SALALE UNIVERSITY

COLLEGE OF NATURAL SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

INTRODUCTION TO DISTRIBUTED SYSTEMS

GROUP ASSIGNMENT

CoSc4038

GROUP 2

Name	ID. Number
1. LEUL TSEHAYE	RU/1516/14
2. TAMENE TILAHUN	RU/1482/14
3. ARSEMAABDISA	RU/2354/14
4. AMANUEL TUFA	RU/1971/14
5. AJEBA DERESA	RU/2394/14
6. MAHDI BUSERI	RU/2341/14
7. LEMI BEKELE	RU/0219/14

Submission date: 06/05/2025

Submitted to: *Mr. Nega F.*

Table of Contents

1.Define and explain global states?

Global State of a Distributed System is the general status or state of all the components of a distributed system at a given moment. Any component in a distributed system, which consists of multiple independent nodes or computers working together to complete a common operation, can theoretically possess a state or data. the world state is the sum total of all of these individual states at any given moment [1].

Understanding the world is crucial to ensuring the consistency, reliability, and correctness of operations in the distributed system because it allows for adequate coordination and synchronization of the system's components.

Description of Global States in Distributed Systems

A distributed system comprises a number of processes running on various machines and communicating with each other via message passing. In a distributed system, there is no shared memory or universal clock for processes and therefore, it is not simple to maintain the information concerning the global state of the system at some point in time. For debugging or comprehending such a system, we require the concept of a global state. Itis a snapshot of the entire distributed system at a given instant. It has two components:

Local State of Each Process: It is the internal state of a process, i.e., variable values, program counter, buffers, and any other information that describes what the process is doing at the moment.

State of Communication Channels: This includes all the messages enroute. In other, messages sent by one process but not yet received by the other.

As distributed systems are asynchronous (there is no shared clock to synchronize them), obtaining a global state is not straightforward. It is not possible or even feasible to halt all the processes at the same time. There are special snapshot algorithms like the Chandy-Lamport Algorithm that are used to record a consistent global state without pausing the system.

A consistent global state is desirable because it represents a potential actual state of the system. For instance, it must not indicate that a message has been received but not sent. It is essential to maintain this consistency for operations like:

- Checkpointing and rollback (storing and recovering system progress).
- Deadlock detection (detecting circumstances where processes are blocked waiting for one another).
- Termination detection (deciding when all processes have finished and there are no messages enroute).
- Debugging and monitoring (watching system behavior for errors or inefficiencies).

2. Explain distributed deadlock detection?

In a distributed system, processes run on different computers and interact by exchanging messages to each other. As with any computing platform, processes in distributed systems also have the tendency to need access to shared resources like files, printers, databases, or locks. A deadlock occurs when a group of processes are locked up, where each process waits for a resource possessed by another, leading to a circular waiting condition. When this happens in a distributed system, it is referred to as a distributed deadlock [2].

A distributed deadlock is a situation where two or more processes across multiple systems are blocked forever, each waiting for the other to release a resource. Unlike centralized systems, distributed systems do not have a global view of all processes and resource allocations, which makes detecting and resolving deadlocks more complex.

Key Concepts in Distributed Deadlock Detection

In distributed systems, deadlocks can be represented using a Wait-For Graph (WFG), where nodes represent processes, and edges indicate one process waiting for another's resources. A cycle in the WFG implies a deadlock. Since each site maintains its local WFG, these must be shared to detect global cycles in a distributed environment.

Approaches to Distributed Deadlock Detection

1. Centralized Approach: A single node collects WFGs from all other nodes, constructs a global WFG, and detects cycles. This is simple but not scalable and has a single point of failure.
2. Hierarchical Approach: Nodes are arranged in a hierarchy, with lower-level nodes reporting to higher-level nodes. This reduces load on a single node but can still have bottlenecks.
3. Fully Distributed Approach: Each node cooperates in deadlock detection. The Chandy-Misra-Haas algorithm uses probe messages to detect cycles. If a process receives its own probe, a deadlock is confirmed.

Deadlock Resolution

Once a deadlock is detected, it can be resolved by:

- Terminating processes in the cycle.
- Rolling back processes to a safe state.
- Resource preemption, where resources are forcibly reassigned.

3. Explain distributed termination detection?

Distributed termination detection is a significant issue in distributed computation, where one attempts to determine whether a distributed computation has terminated completely. In distributed systems, there are various processes executing concurrently on different machines or nodes, typically communicating through message passing. Unlike centralized systems, there is no global controller or monitor in distributed systems to know the global state of the computation. This renders termination detection extremely challenging because each process has only partial knowledge of the system state, and there is no global view present by default.

Termination detection, in plain terms, refers to being able to determine that all computation and communication in a system has ceased, and there will be no further computation or communication. Termination can be said to have happened if all processes within the system idle (i.e., they don't have an operation to carry out) and there are no messages in progress between any two nodes. It is not possible for the system to be terminating if one single message is still in transit on the network or if a process is still executing. Detection of termination therefore entails precise coordination across all the participant processes.

Termination can be classified into two types: local termination and global termination.

Local termination: refers to when a single process or a group of processes finish their designated tasks. This is not always the situation when the entire system is terminated since other processes within the system are either running or awaiting communication.

global termination: is achieved only when all the processes in the distributed system have completed their computation and are in the passive state, and there are no in-transit messages between any two processes. Only then can it be said that the distributed system has actually terminated.

Detection of worldwide termination is specially advanced in asynchronous systems, wherein messages can get delayed and the order of activity can be different from node to node. More importantly, since there is no world clock that can be used to synchronize activities or states of processes, systems have to rely exclusively on protocols of message passing to infer whether termination has been achieved. These protocols need to be constructed to avoid premature detection (announcement of termination when not finished) and are required to ensure correctness in the presence of network delays or partial failures.

Because of these challenges, there have been algorithms and methods uniquely designed to perform distributed termination detection. These include techniques such as token-based detection, control message propagation, and distributed snapshot algorithms, which all help to synchronize the processes and confirm that all of them have completed their task and the computation has ended appropriately. Proper termination detection is important for correctness in applications like

distributed databases, parallel processing, and cloud computing, where proper understanding of the system state is essential for efficient allocation of resources and correct execution.

4.Explain clocks, events and process states?

In distributed systems, where multiple processes run on separate machines and communicate over a network, it is a challenging but required task to know the order and relationship of events. Since there is no single global clock shared by all nodes, there is a need to employ special mechanisms so that a consistent and correct view of time and event order can be maintained. This is where events, process states, and logical clocks play the central role in ensuring coordination and correctness of distributed computations. Below we define those in one by one.

1. **Clocks in Distributed Systems**:

Clocks are used in distributed systems to provide proper event ordering on events occurring in different processes that are on different machines. Since there is no shared clock in such systems, and physical clocks of different nodes cannot be perfectly synchronized, distributed systems use logical clocks to assign timestamps to events. Logical clocks help identify the order of events rather than the exact time when they occurred. Lamport Clocks and Vector Clocks are two notable types of logical clocks. Lamport Clocks give a partial ordering of events by assigning increasing numbers to each event, but they do not specify whether two events are causally dependent. Vector Clocks, however, maintain a vector of integers, one for each process, so that the system can determine both the order and the causality between events. These clock mechanisms are crucial for coordination and consistency in distributed systems [3].

2. **Events in Distributed Systems**:

An event in a distributed system is any event or action taking place within a process while it is running. Events are the basic units of activity in a distributed system and can be grouped into three classes: internal events, send events, and receive events. Internal events are activities performed within a single process, such as calculations or variable updates. Send events happen when one process sends a message to another process, while receive events happen when a process gets a message. Because processes in a distributed system do not share memory or have a global clock, it is difficult to say the exact order of events. In order to address this, logical clocks are used to time-stamp every event so that the system is aware of the order and relationships between events even when events occur on different machines [4].

3. **Process States in Distributed Systems**:

Process states in a distributed system are the state or phase a process is in as it runs. As processes perform work and exchange messages with each other, processes transition from state to state. The main process state of running, waiting, and idle (or ready). A process runs instructions when it is in the running state. It is in the waiting state when waiting and suspended from running for an

event to occur, for example. waiting for a message sent by another process. The process is in an idle or ready state when the process is ready to run but not running on the processor. These states help the system control execution and communication efficiently, especially in distributed systems where coordination of processes is critical. By observing process states, the system can control resource allocation, communication, and synchronization efficiently between different nodes.

References

[1] 10 4 2025. [Online]. Available: https://www.geeksforgeeks.org/what-is-the-global-state-of-a-distributed-system/.

[2] 12 4 2025. [Online]. Available: https://www.geeksforgeeks.org/deadlock-detection-in-distributed-systems/.

[3] 12 4 2025. [Online]. Available: https://www.geeksforgeeks.org/clock-in-distributed-system/.

[4] 11 4 2025. [Online]. Available: https://www.bing.com/search?q=in+distributed+system+what+is+event&FORM=AWRE1.