**AOS ASSIGNMET**

**SUBMITTED TO :**

**Dr. Prashant Kumar**

**Assistant Professor**

**CSE Department**

**SUBMITTED BY :**

**Ankit Goyal**

**Roll No. : 17103011**

**Branch : CSE 3rd year**

**Group : G1**

1. **Give an Example to show that in the AND request model, false deadlocks can occur due to deadlock resolution in distributed systems. Can something be done about this or they are bound to happen??**

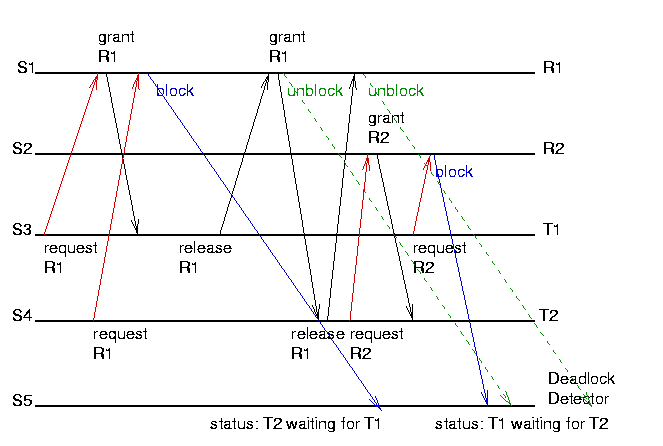
**AND Request Model**

* In the AND model, a process can request for more than one resource simultaneously and the request is satisﬁed only after all the requested resources are granted to the process.
* The out degree of a node in the WFG for AND model can be more than 1.
* The presence of a cycle in the WFG indicates a deadlock in the AND model.
* Since in the single-resource model, a process can have at most one outstanding request, the AND model is more general than the single-resource model.

**Example :**

Consider two transactions running parallel. If each site maintaining its own WFG locally and sending the information to the control site periodically, we may have false deadlocks:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *T*1: lock *R*1 | *T*2: lock *R*1 *T*1: unlock *R*1 | *T*2: unlock *R*1 *T*1: lock *R*2 | *T*2: lock *R*2 *T*1: unlock *R*2 | *T*2: unlock *R*2 |



The Chandy–Misra–Haas algorithm resource model is an edge chasing algorithm for [deadlock](https://en.wikipedia.org/wiki/Deadlock) in a [distributed system](https://en.wikipedia.org/wiki/Distributed_system). This algorithm does not report any false deadlock (also called phantom deadlock)

In edge chasing algorithm, a special message called *probe* is used in deadlock detection. A *probe* is a triplet *(i, j, k)* which denotes that process Pi has initiated the deadlock detection and the message is being sent by the home site of process Pj to the home site of process Pk.

The probe message circulates along the edges of WFG to detect a cycle. When a blocked process receives the probe message, it forwards the probe message along its outgoing edges in WFG. A process Pi declares the deadlock if probe messages initiated by process Pi returns to itself.

1. **Discuss the Chandi-Misra-Hass distributed deadlock detection algorithm. Discuss its complexity in terms of number of messages exchanged and delay in detecting the deadlock.**

**Chandy-Misra-Hass Detection Algorithm**

This is considered an edge-chasing, probe-based algorithm.

It is also considered one of the best deadlock detection algorithms for distributed systems.

If a process makes a request for a resource which fails or times out, the process generates a probe message and sends it to each of the processes holding one or more of its requested resources.

Each probe message contains the following information:

* the id of the process that is blocked (the one that initiates the probe message);
* the id of the process is sending this particular version of the probe message; and
* the id of the process that should receive this probe message.

When a process receives a probe message,

    it checks to see if it is also waiting for resources.

If not, it is currently using the needed resource

    and will eventually finish and release the resource.

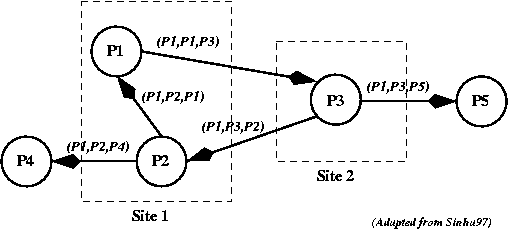
If it is waiting for resources, it passes on the probe message to all processes it knows to be holding resources it has itself requested.

The process first modifies the probe message, changing the sender and receiver ids.

If a process receives a probe message that it recognizes as having initiated,

    it knows there is a cycle in the system

    and thus, deadlock.



The advantages of this algorithm include the following:

* It is easy to implement.
* Each probe message is of fixed length.
* There is very little computation.
* There is very little overhead.
* There is no need to construct a graph, nor to pass graph information to other sites.
* This algorithm does not find false (phantom) deadlock.
* There is no need for special data structures

1. **Explain the following sentence: “Consistency, availability and performance tend to be contradictory forces in a distributed file system.”**

**Consistency** restricts the order in which reads and writes appear to occur. Formally, a

consistency semantics is a test on an execution—if the test for consistency C passes on an

execution, we say is C-consistent. An execution comprises of a set of nodes and a sequence of read and write operations at each node. We abstract the details of an execution and model read and write operations as follows: Write = (nodeId, objId, value, startTime, endTime) Read = (nodeId, objId, writeList, startTime, endTime) For a read operation thewriteList is a list of write operations that produced the values a read returns. Note that we permit a read operation to return multiple results, which, as we discuss below, provides a clean way to handle logically concurrent updates without making restrictive assumptions about conflict

resolution. The startTime and endTime fields indicate the real time at which an operation starts and finishes. Note that in an asynchronous system, this absolute global time is not visible to the nodes but our model includes it so that we can reason about semantics such as linearizability and real time causal, which restrict legal orderings to be consistent with real-time clocks. Note that we assume serial execution at each node so that one operation at a node ends before the next one starts. We say that a consistency semantics Cs is stronger than another consistency semantics C wiff the set of executions accepted by Cs is a subset of the set of executions accepted by Cw( ECS⊂ECW where EC denotes the set of executions accepted by a consistency semantics C). We say that two consistency semantics are incomparable iff neither of them is stronger.

**Availability**, informally, refers to an implementation’s ability to ensure that read and write

operations complete. The availability of an implementation is defined by describing the

environment conditions (network, local-clocks etc.) under which all issued operations complete. An implementation is always available if for any workload, all reads and writes can complete regardless of which messages are lost and which nodes can communicate.

**Performance** is measured as the average amount of time needed to satisfy client requests. This time includes CPU time + time for accessing secondary storage + network access time. It is desirable that the performance of a distributed file system be comparable to that of a centralized file system.

Replication is a technique frequently used to improve availability of data in distributed

file system. In replication, many replicas of data are stored at different servers. If one server fails, data still can be obtained from another server. The most serious problem here is how to keep replicas consistent and how detect inconsistencies in replicas of a file and recover them. Some typical situations cause inconsistencies in replica and replica is not updated due to failure of server storing it or may ne all file servers storing replica of a file are not

reachable. So, ironically, potential inconsistency problems may preclude file updates, thereby decreasing availability as level of replication is increased. This is why “ **Consistency, availability and performance tend to be contradictory forces in a**

**distributed file system.**”

1. **Explain the Sun file systems. Compare the sun file system with the Andrew file systems.**

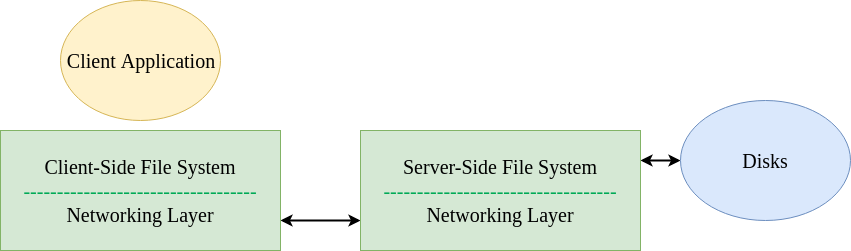
Network File System ( NFS ) is a distributed file system ( DFS ) developed by Sun Microsystems. This allows directory structures to be spread over the net- worked computing systems.

A DFS is a file system whose clients, servers and storage devices are dis- persed among the machines of distributed system. A file system provides a set of file operations like read, write, open, close, delete etc. which forms the file services. The clients are provided with these file services. The basic features of DFS are multiplicity and autonomy of clients and servers.

NFS follows the directory structure almost same as that in non-NFS system but there are some differences between them with respect to:

* Naming
* Path Names
* Semantics

NFS is a low-cost solution for network file sharing that is easy to setup as it uses the existing IP infrastructure. A significant advantage of NFS is that it allows for central management, decreasing the need for added software and disk space on individual user systems. NFS is user-friendly, allowing users to access files on remote hosts in the same way they access local files. This reduces the need for [removable media](https://searchdatabackup.techtarget.com/definition/removable-media) storage devices and increases security as fewer CDs, DVDs, [Blu-Ray](https://whatis.techtarget.com/definition/Blu-ray) disks, diskettes and [USB drives](https://searchstorage.techtarget.com/definition/USB-drive) are in circulation.



|  |  |
| --- | --- |
| **AFS** | **SFS** |
| AFS provide both location independencies and location transparency | SFS provides only location transparency.Callback: |
| AFS is implemented as two software components that exist at UNIX processes called Vice and Venus | SFS build on the Open Network Computing Remote Procedure Callsystem |
| AFS use disk caching that reduces file server and network load and server callbacks guarantee cache consistency and attributes are cached several hours then flushed to the disk. | NFS has memory caching with small buffers and attributes are cached between 3-30 seconds .There for time-based cache consistency may cause inconsistencies to occur. |
| AFS has stateful servers | NFS has Stateless server |
| AFS make use of KerBeros | Has three layer : Upper kernel layer, Lower kernel layer and Middle kernel layer |
| AFS provides excellent performance in wide-area configurations | NFS is inefficient in wide-area configurations |
| AFS uses the access control lists for fine tuning directory access and uses standard Unix mode | NFS uses Access control with standard UNIX mode bits on files and directories. |

1. **Why we need fault tolerance in distributed systems? What are the different failure models? What is the problem of consensus?**

Fault-tolerance or graceful degradation is the property that enables a system (often computer-based) to continue operating properly in the event of the failure of (or one or more faults within) some of its components.

Fault Tolerance is needed in order to provide 3 main feature to distributed systems:

1. Reliability-Focuses on a continuous service without any interruptions.
2. Availability - Concerned with read readiness of the system.
3. Security-Prevents any unauthorized access.

**Failure Models :**

1. **Crash failure** A server halts, but is working correctly until it halts
2. **Omission failure** A server fails to respond to incoming requests

* **Receive omission** A server fails to receive incoming messages
* **Send omission** A server fails to send messages

1. **Timing failure** A server's response lies outside the specified time interval
2. **Response failure** The server's response is incorrect

* **Value failure** The value of the response is wrong
* **State transition** failure The server deviates from the correct flow of control

1. **Arbitrary failure** A server may produce arbitrary responses at arbitrary times

The consensus problem requires agreement among a number of processes (or agents) for a single data value. Some of the processes (agents) may fail or be unreliable in other ways, so consensus protocols must be [fault tolerant](https://en.wikipedia.org/wiki/Fault_tolerant) or resilient. The processes must somehow put forth their candidate values, communicate with one another, and agree on a single consensus value.

The consensus problem is a fundamental problem in control of multi-agent systems. One approach to generating consensus is for all processes (agents) to agree on a majority value. In this context, a majority requires at least one more than half of available votes (where each process is given a vote). However, one or more faulty processes may skew the resultant outcome such that consensus may not be reached or reached incorrectly.

Protocols that solve consensus problems are designed to deal with limited numbers of faulty [processes](https://en.wikipedia.org/wiki/Process_(computing)). These protocols must satisfy a number of requirements to be useful. For instance, a trivial protocol could have all processes output binary value 1. This is not useful and thus the requirement is modified such that the output must somehow depend on the input. That is, the output value of a consensus protocol must be the input value of some process. Another requirement is that a process may decide upon an output a value only once and this decision is irrevocable. A process is called correct in an execution if it does not experience a failure. A consensus protocol tolerating halting failures must satisfy the following properties.

Termination

Eventually, every correct process decides some value.

Integrity

If all the correct processes proposed the same value v , then any correct process must decide v .

Agreement

Every correct process must agree on the same value.

Variations on the definition of integrity may be appropriate, according to the application. For example, a weaker type of integrity would be for the decision value to equal a value that some correct process proposed – not necessarily all of them. The Integrity condition is also known as validity in the literature.

A protocol that can correctly guarantee consensus amongst n processes of which at most t fail is said to be t-resilient.

In evaluating the performance of consensus protocols two factors of interest are running time and message complexity. Running time is given in [Big O notation](https://en.wikipedia.org/wiki/Big_O_notation) in the number of rounds of message exchange as a function of some input parameters (typically the number of processes and/or the size of the input domain). Message complexity refers to the amount of message traffic that is generated by the protocol. Other factors may include memory usage and the size of messages