Atlantic International University
A New Age for Distance Learning

OKELLO THOMAS

STUDENT ID: UB69639SSO78773

PROGRAM

BACHELOR'S IN SOFTWARE ENGINEERING

COURSE NAME

(DISTRIBUTED SYSTEMS DEVELOPMENT)

ATLANTIC INTERNATIONAL UNIVERSITY

SUBMISSION PERIOD: FEBRUARY / 2022

Table of Contents

1.0 Introduction

Distributed computability of a networked system of computers is a computer science and obviously software engineering related field which deals with the study and implementation of distributed systems. In software development for large, complex, and distributed systems, agent systems are seen as the next revolution (Paprzycki, Ganzha, & Essaaidi, 2012).

A distributed system has its components located on several computers which are networked. The size of a distributed system varies from a LAN (Local Area Networks) through MAP (Metropolitan Area Networks) to WAN (Wide Area Networks).

They communicate and coordinate their activities by exchanging messages within this interconnected system of computers according to the design goal (Wikipedia, Wikipedia, 2022). A distributed system is therefore a collection of autonomous computers which are linked by a network utilizing software to generate an integrated facility of computation.

A distributed program is a computer program which runs within a distributed system and distributed programming is the process of writing those programs. Distributed computing is employing distributed systems to resolve computing instances. High performance for shared memory multiprocessor parallel computing, employs parallel algorithms while distributed algorithms coordinate large scale distributed systems. Distributed computing is the simultaneous utilization of more than a computer to resolve a computational problem (Pierfederici, 2016).

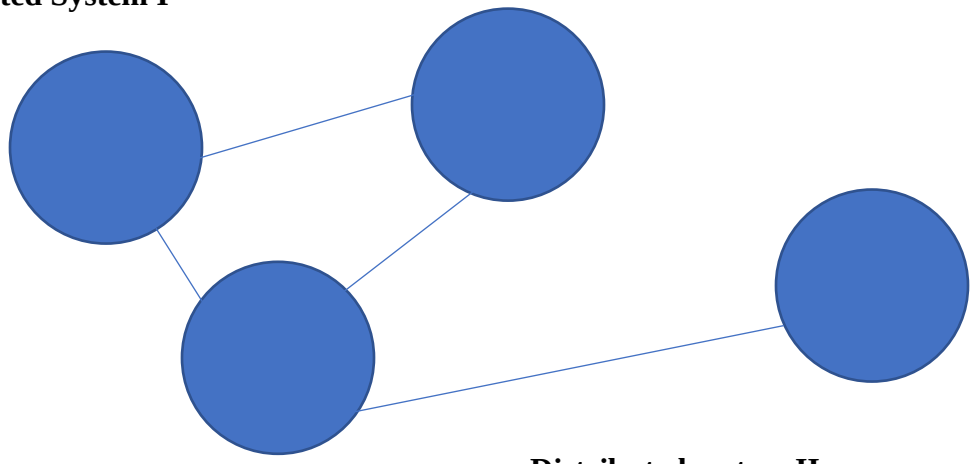
2.0 Distributed and Parallel Computing Systems

There is no clear distinction between parallel and distributed computing but possibly one can define and classify concurrent systems as parallel or distributed by using the parameters below:

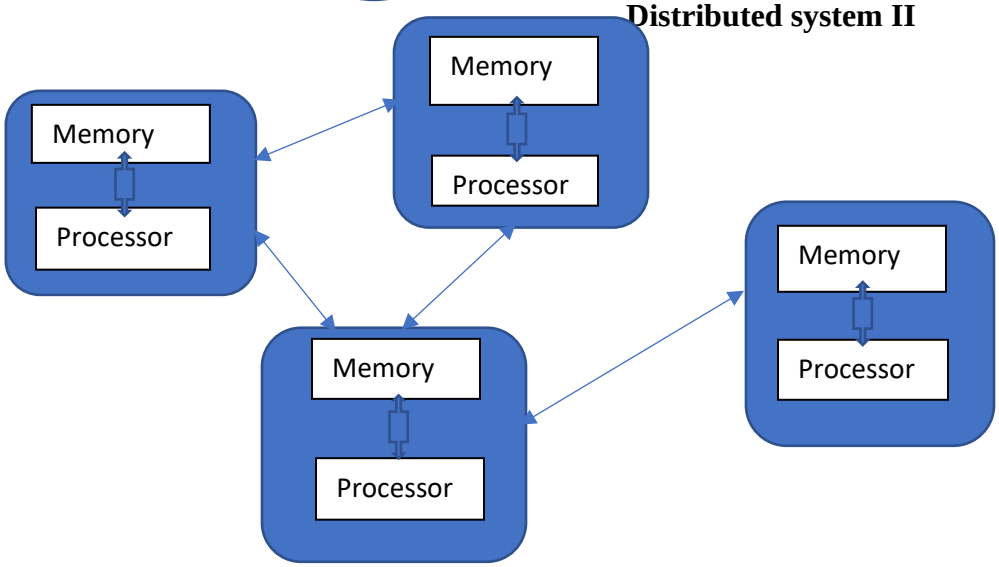
- For parallel computing systems, all processors may have access to a shared memory for information exchange between processors.
- For distributed computing system, each processor has access to its individual memory known as **distributed memory** for information exchange between processors.

This can be depicted by the figures below showing differences between parallel and distributed systems of computing.

Distributed System I

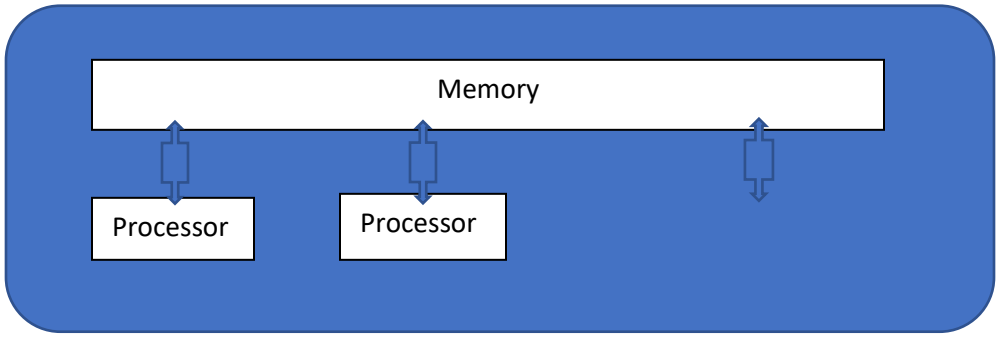


Distributed system II



Distributed system I is depicted as a network topology where individual nodes are computers and the connections between the nodes are communication links. In more detail, **distributed system II** is depicted where each computer is localized with its memory and through the communication links information can be exchanged between the nodes.

Parallel System



The **Parallel system** above depicts individual processors directly accessing a shared memory. High performance in parallel computing where multiple processors have direct access to shared memory employ parallel algorithms while large scale coordination of distributed systems utilize distributed algorithms.

2.1 Distributed System Properties

The following are important attributes of distributed systems:

- Sharing of resources and resource managers
- Openness
- Concurrency
- Scalability

- Fault-tolerance
- Transparency

2.2 Reasons for Deploying Distributed Systems and Computing

Reasons may include the following amongst others:

If running an application which requires the use of a communication network that connects several computers. The nature of this application being developed requires a distributed system. For instance, computers producing data in each physical location and required in another location. Example: instant messaging and video conferencing applications.

It is cost effective to obtain the desired level of performance by utilizing a cluster of several low-end computers compared to a single high-end computer. A distributed computing system provides more reliability and expandability than a non-distributed computing system as it presents failure tolerance and relatively easy to extend and manage. Example: rendering of 3D animation movies like Pixar and DreamWorks.

2.3 Distributed Computing with Python

Threads are typical tools employed in parallel applications. On Python systems, threads present considerable limitations prompting programmers to launch subprocesses by forking. These subprocesses substitute or complement threads by running alongside the main application process. There are two architectural techniques employed to overcome these thread limitations.

- Multithreaded programming
- Multiprocessing

The two techniques are subject of choice depending on the desired level of computability. This means that, there are several situations where multiple processing is preferred to multiple threading and vice versa. These computability executions both run in a single computer as it has been ascertained earlier that there is no clear distinction between parallel and distributed computing.

An example of a shared-memory architecture is a **multithreaded application,** and a distributed-memory architecture is a **multiprocessor application**. CPU-GPU combination is yet another example of an interesting distributed system where graphics cards are very sophisticated computers in their own ways.

For many compute-intensive problems, CPU-GPU interactions generate a highly parallel and compelling performance in addition to displaying images on the screen. For programmers to best-employ GPUs for general purpose-computing, tools and libraries are available.

3.0 Distributed Systems and Computing Applications

Examples of distributed systems and applications includes amongst others, the following (Kuzmiakova, 2020):

- **WAN (Network) applications such as:**
- Multimedia information access and conferencing applications
- Multimedia/teleconferencing over networks
- The world wide web and peer-to-peer networks
- Electronic mail
- Bulletin board systems
- Netnews (group discussions on single subject)

- Gopher (text retrieval service)

❖ **Distributed Unix software**

❖ **Telecommunication Networks**

- Telephone networks and cellular networks
- Wireless sensor networks
- Routing algorithms

❖ **Network Applications (Distributed information processing systems)**

- Banking Systems
- Airline reservation system

❖ **Real-time process control systems**

- Aircraft control systems
- Industrial control systems

❖ **Parallel Computation.**

- Scientific computing, cluster computing, grid computing and cloud computing
- Distributed rendering in computer graphics

4.0 Sharing of resources and resource managers

Sharing of computer hardware and software resources encompasses components such as disks and printers. Software resources include components such as files, windows, and data objects. Hardware resource sharing is useful for convenience and cost reduction while software sharing like for instance data sharing is for consistency in libraries and compilers, information exchange in databases and cooperative work such as groupware.

A set of resources is managed by a software module. Server processes in a client-server framework function as resource managers for a set of resources and clients. Every resource is linked to its own management policies and methods. Consider the following analytical insight of resource sharing and management.

Openness

An open distributed system has published specifications. Uniform interprocess communications and published interfaces for access are provided. All vendor conformity to published standards must be tested and certified if users are to be protected from the responsibility of troubleshooting system integration problems.

Concurrency

This is where multiple programs and processing are running. This can be parallel executions in a distributed system. Many users accessing the same resources, applications and multiple servers responding to client requests.

Scalability

This how the system responds to expansion. The software should be resilient and robust to handle growth and therefore centralisation to support scalability should be avoided.

Fault-tolerance

Due to computer failure, hardware redundancy and software recovery is necessary.

Transparency

Transparency of the following key factors:

- Access
- Location
- Concurrency
- Replication
- Failure
- Migration
- Performance
- Scaling

The key design goals of the distributed systems include:

- High performance
- Reliability
- Scalability
- Consistency
- Security

5.0 User Requirements

The user requirements specification includes the following:

Functionality

Minimally, the distributed system services should be accessed through the functions of single computer such as hardware resource sharing and use of distributed resources for parallel processing and fault-tolerance cooperative working environments.

Reconfigurability

Distributed systems can be subjected to short- and long-term changes and hence must be corrected accordingly. This can be accomplished by exploring a few options briefly mentioned below.

Through troubleshooting, replacement, advancement of services, changed roles of machines and resources on existing machines.

Such dynamics for short term may include:

- Failed processes or components
- Computational load shift
- Network overhead increase

For general long-term changes may include replacement and advancement of new machines and services including machine role changes and resource changes on existing machines.

Quality of Service

Needs of the users should be well defined and guaranteed through, distributed system computing performance, reliability, availability, and security.

5.1 Architectural Frameworks

Variations of hardware and software frameworks exist and can be employed for distributed computing. Lower levels necessitate interconnecting multiple CPUs with the designed network framework disregarding the fact that a network is printed onto the circuit board or a build-up of connected devices and cables. Higher levels necessitate interconnecting processes that are running on those CPUs with the required communication system.

Several architectural frameworks on which distributed programs can be installed and run, exist. They are outlined and briefly explained below:

- Client-server model
- Three-tier model
- n-tier model
- Peer-to-peer model

Client-server model is an architectural framework where the client computer communicates to the server, and it responds to the client by displaying the output to the users.

Three-tier model is a framework which initiates the stateless clients to be utilized by moving the client intelligence to the middle tier and this simplifies application deployment for most web applications as they are three-tier.

n-tier model is a framework which typically are web applications that execute their functionalities by forwarding their requests to the application servers.

Peer-to-peer model is a model whereby peers (computers) serve as both clients and servers and tasks are uniformly distributed amongst the computers called peers. Examples are BitTorrent and bitcoin network.

A central database architectural framework is an alternative that can be employed by utilizing a shared database to enable distributed computing to be executed without any form of direct interprocess communication. Centralised database architecture offers a relational processing analytics in a schematic architecture allowing for a live environment transmission enabling distributed computing functionalities both within and beyond the settings of networked database.

6.0 Computability and Computational Complexity Theories

Computability and computational theories are a computer science concepts which aid in understanding how computational problems known as problem instances can be resolved by efficiently employing a computer. Algorithms are computer programs designed to generate a correct solution to computational problems.

These computer programs run on the general-purpose machines or computers by reading the problem instance from input, performs computational activities and generates appropriate solution as output. Turing machines are random access machines employed as abstract models of a sequential general-purpose computer that can execute an algorithm.

The following section will discuss specifically issues on multiple computers although these issues are the same for concurrent processes running on one computer. The enlisted algorithm frameworks below, will be briefly discussed.

- Shared-Memory framework parallel algorithms
- Message-Passing framework parallel algorithms
- Message-Passing framework distributed algorithms

Shared-Memory framework parallel algorithms

The designer of the algorithm will choose the program executed by each processor and all processors access the shared memory. Parallel random-access machines (PRAM) are a theoretical model that is employed, and additionally classical PRAM framework takes synchronous access to the shared memory.

If the underlying **operating system encapsulates** the **communication** between **nodes and virtually unifies the memory** across all **individual systems**, then shared-memory programs(parallel systems) can be extended to distributed systems.

Compare-and-Swap (CAS) is such an example of a model of asynchronous shared memory which is closer to the behavior of real-world multiprocessor machines with the capability of considering the use of machine instructions.

Message-Passing framework parallel algorithms

The designer of the algorithm chooses the network topology and the program executed by each computer. Boolean circuits and sorting networks are examples of such models that can be employed. A Boolean circuit can be viewed as a computer network where each gate is a computer that runs a completely simple computer program and similarly a sorting network can be viewed as a computer network where each computer is a comparator.

Message-Passing framework distributed algorithms

The designer of the algorithm only chooses the computer program, and this same program runs on all computers. Regardless of the network topology, the system must work correctly. A graph is a commonly employed framework with one finite-state machine per node. For distributed algorithms, problem instances are typically related to graph which gives a description of the computer network topology.

There is considerable interaction between distributed and parallel algorithms although the two algorithms have different points of focus. For instance, the Cole-Vishkin algorithm employed in

graph coloring, initially was presented as a parallel algorithm but can also be used as a distributed algorithm technique.

The implementation of parallel algorithm can be achieved in parallel systems using shared memory or distributed systems using message passing. The boundary between parallel and distributed algorithms (take a suitable network versus a run in any given network) lies in a different place compared to the boundary between parallel and distributed systems (take a shared memory versus message passing).

6.1 Measures of complexity

In addition to time and space, the number of computers is considerably another resource but always there is a trade-off between the number of computers and the running time. The problem instance can be resolved faster if more computers are running in parallel and if the decision problem is resolved in polylogarithmic time using a polynomial number of processors, then the problem is in class **NC (Nick's Class)** (Wikipedia, Wikipedia, 2022). The class NC can be defined by using the PRAM formalization or Boolean circuits. Boolean circuits can be efficiently simulated by PRAM machines and vice versa.

Emphasis is usually put on communication operations other than computational steps. Probably, the simplest framework of distributed computing is **asynchronous system** where all nodes operate in a **lock state setting** commonly referred to as **local model**. During every communication round, all nodes in parallel will:

- Receive the latest messages from their neighbors
- Perform arbitrary local computations

- Send new messages to the neighbors

In this kind of system, the number of synchronous communication rounds required to complete the task is the central complexity measure. This complexity measure is closely related to the diameter of the network. Considering D to be the diameter of the network in one situation, any problem instance can be solved **trivially** in a synchronous distributed system in the approximation of 2D communication rounds by gathering all information in one location(D-rounds), resolve the problem and gives feedback to each node about the given solution.

By considering the running time of the algorithm in another situation, to be smaller than the D communication round, the network nodes will generate their output without having the information about distant parts of the network and so network nodes will make consistent global decisions based on the available information in their local D-neighborhood.

Most distributed algorithm run time, are smaller than the D-rounds and one central research question in this field is understanding which problem can be resolved by such algorithms. In a network size, a typical algorithm which resolves a computational problem instance in a polylogarithmic time is efficient in this model.

Regarding communications complexity, the total number of bits transmitted in the network, is another commonly used measure. This concept is common with the congest (B) model also defined as the local model but where single messages can only contain the B bits.

6.2 Related Computational Instances

From the traditional dimensions of computational instances, a user asks a question, a computer or distributed system processes the question, generates an answer, and terminates the process but

there are other problem instances where the distributed system is required to continuously coordinate the use of shared resources so that conflicts or deadlocks do not occur. Such problem instances include:

- Dining philosophers' problems is a problem example used concurrent algorithm design to illustrate synchronization issues and issue-resolving techniques (Wikipedia, Wikipedia, 2022).
- Mutual exclusion problem is concurrency control property meant to prevent race conditions. When the concurrent thread of execution is already accessing the critical section, another thread of execution should never enter this critical section (Wikipedia, 2021).

Distributed computing is also subject to other major challenges such as those related to fault tolerance and includes amongst others:

Consensus problem resolution in distributed and multi-agent computing is achieving overall system reliability in the presence of several faulty processes by coordinating processes to agree on some data value that is needed during computation (Wikipedia, Wikipedia, 2022).

Byzantine fault tolerance is distributed computing system condition where components may fail and there is no perfect information on failure (Wikipedia, Wikipedia, 2022).

Self-stabilization is a distributed computing system fault-tolerance concept that given any initial state, a self-stabilizing distributed system will end up in a correct state in a finite number of steps of execution (Wikipedia, Wikipedia, 2022).

Considerable research work is being carried on the asynchronous nature of distributed systems and covering concepts such as:

- Synchronizers which are employed in asynchronous systems to run synchronous algorithms.
- Synchronization of clock algorithms which provide a global consistent physical time stamp.
- Logical clock application provides a casual happened-before event ordering.

7.0 Coordinator Election

This is the process of assigning a particular process to be the organizer of a given task distributed amongst several nodes. Prior to running the task, all network nodes neither know which node will serve as the leader of the task nor communicate with the current coordinator. After running the leader election algorithm, all the network nodes are reorganized, and a particular unique node is recognized as the task coordinator = leader election.

For the network nodes to decide on which one of them will enter the **coordinator state,** the network nodes, use some method to communicate amongst themselves to break the symmetry amongst them. As an example, nodes may have unique and comparable identities and the nodes will compare these identities. The node with the highest identity assumes the coordinator role.

In distributed computing, coordinator algorithms are designed to be cost effective in terms of transmission of total bytes and time. The algorithm for general undirected graphs as suggested by Gallager, Humblet and Spira presents a strong impact on the design of distributed algorithms which won the **Dijkstra Prize**.

8.0 Conclusion

Distributed computing focusses on application of a designed computational algorithm to solve a computation problem in relatively efficient way. Decision can be taken on which algorithm to be employed for execution in large networks as there is no efficient centralized, parallel, or distributed algorithm.

A distributed system may be subject to common goal like solving a large computational problem where users view the collection of autonomous processors as a single unit. A computer may have a user specified needs and the distributed system will be designed to coordinate the utilization of shared resources by offering communication services to the users.

The most complex structure of the IEC 61499 architecture of distributed systems is system configurations. In this context, the description of system configuration is a set of devices(instances of predefined device types) populated by functionality blocks in one or several applications. It encompasses device types of instances along with block applications functionalities. Application and system can be summarized as below:

Application = Functional Block Network

System = Communication Network + Devices + Process/Machines

9.0 Bibliography

Kuzmiakova, A. (2020). Computer Science and Web Technologies. Ashland: Arcler Press.

Paprzycki, M., Ganzha, M., & Essaaidi, M. (2012). Software Agents,Agent Systems and their Applications. Amsterdam: IOS Press.

Pierfederici, F. (2016). Distributed Computing with Python. Birmingham: UK : Packt Publishing.

Wikipedia. (2021, August 27). Wikipedia. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Mutual_exclusion

Wikipedia. (2022, January 28). Wikipedia. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Dining_philosophers_problem

Wikipedia. (2022, January 29). Wikipedia. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Consensus_(computer_science)

Wikipedia. (2022, February 1). Wikipedia. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Byzantine_fault

Wikipedia. (2022, January 30). Wikipedia. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Self-stabilization

Wikipedia. (2022, January 3). Wikipedia. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/NC_(complexity)

Wikipedia. (2022, January 12). Wikipedia. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Distributed_computing#Introductio