CHAPTER 6

PPS-ADS: An Approach for Handling Big Data

6.1 Introduction

The agile proliferation of voluminous data due to the technological advancements has led to the development of novel tools, techniques and systems for the effective management of such huge datasets. Conventional methods and tools are inadequate to handle the intricate nature of these data. The unstructured nature and heterogeneity of such datasets adds to the complexity of the management systems. This chapter aims to provide an improvement to the Atrain Distributed System (ADS) [30] by incorporating new security features and improved data storage, retrieval and routing mechanisms. ADS was developed in [30] for effective handling of big data in any 4 V’s. “ADS” at the core is like a train with one engine (or pilot computer) and several coaches (distributed computers). In order to represent ADS in computer memory, the author in [30] has also proposed two novel data structures by the name “r-train” [28-30] and “r-atrian” [28-30] for managing homogenous and heterogeneous big data respectively. Although ADS fairs very well in effective storing of big data, it lacks the security and privacy features. Security and privacy of data are the imperative requirements for any data management system. In order to overcome this limitation of ADS, this chapter proposes a modified version of ADS called by the name “PPS-ADS (Privacy-Preserved and Secured Atrain Distributed System)” [4]. The current chapter is divided into four sections. The chapter starts with a brief description of ADS and the big data structures “r-train” and “r-atrian” in section 6.3. The section 6.5 presents an improved version of r-train data structure by incorporating effective storing and searching mechanisms. Then section 6.6 talks about the notable security issues in the current ADS. The next section 6.7 presents the architecture of the proposed PPS-ADS. In this section, the various components of PPS-ADS are explained in detail and their working mechanism has been discussed.

6.2 A Brief Introduction of Classical ADS

A distributed computer system (DCS) is one wherein we have several (more than two) independent computers with a provision to connect and communicate with each other using
any mutually acceptable medium on a network. These computers may be available at single site or geographically separated. The core idea of a DCS is to promote resource and computation sharing among all the connected computers to give an impression of one single working unit.

ADS is a new kind of distributed system proposed in [30]. ADS primarily follows a master slave architecture wherein the master is known as a pilot computer (PC) and slaves are the other distributed computers (DCs). The PC is directly connected to all the DCs. A DC is connected to other DC via a forward and backward link. It can be noted that each DC stores the address of next DC which is connected to it forming a chain like structure. The ADS has the ability to expand in both width and depth. If the ADS is expanding in width, the addition of more DCs are allowed only at the end. And if the ADS is expanding in depth, the addition of DCs are allowed from left to right. Thus we can say that an ADS can have a uni-tier or multi-tier configurations. Furthermore, if in a uni-tier arrangement of ADS, the last DC is connected to the first DC, then it is said to be cyclic ADS. ADS can also be seen as a linked structure of computers which is analogous to the classical linked list data structure. In ADS, we can have “Z” number of DCs (where Z is any natural number) at any tier, however we can have only one and unique PC [28-30].

Figure 6.1 and 6.2 given here depicts the structure of a typical Uni-tier and Two-tier ADS.
It can be noticed from these arrangements of the PC and the DCs in the above figures that they are following a pattern of arrangement. Although apparently it looks similar to a tree structure, however it can be observed that it is not exactly following the tree structure as we have a connection from one DC to another, this is absent in a classical tree structure. If we have an arrangement of PC and DCs such that the last DC of the system stores an invalid address, then this type of arrangement is termed as “Multi-horse Cart Topology [30]. Furthermore, if the last of DC stores the address of the first DC of the system, then this type of arrangement is known as Cycle-Topology [30]. These are the two new types of topologies defined in [30]. Figure 6.3 and 6.4 show the multi-horse cart topology and cycle topology respectively.
For representing ADS in computer memory, the author in [30] developed “r-train” and “r-atrain” data structures. r-train is used for organizing homogenous big datasets while r-atrain is used to store heterogeneous big datasets. The next section gives the description of these two data structures.

6.3 r-train and r-atrain: Two Data Structures for Big Data

r-train and r-atrain at the core looks similar to a linked list, however the primary difference is that, these data structures can be distributed and scaled to any desired amount and has the ability to store the data in a hierarchical manner which is lacking in a classical linked list [28-30]. In the classical array and linked list data structures we cannot leave any blank space in between two data elements while storing. But one or more blank storage spaces (empty elements) between the data elements in the r-train and r-atrain data structures are allowed to be stored. An r-train or r-atrain is a collection of linked coaches. Each coach can store one or more data elements and can have one or more blank storage spaces (empty elements). The blank storage space is represented by ‘ε’ (null element). The letter ‘r’ in r-train represents the
number of coaches in the r-train. Another point to note here is that, the space required by ‘ɛ’ element is exactly the same as required by the other elements of the coach. In future it can store the similar type of data as other elements of the coach and no other type of element is allowed for storage. This means that if the non-empty elements of the coach are all integers, then all the ‘ɛ’ spaces in the coach can store only integer type elements. A coach also stores the information about the number of ‘ɛ’ elements in it, i.e. about the number of vacancies in a real time manner [28-30].

6.3.1 Terminologies of r-train and r-train Data Structures

This section provides the various terminologies and their descriptions [28-30].

6.3.1.1 larray

For representing a coach, the work in [30] has given the notion of a new term “larray” (like-array). A larray is a kind of array with the ability to store one or more ‘ɛ’ elements (null elements). The total number of elements (including ‘ɛ’) gives the size or length of the larray. Given below are some examples of larrays.

i. LA-1 = < 12, 26, 7, ɛ, 38, ɛ, ɛ >, the length of ‘larray’ LA-1 is 7

ii. LA-2 = <12, 15, ɛ, ɛ, 47, 1700, 18, 19, ɛ >, the length of ‘larray’ LA-2 is 10

iii. LA-3 = < ɛ, ɛ, ɛ, ɛ, ɛ, ɛ >, the length of ‘larray’ LA-3 is 6

iv. LA-4 = <13, 15, 82, 14, 7>, the length of ‘larray’ LA-4 is 5

v. LA-5 = < 32 >, the length of ‘larray’ LA-5 is 1

vi. LA-6 = < ɛ, ɛ >, the length of ‘larray’ LA-6 is 2

vii. LA-7 = < ɛ >, the length of larray LA-7 is 1

viii. LA-8 = < >, LA-8 is an empty larray.

ix. LA-9 = < 2, ɛ >, the length of larray LA-9 is 2

x. LA-10 = < 12, ɛ, 5 >, the length of larray LA-10 is 3

xi. LA-11 = < 4.5, 6.7, 8.3, 2.7, ɛ, ɛ >


xiii. LA-13 = < 8.5, ɛ >
xiv. \( \text{LA-14} = \langle \text{z}', \varepsilon \rangle \)

xv. \( \text{LA-15} = \langle 124.6 \rangle \)

Now considering the above defined larrays (from i to x), if we have all the elements of the larray as ‘\( \varepsilon \)’, then it is said to be “null larray”. Here the larrays LA-3, LA-6 and LA-7 are “null- larrays”. The sizes of these larrays are 6, 2 and 1 respectively. Thus we can say that, null-larrays are not unique. Also larray LA-8 does not contains any element. Thus it is known as an “empty” larray. Thus empty larray and null larray are two different objects. Also larrays LA-1 to LA-10 are integer larrays, larrays LA-11, LA-13 and LA-15 are double larrays and larrays LA-12 and LA-14 are character larrays.

6.3.1.2 Coach of an r-train

The coach of an r-train is represented by ‘C’. It consists of two items. One is the larray and other is the address of the next coach. In other words we can say that a Coach(C) is a set consisting of larray items and address of the next coach. One point to remember is that, the address of the next coach will always be stored as the last element of the coach and it will not be counted as a number of elements of the larray stored in that coach [28-30]. More formally, a “Coach (C)” = (LA, add) where “LA” is a “non-empty larray” (can include a “null larray” also) and “add” is the address of the next coach of the “r-train”. Figure 6.5, 6.6 and 6.7 show the example of a coach storing integer, floating and character type larray items respectively.

![Coach storing integer type larray items](image1)

**Fig 6.5** Coach storing integer type larray items

![Coach storing floating point type larray items](image2)

**Fig 6.6** Coach storing floating point type larray items

![Coach storing character type larray items](image3)

**Fig 6.7** Coach storing character type larray items

6.3.1.3 Status of Coach
The status of the coach signifies the number of blank empty spaces (‘ɛ’ elements) in it. It is denoted by “s (small ‘s’))”, where 0 ≤ s ≤ r. If s = 0, it means no more elements can be added in the coach. If s = r, it means that a maximum of ‘r’ elements can be stored in the coach [28-30].

Let us take few examples to discuss about the status of coaches.

Consider five coaches C1, C2, C3, C4 and C5 as below:

C1 = < LA1, add2 >, C2 = < LA2, add3 >, C3 = < LA3, add4 >, C4 = < LA4, add5 >
C5 = < LA5, add invalid >

And LA1, LA2, LA3, LA4 and LA5 are given as:

LA1 = < 5, 4, 7, 9, ɛ, 85, ɛ, ɛ >
LA2 = < 6, 7, 21, ɛ, ɛ, ɛ, ɛ, ɛ >
LA3 = < 32, 54, 17, 87, 43, ɛ >
LA4 = < 5, 9, 43, 75 >
LA5 = < 97, ɛ, 61, ɛ, 9 >

Therefore, the status of C1, C2, C3, C4 and C5 are 3, 4, 1, 0 and 2 respectively i.e.

s(C1) = 3, s(C2) = 4, s(C3) = 1, s(C4) = 0, s(C5) = 2.

6.3.1.4 Tagged Coach of an r-train

A tagged coach is one wherein the status of the coach is attached with the coach. It is denoted by TC and represented by a pair. TC = (C,s), where ‘C’ is the coach and ‘s’ is its corresponding status [28-30]. Given below are few examples of tagged coaches.

Let us have the coaches as C1, C2, C3, C4 and C5.

C1 = < LA1, add2 >, C2 = < LA2, add3 >, C3 = < LA3, add4 >, C4 = < LA4, add5 >
C5 = < LA5, add invalid >.

And LA1, LA2, LA3, LA4 and LA5 are given as:

LA1 = < 5, 4, 7, 9, ɛ, 85, ɛ, ɛ >, LA2 = < 6, 7, 21, ɛ, ɛ, ɛ, ɛ >
LA3 = < 32, 54, 17, 87, 43, ε >, LA4 = < 5, 9, 43, 75 >
LA5 = < 97, ε, 61, ε, 9 >.

Therefore, the Tagged Coaches of the Coaches C1, C2, C3, C4 and C5 are given as:
TC1 = (C1, 3), TC2 = (C2, 4), TC3 = (C3, 1), TC4 = (C4, 0), TC5 = (C5, 2).

6.4 General Representation of r-train Data Structure

Suppose we have to create an r-train with seven coaches C1, C2, ..., C7. Then this type of r-train is represented as
T = < (C1, s1), (C2, s2), (C3, s3), (C4, s4), (C5, s5), (C6, s6), (C7, s7) >

where s1, s2, s3, s4, s5, s6 and s7 are status of the coaches C1, C2, C3, C4, C5, C6, C7 respectively. In general we can represent an r-train with ‘z’ number of coaches as follows:
T = < (C1, s1), (C2, s2), (C3, s3), . . . . . .(Cz, sz) >

In the above figure 6.8, C1, C2, C3, C4, ... Cz are the coaches. LA1, LA2, LA3, ... , LAz are the larrays, d1, d2, d3, .... , d2z are the data items and add2, add3, add4, ...., addz+1 are the addresses of the next coaches [28-30].

Let us take an example of an r-train (T) in which we have 5 coaches with 3 elements each as shown in figure 6.9.
T = < (C1, 3), (C2, 2), (C3, 1), (C4, 0), (C5, 1) >
where C1 = < ε, ε, ε, add2 >, C2 < 14, ε, ε, add3 >, C3 = < 12, 45, ε, add4 >, C4 = < 37, 3, 6, add5 > and C5 = < 72, ε, 8, Invalid Address >
6.5 Sorted r-train Data Structure

This section provides the details about the proposed sorted r-train data structure. Sorted r-train is an improved version of the classical r-train data structure with efficient insertion, deletion, updation and searching mechanisms.

6.5.1 Introduction

The enormous expansion of technology has led to the generation of massive volumes of data with highly intricate properties and characteristics ranging in all 7 V’s of big data. The classical ways of data organizations are inadequate to handle such gigantic datasets. There are extensive researches going around the globe with a common aim to find ways and means to effectively store, organize and manage big data. One such proposal was given in [30]. The proposal highlighted two new kinds of data structure called by r-train and r-atrain for handling homogeneous and heterogeneous big data. r-train and r-atrain exploited the advantages of both array and linked list but are much more robust, dynamic and scalable.

Since the inception of IoT technologies, the real time systems have seen phenomenal transformations. The core aim of real time system is to handle the user requests as they arrive and give the response immediately. For real time systems, it is a critical requirement to have an optimal data retrieval mechanism which is as fast as possible. Since the amount of data handled by IoT systems consists of huge piles of massive datasets, effective ways to search the requested data item or service is pivotal. Therefore, a good real-time system must have an efficient searching mechanism. This section proposes an improvement in the classical r-train data structure in order to provide efficient storage, retrieval and searching mechanisms. Since the core aim of r-train is to store huge datasets, it is generally done in a distributed manner so that the coaches of the r-train are evenly balanced. We have applied effective sorting and searching techniques in order to achieve these refinements.

The unstructured nature of big data makes it even more difficult to organize it efficiently.
Furthermore, the varying characteristics of big data which are rapidly expanding in all 7 V’s (described previously) must be observed and analyzed critically as most of these characteristics are overlapping and can cause confusion while appropriately differentiating them from one another. If we can devise techniques and mechanisms to individually identify and characterize these V’s we can come up with state-of-the-art solutions to our problems.

### 6.5.2 Classical r-train Data Structure

The r-train is a dynamic homogeneous data structure which is used to store big data [28-30]. In an r-train, the data is arranged in the form of coaches of a train where each coach has a predefined capacity. A coach is not allowed to store more elements than its predefined capacity. r-train supports distributed data storage and its coaches can be placed at multiple sites. These coaches are connected together via a linked address (just like typical Linked Lists data structure, with a difference that Linked Lists do not support distributed storage). An r-train can be linear, circular or hierarchical. The last coach of the linear r-train contains an invalid address. The last coach of a circular ‘larray’ contains the address of the first coach and the last coach of r-train at each level of hierarchy contains an invalid address. A coach can be thought of as a computer node with a memory unit which is used to store this big data. The data inside a coach is organized using the concept of larrays [28-30]. A larray is a dynamic collection of identical elements in which we have the provision of having some empty spaces along with the non-empty elements. This provision makes it superior than conventional arrays. The last element of the larray is the address of the next coach therefore it cannot be ‘empty’. The term ‘dynamic’ means that the size of the larray can be increased or decreased as and when required. The empty spaces in the larray are represented by “ε”. The length of the larray is defined as the number of non-empty elements along with the number of ‘ε’ elements in the larray [28-30].

### 6.5.3 Sorted r-train: The Proposed Approach [8]

The core aim of sorted r-train data structure is to improve the insertion, deletion, modification and searching capabilities of classical r-train data structure ([4-5], [30]). More specifically we aim to improve the time complexity of addition, deletion searching and modification operations. The time complexity of searching a data element in native r-train is $O(n^2)$, but with the proposed approach we are reducing it to $O(\log(n))$. Similarly the insertion operation takes $O(n)$ time, but in our approach it takes a constant time $O(1)$. Another improvement of Sorted r-train is the ability to remove a coach from anywhere in the r-train as opposed to
classical r-train which allowed only the last coach to be deleted. The fundamental mechanism of our approach is to ensure that the data elements are stored at its proper (appropriate) position in the r-train so that it follows some order or arrangement (ascending or descending). Furthermore, with every deletion of data element, the elements of r-train are reorganized so that the remaining data items are at their proper positions. Since this reorganization is done after the deletion operation, the overhead involved is not a concern for the end user. It should be noted that deleting the coach of the r-train does not mean deleting it physically from the memory, rather it means that all the elements in this coach must be ‘ε’ elements (blank spaces). The proposed operations are described below ([4-5], [30]).

6.5.4 Pre-Processing and Assumptions

1. Let “FIRST” denotes the address of the first Coach (C1).
2. Let the length (total number of elements in the coach) of coach is ‘k’.
3. Define the first Coach C1 in the memory which contains all “ε” elements in it and “add_inv” as an invalid address.
   For example let the coach C1 is defined as
   \[ C_1 = < e_{i11}, e_{i12}, e_{i13}, e_{i14}, \ldots\ldots, e_{i1k}, add_{inv} >, \] where \( e_{i11}, e_{i12}, \ldots, e_{i1k} \) are empty element and add_{inv} is the address of this coach.
4. Find the Status (s) of C1. Here s = k. (since all elements in C1 are ‘ε’).
5. Create the “tagged coach (TC1)” of C1.
   - TC1 = (C1, s) or TC1 = (< e_{i11}, e_{i12}, e_{i13}, e_{i14}, \ldots\ldots, e_{i1k}, add_{inv} >, k)
6. Let ‘begin’, ‘mid’ and ‘end’ be the indexes of first, last but one and middle elements of the array of the coach respectively.

6.5.5 Proposed Insertion Mechanism

Insertion in Sorted r-train is of two forms:

i. Insertion of a “New data item” in a “Coach”.
ii. Insertion of a “New Coach” in the “Sorted r-train”.

6.5.5.1 Insertion of a “New data element” in Sorted r-train

For inserting a new element, we follow the process given below.

Algorithm: Sorted-r-train-Insertion
Let “CS” be the set of coaches in the r-train. The set CS can contain any number of Coaches.

Let $S_c$ be the status of Coach C.

Let $L_c$ be the length of Coach and $L_c = K$

---

**Algorithm: Sorted r-train Insertion**

1. FOR ALL New Elements, DO
2. FOR ALL Coaches in CS, DO
3. IF ‘$S_c$’ = 0 THEN
4. Print “Coach Full, Data Cannot be inserted in this Coach.”
5. Goto Step 2.
6. END IF
7. ELSE IF $S_c = L_c$ THEN
8. Insert the Element at $LA[mid]$ (Middle location in the LArray) in the Coach
9. Sort (LA)
10. Update the $S_c$ as $S_c = S_c - 1$
11. END IF
12. ELSE IF $(0 < S_c < K)$
13. IF $(LA[mid] = ε)$
14. Insert the Element at $LA[mid]$
15. Sort (LA)
16. Update the $S_c$ as $S_c = S_c - 1$
17. ELSE
18. Loc = SearchLocation (LA)
19. Insert the element at the location ‘Loc’ in the array
20. Sort (LA) // Sort the array
21. Update the $S_c$ as $S_c = S_c - 1$
22. END IF
23. END FOR
24. END FOR

---

**Function: SearchLocation(LArray)**

```java
int SearchLocation(LA)
{
    return (LA.IndexOf(ε)) // returns First index of ‘ε’ in the array
}
```

---

Let us take few example for showing the insertion operation.

**Example 1:**

Let the given array be $LA = < ɛ, ɛ, ɛ, ɛ, ɛ, ɛ, ɛ, ε, ε, ε add >$.

Let elements to be inserted are 5, 6, 4, 2, 9, 3 and 1.
We can easily find out the following for the given LA:
Length of larray LA = 7 (No. of elements in larray)
S_c = 7 (No. of ‘ɛ’ elements in larray)
begin = 0
end = 7
mid = (begin + end)/ 2 = (0 +7) = 3
Since S_c > 0, we can insert the element in this coach.

1. Insert 5 in the LA

Element 5 will be inserted at location 3. Therefore the larray after insertion of element 5 will be: LA = < ɛ, ɛ, ɛ, 5, ɛ, ɛ, ɛ, add >
   Now, Sort the LA.
   After sorting, the larray LA = < ɛ, ɛ, ɛ, ɛ, ɛ, ɛ, 5, add >
   Update the Status as S_c = S_c − 1 = 7 − 1 = 6

2. Insert 6

Element 6 will be inserted at location 3. Therefore the larray after insertion of element 6 will be: LA = < ɛ, ɛ, ɛ, 6, ɛ, ɛ, 5, add >
   Now, Sort the LA
   After sorting, the larray LA = < ɛ, ɛ, ɛ, ɛ, ɛ, 5, 6, add >
   Update the status as S_c = S_c − 1 = 6 − 1 = 5

3. Insert 4

Element 4 will be inserted at location 3. Therefore the larray after insertion of element 4 will be: LA = < ɛ, ɛ, ɛ, 4, ɛ, 5, 6, add >
   Now, Sort the LA
   After sorting, the larray LA = < ɛ, ɛ, ɛ, 4, 5, 6, add >.
   Update the status as S_c = S_c − 1 = 5 − 1 = 4

4. Insert 2

Element 2 will be inserted at location 3. Therefore the larray after insertion of element 2 will be: LA = < ɛ, ɛ, ɛ, 2, 4, 5, 6, add >
   Now, Sort the LA
   After sorting, the larray LA = < ɛ, ɛ, 2, 4, 5, 6, add >
   Update the status as S_c = S_c − 1 = 4 − 1 = 3
5. Insert 9
Element 9 will be inserted at location 0. Since LA[mid] is already filled. Therefore the larray after insertion of element 6 will be:

\[ \text{LA} = \langle 9, \epsilon, \epsilon, 2, 4, 5, 6, \text{add} \rangle \]

Sort the LA
\[ \text{LA} = \langle \epsilon, \epsilon, 2, 4, 5, 6, 9, \text{add} \rangle \]
\[ S_c = S_c - 1 = 3 - 1 = 2 \]

6. Insert 3
Element 3 will be inserted at location 0. Since LA[mid] is already filled. Therefore the larray after insertion of element 3 will be:

\[ \text{LA} = \langle 3, \epsilon, 2, 4, 5, 6, 9, \text{add} \rangle \]

Sort the LA
\[ \text{LA} = \langle \epsilon, 2, 3, 4, 5, 6, 9, \text{add} \rangle \]
\[ S_c = S_c - 1 = 2 - 1 = 1 \]

7. Insert 1
Element 1 will be inserted at location 0. Since LA[mid] is already filled. Therefore the larray after insertion of element 1 will be:

\[ \text{LA} = \langle 1, 2, 3, 4, 5, 6, 9, \text{add} \rangle \]

Sort the LA
\[ \text{LA} = \langle 1, 2, 3, 4, 5, 6, 9, \text{add} \rangle \]
\[ S_c = S_c - 1 = 1 - 1 = 0 \]

Note that, Now the status of Coach = 0. This means that no more elements can be further added in this coach.
The final larray after insertion is \( \text{LA} = \langle 1, 2, 3, 4, 5, 6, 9, \text{add} \rangle \)

*Example 2:*
Let the given larray be \( \text{LA} = \langle \epsilon, \epsilon, \epsilon, \epsilon, \epsilon \text{add} \rangle \).
Let elements to be inserted are 15, 62, 14, and 2.
We can easily find out the following for the given LA:

Length of larray \( \text{LA} = 5 \) (No. of elements in larray)
\[ S_c = 5 \] (No. of ‘\( \epsilon \)’ elements in larray)
\[ \text{begin} = 0 \]
end = 5
mid = (begin + end)/2 = (0 + 5)/2 = 2
Since $S_c > 0$, we can insert the element in this coach.

**Insert 15**
Element 15 will be inserted at location 2. Therefore the array after insertion of element 15 will be:

$LA = \langle \varepsilon, \varepsilon, 15, \varepsilon, \varepsilon \rangle$

Now, Sort the LA
After sorting the array $LA = \langle \varepsilon, \varepsilon, \varepsilon, 15, \varepsilon \rangle$
Update the status as $S_c = S_c - 1 = 5 - 1 = 4$

**Insert 62**
Element 62 will be inserted at location 2. Therefore the array after insertion of element 62 will be:

$LA = \langle \varepsilon, \varepsilon, 62, \varepsilon, 15 \rangle$

Now, Sort the LA
After sorting the array $LA = \langle \varepsilon, \varepsilon, 15, 62, \varepsilon \rangle$
Update the status as $S_c = S_c - 1 = 4 - 1 = 3$

**Insert 14**
Element 14 will be inserted at location 2. Therefore the array after insertion of element 14 will be:

$LA = \langle \varepsilon, \varepsilon, 14, 15, 62 \rangle$

Now, Sort the LA
After sorting the array $LA = \langle \varepsilon, \varepsilon, 14, 15, 62 \rangle$
Update the status as $S_c = S_c - 1 = 3 - 1 = 2$

**Insert 2**
Element 2 will be inserted at location 1. Since $LA[mid]$ is already filled.
Therefore the array after insertion of element 2 will be:

$LA = \langle 2, \varepsilon, 14, 15, 62 \rangle$

Now, Sort the LA
After sorting the array $LA = \langle \varepsilon, 2, 14, 15, 62 \rangle$
Update the status as $S_c = S_c - 1 = 2 - 1 = 1$

This completes the insertion operation.

6.5.5.2 **Insertion of new coach at the end in Sorted r-train**

In order to insert a new coach in the r-train we proceed as follows. Note that the size of the new coach will be same as the size of the other coaches in the r-train as per definition ([4-5], [30]). The new coach will be having all elements as ‘ε’ elements.

---

**Algorithm: Insert New Coach**

1. Define the new Coach $C_n$ with all ‘ε’ elements
2. Update the PC by including (storing) the metadata of $C_n$ in its memory.
3. Modify the address of the previous Coach ($C_{n-1}$) by assigning the address of Coach $C_n$ to it.
4. Set the address of Coach $C_n$ to an invalid address value.
5. Set the status of Coach $C_n = k$.

---

6.5.6 **Searching mechanism of Sorted r-train**

Let the given LArray of Sorted r-train is

$LA = \langle \varepsilon, \varepsilon, \varepsilon, a, b, c, d, \text{add} \rangle$, where $a, b, c$ and $d$ are arbitrary numbers and add is the address.

$T = \langle TC_1, TC_2, TC_3, TC_4, \ldots, TC_n \rangle$, where $TC_1, TC_2 \ldots TC_n$ denote the tagged coaches of the Sorted r-train.

The coaches $C_1, C_2, C_3, \ldots C_n$ are given by $(LA_1, \text{add}_1), (LA_2, \text{add}_2), (LA_3, \text{add}_3), \ldots (LA_n, \text{add}_n)$

Each of these coaches has some status associated with them denoted by $s_1, s_2, s_3, \ldots s_n$ respectively.

Let the element to be searched is “ele”.

For searching the element in the coach we can have two ways.
1. The first way is fairly straightforward, when coach number is known to us. In this case, directly go to the coach and apply the proposed searching strategy to obtain the result.

2. The second case is slightly tricky. It is when the coach number is not previously known to us. In this case we have to start from the beginning of the Sorted r-train for searching the particular element. The proposed searching procedure is given below.

---

**Algorithm: Sorted r-train Searching**

1. **FOR** Each Search Request, **DO**
2. **FOR** Each Coach, **DO**
3. Find Low and High Index of Each array in the Coach
4. Find Mid = (Low + High) / 2
5. **WHILE** (Low < High)
6. Print Element Not Present
7. **IF** (LA [Mid] = = element)
8. **RETURN** mid
9. **ELSE IF** (LA [Mid] = = ε)
10. Low = Mid + 1
11. **IF** (LA [Mid] > num)
12. High = Mid - 1
13. **ELSE**
14. Low = Mid + 1
15. **END WHILE**
16. **END FOR**
17. **END FOR**

---

Some Examples to show the working of Sorted r-train Searching approach

Example 1:
Let LA = < ε, ε, ε, 7, 8, 12, add >
Suppose we want to search for element 8
For the sake of understanding let us create a block diagram for showing the steps
The given array LA has been represented below.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

Therefore, we have

\[
\begin{align*}
\text{begin} & = 0 \\
\text{end} & = 6 \\
\text{mid} & = \frac{\text{begin} + \text{end}}{2} \\
& = \frac{0+6}{2} = 3
\end{align*}
\]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

Is \((8 = \text{LA}[\text{mid}])\) ?

No, because \(\text{LA}[\text{mid}] = \varepsilon\)

Therefore, \(\text{begin} = \text{mid} + 1\).

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>ε</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

Now we have the updated values as

\[
\begin{align*}
\text{begin} & = 4 \\
\text{end} & = 6 \\
\text{mid} & = \frac{\text{begin} + \text{end}}{2} \\
& = \frac{4+6}{2}
\end{align*}
\]
Is \((8 = LA[\text{mid}])\)?

YES, because \(LA[\text{mid}] = 8\).

The search stops here and the location (\text{mid}) is returned which can be used to fetch the data item.

**Example 2:**

Let \(LA = \langle \varepsilon, \varepsilon, 2, 4, 6, 10, 12, 19, 20, \text{add} \rangle\)

Suppose we want to search for element 4

For the sake of understanding let us create a block diagram for showing the steps

The given array \(LA\) has been represented below.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>(\varepsilon)</td>
<td>(\varepsilon)</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td>mid</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Therefore we have

\[
\begin{align*}
\text{begin} &= 0 \\
\text{end} &= 8 \\
\text{mid} &= (\text{begin} + \text{end})/2 \\
&= (0+8)/2 = 4
\end{align*}
\]
Is \((4 = = \text{LA}[\text{mid}])\)?

No, because \(\text{LA}[\text{mid}] = 6\), Now since \(\text{LA}[\text{mid}] > \text{ele}\)

Therefore end = \(\text{mid} - 1\).

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>Begin</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Therefore we have the updated values as:

\[
\begin{align*}
\text{begin} &= 0 \\
\text{end} &= 3 \\
\text{mid} &= (\text{begin} + \text{end})/2 \\
&= (0+3)/2 \\
&= 1
\end{align*}
\]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td>mid</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is \((4 = = \text{LA}[\text{mid}])\)?

No, because \(\text{LA}[\text{mid}] = \varepsilon\), Therefore, \(\text{begin} = \text{mid} + 1\);

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now we have the updated values as:

\[
\begin{align*}
\text{begin} &= 2 \\
\text{end} &= 3 \\
\text{mid} &= (\text{begin} + \text{end})/2
\end{align*}
\]
= (2+3)/2 = 2

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td>mid</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is \((4 = \text{LA}[\text{mid}])\)?
No, because \(	ext{LA}[\text{mid}] = 2\). Since, \(\text{LA}[\text{mid}] < \text{ele}\), Therefore, \(\text{begin} = \text{mid} + 1\).

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td>mid</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Therefore we have the updated values as:
\(\text{begin} = 3\)
\(\text{end} = 3\)
\(\text{mid} = (\text{begin} + \text{end})/2\)
\(= (3+3)/2\)
\(= 3\)

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>ε</td>
<td>ε</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>Add</td>
</tr>
<tr>
<td>Positions</td>
<td>begin</td>
<td>mid</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is \((4 = \text{LA}[\text{mid}])\)?
YES, because \(	ext{LA}[\text{mid}] = 4\).
The search stops here and the location (mid) is returned which can be used to fetch the data item.

### 6.5.7 Deletion Mechanism in Sorted r-train

Sorted r-train allows three kinds of deletion operations ([4-5], [30]).

1. Deleting data element from a Coach.
2. Deleting last Coach of the Sorted r-train.
3. Deleting a Coach that lies between the Two Coaches

6.5.7.1 Deletion of the data element from a coach

1. Suppose we want to delete the element “ele”.
2. If we know the coach number of the element “ele” beforehand, we can directly access the coach by querying the pilot computer and search for “ele” using search mechanism technique discussed above.
3. After identifying the position of the element “ele” inside the coach, replace its content with “ε”.
4. Update the status of the coach as s = s+1.
5. Sort the larray (LA).
6. If the coach number of the element “ele” is not known, we will start from the first coach, and search for the element “ele” in every coach till the element is found using the search mechanism discussed above.
7. After finding ‘ele’ within any coach, we will update its contents and insert a “ε” value at its place.
8. Sort the larray (LA).
9. Change the status of the coach C as s = s+1.

The reorganization (sorting) of elements after every deletion operation ensures that all the elements are always present at their proper positions. This makes the process of insertion and searching less intricate.

6.5.7.2 Deletion of the last Coach of Sorted r-train

We can delete a coach if and only if it is null (or empty) coach (i.e. all the elements in the coach are “ε” elements, as depicted here under in Figure 6.10). Here deletion refers to deleting the coach from the memory.

| ε | ε | ε | ε | ε | ε | Invalid Address |

Fig 6.10 Example of an Empty Coach
In order to delete the last coach, the following procedure is adopted

a) Identify the coach to be deleted. Let the coach be $C_n$.
b) Modify the address “add_1” of the coach which is previous to the coach $C_n$ by replacing its value with an “invalid address”.
c) Delete $C_n$ from the Sorted r-train.
d) Delete the metadata of $C_n$ from the pilot.

Figure 6.11 (a) and 6.11 (b) shows the coach before and after the deletion operation has been performed.

<table>
<thead>
<tr>
<th>Coach $(C_1, S_{C_1})$</th>
<th>Coach $(C_2, S_{C_2})$</th>
<th>Coach $(C_3, S_{C_3})$</th>
<th>\ldots</th>
<th>Coach $(C_{i-1}, S_{C_{i-1}})$</th>
<th>Coach $(C_i, S_{C_i})$</th>
</tr>
</thead>
</table>

Fig 6.11 (a)  Sorted r-train before deletion of Coach $(C_1, S_{C_1})$

<table>
<thead>
<tr>
<th>Coach $(C_1, S_{C_1})$</th>
<th>Coach $(C_2, S_{C_2})$</th>
<th>Coach $(C_3, S_{C_3})$</th>
<th>\ldots</th>
<th>Coach $(C_{i-1}, S_{C_{i-1}})$</th>
</tr>
</thead>
</table>

Fig 6.11 (b)  Sorted r-train after deletion of Coach $(C_1, S_{C_1})$

6.5.7.3  \textit{Deletion of any Coach that lies between two Coaches in the Sorted r-train}

Sorted r-train has the provision to delete a coach that lies between any two coaches. In order to delete such coach, the following procedure is adopted.

a) Identify the coach to be deleted. Let this coach be $C_i$.
b) Modify the address “add_1” of the coach which is previous to “$C_i$” by replacing its value with the address of the coach which is next to “$C_i$”.
c) Delete $C_i$ from the Sorted r-train.
d) Delete the metadata of $C_i$ from the pilot.

Fig 6.12 (a)  An r-train before deletion operation
Figures 6.12 (a) and 6.12 (b) given above shows the deletion operation. Here the Coach $C_i$ has been deleted from the r-train.

“r-train” [28-30] is a robust extensible data structure which has the capabilities of effective management of big data. Furthermore, it has the potentials for performing parallel data processing [10]. The proposed refinement named as “Sorted r-train” has significantly improved the time complexity for search operations which in turn has improved the deletion and insertions of the native r-train data structure ([4-5], [30]).

Since we are storing the data elements in a sorted order, the worst case complexity of the proposed searching mechanism is $O(n \log(n))$ as opposed to $O(n^2)$ in case of classical r-train.

Similarly the deletion operation in Sorted r-train also takes $O(n \log(n))$ while it takes $O(n^2)$ in classical r-train. Insertion operation takes identical amount of time in both r-train and Sorted r-train. It is the restructuring of data items after the insertion that makes the Sorted r-train superior than classical r-train. Considering the significant improvements, the overhead of restricting the data elements can be neglected. Since the data searching and retrieval mechanism is extremely fast in sorted r-train, this data structure can be effectively used to handle real-time requests like in IoT devices and other real time systems [4-5], [30], [51].

6.6 Security issues in classical ADS

Along with all the advantages of ADS, there are some security issues involved in ADS. The data stored in the classical ADS are not secured at all. The absence of proper authentication and security mechanism makes ADS vulnerable to severe privacy and security threats. This section highlights these issues in details [4-5], [30].

1. No Provision of End-to-End security of data
2. Absence of User Authentication
3. Absence of Verification and Validation of data and users
4. Absence of Access Control mechanism
5. Absence of Data ownership Provisioning

6.6.1 No Provision of End-to-End security of data

In ADS, there is no provision of encrypting the data before storing or transferring it. This exposes the data and makes the data vulnerable to thefts and cybercrimes. Non availability of proper security mechanism in ADS invites a severe threat to the data. We can have a situation when the data is accessed and altered by any illegitimate personnel to cause serious damage to the organization, system or an individual. In any good data management system, end-to-end security is a must.

6.6.2 Absence of User Authentication

Non availability of an stringent authentication mechanism can cause severe damage to the business or organizations. Without authentication, anyone can access the system and alter the private and sensitive data stored in the system.

6.6.3 Absence of Verification and Validation of Data and Users

Non availability of verification and validation of data and users is a serious threat to the data management system. Without the presence of a sophisticated verification and validation mechanism, any illegitimate user can alter or store any kind of data into the system. The hackers can plant viruses into the system and can shut down the whole system.

6.6.4 Absence of Access Control Mechanism

A good distributed system must have the provision to keep a check on “who is accessing what data at what time and from where”. “ADS” in its current form lacks this ability. Furthermore, there is no concept of role based access in the current ADS.

6.6.5 Data Ownership Issues

In the ADS, the data resides at multiple locations in the DCs. This means that for any storage or retrieval requests, the data has to travel across the network. While the data is on the move, anyone can access and alter its content without the knowledge of the sender or the receiver. The question like “who owns the data and at what time” are not answered in the current ADS. Furthermore, in case of IoT systems, the pervasive nature of data brings forth critical privacy
issues since the data movement and pre-processing is very fast. Also, the context of the “sensitivity of data” varies from person to person and situation to situation. Therefore predefined data ownership, role-based access and time-based authorization must be provided to the users, which is lacking in the current literature on ADS ([4], [30]).

6.7 PPS-ADS: The proposed Architecture

This section presents the proposed PPS-ADS architecture. The ADS in its current form do not have the provision to keep track of the users and the data accessed by these users. The main aim of this proposed improvement is to incorporate security and reliability features in ADS and to provide a privacy-preserved and agile framework for effective and efficient storage and retrieval of big data. Figure 6.13 gives the architecture of PP-ADS approach [4].
For the sake of understanding the proposed approach has been divided into four parts. In order to improve the storage and retrieval mechanism, this chapter proposes an improvement.
to the r-train data structure as a first part of the proposal. The second part discusses the authentication mechanism. The third part talks about the improved data storage mechanism and the final part discusses the data retrieval approach.

In order to use the proposed PPS-ADS, the user needs to register with the system. Only the registered and legitimate users are allowed to use the PPS-ADS. When a user requests for using the PPS-ADS, he/she is asked to provide the credentials. The access to the system is granted only if the credentials provided by the users match with the credentials stored with the system. Otherwise the access to the system is denied. The system gives a role-based access to the users. Which means that the users can only access the data and resources which lies in its domain. Any attempt to access the data or resources outside its domain is prohibited. At the background of the authentication mechanism lies the OAuth 2.0 protocol ([35-36],[64], [77], [82], [95]). Using this protocol access is granted to the users. The main advantage of using OAuth 2.0 is the granting of time-based access tokens to the users. These tokens expire after a prefixed amount of time. This means that if a user is granted an access token for one hour, he/she can use the system only for one hour. As soon as the token expires, the access is revoked ([35-36], [64], [77], [82], [95]). Figure 6.14 gives the overview of the proposed authentication mechanism.

![Diagram](image)

**Fig 6.14**  User Authentication and Authorization Module

The classical ADS does not keep track of the users and data accessed by the users. However, by incorporating OAuth 2.0 and granting the access token serves two purposes. First the user is getting authenticated and secondly, the access token is giving insights about the portions of data that a user is eligible to access, thus keeping track of its every access.
6.7.1 Components of PPS-ADS

PPS-ADS is composed of five main components. The details of these components are given below [4].

6.7.1.1 Pilot Computer (PC):

The Pilot Computer (PC) is called as the brain of PPS-ADS. It is responsible for controlling every other component of the system. Each and every decision about data storage, retrieval, replication, routing, and encryption/decryption etc. is controlled by it. Furthermore, it also serves as the repository of metadata of the data and the coaches of “r-train” or “r-atrain”. This means that the PC stores the information like status of each coach, length of each coach, distance of each coach from the PC, paths leading to each coach etc. The pilot computer is also responsible for saving the image of the system after a prefixed interval of time and saving it at alternate site. This is useful in cases like system crashes, updates or recovery. The pilot computer identifies one DC among the connected DCs to store this image file. The decision is based on the distance of the DC from the PC, so that, in case of any faults or errors, the system can be recovered as soon as possible [4].

6.7.1.2 User Interface (UI):

This unit serves as an interface between the user and the system. The end user interacts with the UI to request for any data storage or retrievals. The UI gives token based access only to the legitimate users and denies the access to any unauthorized person.

6.7.1.3 Encryption and Decryption Unit (EDU):

The encryption and decryption of the data stored in PPS-ADS is carried out using Twofish cryptographic technique ([151], [155-156], [179]). Twofish is based on symmetric key cryptographic approach which has the capability to use variable length keys between 128 to 256 bits. It follows a Feistel network structure [65]. In order to encrypt the data, it is divided into multiple blocks of 128-bits and then it is fed into the encryption unit along with the key.

6.7.1.4 Software Defined Networking (SDN) unit:
PPS-ADS uses SDN in order to dynamically control the network configurations and parameters. As we are aware that SDN separates data plane from the control plane, thus the control of network configurations and management lies directly in the hands of the administrator (which is the pilot computer in this case). Using SDN, the network devices work only as the forwarding devices which receive the instructions from the PC and act accordingly. There are three core components of SDN namely SDN controller, Southbound API and Northbound API. Further details about SDN can be found in ([37], [52], [69], [76], [100], [102], [105], [132], [134], [165], [198]).

6.7.1.5 Distributed Computers (DC):

A distributed computer is one in which the ADS actually stores the data. These computers may be located at different sites within the organization or geographically distributed and connected through a secured network. Along with the data, each DC also stores the information about its capacity and the type of data stored in it. This information is periodically communicated to the PC in order to keep the PC updated about their current status [4]. All these components work in synchronization with each other and this synchronization is governed by the pilot computer (PC).

6.7.2 User Authentication and Authorization

The user authorization and authentication in PPS-ADS is governed by OAuth 2.0 framework [35-36]. Whenever the user requests for using PPS-ADS system, they are required to fill in their credentials on the PPS-ADS user interface. As soon as the credentials are entered by the user, it gets cross-checked and verified by the system by matching it with the credentials already stored in the system. If there is a match, the system assigns a secure login token to the user. This token is time-based limited-access grant to the user. That is an access token is valid only for a limited period of time and the user can use only the allowed services of the system upto this period only. After the time period expires, the system prompts the user to request for re-issuing the token or otherwise the access to the system is revoked. This kind of authorization and authentication mechanism is generally used in cases where access needs to be granted to third party service from within the system or outside the system.
The formal steps involved in OAuth 2.0 are given below [35-36].

1. The user request for access token from the system.
2. The system identifies the User on the basis of its credentials.
3. If found authentic, the system issues an Access token to the user based on the credential details (level of access provided).
4. Once the user gets the Access token, they can use the system by providing their access token. (This access token is a kind of time-bound pass that enables the user to use only the allowed services of the system with this time period).
5. As soon as the time period of the access token expires, the system prompts the user to request for re-issuing it.
6. If the user wants to further use the system, it can request for re-issue of the access token or extending the time-period of the current token.
7. The system analyses the request from the user and if deemed fit, extends the time period of the current access token or instructs the user to request for a fresh access token.
8. Once the user finishes working on the system before the expiry of the access token, it notifies the system.

The system on receiving this request from the user revokes the access grant and closes the connection.

6.7.3 Data Storage in PPS-ADS [4]

In order to store the data the user communicates with PPS-ADS by using the access token. When the PC receives the request to store the data, it takes the following decisions with respect to the data.

- Where to store the data (in which DC, the data shall be stored)?
- How to store the data?
- Which path to follow for storing the data in the DC?

These decisions are based on three vital parameters given below.

- Data Type (Structured or Unstructured).
- Status of the Coach.
- Distance of Coach from the PC.

The storage of data elements in the coaches of PPS-ADS is almost similar with Hadoop. This means that the data elements in PPS-ADS are also stored in the form of data-blocks. The
The main difference in PPS-ADS storage mechanism is that the data blocks here can be of variable size lengths (which are of fixed size in classical Hadoop). Furthermore, each data-block in PPS-ADS stores three important header fields, which are helpful in retrieving the stored data at later stages. Figure 6.15 shows the fields of the typical data-block in the proposed PPS-ADS architecture and figure 6.16 shows a simple Coach/DC with multiple data-blocks (of variable sizes).

<table>
<thead>
<tr>
<th>Coach_Id</th>
<th>Coach_Sub_Id</th>
<th>Block_Id</th>
<th>Data_Item</th>
<th>Length</th>
<th>Offset</th>
</tr>
</thead>
</table>

Fig 6.15  Example of a Data-Block in PPS-ADS

The various fields of Figure 6.15 are:

- **Coach_Id**: The Id of the coach in which the data is stored.
- **Coach_Sub_Id**: Id of the coach, wherein the replication of the data item is stored.
- **Block_Id**: The Id of the block wherein the data is being stored inside a coach.
- **Length**: The size or length of the data item.
- **Offset**: An integer value that gives the distance with respect to the base address in the block.

The Coach_Id, Coach_Sub_Id, and Block_Id in the proposed ADS take two bytes each.

Fig 6.16  Example of a Coach/DC with variable size Data-Blocks

A coach stores the information like which data element is stored in which data block. If the Coach_Id and Coach_Sub_Id are same for any data block, it means that this particular coach itself has multiple copies of the same data element.
Whenever there is a request for data storage from the user, the pilot computer checks the metadata records of each coach available in its memory in order to find the respective status of the coaches and their distances from the PC. It then instructs the SDN controller to select the optimal route for data transfer from the client side to the DC. As there are multiple routes for reaching a particular DC, the SDN makes sure to choose the most appropriate path in order to reach the destination DC. Since there can be multiple data storage or retrieval requests from different clients, there are chances of network traffic and congestions in routes [65]. The SDN dynamically updates its routing table to ensure real time traffic management and congestion control. Thus, ensuring fastest possible data transfer (considering all latencies and network delays).

Once the Coach in which the data will be stored and the path for data transfer is selected, the PC receives the data from the user and forwards it to the Encryption and Decryption Unit (EDU) for encrypting it using the Twofish cryptographic technique. The encryption of data is done to protect it from any security breaches, data theft and unauthorized access or alterations. PPS-ADS have adopted Twofish because of the following reasons:

- Uncomplicated and straightforward implementation.
- Intricate Internal Architecture.
- Variable and large size Key Support (from 128 to 256 bits).
- Highly Optimized Architecture.
- Support for API and plugins for various platforms.
- Freely available.
- Extremely fast and highly secure.
- Can be hardcoded in smart cards, microprocessors and other dedicated miniature hardware.
- Accepts all standard encryption modes.
- Allow various performance tradeoffs on encryption-speed, hardware gate count and memory usage.

Detailed information about the working of Twofish can be found in ([151], [155-156], [179]).
6.7.3.1 Insertion of New data item in PPS-ADS

The various steps involved in data storage are given below:

```
Algorithm: PPS-ADS Insertion

1. FOR Every Request for Data Storage, DO
2. Check the Status of all the Coaches/DCs
3. IF Status of the DC/Coach is not Full
4. Retrieve the distance of DCs/Coaches whose status is not Full
5. Identify and Select the DC/Coach which is nearest to the PC
6. Identify and Select the optimal route from PC to the selected DC
7. Forward the Data to the EDU using the above selected path
8. Encrypt the data element
9. Insert the Encrypted data element in selected DC/Coach using the pre selected optimal path.
10. Update the status of the DC/Coach
11. ELSE
12. Create a new DC/Coach
13. Encrypt the data element
14. Insert the encrypted data element in this DC/Coach
15. Update the status of this newly created DC/Coach
16. Notify about the Status and distance of this newly created DC/Coach to the PC
17. END FOR
```

6.7.4 Data Consistency and Reliability in PPS-ADS

In order to ensure secured and reliable delivery of data, PPS-ADS proposes two kind of acknowledgements. These are called by the name: “Forward Positive Acknowledgement” and “Backward Positive Acknowledgement”.
Whenever the data is arrived and stored successfully in the coach, the coach notifies the PC by sending an acknowledgement about the successful arrival and storage of data. This type of acknowledgement is known as “forward-positive acknowledgement”.

Once the PC receives the forward-positive acknowledgement from the Coach of the DC, it notifies the User/Client by sending an acknowledgement. This type of acknowledgement which is sent by the PC to the User/Client is known as “backward-positive acknowledgement”.

Both these acknowledgements must reach their respective destination within a stipulated amount of time, otherwise, the data is assumed to be corrupted and the client/user is required to resend the data. This 'stipulated time' is dynamically calculated with respect to the type of data being transferred. The concept of acknowledgements ensures all time reliability and consistency of the data.

### 6.7.5 Data Availability in PPS-ADS

For maintaining all time availability of the data items, the data-blocks in the coaches are replicated and stored at multiple sites (locations).

The information about the replicated data items in a coach is stored in DC’s metadata. The redundancy of data items ensures higher availability in cases of network or DCs failure. Since ADS houses several coaches there are chances that a coach may fail or loses connectivity. Data replication ensures that the data is available all the time to the users even in such unfortunate but possible cases.

For ensuring higher availability of data items, PPS-ADS works as follows:

It is the duty of each DC to notify the PC about its latest capacity after a prefixed amount of time. After analyzing this information, the PC takes the decision about the number of replicas of each data block. The PC enforces the restriction to ensure that there must be exactly three replicas of each data block at any given time (which is analogous to Hadoop). This deliberate redundancy of data blocks ensures higher data availability irrespective of any failures since the same copy of the data is stored at multiple locations in different DCs. In case, any DC goes down or fails (temporarily or permanently), the client’s requests can be served by any other DC consisting of the relevant information about the request.
6.7.6 Data Encryption in PPS-ADS

The process of encryption and decryption of data elements in PPS-ADS is handled by EDU. The EDU adopts Twofish algorithm to perform the task of encryption and decryption. The following steps show the working of Twofish algorithm in PPS-ADS ([151], [155-156], [179]).

1. The input from the pilot computer is divided into data blocks of 128 bits each.
2. Each of these 128-bit blocks are further divided into four parts P0, P1, P2, P3 of 32-bits each. These parts (P0, P1, P2, P3) along with the key 'K' serves as the input to the Twofish encryption unit (where the length of the key can be between 128 to 256 bits).
3. Let us consider the key 'K' of 128 bits. The key 'K' is also divided into four parts K0, K1, K2, K3 of 32-bits each and is XOR-ed with the four parts P0, P1, P2, P3 respectively to produce the output as R0, R1, R2, R3.
   i.e. $K_i \cdot XOR \cdot P_i = R_i$, for $i = 0,1,2,3$. This process is known as input whitening.
4. In the next step, the outputs R0 and R1 along with the round-number 'r' are passed through the 'F' function which results in F0 and F1 respectively.
5. R2 is XOR-ed with F0 and this result is rotated right by one bit to produce C2
6. R3 is rotated left by one bit and then XOR-ed with F1 to produce C3.
   This completes one round of Twofish. The results of round-one are R0, R1, C2, C3.
   These results are then exchanged in such a way that it gives the resultant as C2, C3, R0, R1, which is then sent to round two. The same process is continued for 16 rounds. The F function takes three inputs (R0, R1, r), where r is the round-number. It also consists of a sub-function known as g function (which is explained later).
7. R0 is passed to the g function to produce T0 as its output.
8. R1 is rotated to left by 8 bits and then passed to the g function, which gives T1 as its output.
9. Then T0 and T1 are mixed in a PHT to give T0' and T1'.
   [A PHT is a simple mixing operation and its value can be calculated as described below:]
   For any two inputs p and q in a PHT,
   $p' = p + q \mod 2^{12}$ and $q' = p + 2q \mod 2^{12}$
10. The results T0' and T1' are then combined with the two words of the expanded key (K_{2n+4} and K_{2n+5}), where r is the round-number, to give F0 and F1, which is the output of the F function.
The ‘g’ function constitutes the pivot of the Twofish algorithm. Here the input is divided into groups of 4 bytes and each byte is individually passed through the S-Boxes. The outputs of the S-Boxes are then passed through the MDS to produce a single resultant. Finally the output whitening is performed on the above resultant to produce the final encrypted data. Further details about S-Boxes and MDS can be found in ([151], [155-156], [179]). Once all the blocks of input data are encrypted, they are merged to get the final encrypted data which is then passed to the next unit for further processing. For the decryption of the data items, the above steps are performed in reverse order. After all the blocks of input data are encrypted, they are merged to get the final encrypted data which is then passed to the DCs for storage. For the decryption of the data items, the above steps are performed in reverse order.

### 6.7.7 Data Retrieval in PPS-ADS

In order to retrieve a data item, the legitimate user sends a request to the PC. On receiving the request for retrieval of any data item, the PC sends a broadcast message to all the DCs asking for the location and updated information about the block Id in which the requested data is stored. The DCs which contain the requested data, reply back to the PC with their location_Id and block_Id of the data item. After receiving the replies from the DCs, the PC selects the DC which is nearest and has minimum congestion and traffic in its route. Then, the PC retrieves the data item from the DC and forwards it to the EDU for decrypting it so as to convert it into its original form. After decrypting the data, it is finally sent to the user/client [4]. The formal steps involved in the process of data retrieval are given below:

1) **Retrieval of Data Element from the Coach/DC**
Algorithm: Data Retrieval

1. **FOR** every Retrieval request, **DO**
2. Find the location of the DC/Coach wherein the data is stored.
3. Find the location of the nearest Data-Block (in a DC/Coach) wherein the requested data item is stored.
4. Find the optimal path for reaching the DC (which holds the requested data) and back.
5. Fetch the data item from that Data-Block within the DC.
6. Forwards the retrieved data item to the EDU for decrypting it.
7. Decrypt the data item.
8. Send the data item to the client.
9. **END FOR**