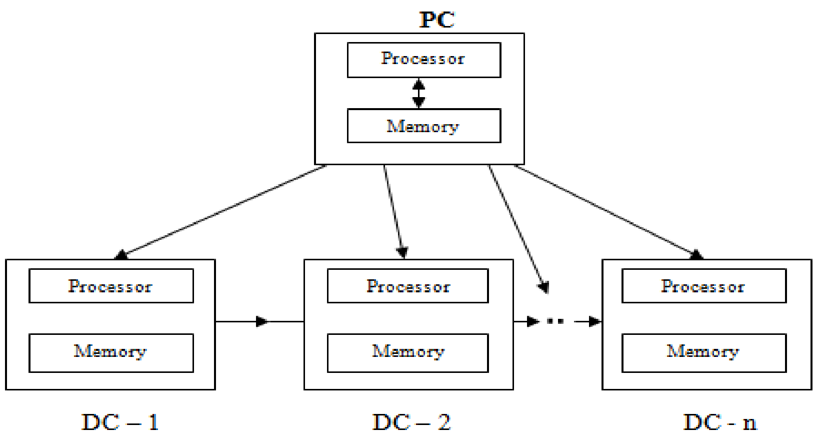


Fig. 7. An Atrain Distributed System (i.e. unitier ADS) in Multi-horse Cart Topology

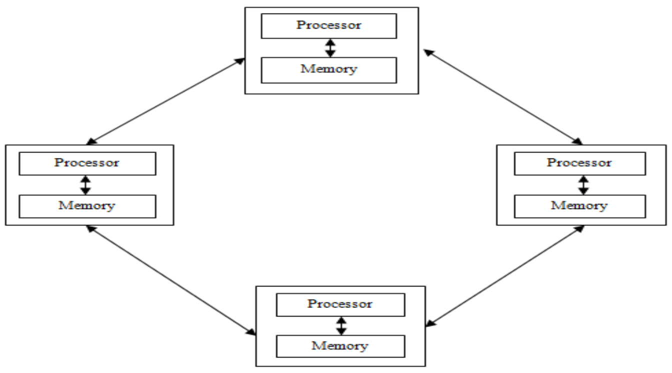


Fig. 8. A Distributed System which is not an ADS

A **multi-tier ADS** can be defined recursively, where every DC can also act as a PC having its own DCs by branching. Thus a multi-tier is a tree having at least one subtree which is too a multi-tier (or at least an uni-tier). The Fig.9 and Fig.10 below show multi-tier ADS of 2-tier and 3-tier respectively in multi-horse topology of network.

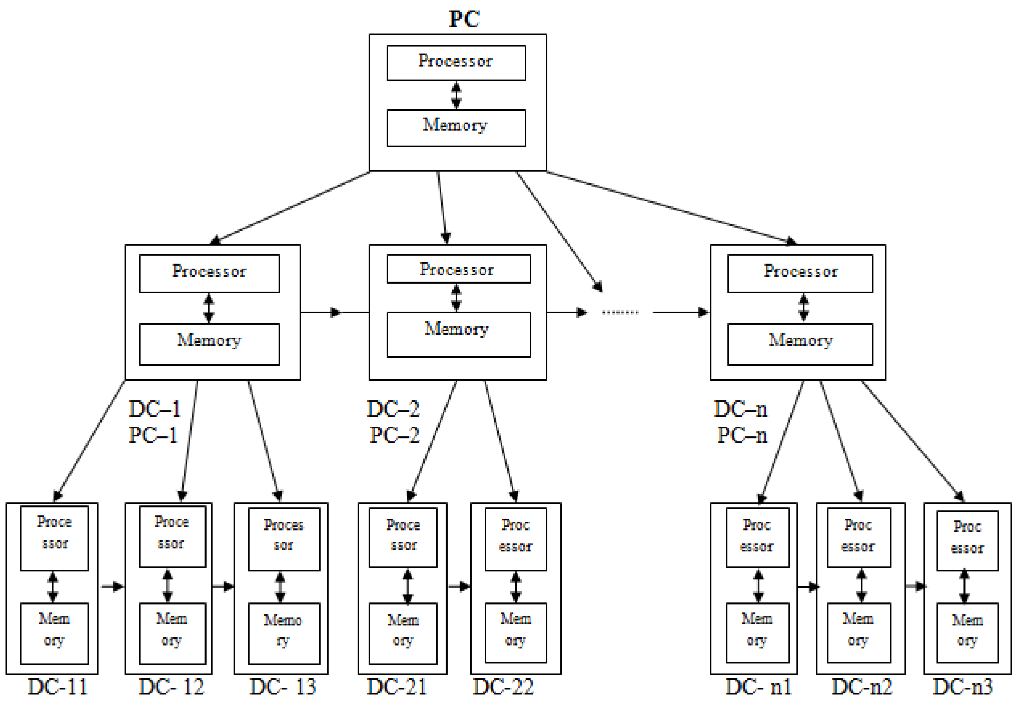


Fig. 9. A multi-tier ADS (2-tier) in Multi-horse Cart Topology

The distributed system ADS is designed for the data structures r-atrain and r-train for processing big data, whereas the 'multi-tier ADS' is designed for the data structures MA and MT (not for r-atrain or r-train) for processing too big data. However, all the data structures r-atrain, r-train, MA and MT can be well useful in an autonomous computer too, for small/large volume of data or sometimes for big data depending upon the code of the datatype (see CD-Table in Table-4 in subsection 5.1 below). While implementing ADS, if the last DC is connected to an invalid address then the ADS is in multi-horse topology, otherwise if it is connected to the first DC of its siblings then the ADS is in cycle topology. The link address **e** could be twin address or double twin address for both the topologies.

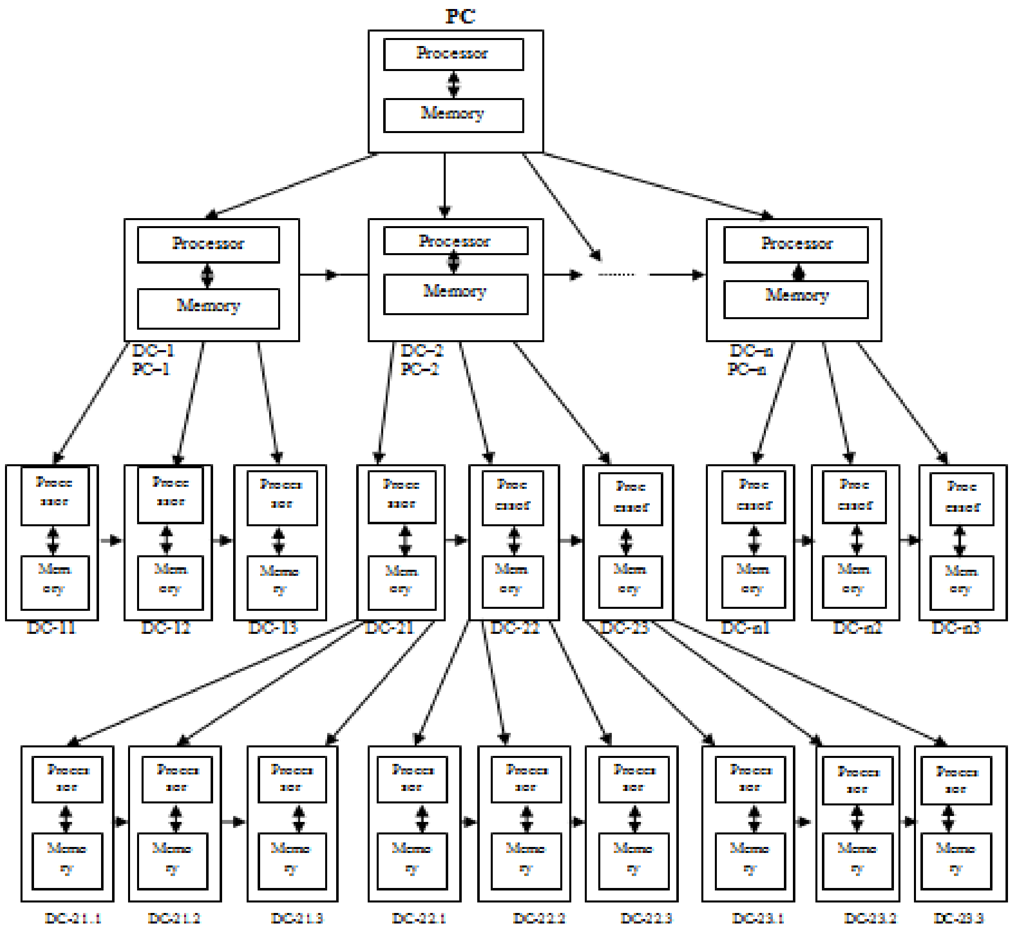


Fig. 10. A multi-tier ADS (3-tier)

The homogeneous data structure r-train (train) and the heterogeneous data structure r-atrain (atrain) are introduced for large data and implemented in details in [8,9] in an autonomous processor system.

4.3 Characterization of the Data Structures Train & Atrain

Every coach in a train/atrain is linked to the immediate next coach by the link address **e**. Although all the coaches can be reached directly from the Pilot but there is a forward link from each coach to its immediate next coach. However, the last coach is connected to an invalid address.

4.3.1 Cyclic Train & Cyclic Atrain

A train (atrain) is called to be a cyclic train (atrain) if the last coach is connected to the first coach.

4.3.2 Train/Atrain with Doubly Linked Address e

Here the node of the link address **e** consists of two fields called by "Predecessor" and "Successor". The Predecessor field contains the address of the previous coach and the Successor field contains the address of the next coach. In a cycle topology, the Predecessor field of the first coach is filled up with the address of the last coach (although in a cycle topology there is no significance of first or second coach, etc. as all the coaches are in a cycle, nevertheless we assume that the coaches are identified as 1st, 2nd, 3rd, etc.). However, for a multi-horse topology, the Predecessor field of the first coach is filled up with an invalid address, like the successor field of the link address **e** of the last coach.

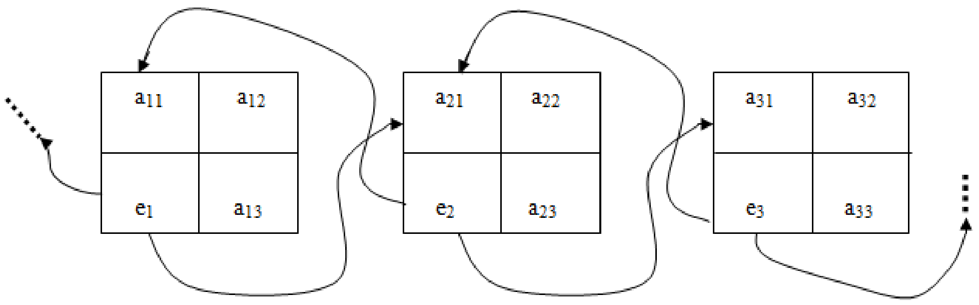


Fig. 11. Doubly linked coaches in a train/atrain

However in a doubly linked train/atrain, both forward and backward movements are possible from any coach while in cycle topology; for multi-horse cart topology backward movement is not possible from the first coach and forward movement is not possible from the last coach. It is advisable that unless not required, the doubly linked train/atrain may be avoided and simple train/atrain data structure could be used.

5 Helatrix (Hematrix) in an Atrain Distributed System (ADS) using the Heterogeneous Data Structure 'r-Atrain'

For implementation of the data structures train/atrain in [8,9] in an autonomous processor system (not in a distributed system), the notion of CD-Table, Coach, Status of a coach etc. are introduced and explained in details. In this section a method is presented on implementation of helatrix (hematrix) in an Atrain Distributed System (ADS) using the heterogeneous data structure 'r-Atrain'. For an atrain distributed system, the notion of CD-Table, Coach, Status of a coach, etc. are almost same as those for an autonomous processor system but with slight adjustment which are presented below.

5.1 Code of a Datatype and CD-Table for an ADS

A table is to be created by the user (i.e. by the concerned organization) to fix unique integer code for each datatype which are under use in the organization. This is not an absolute set of codes to be followed universally by every organization, but it is a local document for the concerned organization. For different organizations, this table could be different. But once it is fixed by an organization it should not be altered by this organization, except that addition of new datatypes and corresponding codes may be decided and be incorporated in the table at any stages later retaining the existing records. This table is called **Code of Datatype Table** or **CD-Table** (in short). A sample CD-Table of a hypothetical organization is shown below for the sake of understanding :-

Table. 4. A hypothetical example of a **CD-Table** of an organization

Sr. No.	Datatype	Space required in bytes (n)	Code of Datatype (c)
1	Character	1	0
2	Integer	2	1
3	Real	4	2
4	String-1	10	3
5	String-2	20	4
6	String-3	50	5
7	File-1	100 KB	6
8	File-2	1 MB	7
9	File-3	10 MB	8
10	File-4	25 MB	9
11	....	....	....
12	....	....	....

It may be noted that for any organization, for the datatypes character, integer, boolean, etc. the individual space requirement respectively are absolutely fixed. But for a particular organization, for the datatypes String-1, String-2 and String-3 (String type appearing thrice in this case) the space requirement in the above table has been fixed at 10 bytes for one kind, 20 bytes for another and 50 bytes for another kind. Similarly there are four types of file : File-1, File-2, File-3 and File-4, in the CD-Table and the space requirement for them have been fixed at 100 KB, 1 MB, 10 MB and 25 MB respectively, fixed by choice of the concerned organization (developers).

5.2 Coach of a r-Atrain in an ADS

By a coach C of a r-atrain we mean a pair (A,**e**) where A is a non-empty larray (could be a null larray) and **e**, called by 'twin address', is an larray of two addresses.

5.2.1 'Twin Address' and 'Double Twin Address' e

The twin address **e** is stored in an address node having two fields : Computer Address Field and Memory Address Field, as shown in Table.5 below.

Table. 5. A Node for 'Twin Address' **e** in an ADS

Computer Address Field	Memory Address Field

The Computer Address Field contains the address of a Computer (immediate next one) and the Memory Address Field contains the address of a memory element inside that computer in the atrain distributed system.

Table. 6. Twin Address **e** in an ADS

S_i	s

Here the twin address **e** is basically a kind of link address. Its significance is **:** it says location and the address of the immediate next coach, and thus it links two coaches in the r-atrain.

However, a **double twin address** can also <u>sometimes</u> play a better role to the developers (although not always). The double twin address **e** is stored in an address node having four fields : Successor Computer Address Field, Successor Memory Address Field, Predecessor Computer Address Field, Predecessor Memory Address Field, as shown in Table.7 below.

Table. 7. A Node for 'Double Twin Address' **e** in an ADS

Successor Computer Address Field	Successor Memory Address Field	Predecessor Computer Address Field	Predecessor Memory Address Field

In the 'Double Twin Address' e in the DC-i, the Successor Computer Address Field contains the address of the immediate next Computer DC-(i+1) and the Successor Memory Address Field contains the address of a memory element inside that computer, whereas the Predecessor Computer Address Field contains the address of the previous Computer DC-(i-1) and the Predecessor Memory Address Field contains the address of a memory element inside that computer, in an atrain distributed system.

Table. 8. Double Twin Address **e** in DC-i in an ADS

$\mathbf{S_{i+1}}$	s_{i+1}	$\mathbf{S_{i-1}}$	s_{i-1}

Here the twin address **e** is basically a kind of link address. Its significance is**:** it says

location and the address of the immediate next coach (first coach of the immediate next computer), and also it says location and the address of the previous coach (first coach of the previous computer). However, it is the choice of the developer whether to use twin address system or double twin address system for the link address **e**. Since each coach can store big data, the excess amount of space required by double twin address system (compared to twin address system) is negligible. The double twin address system in an ADS allows horizontal movement fluently both forward and backward. However it is advisable that if twin address system in ADS suffices the purpose then double twin address system is to be avoided by the developers.
For a cycle topology, the fields corresponding to the Predecessor Computer Address and the Predecessor Memory Address of the link address **e** of DC-1 are filled up with the addresses from the last DC of the siblings (although in a cycle topology there is no significance of first or second DC, etc. as all the DCs are in a cycle, nevertheless we assume that the DCs are identified as 1st, 2nd, 3rd, etc. in the siblings). However, for a multi-horse topology, the fields corresponding to the Predecessor Computer Address and the Predecessor Memory Address of the DC-1 are filled up with invalid addresses, like the Successor Computer Address field and the Successor Memory Address field of the link address **e** of the last DC in their siblings.

A coach in a r-atrain can store homogeneous data only, not heterogeneous data. But different coaches of a r-atrain store data elements of different datatypes, and thus the data structure r-atrain is a kind of heterogeneous data structure. For constructing a coach for a r-atrain in an organization, we must know in advance the datatype of the data to be stored in it. For this we have to look at the CD-Table of the organization, and reserve space accordingly for r number of data.
In a atrain distributed system dealing with big data, different coaches are stored in different computers. It is desired so because the data inside a coach is homogeneous but coach to coach is heterogeneous, and hence the GETNODE will be different for different distributed computers but in accordance with the CD-Table only. If desired by the developer of the concerned organization, one distributed computer may store files of approximately 10 MB size, another distributed computer may store files of approximately 100 MB size, another distributed computer may store integers, and so on. But it is not a strict policy, sometimes two or more coaches may be stored in one computer too if desired by the developer (although not recommended), and in that case GETNODE module needs to be designed accordingly for that distributed computer. These type of decisions is taken by the developer of the concerned organization. If the coach C is the only coach of the r-atrain then the fields of the twin address **e** will be put equal to invalid addresses and if the coach C is the last coach of the r-atrain then the fields of the twin address **e** will be put either equal to invalid addresses or linked to the first coach **(in case circular link is desired).** Otherwise **e** will be the twin address of the next coach. The address S_i is the address

of a computer and the address s is the address in memory of the computer S_i. Thus the next coach is stored in the computer S_i at the memory address s.

Suppose that the larray A has r number of elements in it of a given datatype. If each element of A is of size x bytes (refer to CD-Table) and if the data **e** requires 2+2 = 4 bytes to be stored in memory, then to store the coach C in memory (r.x + 4) number of consecutive bytes are required, and accordingly the coach be created by the programmer (concerned organization). In our discussion henceforth, by the phrase "datatype of a coach" we shall always mean the datatype of the data elements of the coach. A coach stores and can store only homogeneous data (i.e. data of identical datatype), but datatype may be different for different coaches in a r-atrain.

5.3 Status of a Coach and Tagged Coach (TC) in a r-Atrain in an ADS

The status s of a coach in a r-atrain is a pair of information (c, n), where c is a non-negative integer variable which is the code of datatype (with reference to the concerned CD-Table) of the data to be stored in this coach and n is a non-negative integer variable which is equal to the number of ε elements present in it (i.e. in its larray) at this point of time. Therefore, $0 \leq n \leq r$. In the status s = (c, n) of a coach, the information c is called the **"code-status"** of the coach and the information n is called the **"availability-status"** of the coach at this time. The significance of the variable n is that it informs us about the exact number of free spaces available in the coach at this point of time. If there is no ε element at this time in the larray of the coach C, then the value of n is 0. Thus, without referring the CD-Table, the status of a coach can not be and should not be fixed.

If C = (A,**e**) is a coach in a r-atrain, then the corresponding tagged coach (TC) is denoted by the notation [C,s], where s = (c, n) is the status of the coach. This means that C is a coach tagged with the following two information :

(i) one signifying the datatype of the data of the coach.

(ii) the other reflects the total amount of available free spaces (here it is termed as ε elements) inside the coach at this time.

For example, consider the CD-Table of Table.4. Clearly a TC with the larray a = < 5, 2, ε ,13,25, ε ,2, > will be denoted by [C,(1,2)]; a TC with the larray b = < 6, 9, 8> will be denoted by [C,(1,0)]; a TC with the larray d = < ε , ε , ε , ε > will be denoted by [C,(3,4)] assuming that this coach will accommodate strings only, and so on.

5.4 Heterogeneous Data Structure r-Atrain in an ADS

A r-atrain is basically a linked list of tagged coaches of various datatypes. This linked list is called the 'pilot' of the r-atrain. The pilot linked list is always controlled in the pilot computer (PC) of the atrain distributed system. The implementation of this pilot in the PC memory can also be done using the data structure array in some cases. The pilot of an r-atrain is thus, in general, a linked list. But, if we are sure that there will be no requirement of any extra coach in future, then it is better to implement pilot as an array. The number of coaches in a r-atrain is called the 'length' of the r-atrain which may increase or decrease time to time. A 'r-atrain' may also be called by the name 'atrain', if there is no confusion.

A r-atrain T of length l (> 0) will be denoted by the following notation

$T = < [C_1, s_{C1}], [C_2, s_{C2}], [C_3, s_{C3}], \ldots\ldots, [C_l, s_{Cl}] >$,
where the coach C_i is $(A_i, \mathbf{e_i})$ with A_i being a larray of length r, $\mathbf{e_i}$ being the twin address of the next coach C_{i+1} (an invalid address or address to the first coach if circular property desired in case C_i is the last coach) and s_{Ci} being the status (c_i, n_i) of the coach C_i, for i = 1, 2, 3, …., l.
For a r-atrain, **START** is the address of the pilot (viewing its implementation in the PC memory as a linked list). Thus START points at the coach C_1 in memory. The length l of the pilot could be any natural number, but the larrays of the TCs are each of fixed length r which store data elements (including ε elements) of heterogeneous datatype, although each coach itself can accommodate homogeneous data only, not heterogeneous data, where r is a natural number. Thus, by name, 1-atrain, 40-atrain, 64-atrain, etc. are few instances of r-atrain, where the term 0-atrain is undefined.
The notion of the heterogeneous data structure r-atrain in an atrain distributed system is a new but very simple data structure, a type of 2-tier data structure, having very convenient methods of executing various fundamental operations like insertion, deletion, searching etc. for heterogeneous big data, in particular for parallel computing. The most important characteristic of the data structure r-atrain for atrain distributed system is that it can handle the 4V issue of big data without any problem by making it scalable horizontally i.e. by adding more number of distributed computers (DCs). In a r-atrain, the data can be well accessed by using the indices. The coach names $C_1, C_2, C_3, \ldots, C_l$ do also mean the addresses (pointers) serially numbered as in case of arrays, for example :- the array z = (image-1, image-2, image-3, image-4) can be easily accessed calling by its name z only**.** The second information in the status of a coach is a dynamic information which reflects the availability of a seat (i.e. whether a valid data element can be stored now in this coach or not) while the first information is always static but can be different for different coaches. The first information 'code-status' of a coach does never alter by virtue of the construction principle of a coach, but the status of this coach may vary with time as the second information 'availability-status' may vary with time dynamically. Every coach of a r-atrain can accommodate exactly r number of homogeneous data elements serially numbered, each data element being called a **passenger**. Thus each coach of a r-atrain points at a larray of r number of passengers. By definition, the data e_i is not a passenger for any i. Consider a coach $C_i = (A_i, \mathbf{e_i})$ of a r-atrain. In the larray $A_i = <e_{i1}, e_{i2}, e_{i3}, \ldots, e_{i(r-1)}, e_{ir}>$, the data element e_{ij} is called the "j th passenger" or "j th data element" for j = 1, 2, 3, 4,..., r. Thus we can view a r-atrain as a linked list of heterogeneous larrays. Starting from any coach, one can visit the inside of all the next coaches but not any of the previous coaches. The r-atrain is a forward linear object, not a type of circular one. However, if desired by the developer of the concerned organization, it can be made circular by storing the twin address of the first coach in the address node of the last coach. The r-atrain data structure is neither a dynamic array nor a HAT. It has an added advantage over HAT that starting from one data element at any DC, all the next data elements of subsequent DCs can be read well without referring to any hash table or the pilot at the PC. Any linked list is a 1-atrain where the coaches are having a common status (c, 0), c being the code of the data-

type.
But the notion of 1-atrain is not a generalization of 'linked list'. It may be mentioned here that, by default, any heterogeneous data structure can be used as a homogeneous data structure, although not preferred in general.

5.4.1 Example of a r-Atrain in ADS

Refer to the CD-Table of Table.4. Consider a 3-atrain T of length 3 given by

$$T = < [C_1, (8,0)], \quad [C_2, (3,1)], \quad [C_3, (9,1)] >,$$

where $C_1 = < F11, F12, F13, \mathbf{e_1} >$, $C_2 = < CALCUTTA, \varepsilon, DELHI, \mathbf{e_2} >$, and $C_3 = < F31, F32, \varepsilon$, an invalid twin address >.

We consider multi-horse topology of ADS. Here $\mathbf{e_1}$ is the twin address of the coach C_2 (i.e. address of larray A_2) in this 3-atrain, and $\mathbf{e_2}$ is the twin address of the coach C_3 (i.e. address of larray A_3). Since it is a 3-atrain, each coach C_i can accommodate exactly three passengers (including ε elements, if any). In coach C_1, the status is (8, 0) which means that this coach can accommodate file of size 10 MB or less (with reference to the CD-Table) and there is no free space in this coach at this point of time. The larray is $A_1 = < F11, F12, F13 >$, which means that according to the CD-Table the first passenger is the file F11, second passenger is the file F12, and the last/third passenger is the file F13; all these three files are of size 10 MB or less, and the data $\mathbf{e_1}$ being the twin address of the next coach C_2. Thus, T is a larray of three TCs which are $[C_1, (8,0)]$, $[C_2, (3,1)]$, $[C_3, (9,1)]$. The logical diagram of this 3-atrain T is shown below where data in coaches are to be read clockwise starting from e_{11} for the coach C_1, from e_{21} for the coach C_2, from e_{31} for the coach C_3 :-

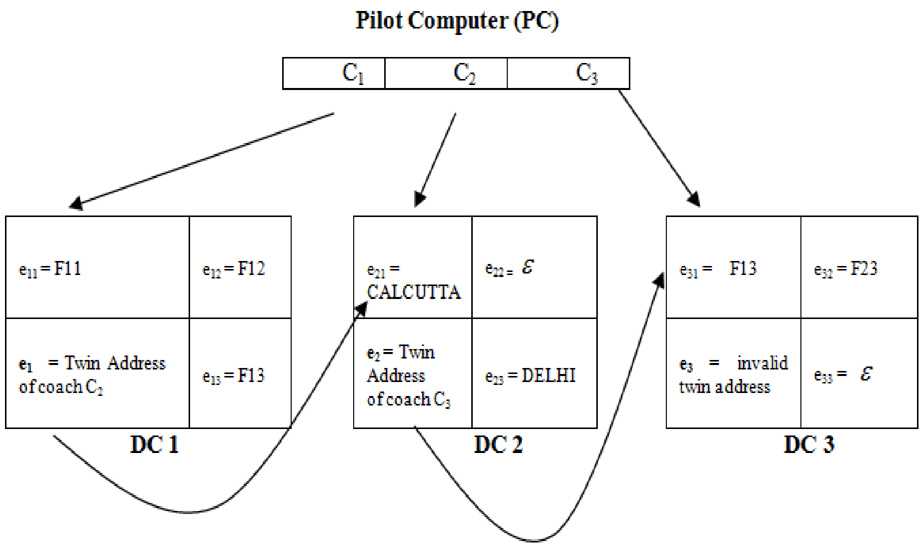


Fig. 12. A 3-atrain with 3 coaches in an ADS

The following figure shows a r-atrain with 30 number of coaches in 30 DCs :-

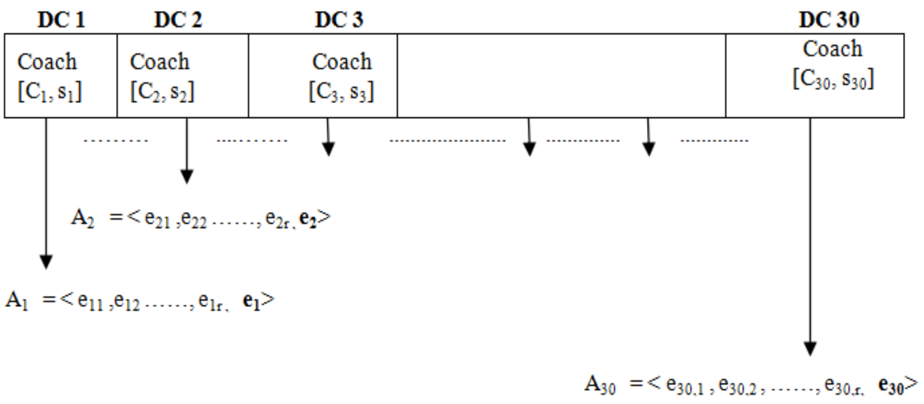

DC 1
DC 2
DC 3
DC 30
Coach [C1, s1]
Coach [C2, s2]
Coach [C3, s3]
Coach [C30, s30]
A2 = <e21 ,e22 ......, e2r, e2>
A1 = <e11 ,e12 ......, e1r, e1>
A30 = <e30.1 , e30.2 , ..., e30.r, e30>

Fig. 13. A r-atrain with 30 coaches in 30 DCs in an ADS

The following figure shows one coach (ith coach) of a 11-atrain, where data are to be read clockwise starting from e_{i1} :-

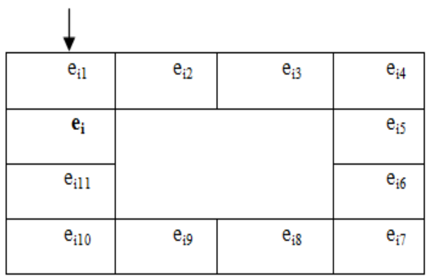


Fig. 14. A coach C_i of a 11-atrain

5.4.2 Full Coach in a r-Atrain

A coach in a r-atrain is said to be a full coach if it does not have any passenger ε, i.e. if its status s is (c, 0) where c is the code of its datatype.
In a full coach, we can not insert (insertion operation is explained in later section here) any more data (passenger) at this point of time (however, may be possible at later point of time). See the following example :

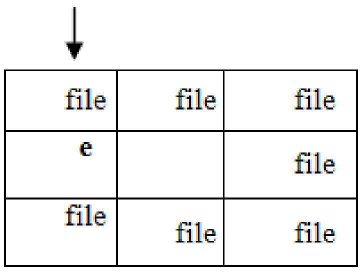


Fig. 15. A full coach of a 7-atrain in an ADS

It is a full coach of a 7-atrain with status (8,0) as per CD-Table of Table.4. Clearly, the coaches of any classical linked list (which is a 1-atrain) are all full.

5.4.3 Empty Coach in a r-Atrain

A coach in a r-atrain is said to be an empty coach if every passenger of it is ε, i.e. if the corresponding larray is a null larray.

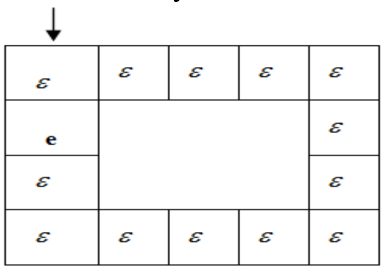


Fig. 16. An empty coach of a 13-atrain in an ADS

Thus for an empty coach of a r-atrain, the status is equal to (c,r). A coach may be sometimes neither empty nor full (see Fig.17 below).

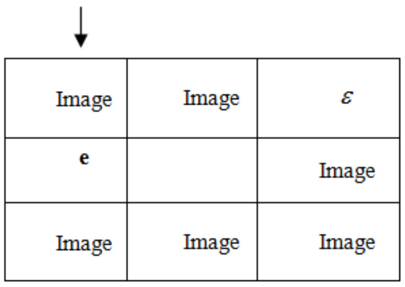


Fig. 17. A coach of a 7-atrain which is neither empty nor full in an ADS

5.5 Fundamental Operations on 'r-Atrain' in an ADS for Big Data

The three fundamental operations on the data structure r-atrain (atrain) in an atrain distributed system are 'insertion', 'deletion', 'search', which are explained below assuming that the ADS is in multi-horse network topology and the link address **e** is in twin address system.

(*If the link address **e** is in double twin address system then the definition of these three basic operations can be adjusted appropriately. Even, if the ADS in Cycle topology instead of multi-horse topology, the implementation method needs slight adjustment as the last address here does not link to invalid address but to the address of the first DC of its siblings DCs*).

5.5.1 Insertion

There are three types of insertion operations in the data structure r-atrain in an atrain distributed system : -

(i) insertion (addition) of a new coach in a r-atrain.

(ii) insertion of a data element (passenger) in a given coach of a r-atrain.
(iii) insertion of a data element (passenger) in a r-atrain.

(i) Insertion of a new coach in a r-atrain

Insertion of a new coach in a r-atrain in an atrain distributed system involves both hardware and software activities. Insertion of a new coach can be done at the end of the pilot, nowhere else. Therefore, assuming the policy that one coach to be stored in one DC, it needs addition of one extra computer connected with the last DC and then adjustment of software both at PC and DC levels. The first job is to decide about the datatype of the coach which is required now to be inserted.
Consider the r-atrain T = < [C_1, s_{C1}], [C_2, s_{C2}], [C_3, s_{C3}], ……, [C_k, s_{Ck}] >, with k number of coaches, where the coach $C_i = (A_i,e_i)$ is stored in DC-i (for i = 1, 2, 3, …., k). The newly connected computer is DC_{k+1} where the new coach is to be stored.
After insertion of a new additional coach, the updated r-atrain in PC immediately becomes the r-atrain : T = < [C_1, s_{C1}], [C_2, s_{C2}], [C_3, s_{C3}], ……, [C_k, s_{Ck}], [C_{k+1},r] >.
Initially at the time of insertion, we create C_{K+1} as an empty coach in DC_{k+1} with status = (c,r) where c is the code of the intended datatype of the coach and since the new coach is empty therefore the number of available space is r at this time, but the coach is likely to get filled-up with non-ε passengers (data) later on with time.

For insertion of a new coach C_{K+1} in a r-atrain, we need to do the following steps : -
(i) Complete the hardware connection activities.
(ii) Read the CD-Table for the code c of the datatype of the new coach intended for insertion. If the code c is not available in the CD-Table, expand CD-Table accordingly.
(iii) Update the pilot (linked list) in the PC.
(iv) $\mathbf{e_k}$ in C_k is to be updated and to be made equal to the twin address of C_{K+1}.
(iv) Set $e_{k+1,\,j} = \varepsilon$ for j = 1, 2, ….., r
(v) Set $\mathbf{e_{k+1}}$ = an invalid twin address.
(vi) Set s_{Ck+1} = (c,r).

(ii) Insertion of an element x inside the coach of a r-atrain in an ADS

Insertion of an element (a new passenger) x inside the coach C_i is feasible if x is of same datatype (like other passengers of the coach C_i) and if there is an empty space available inside the coach C_i. For doing this type of insertion one has to reach the coach C_i from the pilot directly (although traverse from the first coach C_1 or from any coach C_j where j<i to the coach C_i is also feasible but it could be costly). If the availability-status n of C_i is greater than 0 then data can be stored successfully in this coach, otherwise insertion operation fails at this moment of time. For insertion of x, we can replace the lowest indexed passenger ε of C_i with x. After each successful insertion, the availability-status n of the coach is to be updated by doing n = n–1, and thus by updating the status S_{Ci} = (c,n) by its new value given by S_{Ci} = (c,n-1).

(iii) Insertion of an element x in a r-atrain in an atrain distributed system

In this case too, the code c (with reference to the CD-Table) corresponding to the datatype of the data x plays an important role in the process of insertion. An initial search is done for the coaches (starting from C_1 onwards) which are having the same code c in their status. Suppose that, the array of the code-matched coaches so extracted from the pilot is Z = ($Ç_1$, $Ç_2$, $Ç_3$, ……, $Ç_t$). If the array Z is a null array or if the availability-status is zero for each and every member of Z, then the insertion operation is to be done by inserting a new coach C_μ, first of all, as per steps mentioned above then by performing the insertion operation. Otherwise we find out the coach $Ç_k$ in Z with lowest index k for which the availability-status n is greater than 0, and then perform the insertion operation as per steps mentioned above.

5.5.2 Deletion

There are two types of deletion operations in the data structure r-atrain in an atrain distributed system:-

(i) Deletion of a data element ($\neq \varepsilon$) from any coach of the r-atrain.

(ii) Deletion of the last coach C_i, if it is an empty coach, from a r-atrain.

(i) Deletion of a data e_{ij} ($\neq$ ε) from the coach C_i of a r-atrain in an ADS

Deletion of e_i from the coach C_i is not allowed as it is the link (twin address). But we can delete a data element e_{ij} from the coach C_i in the DC-i. Deletion of a data (passenger) from a coach means replacement of the data by an ε element (of same datatype). Consequently, if e_{ij} = ε, then the question of deletion does not arise. Here it is pre-assumed that e_{ij} is a non-ε member element of the coach C_i .

For j = 1, 2, …., r, deletion of e_{ij} is done by replacing it by the null element ε, and updating the availability-status n by doing n = n+1. Deletion of a data element (passenger) does not effect the size r of the coach. For example, consider the tagged coach [C_i, (c_i,m)] where $C_i = < e_{i1}, e_{i2}, e_{i3}, e_{i4}, ..., e_{ir} >$.

If we delete e_{i3} from the coach C_i, then the updated tagged coach will be [C_i, (c_i, m+1)], where C_i = $< e_{i1}, e_{i2}, \varepsilon , e_{i4}, ..., e_{ir} >$.

(ii) Deletion of the last coach C_i from a r-atrain in an ADS

Deletion of coaches from a r-atrain is allowed from the last coach only and in backward direction, one after another. An interim coach can not be deleted. Advertently, we avoid here any kind of deletion of interim coach. Also there is no need to disconnect the concerned DC. The last coach C_i can be deleted if it is an empty coach (as shown below) : -

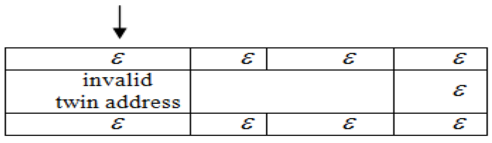


Fig. 18. An empty last coach of a 9-atrain which can be deleted from the ADS

If the last coach is not empty, it can not be deleted unless its all the passengers are deleted to make it empty. To delete the empty last coach $C_{i,}$ of a r-atrain, we have to do the following actions : -

(i) update e_{i-1} of the coach C_{i-1} by storing an invalid twin address in it.

(ii) delete $[C_i, (c_i,r)]$ from the r-atrain

$T = < [C_1, s_{C1}], [C_2, s_{C2}], [C_3, s_{C3}], \ldots\ldots, [C_{i-1}, s_{Ci-1}], [C_i, (c_i,r)] >$, and get the updated r-atrain $T = < [C_1, s_{C1}], [C_2, s_{C2}], [C_3, s_{C3}], \ldots\ldots, C_{i-1}, s_{Ci-1}] >$.

(iii) update the pilot in the PC.

5.5.3 Searching for a data x in a r-Atrain T of length k

Searching a given data from a database or storage of big data is one of the important issues to the users dealing with big data. Searching for a data x in a r-atrain T in an atrain distributed system is very easy. If we know in advance the coach number Ci of the passenger x, then by visiting the pilot in PC we can enter into the coach Ci (DC-i) of the r-atrain directly and then can read the data-elements e_{i1}, e_{i2}, ….., e_{ir} of the larray A_i for a match with x. Thus we need to visit only one DC of the atrain distributed system. Otherwise, the code c (with reference to the CD-Table) of the datatype of the data x plays an important role in the process of searching. The initial search is done for the coaches (starting from C_1 onwards) which are having the same code c in their status. Suppose that the array of the code-matched coaches so extracted is $Z = (Ç_1, Ç_2, Ç_3, \ldots\ldots, Ç_{t-1}, Ç_t)$. If the array Z is a null array, the search fails. Otherwise we start searching inside, beginning from the coach $Ç_1$ onwards till the last coach $Ç_t$ of Z. The search may lead to either success or failure. We need not go back to the pilot for any help during the tenure of our searching process. Here lies an important dominance of the data structure r-atrain over the data structure HAT introduced by Sitarski [11]. The searching can be done in parallel very fast monitoring from the PC, which is obvious from the architecture of the data structure r-atrain in an atrain distributed system.

6 Heterogeneous Data Structures 'MA' for Solid Helatrix

Today's supercomputers or multiprocessor systems which can provide huge parallelism has become the dominant computing platforms (through the proliferation of multi-core processors), and the time has come to stand for highly flexible advanced level of data structures that can be accessed by multiple threads which may actually access any large volume of heterogeneous data simultaneously, or even that can run on different processors simultaneously. In most of the giant business organizations, the system has to deal with a large volume of heterogeneous data, heterogeneous type of big data for which the data structures of the existing literature can not lead to the desired solution for thirst or desired optimal satisfaction always. The very common and frequent operations like Insertion, deletion, searching, etc. are required to be faster for the big data. Such situations require some way or some method which work more efficiently than the simple rudimentary existing data structures. Obviously,

there is a need of a dash of creativity of a new or better performed heterogeneous data structure for big data which at the same time must be of rudimentary in nature.
In this section we propose a very powerful and dynamic real time heterogeneous data structure MA to deal with big data of heterogeneous datatype, and then we present a generalized type of application of MA. MA is the abbreviation for 'Multi Atrains', as it is an extension of the heterogeneous data structure 'Atrain' proposed by Biswas in [8,9]. For details about the properties, operations, algorithms and applications of heterogeneous data structure 'Atrain' and of the homogeneous data structure 'Train', one could see [8,9]. In the heterogeneous data structure Atrain, there are logically two layers **:** the pilot is the lower layer and the coaches are in the upper/inner layer. We extend the notion of Atrain by incorporating nil or one or more number of intermediate layers between the pilot (lower layer) and linked-coaches (upper layer) to develop a new heterogeneous data structure 'Multi Atrains (MA)'. The term 'Atrain' stands for "**A**dvanced **train**'. The intermediate layers are usually Atrains, but could be pilots, linked-coaches, or larrays too. Type of the various layers, according to the construction-needs for the problems under study, are decided by the developers on behalf of the organization concerned. Thus Atrain may be regarded as a special case of MA, where there is(are) no intermediate layer(s) between the upper layer and the lower layer. If the total number of layers is called the **height**, then height(Atrain) = 2, and height(MA) $\geq$ 2. Clearly, an 'MT of height h' is a particular case of an 'MA of height h'. The Solid Hematrix/Helatrix is useful if the big data is a temporal big data. Otherwise, the logical storage structure '2-D Helatrix' (2-D Hematrix) is the appropriate model to store heterogeneous big data as the number of rows/columns in a 2-D helatrix can be scalable upto any big extent. Consequently, the data structures MT or MA are useful for temporal big data only. Implementation method of a 3-SHL(3-SHM) using MA in an autonomous system is similar to the implementation of a 3-SL(3-SM) using MT in an autonomous system as explained earlier in section-2.

file	file	file	file
file	file	file	file
%	$	(	@

Layer M_1

file	file	file	file
9	4	7	2
file	file	file	file

Layer M_2

Fig. 19. Two layers of a hematrix of height 2

Here each of M_1 and M_2 are twin addresses given by M_1 = (DC-1, T_1) and M_2 = (DC-2, T_2) respectively. The elements $C^1{}_1$, $C^1{}_2$ and $C^1{}_3$ in T_1 are the twin addresses (DC-11, A21B0h), (DC-12, B0072h) and (DC-13, E0104h) respectively. Similarly the elements $C^2{}_1$, $C^2{}_2$ and $C^2{}_3$ in T_2 are the twin addresses (DC-21, 02AB5h), (DC-22, CB082h) and (DC-23, BA26Dh) respectively.

7 Conclusion

The 'Theory of Solid Matrices/latrices' [11] is an extension of the classical theory of matrices. A new matrix is introduced called by 'Hematrix'/'Helatrix'. The name 'Hematrix' stands for Heterogeneous Matrix, and the name 'Helatrix' stands for Heterogeneous Latrix. Each row in a hematrix/helatrix contains data of homogeneous datatype, but different rows contain data of heterogeneous datatype. This mathematical model is designed to make a logical storage structure for big data. The number of rows/columns in a hematrix/helatrix are scalable, i.e. can be made as large as desired. The homogeneous data structure r-train or the heterogeneous data structure r-atrain is an appropriate tool for implementing BFS, DFS or any Search algorithms, Divide and Conquer algorithms, Branch and Bound type algorithms, etc. for big data. Recently Bashir [2] has applied the r-train data structure in matrix multiplication method using a parallel processing technique. The data structure atrain (train) should not be confused with a thought that it is just a linked list of arrays, but it is much more [8,9]. A hematrix/helatrix of big size may not be always possible to be implemented in a limited memory space of an autonomous computer. For storing big data we must have big amount of big spaces in our availability for which we need big number of memory spaces to work in an integrated way. Consequently, there is a need of an appropriate and efficient new model for distributed system. The new type of distributed system introduced in this paper called by 'Atrain Distributed System' (ADS) which could be unitier or multitier. An ADS will have a single unique Pilot Computer (PC) and several Distributed Computers (DCs) connected by new type of network topologies called by 'Multi-horse Cart Topology' and 'Cycle Topology'.

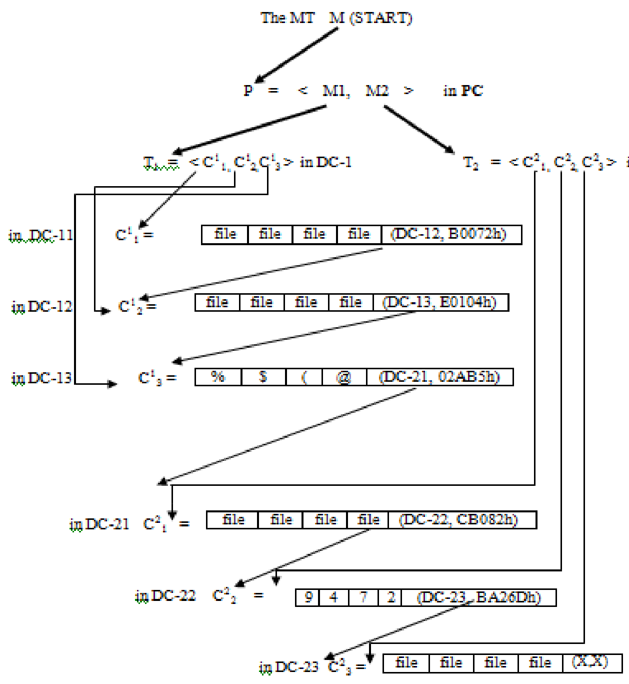

The MT M (START)
P = < M1, M2 > in PC
T1 = < C11, C12 C13 > in DC-1
T2 = < C21, C22, C23 > in DC-2
in DC-11 C11 = file file file file (DC-12, B0072h)
in DC-12 C12 = file file file file (DC-13, E0104h)
in DC-13 C13 = % $ (@ (DC-21, 02AB5h)
in DC-21 C21 = file file file file (DC-22, CB082h)
in DC-22 C22 = 9 4 7 2 (DC-23, BA26Dh)
in DC-23 C23 = file file file file (X,X)

Fig. 20. Implementation of the 3-SHM S of height 2 using the data structure MA in an ADS

A 'Multi-horse Cart Topology' or a 'Cycle Topology' is neither a tree topology nor a bus/ring/star/mesh/hub/hybrid topology [1]. A cycle topology looks like a ring (wheel) of a riding pedal cycle connected to the centre PC by spokes. Both the new type of topologies are implemented in an ADS either by twin address **e** or by doubly twin address **e**, choice being of the developer of the concerned organization /institution. The doubly twin address facilitates the sibling DCs to communicate with their respective next and respective previous DCs (i.e. both forward and backward communication possible). The Solid Hematrix/Helatrix is useful if the big data is a temporal big data. Otherwise, the logical storage structure '2-D Helatrix' (2-D Hematrix) is the appropriate model to store heterogeneous big data as the number of rows/columns in a 2-D helatrix can be scalable upto any big extent. Consequently, the data structure MT is useful for temporal homogeneous big data and the data structure MA is useful for temporal heterogeneous big data only, whereas the Atrain (train) data structure can easily handle heterogeneous/homogeneous big data for storage in an ADS for immediate or future processing. Our future research work will be to find a method to solve the issue of memory fragmentation.

References

[1] Andrew S. Tanenbaum, Computer Networks (3rd Edition), Pearson Education India, 1993.

[2] Bashir Alam, Matrix Multiplication using r-Train Data Structure, AASRI(Elsevier) Procedia 5(2013)189–193, doi:10.1016/j.aasri.2013.10. 077 of the Conference on Parallel and Distributed Computing Systems.

[3] David Feinleib, "Big Data Demystified: How Big Data Is Changing The Way We Live, Love And Learn", The Big Data Group Publisher, LLC San Francisco, USA, 2013.

[4] Jeffrey Needham, "Disruptive Possibilities: How Big Data Changes Everything", O'reilly Publisher, Cambridge, 2013.

[5] Joel L Franklin, Matrix Theory, Mineola, N.Y., 2000.

[6] Jules J Berman, "Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information", Morgan Kaufmann (Elsevier) Publisher, USA, 2013.

[7] Phil Simon, "Too Big to Ignore: The Business Case for Big Data", John Wiley & Sons, New Jersey, 2013.

[8] Ranjit Biswas, Heterogeneous Data Structure "R-Atrain", INFORMATION : An International Journal (Japan), Vol.15(2) February'2012, pp 879-902 (©2012 International Information Institute of Japan & USA).

[9] Ranjit Biswas, Heterogeneous Data Structure “r-Atrain”, Chapter-12 in “Global Trends in Knowledge Representation and Computational Intelligence” by B. K. Tripathy and D. P. Acharjya, IGI Global, USA, 2013. (DOI : 10.4018/978–1–4666–4936–1, ISBN13: 9781466649361, ISBN10: 1466649364, EISBN13: 9781466649378).

[10] Ranjit Biswas, Region Algebra, Theory of Objects & Theory of Numbers, International Journal of Algebra, Vol. 6(8), 2012, 1371 – 1417.

[11] Ranjit Biswas, Theory of Solid Matrices & Solid Latrices, Introducing New Data Structures MA, MT : for Big Data, International Journal of Algebra, Vol.7(16) (2013) pp 767–789, http://dx.doi.org/10.12988/ija.2013.31093.

[12] Sitarski, Edward, Algorithm Alley, "HATs: Hashed array trees", Dr. Dobb's Journal 21(11), 1996. http://www.ddj.com/architect/184409965?pgno=5

[13] Viktor Mayer-Schönberger and Kenneth Cukier, “BIG DATA : A Revolution That Will Transform How We Live, Work, and Think”, Eamon Dolan/ Houghton Mifflin Harcourt Publisher, 2013.

Received: March 25, 2014