A Hypothetical Model of Data Structures and Algorithms in the View of Modern Concurrent Environment

Shivshankar Kumar[1], Debjana De Biswas[2], Anirban Bhar[3], Suchismita Maiti[4]

[1,2] *B. Tech student, Department of Information Technology, Narula Institute of Technology, Kolkata, India.*
[3] *Assistant Professor, Department of Information Technology, Narula Institute of Technology, Kolkata, India.*
[4] *Associate Professor, Department of Information Technology, Narula Institute of Technology, Kolkata, India.*

ABSTRACT

Early computer scientists agreed on a straightforward standard model of computation and a fundamental canon of all-purpose algorithms and data structures that complemented it. Analysis of Data Structures and Algorithms has typically been theoretical and mathematical that limits the scope and scale of undertaking concepts. Concurrent data structure algorithms have drawn more interest recently as multi-core CPUs have proliferated. Concurrent data structures are substantially more challenging to design and validate as proper than their sequential equivalents due to a number of shared-memory multiprocessor characteristics. Software engineers use novel models of computation and real-world constraints to create efficient algorithms and data structures. Concurrency is the main cause of the new challenge in the distributed system. The creation of sophisticated algorithms with broad applicability is encouraged by these limits.

Keywords: *Concurrent data structure, Complexity, Shared memory, Real-time algorithms, Distributed system.*

1. INTRODUCTION

Early computer scientists agreed on a straightforward standard model of computation and a fundamental canon of all-purpose algorithms and data structures [1],[2]. A specific method of storing and organizing data such that it can be accessed by numerous computing threads (or processes) on a computer is known as a concurrent data structure. The practice of concurrent programming has undergone substantial modifications as a result of the widespread use of commercial shared-memory multiprocessor systems. The traditional computing model has a single processor that executes one instruction at a time, a large amount of memory that is uniformly accessible with minimal latency, easily accessible persistent storage with significant latency, input and output that is insensitive to content or context and orders of magnitude slower, resources and connections that do not change, and little concern for energy use or component failure, either intermittent or permanent. The practice of concurrent programming has undergone substantial modifications as a result of the widespread use of commercial shared-memory multiprocessor systems. These devices will inevitably become more prevalent as low-cost chip multithreading (CMT) becomes more popular. Shared-memory multiprocessors are computer systems that run several threads of computation simultaneously. These threads interact and synchronize using shared memory data structures. It is much more difficult to design and ensure the accuracy of concurrent data structures than it is for their sequential equivalents. The difficulty of concurrency is made more complex by the asynchronous nature of threads, which are vulnerable to page faults, interruptions, and other problems. Multithreaded programs require synchronization to ensure thread-safety by coordinating the concurrent accesses of the threads in order to manage the complexity of concurrent programming. To take use of the parallel processing capabilities of modern architectures, numerous operations must be let to go simultaneously and finish without interruption.

2. PREVIOUS WORK

Lists, quicksort, trees, the Boyer-Moore string search, hash tables, and other algorithms are examples of canonical data structures and algorithms. Even before that, Knuth started what he meant to be a comprehensive encyclopedia of computers [3]. There haven't been any new and improved sort procedures that are also broadly applicable, despite the editors of the Dictionary of Algorithms and Data Structures receiving them frequently [4]. The skip list, developed in 1989, is the newest all-purpose data structure. Specialized fields like graphics, distributed systems, user interaction, cyber-physical systems, and artificial intelligence are still under development [5]. General textbooks follow the same model and cover the same material, with the possible exception of quantum computing methods.
Memory blocks can only be deleted up to 100,000 times before there are too many errors [6]. The entire memory block is then rendered useless. Many flash memory subsystems employ sophisticated algorithms to distribute writes across memory in order to increase their usable life while preserving the abstraction of permanent memory that is insensitive to write locations. This is frequently done using log-structured or journaling file systems, whose write patterns closely resemble flash memory patterns [7]. Utilizing data structures designed specifically for flash storage, like [8] or [9], may allow the software developer to achieve additional efficiency. Consideration of the device's features can increase solid state disc responsiveness by 44% [6].
Similar challenges and opportunities are brought about by multicore machines, which enable parallelism on every desktop, distributed computing, which allows for the quick use of hundreds of machines, and cloud computing, which makes extra processing always available for a fee but requires some time to request and provision additional resources. Furthermore, applications that consider input as a stream of values to be processed rather than a static set of data make the maximum use of micropower and ubiquitous sensors. The relevance of power and thermal awareness in computing is rising in both small mobile devices and massive server systems. Today, despite speed-of-light delays, we want computer systems dispersed throughout the entire planet to work as a single, cohesive unit, which frequently necessitates careful caching [10].

3. DATA STRUCTURES

Many alternative structures can be used to arrange data. A "data structure" is a logical or mathematical representation of a specific data organization. The data must be organized using an effective data model. Two factors determine the specific data model to use. The structure must be both complex enough to reflect the links between data in the real world and simple enough to allow for efficient data processing when necessary. The choice of a particular data structure depends mainly on how frequently certain operations are carried out in a given circumstance. Traversing, searching, inserting, removing, and a few unique operations like merging and sorting are the most frequently used operations on data structures. Without using data structures in their algorithms, the aforementioned action increases in time complexity, introduces numerous errors and flaws, and leaves the process with numerous unresolved issues. This demonstrates a negative impact on the system's performance. The creation of effective algorithms is the response to all of these issues. By reducing the complexity of their algorithms, data structures that are appropriate for the scenario are adopted, which results in effective data administration and maintenance and making efficient use of system resources. The operational tasks of practically every program or software system use data structures. Data structures, as opposed to algorithms, are sometimes emphasized in programming languages as the primary organizing principle in software development. We feel it is important to clearly categorize the different categories of data structures before beginning the overview of data structures and their applications. Primitive and non-primitive data structures are included in the standard classification of data structures [11]. The categories of data structures and their subcategories are depicted in the following Figure 1.

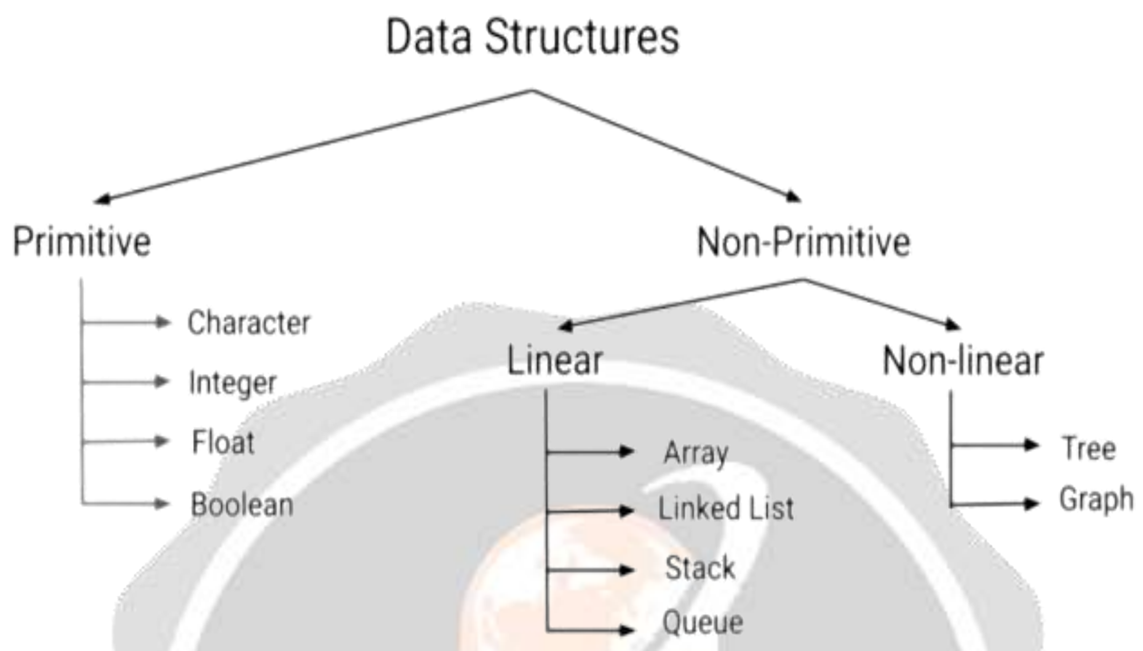

Fig -1: The data mining tasks

As was indicated in the section above, a data structure is a specific format for organizing and storing data. The array, the file, the record, the table, the tree, and other common data structure types are only a few examples. Any data structure is made to arrange data in a way that serves a specific purpose and can be accessed and used in the right ways. A data structure may be chosen or created in computer programming to store data in order to process it using different methods. The only data structures that are directly modified by machine instructions are primitive ones. Integers, real data, logical data, character data, and pointer data are examples of primitive data structures. The operating system defines storage structures for various sorts of data with help from primitive data structures. Comparatively, data types differ from one another in terms of their storage structures, as do the operating systems and software platforms [12].

Non-primitive data structures are those that are manually created using algorithms to meet any application requirements rather than being deployed directly by machine level instructions. Linear data structures and non-linear data structures are the next categories for non-primitive data structures. A concept known as the principle of alignment is used to define the relationship between linear and non-linear data structures. Whether data is stored next to each other or not is determined by the alignment principle. Lists, queues, stacks, unions, files, and other linear data structures are just a few examples.

It is possible to create linear data structures in memory as a continuous arrangement of data elements. It can be built using the array data type. The relationship of adjacency is preserved between the Data elements in linear data structures.

The most common types of non-linear data structures are trees and graphs. A non-linear data structure can be created by joining together a number of randomly dispersed sets of data items with the aid of a unique pointer. The relationship of adjacency between the Data elements is not maintained in a non-linear data structure.

Dictionary, skip list, hash table, heap, leftist tree, winner tree, loser tree, balanced search tree such AVL tree, Red-black tree, splay tree, B-tree, etc. are some examples of the more complex data structures.

The following is a discussion of some of the significant applications of multiple data structures in the field of computer science.

3.1 Arrays

Arrays are homogeneous, linear data structures that order data objects in a continuous block of memory in a sequential order [13]. A number of identical pieces of information can be stored in an array. Arrays are thought of as the foundation of data structures. All of the fundamental and sophisticated data structures may be implemented using arrays, which oftentimes makes code simpler. Arrays can occasionally even complete jobs that are difficult to complete using other techniques. In addition to its repetitive and conditional processing capabilities, an array can be quite powerful and have a wide range of uses.

Applications:

The array is comparable to a universal data structure that can be used to generate any other data structure.

- The arrays are particularly effective for searching and finding a specific target value, such as the maximum, minimum, mean, median, average, or count from a collection of data objects.
- It is simpler to perform operations like sorting, combining, traversing, and retrievals when things are arranged at evenly spaced and sequential addresses in computer memory.
- An indexable variable can be specified in a computer language using the array data type.
- Matrix-based numerical representations are simple to store in a computer's memory, making it possible to solve a variety of challenging mathematical problems and perform image processing transformations.

3.2 Stacks

The stack data structure, which receives values in last-in, first-out order, is homogeneous, linear, and recursive. As a result, they are known as LIFO lists. The PUSH operation is represented by adding an element to the stack, and the POP operation is represented by removing an element from the stack. A stack is a data structure with restricted access, allowing only additions and deletions at the top. Consider a stack of books; you can only remove the top book, and you can also place a new book on top of the stack as an analogy. Reversing a word is the simplest use of a stack. You add letters one at a time to a particular word, then remove them from the stack.

Applications:

- The compiler and operating system can keep local variables used inside a function block on a stack, where they can be removed once control leaves the function block.
- In text editors, a stack can be used as "undo" mechanism; this function was made possible by storing all text changes in a stack.
- Stacks are in handy when you need to obtain the most recent data piece in a list of items, a process known as "backtracking."
- Parsing is a crucial area in which stacks are used.
- It can be used to execute recursive functions and handle function calls. The final executed function will return the result by popping the stack after pushing the function's return values and addresses into memory.
- Internal stacks are used in language processing to generate room for parameters and local variables. Stack is used to implement the syntax check for matching braces in the compiler.

3.3 Queues

Queue is a linear and homogenous data structure in which insertions and deletions are done at different ends in First in first out order (FIFO). Hence, they are called FIFO lists [14]. In queue all the elements are inserted at rear end whereas all deletions are performed at another end called as front end. General applications of queues include transport, and operations research where various entities such as data, objects, persons, or events are stored and held to be processed later. In these contexts, the queue performs the function of a buffer.

Applications:

- Queues are frequently used in programs, where they are implemented as abstract data structures, as data structures with access functions, or as classes in object-oriented programming languages. Linked lists and circular buffers are common implementations.
- In the operating system, interrupts can be stored in queues.
- An application program makes use of it to store incoming data.
- Operating systems execute synchronization via queues.
- Both CPU job scheduling and disc scheduling require queues.

This data structure is typically used to simulate a variety of operating system capabilities, such as multiprogramming platform systems and various scheduling methods that employ queues. Queue data structures are used by printer server routines and other software programs.

Example:

Queues are utilized to accomplish the round robin technique. It serves the time-sharing system specifically. These algorithms are implemented using the circular queue.

3.4 Linked Lists

A linked list is a type of linear data structure made up of a collection of nodes, each of which has information and a link to the node after it in the list. A linked list arranges data on a storage device based on th e logical, rather than the physical, order of the data.

The most practical linear data structure is the linked list, which finds utility in a wide range of system operations.

Applications:

- Linked lists are employed in dynamic memory management operations like run-time memory allocation and release.
- To execute operations like addition and multiplication with polynomials, a linear link list can be used to represent and manage a polynomial.
- In Symbol Tables, linked lists are used to balance parentheses and express sparse matrices.
- Implementing a linked list of file names, undo functionality in Photoshop, would be a relatively excellent choice.
- Using the browser's cache, it is possible to implement a linked list of URLs by pressing the BACK button.
- Stacks, hash tables, and binary trees are examples of data structures that can be built using doubly linked lists.
- Unbounded memory in a system is readily managed using a linked list.

Creating hash tables for collision detection across communication channels, structuring binary trees, constructing stacks and queues in programming, and maintaining relational databases are some typical uses of linked lists.

3.5 Trees

For grouping data objects according to keys, a tree data structure is a potent tool. It can also be used to arrange various data elements according to hierarchical relationships. When data is keyed or has internal structure that enables one piece to be associated to, or "preserved within," another, tree structures are a great alternative to arrays.

Applications:

- The phrase structure of sentences, which is essential for language processing tools, is represented by trees. By examining the program's words and attempting to construct the program's parse tree, the Java compiler examines the grammatical structures of the Java program. When the parse tree is correctly built, the Java compiler uses it as a guide to create the byte code that can be found in the program's class file.
- In many search applications, such as the maps and set objects of various language libraries, where data is constantly added to and removed, trees are employed.
- File folders serve as the nodes of a tree that an operating system maintains on a disc. The tree structure is advantageous since it allows for the simple creation and deletion of folders and files.
- Major compilers can validate syntax using tree data structures, and they can also be used to implement sorted dictionaries.
- The usage of trees is widespread in internet protocols and has many uses in computer networking. For storing router tables, trees can be utilized as every high-bandwidth router.
- Trees can be used for quick traversals and searching of directory structures in the system. Trees can be searched using Dijkstra's algorithm to discover nodes that are closest to the router for the shortest paths.

Trees come in a variety of types and variations, and they can be used in a number of different fields.

3.6 Graphs

A graph is a type of data structure made up of a limited number of ordered pairings of objects known as nodes or vertices and edges or arcs.

A graph can either be directed from one vertex to another or be undirected. Graphs can be used to represent and resolve a variety of real-world computer science issues. Computer science makes extensive use of graph theoretical concepts [15]. Graphs make it much simpler and easier to model data structures using vertices and edges to address issues like resource allocation, scheduling, graph coloring, resource networking, database design, network topologies, etc.

Applications:

- A directed graph could be used to depict the link structure of a website; the web pages that are available there make up the vertices, and a directed edge only connects page A and page B if and only if page A has a link to page B.
- Jobs are supposed to represent the vertices of the graph, and there will be an edge between two jobs that cannot be done simultaneously and a one-to-one relationship between viable scheduling of graphs. This notion of graph coloring can be used to CPU task scheduling difficulties.
- Similar to this, graphs make it simple to address the problem of simultaneous job execution between a set of processors and a set of jobs.

4. CONCURRENCY ALGORITHM IN DATA STRUCTURE

Blocking data structures and algorithms are those that synchronize the data by means of mutexes, condition variables, and futures. The application uses library functions to pause a thread's execution while waiting for another thread to finish its task. Such library calls are referred to as blocking calls because, until the block is removed, the thread cannot carry on. A blocked thread will typically be totally suspended by the OS (and its time slices will be given to another thread) until the proper action of another thread, such as unlocking a mutex, alerting a condition variable, or making a future ready, unblocks it.
Nonblocking refers to data structures and algorithms that don't make use of blocking library functions. However, not all of these data structures are lock-free, so let's examine the different kinds of nonblocking data structures.

4.1 Lock-free data structures

A data structure must permit concurrent access from several threads in order to be considered lock-free. A lock-free queue might let one thread push and another pop, but it will crash if two threads attempt to push new items simultaneously. They don't even have to be able to do the same tasks. Additionally, the other threads must be able to finish their actions without waiting for the suspended thread if one of the threads accessing the data structure is suspended by the scheduler mid-operation.

4.2 Wait-free data structures

A wait-free data structure is a lock-free data structure with the added feature that every thread accessing the data structure can finish its operation in a finite number of steps, regardless of how other threads behave. Therefo re, algorithms that may require an infinite number of retries due to conflicts with other threads are not wait-free.

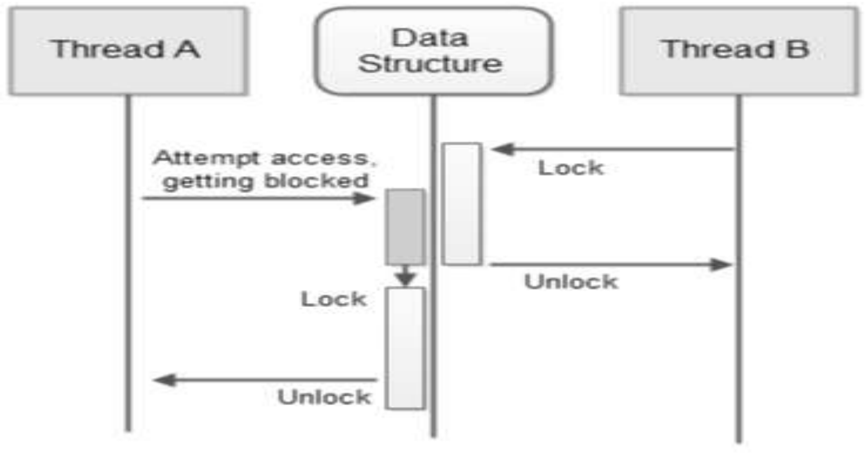

Fig -2: Concurrency Algorithms

5. CONCLUSION

According to concurrency management methods like blocking and non-blocking, algorithms are divided into different categories. The latter can be lock-free, wait-free, or obstruction-free while the former is predicated on locks. Finally, it is clear that the lock-free strategy performs better than the locking-based approach. Novel, elegant, widely applicable algorithms and data structures may result from new computing limitations and opportunities, such as highly parallel, cloud, and energy-aware processing. We demand a resurgence of resources and enthusiasm in the study of fundamental algorithms and data structures. In order for programmers to appreciate the advantages and necessity of such advancements, according to Kamp, they should be taught basic cache-aware data structures and algorithms. More importantly, these data structures and algorithms must be as easily accessible from standard libraries as the current "standard model" techniques. Professionals that are busy rarely take the extra time or effort to find the best tool for the job if a suitable instrument is already available.

6. REFERENCES

[1]. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, McGraw-Hill, 1990.
[2]. G. V. Neville-Neil, "The data-structure canon," Comm. Of ACM, vol. 53, no. 4, pp. 33–34, April 2010.
[3]. D. E. Knuth, "Artistic programming," Current Contents, no. 34, August 1993.
[4]. Dictionary of algorithms and data structures. P. E. Black, ed. [Online]. Accessed 21 April 2020.
[5]. W. Pugh, "Concurrent maintenance of skip lists," April1989. Tech. Report CS-TR-2222, Dept. of Computer Science, U. Maryland.
[6]. L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," In Proc. 42nd Ann. IEEE/ACM Int'l Symp. On Microarchitecture (MICRO 42), pp. 24–33, New York, NY, USA, 2009.
[7]. E. Gal and S. Toledo, "Algorithms and data structures for flash memories," ACM Computing Surveys, vol. 37, no. 2, pp. 138–163, June 2005.
[8]. C.-H. Wu, T.-W. Kuo, and L. P. Chang, "An efficient B-tree layer implementation for flash-memory storage systems," ACM Trans. Embedded Computing Systems, vol. 6, no. 3, art. 19, July 2007.
[9]. N. M. M. K. Chowdhury, M. M. Akbar, and M. Kaykobad, "Disktrie: An efficient data structure using flash memory for mobile devices," In M. Kaykobad and M. S. Rahman, editors, Proc. First Workshop on Algorithms and Computation (WALCOM), pp. 76–87, 2007.
[10]. Cloudflare, Network latency, in How to Make the Internet Faster for Everyone [Online]. Accessed 24 August 2020.
[11]. http://people.cis.ksu.edu/~schmidt/300s05/Lectures/Week 7b.html
[12]. http://en.wikipedia.org/wiki/Data_structure
[13]. Zaizai., "Advanced Array Applications in Clinical Data Manipulation", AstraZeneca Pharmaceuticals David Shen, WCI, Inc.
[14]. Steve First, Teresa Schudrowitz., "Arrays Made Easy: An Introduction to Arrays and Array Processing", Systems Seminar Consultants, Inc., Madison, WI.
[15]. S.G.Shirinivas, S.Vetrivel, Dr. N.M.Elango.,"The applications of graph theory in computer science an – overview", International Journal of Engineering Science and Technology, Vol. 2(9), 2010, 4610-462.