ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE NAME: Distributed Systems

COURSE CODE: CSE7424

Assignment I

Submitted By:

- Khalid Mohammed
- PGR/22331/13
- Section 1

Submitted to: Dr. Ravindrababu
Submission Date: June 6, 2021

Adama, Ethiopia

1. How distributed computing systems are going to be evolved in future and explain it briefly mentioning/citing with proper references.

A distributed computing system is a software system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another from any system in order to appear as a single coherent system to the end-user.[1][2][3]. There are many alternatives for the message passing mechanism, including RPC-like connectors and message queues. The components interact with each other in order to achieve a common goal. The ultimate goal of distributed computing is to maximize performance by connecting users and IT resources in a cost-effective, transparent and reliable manner. [4] Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components. An important goal and challenge of distributed systems is location transparency[1].

Since the current computing/software paradigm relies on “shared memory”, it isn’t scalable but most physical systems work with message passing, so persuasion is needed to let users sacrifice one for the other. Due to the rapid progress in computer hardware, software, internet, sensor networks, mobile communications, and multimedia technologies, distributed computing systems have recently evolved drastically to improve and expand various applications with better quality of services and lower cost, especially those involving human factors. Besides reliability, performance and availability, many other attributes, such as security, privacy, trustworthiness, situation awareness, flexibility and rapid development of various applications, have also become important.

Some of trending(emerging) distributed computing technologies are:

- Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved "in parallel". There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Parallelism has been employed for many years, mainly in high-performance and distributed computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors.

- Another area of computing that is currently on demand for researches is quantum computing. Distributed quantum computing is also one of future evolvement of distributed system. A quantum computer is a computation system that makes direct use of quantum-mechanical phenomena, such as superposition and entanglement, to perform operations on data. Quantum computing is still in its infancy but experiments have been carried out in which quantum computational operations were executed on a very small number of qubits. An accelerated shift of enterprises from on premise to cloud-based solutions primarily accounts for hefty growth in demand for quantum computing. Quantum computing and distributed systems may eneter a mutually beneficial partnership in the future. On the one hand, it is much easier to build a number of small quantum computers rather that a single large one. The best results concerning some of the fundamental problems in distributed computing can potentially be drastically

improved upon by taking advantage of the superior resources and processing power that quantum mechanics offers. [3] It will be in the place for future computing since it works combined with other areas of computing like distributed quantum computing and quantum machine learning, this area of research is developing at such blazing speeds. Both practical and theoretical research continues, and many national governments and military funding agencies support quantum computing research to develop quantum computers for both civilian and national security purposes, such as crypto-analysis.[5]

- Microservice architecture is one of the distributed system future evolvement, every component technically can be replicated in order to balance the load of work as required. This helps increase the reaction's speed to demands and cope with the demands more precisely. In addition, this modularity also helps the system becomes more fault tolerant. That says, if one service faces with an issue, the rest of the system still operate independently. Using automated deployment, the faulty service may be replaced automatically. The replacement can also occur pre-emptively by keeping backup-instances of critical services running. Due to the high independency of microservices, it makes the technology stack and programming languages more flexible. [6]

- The Blockchain is another big thing currently dominating industries and applications. Blockchain technology which allows distributed, decentralized and secure transaction is popular at the present time. Not just cryptocurrencies but their tremors of presence could be felt in almost all major industries. From IoT to telecommunication network, aircraft control systems to scientific computing, they have counted their presence everywhere. Given their quantum of growth, the future of Blockchain based distributed systems seem promising. [7]

- Edge computing serves as the decentralized extension of the networks, data center infrastructures, or the cloud. Consequently, the edge can be a combination of virtualization, cloud, and colocation data center solutions that addresses the need for a decentralized data center or cloud. This edge computing approach is all about placing decentralized computing power closer to the point where data is generated. This addresses the needs of a wide variety of use cases across industries such as meeting the needs of building management, IoT, data distribution, application delivery, and complex-event processing among others. While the cloud is also a major part of that IoT/big data world, businesses require the means for gaining instantaneous access, fast data transport, and needed compute resources that are reliable. Colocation becomes the most efficient and flexible means to manage and analyze the enormous amounts of IoT sensor data for factories, supply chains, power grids, distributed products and even cities. Smart cities have now moved beyond the point of speculation in ways that integrate utilities, services, security, and transportation and much more under the IoT banner.[8]

- Some mobile devices also use the distributed technology resources to make such applications for mobile devices. An open source software platform for supporting Grid systems and applications Amoeba: A distributed operating system that is designed for distributed computing tasks. Green Tea Software: A java based P2P generic distributed network computing platform that transmits code and data on-demand to run on heterogeneous OS. There are some future prospects of Distributed computing in technological World.[9]

Some other notable thematic research topics in Distributed Systems and Cloud computing are:

- ✓ Distributed systems and networks.
- ✓ Distributed Data Clustering.
- ✓ Graphical interactive debugging for distributed systems.
- ✓ High-performance computing.
- ✓ Ubiquitous Computing.
- ✓ Decentralized and Autonomous Systems
- ✓ High Performance Computing

Distributed Computing System will serve and evolved in the long run. With the rapid development of various emerging distributed computing technologies such as Web services, Grid computing, and Cloud computing, computer networks become the integrant of the next generation distributed computing systems. Therefore, integration of networking and distributed computing systems becomes an important research problem for building the next-generation high performance distributed information infrastructure.[9]

In the near future, distributed application frameworks will support mobile code, multimedia data streams, user and device mobility , and spontaneous networking. Looking further into the future, essential techniques of distributed systems will be incorporated into emerging new areas, envisioning billions of communication smart devices forming a world-wide distributed computing systems several orders of magnitude larger than today's internet.

2. Write limitations of following technologies?

A) XML

- ✓ XML syntax is verbose compared to other text-based data transmission formats such as JSON.
- ✓ XML is not compact. There is no official scheme for compressing XML.
- ✓ XML syntax is redundant relative to binary representation of similar data, especially with tabular data.
- ✓ XML is not optimized for access speed. XML documents are meant to be completely loaded, and then used as a data source. The redundancy in syntax of XML affects efficiency as a result of higher storage and transportation cost when the volume of data is large.
- ✓ XML does not support array.

- ✓ The hierarchical model for representation is limited in comparison to an object-oriented graph
- ✓ Representation of overlapping(non-hierarchical) node relationships in XML is not straightforward(requires extra effort)
- ✓ XML file sizes are usually very large due to its verbose nature, it is totally dependent on who is writing it.
- ✓ XML does not have a default processing application thus; XML documents should be converted to HTML by a middleware adding extra overhead. XML is mainly focused on the structure of the content of the document not on the display itself.
- ✓ Distinguishing between XML content and attributes is not obvious thus making XML data structure design difficult
- ✓ Implementing and supporting XML namespaces creates complications in XML parser. The parser is required to do a syntax check every time it reads in the markup. XML parsers expect uncompressed text; should either be put up with large text files, or create a complex mechanism for compressing and decompressing on the fly, which will add to processing overhead.
- ✓ XML's depiction as "self-documenting" ignores critical ambiguities.
- ✓ The XML language has no predefined tags
- ✓ The tags (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document

B) **SOA**

- ✓ High Bandwidth Server: As web service sends and receives messages and information frequently so it easily reaches a million requests per day. So it involves a high-speed server with a lot of data bandwidth to run a web service.
- ✓ Extra Overload In SOA, all inputs square measures its validity before it's sent to the service. If you are victimization multiple services, then it'll overload your system with extra computation.
- ✓ High Cost: It is expensive in terms of human resource, development, and technology.
- ✓ XML format messaging: Messages are large in size due to which more bandwidth and resources are consumed.
- ✓ SOA is not suitable for deployment when an application is a stand alone entity, or does not require call based or message passing mechanism.
- ✓ SOA is not suitable for an application which does not provide full functionality or does not work as a complete system instead serve as a component and have limited scope.
- ✓ SOA does not support application that are tightly coupled
- ✓ It is not recommended to use SOA in a homogenous application environment.

- ✓ SOA face the problem of lack of a uniform testing framework. There are no tools that provide the required features for testing services in a service-oriented architecture.
- ✓ SOA is not suitable for an application that has short living time span
- ✓ Service oriented architecture is not suitable for applications which are based on heavy data exchange.
- ✓ Large upfront investment. SOA architecture is a great choice for further business development. It enables your team to work on several applications simultaneously at a smaller cost. However, its implementation is usually a pricey endeavor.
- ✓ Greater load and increased response time. In SOA, each interaction between services is followed by a full validation of all input parameters. As a result, the load gets extremely heavy, which in turn prolongs the response time.
- ✓ Vast variety of services. SOA has a very special approach to operation. Services constantly exchange messages while executing tasks. Consequently, the number of those messages gets overwhelming, and this peculiarity makes it difficult to ensure decent service management.
- ✓ Due to multiple points of failure, it is hard to determine the reliability of each point. Software debugging can affect both client and server side.
- ✓ Not scalable; It's hard to add a new application or require additional development.

C) SOAP

- ✓ When using standard implementation and the default SOAP/HTTP binding, the XML info set is serialized as XML. To improve performance for the special case of XML with embedded binary objects, the Message Transmission Optimization Mechanism was introduced.
- ✓ When relying on HTTP as a transport protocol and not using Web Services Addressing or an Enterprise Service Bus, the roles of the interacting parties are fixed. Only one party (the client) can use the services of the other. 5
- ✓ SOAP is less "simple" than the name would suggest. The verbosity of the protocol, slow parsing speed of XML, and lack of a standardized interaction model led to the dominance of services using the HTTP protocol more directly. See, for example, REST.
- ✓ SOAP uses serialization by value not by reference which causes a memory problem.
- ✓ SOAP clients do not hold any stateful references with respect to remote objects since it is stateless
- ✓ SOAP is based on the contract, so there is a tight coupling between client and server applications.
- ✓ SOAP is slow because payload is large for a simple string message, since it uses XML format.

D) Micro-services

- ✓ Increased deployment effort. The deployment of microservices entails a lot of work as all services you've included should be able to deploy/undeploy themselves. Another thing to keep in mind is that each service needs to be isolated in case of a crash.
- ✓ Complexity. The system puts a lot of stress on developers as it's easier to run a single program than to juggle different services. Some of this stress is mitigated with tooling, but as the program grows, it becomes harder to run it properly.
- ✓ Challenging testing. Unlike the monolithic system, MSA testers need to check every service for an update, which is rather time-consuming.
- ✓ Communication slow-downs. The more services are added to the system, the harder it becomes to handle errors. It gets especially difficult when distant calls are required.

3. A. Explain working principles of 2 phase locking and 3 phase locking?

A. Two-Phase Locking

Two-phase locking is used in databases to ensure serializability meaning that outcome of concurrent operations can be seen as the outcome of the same operations performed in some sequential order.

Two-phase locking defines how transactions acquire and relinquish locks. The two phases are:

Expanding (Growing) Phase: New locks on data items may be acquired but none can be released.
Shrinking Phase: Existing locks may be released but no new locks can be acquired.

Two types of locks are utilized by the basic protocol: Shared and Exclusive locks. Refinements of the basic protocol may utilize more lock types. Using locks that block processes, 2PL may be subject to deadlocks that result from the mutual blocking of two or more transactions. Cascading rollbacks is also another major drawback.

Disadvantages:

⇒ Performance Issues.

⇒ State Inconsistency

The two-phase locking protocol is governed by the following rules:

- ➢ Two transactions cannot have conflicting locks.
- ➢ No unlock operation can precede a lock operation in the same transaction.
- ➢ No data are affected until all locks are obtained; that is, until the transaction is in its locked point.

Two Phase Locking working principle

Let's see a transaction implementing 2-PL.

	T1	T2
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	…….	……
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10	…….	……

This is just a skeleton transaction which shows how unlocking and locking works with 2-PL. Note for:

Transaction T1:

- Growing Phase is from steps 1-3.
- Shrinking Phase is from steps 5-7.
- Lock Point at 3

Transaction T2:

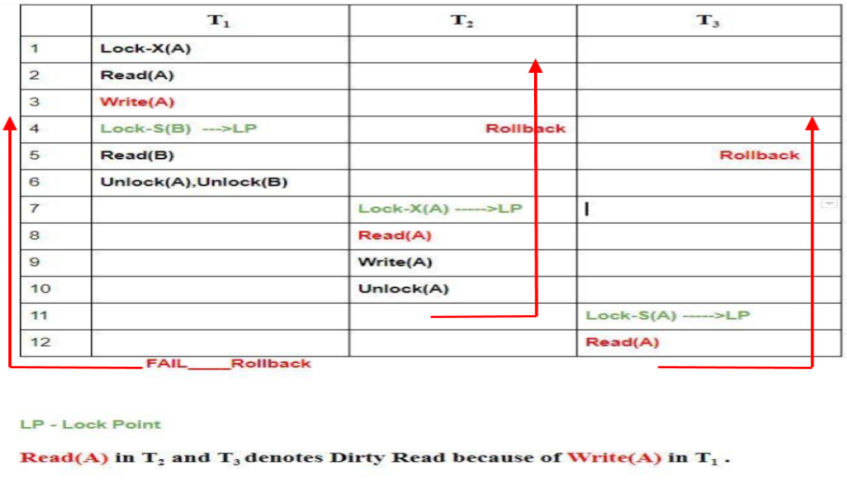
- Growing Phase is from steps 2-6.
- Shrinking Phase is from steps 8-9.
- Lock Point at 6

LOCK POINT is The Point at which the growing phase ends, i.e., when transaction takes the final lock it needs to carry on its work.

Although 2-PL ensures serializablity, but there are still some drawbacks of 2-PL.

- Cascading Rollback is possible under 2-PL.
- Deadlocks and Starvation is possible.

Cascading Rollbacks in 2-PL



LP - Lock Point

Read(A) in T_2 and T_3 denotes Dirty Read because of Write(A) in T_1 .

Because of Dirty Read in T2 and T3 in lines 8 and 12 respectively, when T1 failed, others have to be rollbacked also. Hence **Cascading Rollbacks are possible in 2-PL.** Here skeleton schedules is taken as examples because it's easy to understand when it's kept simple. When explained with real time transaction problems with many variables, it becomes very complex.

Deadlock in 2-PL –
Consider this simple example, it will be easy to understand.Say we have two transactions T1 and T2.

Schedule: Lock-X1(A) Lock-X2(B) Lock-X1(B) Lock-X2(A)

Drawing the precedence graph, the loop may be detected. So Deadlock is also possible in 2-PL.

Two-phase locking may also limit the amount of concurrency that occur in a schedule because a Transaction may not be able to release an item after it has used it. This may be because of the protocols and other restrictions one may put on the schedule to ensure serializability, deadlock freedom and other factors. This is the price to pay to ensure serializability and other factors, hence it can be considered as a bargain between concurrency and maintaining the ACID properties.

The above mentioned type of 2-PL is called **Basic 2PL**. To sum it up it ensures Conflict Serializability but does not prevent Cascading Rollback and Deadlock. Further we will study three other types of 2PL, Strict 2PL, Conservative 2PL and Rigorous 2PL.

Note – If lock conversion is allowed, then upgrading of lock(from S(a) to X(a)) is allowed in Growing Phase and downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

B. Three-Phase locking:

Two phase locking has performance issues when dealing with frequently-changing data in mukti-user environments. 2PL is probably subjected to deadlocks that result from the mutual blocking of transactions between multi-user(distributed) systems. It is designed as a non-blocking protocol as a contrary to the 2PC. Concurrency is also a major issue due to presence of multiple users. Moreover, the large number of data access contributes to the occurrence of deadlocks. Three phase locking is developed in order to solve these problems. It is a concurrency control method which divides the execution phase of a transaction into three parts.

a) Voting phase

b) Pre-commit phase

c) Global decision phase.

This locking method is not a necessity, thus not used a lot. It depends on the nature of the data(frequency of change) and the environment(multi-users). The types of locks used are Read, Write and Write Intent.

Disadvantages

- ✓ Multiple site failures.

Working Principle

- User signs in and requests data from the database
- Read lock is applied to the data
- Once the data is read, the read lock is dropped
- Therecan be multiple read locks by multiple users on the same data/tuple
- When the user indicates that he wishes to edit the data, takeout a write-intent lock
- UPDATE: write intent lock also known as chanfge lock or protect lock
- Other users can still obtin a read lock on the data
- Write lock allowed only to the user who has the write intent lock
- If data locked with write intent lock, then no further write intent locks can be applied on it
- When user finishes editing the data and submits the changes, write lock is applied on the data
- Write intent lock is unlocked
- Transaction is committed and write lock is unlocked

All locks are subject to timeouts, with appropriate actions(unlocks, error/warning messages to users etc) taken in the event of lock failure. This prevents deadlocks.

All access to the data in question should use the dame locking protocol. No other protocol should be applied or considered as it may disrupt the flow of operations.

UPDATE: only write intent locks can be held for any length of time, typically because the record of interest is at the mercy of the user in edit mode.

Most of RDBMS does not support the degree of control involveed here.

b. Explain the differences between Synchronous and Asynchronous messaging mechanisms

Synchronous messaging involves a client that waits for the server to respond to a message. Messages are able to flow in both directions, to and from. Essentially it means that synchronous messaging is a two way communication. i.e. Sender sends a message to receiver and receiver receives this message and gives reply to the sender. Sender will not send another message until it gets a reply from receiver. Synchronous messaging can be best understood as a live one-to-one conversation. A synchronous message has a defined beginning and end.

Asynchronous messaging involves a client that does not wait for a message from the server. An event is used to trigger a message from a server. So even if the client is down, the messaging will complete successfully. Asynchronous Messaging means that, it is a one way communication and the flow of communication is one way only. While synchronous messaging is analogous to a live person-to-person conversation, asynchronous communication doesn't require both parties to be present at the same time during transmission. This is suitable for situations in which messaging might be started, paused or resumed. Additionally, asynchronous messaging provides a better experience than synchronous messaging for complex issues that require more than one sitting to carry out effectively.

Synchronous Messaging	Asynchronous Messaging
Messaging method in which a continuous messages is accompanied by timing messages	Messaging method in which the sender and the receiver use the flow control method.
Synchronous handler do not return until it finishes processing the HTTP request for which it is called.	Asynchronous handler helps you to run a process independently of sending a response to the user.
Users need to wait until it sending finishes before getting a response from the server.	Users do not have to wait until sending completes before receiving a response from the server.
Messages are transmitted at high speed.	Messages are transmitted at low speed.
Is fast and costly.	Is slow and commercial.
The time interval of messaging is constant.	The time interval of messaging is random.
Does not have a gap between messages.	There is a gap between messages.
Does not need any local storage at the terminal end.	Requires local buffer storages at the two ends of the line to assemble blocks.
Should be performed when reliable and long-term storage is required.	Ideal for projects that span across long distances and have a very minimal budget.
Does not need any synchronized clocks.	Needs accurately synchronized clocks at both ends.

4. a. Describe the essentiality of transaction management and explain how transaction management works in distributed platforms.

Transaction management refers to the tasks of processing multiple transactions issued by various clients of a database server in such a way that the ACID contract can be fulfilled, that is, the properties of atomicity, consistency prevention, isolation, and durability of each individual transaction can be guaranteed.

Distributed transaction management deals with the problems of always providing a consistent distributed database in the presence of a large number of transactions (local and global) and failures (communication link and/or site failures). This is accomplished through (i) distributed commit protocol; (ii) distributed concurrency control techniques and (iii) distributed recovery method.

Databases are standard transactional resources, and transactions usually extend to a small number of such databases. In such cases, a distributed transaction may be viewed as a database transaction that should be synchronized between various participating databases allocated between various physical locations. The isolation property presents a unique obstacle for multi-database transactions.

For distributed transactions, each computer features a local transaction manager. If the transaction works at several computers, the transaction managers communicate with various other transaction managers by means of superior or subordinate relationships, which are accurate only for a specific transaction.

Resource managers handle consistent or resilient data and closely cooperate with the distributed transaction coordinator (DTC) to ensure an application's isolation and atomicity. In distributed transactions, every participating element should conform to committing a change action, such as a database update, prior to the transaction. The DTC coordinates the transaction for the participating components and works as a transaction manager for each computer that is meant to manage the transactions. When distributing transactions between various computers, the transaction manager delivers, prepares, commits and aborts messages to each subordinate transaction manager.

In the DTC's two-phase commit algorithm, phase one involves the transaction manager prompting commitment preparation of each enlisted component, whereas in phase two, if all components are prepared to successfully commit, the transaction manager messages the decision to commit.

There are also long lived distributed transactions, for example a transaction to book a trip, which consists of booking a flight, a rental car and a hotel. Since booking the flight might take up to a day to get a confirmation, two-phase commit is not applicable here, it will lock the resources for this long. In practice, long-lived distributed transactions are implemented in systems based on Web Services. Usually these transactions utilize principles of compensating transactions, Optimism and Isolation without Locking.

Several modern technologies, including Enterprise Java Beans (EJBs) and Microsoft Transaction Server (MTS) fully support distributed transaction standards.

Transaction Processing in a Distributed System

Remote SQL Statements

A remote query statement is a query that selects information from one or more remote tables, all of which reside at the same remote node.

A remote update statement is an update that modifies data in one or more tables, all of which are located at the same remote node.

Distributed SQL Statements

A distributed query statement retrieves information from two or more nodes. A distributed update statement modifies data on two or more nodes. A distributed update is possible using a PL/SQL subprogram unit such as a procedure or trigger that includes two or more remote updates that access data on different nodes.

Shared SQL for Remote and Distributed Statements

The mechanics of a remote or distributed statement using shared SQL are essentially the same as those of a local statement. The SQL text must match, and the referenced objects must match. If available, shared SQL areas can be used for the local and remote handling of any statement or decomposed query.

Remote Transactions

A remote transaction contains one or more remote statements, all of which reference a single remote node.

Distributed Transactions

A distributed transaction is a transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.

Two-Phase Commit Mechanism

A database must guarantee that all statements in a transaction, distributed or non-distributed, either commit or roll back as a unit. The effects of an ongoing transaction should be invisible to all other transactions at all nodes; this transparency should be true for transactions that include any type of operation, including queries, updates, or remote procedure calls.

The general mechanisms of transaction control in a non-distributed database are discussed in the Oracle Database Concepts. In a distributed database, the database must coordinate transaction control with the same characteristics over a network and maintain data consistency, even if a network or system failure occurs.

The database two-phase commit mechanism guarantees that all database servers participating in a distributed transaction either all commit or all roll back the statements in the transaction. A two phase commit mechanism also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers.

Database Link Name Resolution

A global object name is an object specified using a database link. The essential components of a global object name are: Object name, Database name and Domain. Whenever a SQL statement includes a reference to a global object name, the database searches for a database link with a name that matches the database name specified in the global object name.

The database searches for a database link called orders.us.acme.com. The database performs this operation to determine the path to the specified remote database. The database always searches for matching database links in the following order:

1. Private database links in the schema of the user who issued the SQL statement.

2. Public database links in the local database.

3. Global database links (only if a directory server is available).

Schema Object Name Resolution

After the local Oracle Database connects to the specified remote database on behalf of the local user that issued the SQL statement, object resolution continues as if the remote user had issued the associated SQL statement. The first match determines the remote schema according to the following rules:

- If the database cannot find the object, then it checks public objects of the remote database.
- If it cannot resolve the object, then the established remote session remains but the SQL statement cannot execute and returns an error.

Global Name Resolution in Views, Synonyms, and Procedures

A view, synonym, or PL/SQL program unit (for example, a procedure, function, or trigger) can reference a remote schema object by its global object name. If the global object name is complete, then the database stores the definition of the object without expanding the global object name. If the name is partial, however, the database expands the name using the domain of the local database name.

b. Explain working principles and limitation of leadership algorithms in distributed systems.

Leader election algorithm

A leader's election is a basic necessity for distributed systems. When a system is chosen as a leader, it should operate as a system's management; make final decisions and the like.[26] Leader election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Leader election algorithm basically determines where a new copy of coordinator should be restarted.

Leader election is a powerful tool for improving efficiency, reducing coordination, simplifying architectures, and reducing operations. On the other hand, leader election can introduce new failure modes and scaling bottlenecks. In addition, leader election may make it more difficult for you to evaluate the correctness of a system.[25]

Leader election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is send to every active process in the distributed system.

Leader election is a common pattern in distributed systems because it has some significant advantages:

- ✓ A single leader makes systems easier for humans to think about. It puts all the concurrency in the system into a single place, reduces partial failure modes, and adds a single place to look for logs and metrics.
- ✓ A single leader can work more efficiently. It can simply inform other systems about changes, rather than building consensus about the changes to be made.
- ✓ Single leaders can easily offer clients consistency because they can see and control all the changes made to the state of the system.
- ✓ A single leader can improve performance or reduce cost by providing a single consistent cache of data which can be used every time.
- ✓ Writing software for a single leader may be easier than other approaches like quorum. The single leader doesn't need to consider that other systems may be working on the same state at the same time.

Leader election also has some considerable downsides:

- ✓ A single leader is a single point of failure. If the system fails to detect or fix a bad leader, the whole system can be unavailable.
- ✓ A single leader means a single point of scaling, both in data size and request rate. When a leader-elected system needs to grow beyond a single leader, it requires a complete re-architecture.
- ✓ A leader is a single point of trust. If a leader is doing the wrong work with nobody checking it, it can quickly cause problems across the entire system. A bad leader has a high blast radius.
- ✓ Partial deployments may be hard to apply in leader-elected systems. Many software safety practices at Amazon depend on partial deployments, such as one-box, A-B testing, blue/green deployment, and incremental deployment with automatic rollback.

There are two leader election algorithms for two different configurations of distributed system.

1. The Bully algorithm

This algorithm applies to system where every process can send a message to every other process in the system.

Algorithm – Suppose process P sends a message to the coordinator.

i. If coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.

ii. Now P sends election message to every process with high priority number.

iii. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.

iv. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.

v. However, if an answer is received within time T from any other process Q,

vi. Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.

vii. If Q doesn't responds within time interval T' then it is assumed to have failed and algorithm is restarted.

Disadvantages

- ✓ It required that every process should know the identity of every other process in the system so it takes very large space in the system.
- ✓ It has high number of message passing during communication which increases heavy traffic, the message passing has order o (n2).

2. **Improved(Modified) bully algorithm**

The main concept of this algorithm is that the algorithm declares the new coordinator before actual or current coordinator is crashed. It overcomes the disadvantages of the original bully.

Algorithm:

This algorithm selects the coordinator before current coordinator is crashed. So it takes extra stages. In this algorithm before the coordinator is failed, the current coordinator tries to gather information about processes in the system and through the current coordinator, declares the next possible coordinator to the processes. With increasing knowledge and get the id of all other process, a process with the bigger id attempts to execute the bully algorithm. If the coordinator is failed, each process that notices this failure compares its id with the id which it has received via the coordinator. And select the new coordinator.

Disadvantages

- ✓ It has complex structure.
- ✓ Every time process updates its database.
- ✓ Large database required to maintain the information of each process in database of every process.

3. **Modified Election Algorithm**

The algorithm resolves the disadvantages of the bully algorithm.

Algorithm

i. When any process p notices that coordinator is not responding, it initiates an election and send election message to all process with higher priority number.

ii. If no process responds, process P wins the election and becomes new coordinator.

iii. Process with the higher priority sends ok message with its priority number to process P.

iv. When process p receive all the response it select the new coordinator with the highest priority number process and sends the grant message to it.

v. Now the coordinator process will broadcast a new coordinator message to all other process and informs itself as a coordinator.

Disadvantages

- ✓ A modified algorithm is also time bounded.
- ✓ It is better than bully but also has o (n2) complexity in worst case.
- ✓ It is necessary for all process to know the priority of other.

4. The Ring algorithm

This algorithm applies to systems organized as a ring (logically or physically). In this algorithm we assume that the link between the processes are unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is active list, a list that has priority number of all active processes in the system.

Algorithm

i. If process P1 detects a coordinator failure, it creates new active list which is empty initially. It sends election message to its neighbor on right and adds number 1 to its active list.

ii. If process P2 receives message elect from processes on left, it responds in 3 ways:

iii. if message received does not contain 1 in active list then P1 adds 2 to its active list and forwards the message.

iv. If this is the first election message it has received or sent, P1 creates new active list with numbers 1 and 2. It then sends election message 1 followed by 2.

v. If Process P1 receives its own election message 1 then active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects highest priority number from list and elects it as the new coordinator.

5. Modified Ring Algorithm

When a node notices that the leader has crashed, it sends its ID number to its neighboring node in the ring. Thus, it is not necessary for all nodes to send their IDs into the ring. At this moment, the receiving node compares the received ID with its own, and forwards whichever is the greatest. This comparison is done by all the nodes such that only the greatest ID remains in the ring. Finally, the greatest ID returns back to the initial node. If the received ID equals that of the initial sender, it declares itself as the leader by sending a coordinate message into the ring. It can be observed that this method dramatically reduces the overhead involved in message passing. Thus, if many nodes notice the absence of the leader at the same time, only the message of the node with the greatest ID circulates in the ring thus, preventing smaller IDs from being sent. If $n\{i1,i2,\cdots ,im\}$ is the number of nodes that concurrently detect the absence of the crashed coordinator and n is the number of nodes in the ring, then the total number of messages passed with an order of O(n2) is as follows:

$T = n\{i1i2,\cdots ,im\} \times n.$

5. a. What are different simulation frameworks available for distributed computing platforms to simulate, and compare and contrast its capabilities.

A simulation technique uses a probability experiment to mimic a real-life situation. Instead of studying the actual situation, which might be too costly, too dangerous, or too time-consuming, scientists and researchers create a similar situation but one that is less expensive, less dangerous, or less time-consuming.

1. SimGrid

SimGrid is a widely used simulation toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. The specific goal of the project is to facilitate research in the area of parallel and distributed large scale systems, such as Grids, P2P systems and Cloud. Its use cases encompass heuristic evaluation, application prototyping or even real application development and tuning. It thereby targets platforms that range from a simple network of workstations to large-scale computational grids. It allows users to easily develop simulation programs by providing APIs for common programming languages. SimGrid consists of three layers: the MSG layer which provides a programming API for users, a SIMIX layer located at the middle to provide synchronization and scheduling of process executions on simulated environment and finally a SURF layer which is a kernel that contains resources for each simulation task.

However, SimGrid have a scalability problem that impedes the simulation of large-scale systems such as grids. The size of the simulated networks is mainly limited by system memory. This memory limit is reached at roughly 4000 hosts on a machine with 8 GB of memory. Although scaling beyond this limit is possible through virtual memory, this results in extensive swapping which cripples the simulator's performance. The large memory requirements of SimGrid are due to the memory-intensive manner of storing the routing information that is used in the simulated network. Another but less important problem is due to the structure of the platform (description) files that describe the network topology in SimGrid. These files are larger than necessary, resulting in significant startup costs of the simulation and decreased manageability of those files.

2. CloudSim

Initial releases of CloudSim used SimJava as the discrete event simulation engine that supports several core functionalities, such as queuing and processing of events, creation of Cloud system entities (services, host, data center, broker, VMs), communication between components, and management of the simulation clock. GridSim is one of the building blocks of CloudSim. However, GridSim uses the SimJava library as a framework for event handling and inter-entity message passing. But in the current release, the SimJava layer has been removed in order to allow some advanced operations that are not supported by it. The CloudSim simulation layer provides support for modeling and simulation of virtualized Cloud-based data center environments including dedicated management interfaces for VMs, memory, storage, and bandwidth. The fundamental issues, such as provisioning of hosts to VMs, managing application execution, and monitoring dynamic system state, are handled by it. CloudSim is better at functionality correctness and memory utilization.

3. Simulation dynamic execution of distributed system

It used to simulate the dynamic execution process of distributed systems with data execution on computer nodes and data transfer along network links. It works in by taking two graphs and mapping scheme: a hierarchical computing modules (nodes) of a virtual workflow is represented by directed a cyclic graph, an overlay network represented by an arbitrary directed weighted graph and work flow mapping scheme that assigns a computing module to a computer node. Its performance is limited by un-optimized buffer size.

b. Describe the challenges and different solutions related to process synchronization in distribute systems

Synchronization in distribute systems deals how processes cooperate and synchronize with one another in a distributed system. In single CPU systems, critical regions, mutual exclusion, and other synchronization problems are solved using methods such as semaphores. These methods will not work in distributed systems because they

implicitly rely on the existence of shared memory.

The main challenge of Synchronization in distribute systems is to decide on relative ordering of events, since it is difficult to determine if events occur on different machines. The main issues raised here are:

I. Clocks synchronization

   How to synchronize events based on actual time?

   How to determine relative ordering?

II. Global State and Election

   What is the "global state" of a distributed system?

   How do we determine the "coordinator" of a distributed system?

III. Sharing in Distributed

   How do we synchronize for sharing

I. Clock synchronization in distributed system

Clock synchronization

Clock synchronization is a method of synchronizing clock values of any two nodes in a distributed system with the use of external reference clock or internal clock value of the node. There are 2 types of clock synchronization algorithms: Centralized and Distributed.

1. **Centralized**: is the one in which a time server is used as a reference.
   a. **Passive Time Server:** methods proposed to improve estimated time value.
      i. Additional information available
      ii. No additional information available (Cristian Method)
   b. Active Time Server

c. Berkeley Algorithm: this algorithm overcomes limitation of faulty clock and malicious interference in passive time server and also overcomes limitation of active time server algorithm. There are following limitations:

i. Due to centralized system single point of failure may occur.

ii. A single time-server may not be capable of serving all time requests from scalability point of view. The readjustment can either be +ve or –ve.

2. **Distributed**: is the one in which there is no centralized time server present. Instead the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like the scalability and single point failure.

a. Global Averaging Algorithm

b. Localized Averaging Algorithm

Achieving agreement on time is not trivial. Clock synchronization need not be absolute. If two processes do not interact, their clocks need not be synchronized. What matters is not that all processes agree on exactly what time is it, but rather, that they agree on the order in which events occur.

i. **Logical clocks**: for algorithms where only internal consistency of clocks matters (not whether clocks are close to real time).

ii. **Physical clocks**: for algorithms where clocks must not only be the same, but also must not deviate from real-time.

a) **Happens-before relation**

If two events happen in different processes that do not exchange messages, these events are said to be concurrent.

Capturing happens-before relation

- ✓ Each process has a local monotonically increasing counter, called its logical clock.
- ✓ Each event that occurs at process is assigned a Lamport timestamp.
- ✓ Happens-before is transitive, and clocks run at different rates.

b) Lamport's algorithm

Lamport solution: corrects the clocks.

i. Between every two events, the clock must tick at least once

ii. No two events occur at exactly the same time.

iii. Lamport Timestamps

iv. The updates have to be ordered the same way across the replicas.

v. This can be achieved by a totally ordered multicast algorithm.

vi. Totally ordered multicasting can be achieved by means of Lamport clocks.

II. Global State

Global state is local state of each process plus message in transit. It is useful to know the global state of a distributed system. Local state depends on the process

- Garbage collection
- Deadlocks
- Termination

Distributed Snapshot: represents a state in which the distributed system might have been.

Consistent global state: If *A* sends a message to *B* and and the system

records *B* receiving the message, it should also record *A* sending it.

Cut of the system's execution: subset of its global history identifying set of events

Election Problem

Many algorithms require one process to act as a coordinator. In general, it does not matter which process takes on this special responsibility, but one has to do it.

General Approach to elect a coordinator

Assumptions:

- ✓ Each process has a unique number to distinguish them, and hence to select one
- ✓ One process per machine: For simplicity
- ✓ Every process knows the process number of every other process
- ✓ Processes do not know which processes are currently up and which are down

Approach: locate the process with the highest process number and designate it as coordinator. Election algorithms differ in the way they do the location. Some are:

- ✓ The Bully algorithm
- ✓ Improved(Modified) bully algorithm
- ✓ Modified Election Algorithm
- ✓ The Ring algorithm
- ✓ Modified Ring Algorithm

III. Sharing in Distributed

Distributed Systems often face mutual exclusion problem. Systems involving multiple processes are often programmed using critical regions. When a process has to read or update certain shared data structure, it first enters a critical region to achieve mutual exclusion, i.e., ensure that no other process will use the shared data structure at the same time. In single-processor systems, critical regions are protected using semaphores, and monitors or similar constructs. To accomplish the same in distributed systems algorithms and protocols are employed.

Application-level protocol

- Enter critical section - block if necessary
- Access shared resources in critical section
- Leave critical section - other processes may enter

Requirements

- ✓ At most one process may execute in the critical section (safety)
- ✓ Requests to enter and exit the critical section eventually succeed(liveness)
- ✓ If one request to enter the critical section happened-before another, then entry to the critical section is granted in that order

Centralized Algorithm (Central Server Algorithm): one process is elected as the coordinator. Simplest and most efficient. A request and a grant to enter, and a release to exit. Only 2 message times to enter.

Advantages

- ✓ Obviously it guarantees mutual exclusion
- ✓ It is fair (requests are granted in the order in which they are received)
- ✓ No starvation (no process ever waits forever)
- ✓ Easy to implement (only 3 messages: request, grant and release)

Shortcomings

- ✓ Coordinator: A single point of failure; A performance bottleneck
- ✓ If processes normally block after making a request, they cannot distinguish a dead coordinator from "permission denied"

Distributed Algorithm: based on a total ordering of events in a system (happens-before relation). It requires *n*-1 request messages, one to each of the other processes, and an additional *n*-1 grant messages. Entry takes 2(n-1) message times assuming that messages are passed sequentially over a LAN

Token Ring Algorithm

If every process constantly wants to enter a critical region (each token pass will result in one entry). The token may sometimes circulate for hours without anyone being interested in it (number of messages per entry is unbounded). Entry delay varies from 0 (token just arrived) to *n*-1 (token just departed)

Pros:

- ✓ Guarantees mutual exclusion (only 1 process has the token at any instant)
- ✓ No starvation (token circulates among processes in a well-defifined order)

Cons:

- ✓ Regeneration of the token if it is ever lost:
- ✓ Recovery if a process crashes

6. a. Write different approaches to achieve high performance in distributed environments.

Distributed environments are characterized by sharing of resources including software, within a network of computers. Distributed environments are widely used in networks like the internet, intranet, telecommunications, and cellular phone networks and so on. One approach used to overcome these delays is replication wherein a replica of data is maintained near to clients so that travel distances can be minimized. Cluster computing, Grid Computing and Cloud Computing are the main approaches used for performance enhancement.

1. Cluster computing

It is a form of computing in which bunch of computers (often called nodes) that are connected through a LAN (local area network) so that, they behave like a single machine. A computer cluster help to solve complex operations more efficiently with much faster processing speed, better data integrity than a single computer and they only used for mission-critical applications.

A computer cluster defined as the addition of processes for delivering large-scale processing to reduce downtime and larger storage capacity as compared to other desktop workstation or computer. Some of the critical Applications of Cluster Computers are Google Search Engine, Petroleum Reservoir Simulation, Earthquake Simulation, weather Forecasting.

2. Grid computing

It is a group of computers physically connected (over a network or with Internet) to perform a dedicated tasks together, such as analyzing e-commerce data and solve a complex problem. Grids are a form of "super virtual computer" that solve a particular application. The grid size may vary from small to large enterprises network. A computing grid is constructed with the help of grid middleware software that allows them to communicate. Middleware is used to translate one node information passed stored or processed information to another into a recognizable format. It is the form of "distributed computing" or "peer-to-peer computing". Grid computing solve Challenging problems such as earthquake simulation and weather modeling.

'Grid computing' is distinguished from the cluster computing, because in Grid computing each node has heterogeneous and geographically dispersed (such as a WAN) and its own resource manager and perform a different task and are loosely connected by the Internet or low-speed networks, but in cluster computing resources are managed in a single location (Like a LAN).

3. Cloud computing

It is the delivery of different services through the Internet. These resources include tools and applications like data storage, servers, databases, networking, and software. Rather than keeping files on a proprietary hard drive or local storage device, cloud-based storage makes it possible to save them to a remote database. As long as an electronic device has access to the web, it has access to the data and the software programs to run it. Cloud computing is a popular option for people and businesses for a number of reasons including cost savings, increased productivity, speed and efficiency, performance, and security

b. Explain the limitations and capabilities of different distributed file systems.

A distributed file system (DFS) is a network file system wherein the file system is distributed across multiple servers. DFS enables location transparency and file directory replication as well as tolerance to faults. A Distributed File System provides a permanent storage in which a set of objects that exist from their explicit creation until their explicit destruction are immune to systems failures. This permanent storage consists of several federated storage resources in which clients can create, delete, read or write files. Unlike local file systems, storage resources and clients are dispersed in a network. Files are shared between users in a hierarchical and unified view: files are stored on different storage resources, but appear to users as they are put on a single location. A distributed file system should be transparent, fault-tolerant and scalable.

INFS: **NFS** stands for Network File System. It is a client-server architecture that allows computer users to view, store, and update files remotely. The NFS protocol is one of several distributed file system standards for **network-attached storage** (NAS).

CIFS: **CIFS** stands for Common Internet File System. CIFS is the accent of SMB. In other words, CIFS is an application of the SIMB protocol designed by Microsoft.

SMB: **SMB** stands for Server Message Block. It is a protocol for sharing files, invented by IMB. The SMB protocol was created to allow computers to perform read and write operations on files sent to remote hosts via a local area network (LAN).

In addition to the functions of the file system of a single-processor system, the distributed file system supports the following:

I. Remote information sharing
II. User mobility
III. Availability
IV. Diskless workstations

There are two important techniques for managing metadata for highly scalable file virtualization:

i. Separate data from metadata with a centralized metadata server
ii. Distribute data and metadata on multiple servers

Lustre: A File System for Cluster Computing

Lustre is a massively parallel, scalable distributed file system for Linux which employs a cluster-based architecture with centralized metadata. It is a scalable cluster file system designed for use by any organization that needs a large, scalable general-purpose file system. This is a software solution with an ability to scale over thousands of clients for a storage capacity of petabytes with high performance I/O throughput. It is basically a network-based RAID. Robustness isn't intriniscly a concern, because each node is internally robust, with its own storage based on a RAID or SAN, and its own back-up system. Lustre does provide tools to facilitate backup, repair, and recovery of the metadata server. Since even the temporary loss of this server can disable the entire system, Lustre supports a standby server, an always-available mirror.

With this approach, bottlenecks for client-to-OST communications are eliminated, so the total bandwidth available for the clients to read and write data scales almost linearly with the number of OSTs in the file system.

- ✓ Lustre uses read-only caching of data in the OSS that enhances performance when multiple clients access the same data set.
- ✓ It also provides other optimization features such as read-ahead and write-back caching for performance improvements.
- ✓ It can stripe files over multiple OSTs for better file I/O speeds. This striping support is a very good way of improving I/O performance for all files, since reading and writing simultaneously increases the available I/O bandwidth.
- ✓ It also supports an innovative file-joining feature that joins files in place, when the file is striped.
- ✓ It has a very good failover feature which adds to the availability of the solution.

GlusterFS

It is an open-source, distributed cluster file system without a centralized metadata server. It is also capable of scaling to thousands of clients, Petabytes of capacity and is optimized for high performance. It is accompanied by a web-based management interface and installer that makes it more suitable for cloud computing even from a user interface perspective.

GlusterFS employs a modular architecture with a stackable user-space design. It aggregates multiple storage bricks on a network (over Infiniband RDMA or TCP/IP interconnects) and delivers as a network file system with a global name space. Unlike Lustre, it does not employ a separate index of metadata. Instead it employs a new technique called Elastic Hash Algorithm, which avoids metadata lookup, adding to better performance. It is also different in its failover architecture, where every cluster is configured to be active and any file in the entire file system can be accessed from any server simultaneously.

The GlusterFS server exports storage volumes to remote clients and the GlusterFS client accesses remote storage volumes using POSIX interfaces. Gluster also employs a mechanism called **translators** to implement the file system capabilities. Translators are programs (like filters) inserted between the actual content of a file and the user accessing the file as a basic file system interface. Translators also allow optimization control at a much finer level as well.

Gluster performs very good load balancing of operations using the I/O Scheduler translators. Adaptive Least Usage (ALU) is one such translator wherein the GlusterFS balances load across volumes. Further, the optimized load is defined by the following “sub-balancers” and that enables one to fine-tune the load balancing feature as per the application-need:

- ➢ Disk-usage: Free and used disk space on the volume
- ➢ Read-usage: Quantum of read operations performed from this volume
- ➢ Write-usage: Quantum of write operations performed done from this volume
- ➢ Open-files-usage: Number of open files from this volume
- ➢ Disk-speed-usage: The disk's spinning speed

GlusterFS also supports file replication with the **Automatic File Replication**(**AFR**) translator, which keeps identical copies of a file/directory on all its sub-volumes

Self healing is another important feature supported by Gluster. In situations of data inconsistency across different copies of a file, the change log mentioned earlier is used to determine the correct copy version.

Gluster is therefore a potential technology that can be used to provide scalable, available and highly performing storage hardware for a cloud storage infrastructure. So, it is usually referred to as a cloud file system and has been used with RackSpace cloud-hosting solution.

General Parallel File System (GPFS)

IBM's General Parallel File System (GPFS) is the high-performance distributed file system developed by IBM that provides support for the RS/6000 supercomputer and Linux computing clusters. GPFS is a multiplatform distributed file system built over several years of academic research and provides advanced recovery mechanisms. GPFS is built on the concept of shared disks, in which a collection of disks is attached to the file system nodes by means of some switching fabric. The file system makes this infrastructure transparent to users and stripes large files over the disk array by replicating portions of the file to ensure high availability. By means of this infrastructure, the system is able to support petabytes of storage, which is accessed at a high throughput and without losing consistency of data. Compared to other implementations, GPFS distributes the metadata of the entire file system and provides transparent access to it, thus eliminating a single point of failure.

MogileFS: Serving Objects

The world will interact with the MogileFS only through the very rich interface of the Web site. It isn't necessary to supported nested directory trees to create a hierarchy for human convenience. Instead, it is only necessary to allow some simpler way to disambiguate files from different applications (or higher-level domains of some kind).

MogileFS works a lot like LustreFS. At a high level, it has the same idea; a distributed RAID. But, it is different in some of the details. It doesn't rely on each node providing robust storage, instead it replicates objects across servers. This reduces the cost of the storage by allowing less expensive components.

MogileFS uses HTTP to server objects from each replica, as opposed to a home-grown protocol, for portability. For the same reason, it keeps its metadata in a standard MySQL database. Since, unlike in Lustre, in-place writes aren't permitted, locks aren't very frequently needed, so the database can maintain a sufficiently high throughput.

It maintains simple namespaces, rather than directory trees. Similarly, the lack of a complex permission/ownership scheme, though less important, helps to keep things simple.

Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on hardware based on open standards or what is called commodity hardware. This means the system is capable of running different operating systems (OSes) such as Windows or Linux without requiring special drivers. HDFS has significant differences from other distributed file systems. It is designed to support Hadoop, an open framework for a specialized type of very scalable distributed computing often known as the Map-Reduce paradigm. It is not designed for user interaction. It is used for batch processing of applications that need streaming access to their datasets. The emphasis is on high throughput of data access rather than low latency of data access.

Distributed file systems constitute the primary support for data management. They provide an interface whereby to store information in the form of files and later access them for read and write operations. Among the several implementations of file systems, few of them specifically address the management of huge quantities of data on a large number of nodes. Mostly these file systems constitute the data storage support for large computing clusters, supercomputers, massively parallel architectures, and lately, storage/computing clouds.

7. a. Explain major distributed platform areas and its algorithmic strengths and weakness

Distributed platforms are an area of computer science which studies distributed systems. It is a system whose parts are located on varied computer networks, they coordinate and communicate their individual processes by exchanging messages with one another. All the components work with each another so that they can reach a common goal. The three outstanding characteristics of distributed systems include independent components failure, components concurrency and lack of a global clock.

1. Parallel computing

Parallel computing refers to the process of breaking down larger problems into smaller, independent, often similar parts that can be executed simultaneously by multiple processors communicating via shared memory, the results of which are combined upon completion as part of an overall algorithm. It includes cluster computing and grid computing.

2. Network computing

Network computing refers to the use of computers and other devices in a linked network, rather than as unconnected, stand-alone devices. Distributed databases and distributed database management systems, World Wide Web and peer-to-peer networks are some examples of network applications.

3. Telecommunication network

Referring to the social process of information exchange, and the transmission of information by various types of technologies over wire, radio, optical or other electromagnetic systems. Routing algorithms, cellular network and wireless sensor network are some its examples.

Strength

- Ring algorithm: When any process notices that the coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down, the sender skips over the successor and goes to the next number along the ring, or the one after that, until a running process is located.

Limitation

Global Averaging Algorithm: Due to large amount of messages network traffic increased. Therefore this algorithm may be useful for small networks only and it does not scale well.

- Active Time Server: Due to communication link failure message may be delayed and clock readjusted to incorrect time.

- Bully algorithm: It required that every process should know the identity of every other process in the system so it takes very large space in the system.

b. Describe the fault- tolerant solutions for distributed systems.

Fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail. The objective of creating a fault-tolerant system is to prevent disruptions arising from a single point of failure, ensuring the high availability and business continuity of mission-critical applications or systems. Any system or component which is a single point of failure can be made fault tolerant using redundancy.

Fault tolerance can play a role in a disaster recovery strategy. For example, fault-tolerant systems with backup components in the cloud can restore mission-critical systems quickly, even if a natural or human-induced disaster destroys on-premise IT infrastructure.

Load balancing and failover: fault tolerance for web applications

In the context of web application delivery, fault tolerance relates to the use of load balancing and failover solutions to ensure availability via redundancy and rapid disaster recovery. Load balancing and failover are both integral aspects of fault tolerance.

Load balancing solutions allow an application to run on multiple network nodes, removing the concern about a single point of failure. Most load balancers also optimize workload distribution across multiple computing resources, making them individually more resilient to activity spikes that would otherwise cause slowdowns and other disruptions.

In addition, load balancing helps cope with partial network failures. For example, a system containing two production servers can use a load balancer to automatically shift workloads in the event of an individual server failure.

Failover solutions, on the other hand, are used during the most extreme scenarios that result in a complete network failure. When these occur, a failover system is charged with auto-activating a secondary (standby) platform to keep a web application running while the IT team brings the primary network back online.

For true fault tolerance with zero downtime, you need to implement “hot” failover, which transfers workloads instantly to a working backup system. If maintaining a constantly active standby system is not an option, you can use “warm” or “cold” failover, in which a backup system takes time to load and start running workloads.

Imperva load balancing and failover solutions

Imperva offers a complete suite of web application fault tolerance solutions. The first among these is our cloud-based application layer load balancer that can be used for both in-datacenter (local) and cross-datacenter (global) traffic distribution.

The solution is provided via a load balancing as a service (LBaaS) model and is delivered from a globally-distributed network of data centers for rapid response and added redundancy.

Intelligent data-driven algorithms (e.g., least pending requests) are used to track server loads in real-time for optimized traffic distribution.

The other side of the coin is our failover solution that uses automated health checks from multiple geolocations to monitor the responsiveness of your servers.

In the event of a server failure, site traffic is instantly rerouted to a backup site within seconds, ensuring uninterrupted availability. The service is delivered from the cloud. As a result, even the execution of a remote failover doesn’t suffer from any TTL-related delays commonly found in other DNS-based solutions.

8. a. Write different clock synchronization and leadership algorithms for distributed platforms.

Clock synchronization

Clock synchronization is a method of synchronizing clock values of any two nodes in a distributed system with the use of external reference clock or internal clock value of the node.

There are 2 types of clock synchronization algorithms: Centralized and Distributed.

1. **Centralized**: is the one in which a time server is used as a reference. The single time server propagates it's time to the nodes and all the nodes adjust the time accordingly. It is dependent on single time server so if that node fails, the whole system will lose synchronization. Examples of centralized areBerkeley Algorithm, Passive Time Server, Active Time Server etc.

A. Passive Time Server

Each node periodically sends a request message 'time=?' to the time-server to get accurate time. Time-server responds with 'time=t'. Assume that client node send a message at time t0 and get a reply at time t1, then message propagation time from server to client node is (t1-t0)/2. Client node receives a reply and it adjusts its clock time to t + (t1-t0)/2.

There are two methods proposed to improve estimated time value.

iii. Additional information available

Assume that to handle interrupt and to process request message sent by the client, time server takes time ti. Hence, better estimated time is (t1-t0-ti)/2. Therefore, clock is readjusted to t + (t1-t0-ti)/2.

iv. No additional information available (Cristian Method)

This method uses time-server connected to a device that receives a signal from a source of UTC to synchronize nodes externally. A simple estimated time t + (t1- t0)/2, is accurate if nodes are on same network.

B. **Active Time Server**

Time server periodically broadcasts its clock time as 'time=t'. Other nodes receive message and readjust their local clock accordingly. Each node assumes message propagation time = ta, and readjust clock time = t + ta.

There are some limitations

i. Due to communication link failure message may be delayed and clock readjusted to incorrect time.

ii. Network should support broadcast facility.

C. Berkeley Algorithm

This algorithm overcomes limitation of faulty clock and malicious interference in passive time server and also overcomes limitation of active time server algorithm. Time server periodically sends a request message 'time=?' to all nodes in the system. Each node sends back its time value to the time server. Time server has an idea of message propagation to each node and readjust the clock values in reply message based on it. Time server takes an average of other computer clock's value including its own clock value and readjusts its own clock accordingly. It avoids reading from unreliable clocks. For readjustment, time server sends the factor by which other nodes require adjustment. The readjustment value can either be +ve or –ve.

There are following **limitations**:

i. Due to centralized system single point of failure may occur.

ii. A single time-server may not be capable of serving all time requests from scalability point of view. The readjustment value can either be +ve or –ve.

2. Distributed: is the one in which there is no centralized time server present. Instead the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like the scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

A. Global Averaging Algorithm

Each node in the system broadcast its local clock time in the form of special 'resync' message when its local time is t0 + IR, where I is for interrupt time and R includes number of nodes in the system, maximum drift rate etc.

After broadcasting, clock process of the node waits for some time period t. During this period it collects 'resync' message from other nodes and records its receiving time based on its local clock time. At the end of waiting period, it estimates the skew of its own clock with respect to all other nodes.

To find the correct time, need to estimate skew value as follows:

a) Take the average of estimated skew and use it as correction of local clock. To limit the impact of faulty clock skews greater than threshold are set to zero before computing average of estimated skew.

b) Each node limits the impact of faulty clock by discarding highest and lowest estimated skews and then calculates average of estimated skew.

There are some limitations as follows:

i. This method does not scale well.

ii. Network should support broadcast facility.

iii. Due to large amount of messages network traffic increased. Therefore this algorithm may be useful for small networks only.

B. Localized Averaging Algorithm

This algorithm overcomes limitation of scalability of global averaging algorithm. Nodes of the system are arranged logically in a specific pattern like a ring or grid. Periodically each node exchange its local clock time with its neighbors and then sets its clock time to the average of its own clock time and the clock time of its neighbors. We have discussed set of physical clock synchronization algorithms. These algorithms can be implemented with the use of time protocol.

Network Time Protocol(NTP)

Network Time Protocol is an Internet protocol used to synchronize the clocks of computers with some time reference. It uses UTC (Coordinated Universal Time) as time reference. NTP is developed by David. Mills. NTP is a fault tolerant protocol that will automatically select the best of several available time sources to synchronize.

Leader election algorithms

Leader election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Leader election algorithm basically determines where a new copy of coordinator should be restarted.

Leader election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is send to every active process in the distributed system.

There are two leader election algorithms for two different configurations of distributed system.

I. **The Bully algorithm: a**pplies to system where every process can send a message to every other process in the system.

II. **Improved(Modified) bully algorithm: t**he main concept of this algorithm is that the algorithm declares the new coordinator before actual or current coordinator is crashed. It overcomes the disadvantages of the original bully.

III. **Modified Election Algorithm:** resolves the disadvantages of the bully algorithm.

IV. **The Ring algorithm:** applies to systems organized as a ring (logically or physically). In this algorithm we assume that the link between the processes are unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is active list, a list that has priority number of all active processes in the system.

V. Modified Ring Algorithm: resolves the disadvantages of the Ring algorithm

b. Explain about the distributed deadlock detection mechanisms and avoidance solutions.

Deadlock Handling in Distributed Systems

In a distributed system deadlock can neither be prevented nor avoided as the system is so vast that it is impossible to do so. Therefore, only deadlock detection can be implemented. The techniques of deadlock detection and avoidance in the distributed system require the following:

- ✓ **Progress:** The method should be able to detect all the deadlocks in the system.
- ✓ **Safety:** The method should not detect false or phantom deadlocks.

Distributed Deadlock Avoidance

As in centralized system, distributed deadlock avoidance handles deadlock prior to occurrence. Additionally, in distributed systems, transaction location and transaction control issues needs to be addressed. Due to the distributed nature of the transaction, the following conflicts may occur −

- Conflict between two transactions in the same site.
- Conflict between two transactions in different sites.

In case of conflict, one of the transactions may be aborted or allowed to wait as per distributed wait-die or distributed wound-wait algorithms.

Let us assume that there are two transactions, T1 and T2. T1 arrives at Site P and tries to lock a data item which is already locked by T2 at that site. Hence, there is a conflict at Site P. The algorithms are as follows −

- ➢ **Distributed Wound-Die**
  - ✓ If T1 is older than T2, T1 is allowed to wait. T1 can resume execution after Site P receives a message that T2 has either committed or aborted successfully at all sites.
  - ✓ If T1 is younger than T2, T1 is aborted. The concurrency control at Site P sends a message to all sites where T1 has visited to abort T1. The controlling site notifies the user when T1 has been successfully aborted in all the sites.
- ➢ **Distributed Wait-Wait**
  - ✓ If T1 is older than T2, T2 needs to be aborted. If T2 is active at Site P, Site P aborts and rolls back T2 and then broadcasts this message to other relevant sites. If T2 has left Site P but is active at Site Q, Site P broadcasts that T2 has been aborted; Site L then aborts and rolls back T2 and sends this message to all sites.
  - ✓ If T1 is younger than T1, T1 is allowed to wait. T1 can resume execution after Site P receives a message that T2 has completed processing.

Distributed Deadlock Detection

Just like centralized deadlock detection approach, deadlocks are allowed to occur and are removed if detected. The system does not perform any checks when a transaction places a lock request. For implementation, global wait-for-graphs are created. Existence of a cycle in the global wait-for-graph indicates deadlocks. However, it is difficult to spot deadlocks since transaction waits for resources across the network.

Alternatively, deadlock detection algorithms can use timers. Each transaction is associated with a timer which is set to a time period in which a transaction is expected to finish. If a transaction does not finish within this time period, the timer goes off, indicating a possible deadlock.

Another tool used for deadlock handling is a deadlock detector. There are three approaches for deadlock detection in a distributed system, namely.

1. **Centralized Deadlock Detector:** One site is designated as the central deadlock detector. In the centralized approach, there is only one responsible resource to detect deadlock. The advantage of this approach is that it is simple and easy to implement, while the drawbacks include excessive workload at one node, single-point failure (that is the whole system is dependent on one node if that node fails the whole system crashes) which in turns makes the system less reliable.
2. **Hierarchical Deadlock Detector:** A number of deadlock detectors are arranged in hierarchy. This approach is the most advantageous. It is the combination of both centralized and distributed approaches of deadlock detection in a distributed system. In this approach, some selected nodes or clusters of nodes are responsible for deadlock detection and these selected nodes are controlled by a single node.
3. **Distributed Deadlock Detector:** All the sites participate in detecting deadlocks and removing them. In the distributed approach different nodes work together to detect deadlocks. No single point failure (that is the whole system is dependent on one node if that node fails the whole system crashes) as the workload is equally divided among all nodes. The speed of deadlock detection also increases.

9. a. Write a short note on evolution of distributed programming with proper references.

In the early years of computing, mainframe-based applications were considered to be the best-fit solution for executing large-scale data processing applications. With the advent of personal computers (PCs), the concept of software programs running on standalone machines became much more popular in terms of the cost of ownership and the ease of application use. With the number of PC-based application programs running on independent machines growing, the communications between such application programs became extremely complex and added a growing challenge in the aspect of application-to application interaction. Lately, network computing gained importance, and enabling remote procedure calls (RPCs) over a network protocol called Transmission Control Protocol/Internet Protocol (TCP/IP) turned out to be a widely accepted way for application software communication. Since then, software applications running on a variety of hardware platforms, operating systems, and different networks faced some challenges when required to communicate with each other and share data. This demanding requirement lead to the concept of distributed computing applications. As a definition, "Distributing Computing is a type of computing in which different components and objects comprising an application can be located on different computers connected to a network distributed computing model that provides an infrastructure enabling invocations of object functions located anywhere on the network. The objects are transparent to the application and provide processing power as if they were local to the application calling them.

Distributed computing technologies

Client-Server Applications

The early years of distributed application architecture were dominated by two-tier business applications. In a two-tier architecture model, the first (upper) tier handles the presentation and business logic of the user application (client), and the second/lower tier handles the application organization and its data storage (server). This approach is commonly called client-server applications architecture. For example, the client-server model has been widely used in enterprise resource planning, billing, and Inventory application systems where a number of client business applications residing in multiple desktop systems interact with a central database server.

Web Services

Nowadays, the adoption of the Internet and enabling Internet-based applications has created a world of discrete business applications, which co-exist in the same technology space but without interacting with each other. The increasing demands of the industry for enabling B2B, applicationto-application (A2A), and inter-process application communication has led to a growing requirement for service-oriented architectures. Enabling service- oriented applications facilitates the exposure of business applications as service components enable business applications from other organizations to link with these services for application interaction and data sharing without human intervention. By leveraging this architecture, it also enables interoperability between business applications and processes. Web services are based on the concept of service oriented architecture (SOA). SOA is the latest evolution of distributed computing, which enables software components, including application functions, objects, and processes from different systems, to be exposed as services.

Message-Oriented Middleware

Although CORBA, RMI, and DCOM differ in their basic architecture and approach, they adopted a tightly coupled mechanism of a synchronous communication model (request/response). All these technologies are based upon binary communication protocols and adopt tight integration across 28 their logical tiers, which is susceptible to scalability issues. Message-Oriented Middleware (MOM) is based upon a loosely coupled asynchronous communication model where the application client does not need to know its application recipients or its method arguments. MOM enables applications to communicate indirectly using a messaging provider queue.

Java RMI

Java RMI was developed by Sun Microsystems as the standard mechanism to enable distributed Java objects-based application development using the Java environment. RMI provides a distributed Java application environment by calling remote Java objects and passing them as argument or return values. It uses Java object serialization, a lightweight object persistence technique that allows the conversion of objects into streams. Before RMI, the only way to do interprocess communications in the Java platform was to use the standard Java network libraries. Java RMI uses Java Remote Method Protocol (JRMP) as the interprocess communication protocol, enabling Java objects living in different Java Virtual Machines (VMs) to transparently invoke one another's methods. Because these VMs can be running on different computers anywhere on the network, RMI enables object-oriented distributed computing.

CORBA

The Common Object Request Broker Architecture (CORBA) is an industry wide, open standard initiative, developed by the Object Management Group (OMG) for enabling distributed computing that supports a wide range of application environments. CORBA differs from the traditional client/server model because it provides an object-oriented solution that does not enforce any proprietary protocols or any particular programming language, operating system, or hardware platform.

Microsoft DCOM

The Microsoft Component Object Model (COM) provides a way for Windows-based software components to communicate with each other by defining a binary and network standard in a Windows operating environment.

Grid computing is a processor architecture that combines computer resources from various domains to reach a main objective. In grid computing, the computers on the network can work on a task together, thus functioning as a supercomputer.

Cloud computing is the delivery of different services through the Internet. These resources include tools and applications like data storage, servers, databases, networking, and software.

Ubiquitous computing is a concept in software engineering and computer science where computing is made to appear anytime and everywhere. In contrast to desktop computing, ubiquitous computing can occur using any device, in any location, and in any format. A user interacts with the computer, which can exist in many different forms, including laptop computers, tablets and terminals in everyday objects.

b. Explain major differences between SOAP and RESTful web services

Although REST is very popular these days, SOAP still has its place in the world of web services.

SOAP and REST both allow you to create your own API. API stands for Application Programming Interface. It makes it possible to transfer data from an application to other applications. An API receives requests and sends back responses through internet protocols such as HTTP, SMTP, and others.

SOAP and REST are two API styles that approach the question of data transmission from a different point of view.

SOAP is a standardized protocol that sends messages using other protocols such as HTTP and SMTP. The SOAP specifications are official web standards, maintained and developed by the World Wide Web Consortium (W3C). As opposed to SOAP, REST is not a protocol but an architectural style. The REST architecture lays down a set of guidelines you need to follow if you want to provide a RESTful web service, for example, stateless existence and the use of HTTP status codes.

As SOAP is an official protocol, it comes with strict rules and advanced security features such as built-in ACID compliance and authorization. Higher complexity, it requires more bandwidth and resources which can lead to slower page load times.

REST was created to address the problems of SOAP. Therefore it has a more flexible architecture. It consists of only loose guidelines and lets developers implement the recommendations in their own way. It allows different messaging formats, such as HTML, JSON, XML, and plain text, while SOAP only allows XML. REST is also a more lightweight architecture, so RESTful web services have a better performance. Because of that, it has become incredibly popular in the mobile era where even a few seconds matter a lot (both in page load time and revenue).

One of the most highly debatable topics is when REST should be used or when to use SOAP while designing web services. Below are some of the key factors that determine when each technology should be used for web services.

REST services should be used in the following instances

- ✓ **Limited resources and bandwidth:** Since SOAP messages are heavier in content and consume a far greater bandwidth, REST should be used in instances where network bandwidth is a constraint.
- ✓ **Statelessness:** If there is no need to maintain a state of information from one request to another then REST should be used. If you need a proper information flow wherein some information from one request needs to flow into another then SOAP is more suited for that purpose. We can take the example of any online purchasing site. These sites normally need the user first to add items which need to be purchased to a cart. All of the cart items are then transferred to the payment page in order to complete the purchase. This is an example of an application which needs the state feature. The state of the cart items needs to be transferred to the payment page for further processing.
- ✓ **Caching:** If there is a need to cache a lot of requests then REST is the perfect solution. At times, clients could request for the same resource multiple times. This can increase the number of requests which are sent to the server. By

implementing a cache, the most frequent queries results can be stored in an intermediate location. So whenever the client requests for a resource, it will first check the cache. If the resources exist then, it will not proceed to the server. So caching can help in minimizing the amount of trips which are made to the web server.

- ✓ **Ease of coding:** Coding REST Services and subsequent implementation is far easier than SOAP. So if a quick win solution is required for web services, then REST is the way to go.

SOAP should be used in the following instances

- ✓ **Asynchronous processing and subsequent invocation** – if there is a requirement that the client needs a guaranteed level of reliability and security then the new SOAP standard of SOAP 1.2 provides a lot of additional features, especially when it comes to security.
- ✓ **A Formal means of communication** – if both the client and server have an agreement on the exchange format then SOAP 1.2 gives the rigid specifications for this type of interaction. An example is an online purchasing site in which users add items to a cart before the payment is made. Let's assume we have a web service that does the final payment. There can be a firm agreement that the web service will only accept the cart item name, unit price, and quantity. If such a scenario exists then, it's always better to use the SOAP protocol.
- ✓ **Stateful operations –** if the application has a requirement that state needs to be maintained from one request to another, then the SOAP 1.2 standard provides the WS* structure to support such requirements.

Soap Advantages

SOAP services can have much higher complexity by supporting arbitrary operations and by potentially tracking state. An example could be a bookstore service having an “addToCart” operation. Every time a client calls “addToCart”, the service tracks the item in the client’s shopping cart. A different client can call “addToCart” and not impact a different client’s shopping cart. The service tracks the state of each client separately. SOAP provides the following advantages when compared to REST:

- ✓ Language, platform, and transport independent (REST requires use of HTTP)
- ✓ Works well in distributed enterprise environments (REST assumes direct point-to-point communication)
- ✓ Standardized
- ✓ Provides significant pre-build extensibility in the form of the WS* standards
- ✓ Built-in error handling
- ✓ Automation when used with certain language products

REST Advantages

REST APIs are more restricted than SOAP by having operations limited to CRUD. Additionally, clients have the burden of tracking their own state.

REST is easier to use for the most part and is more flexible. It has the following advantages over SOAP:

- ✓ No expensive tools require to interact with the web service
- ✓ Uses easy to understand standards like swagger and OpenAPI Specification
- ✓ Smaller learning curve
- ✓ Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
- ✓ Closer to other web technologies in design philosophy
- ✓ Fast (no extensive processing required)

	SOAP	REST
Meaning	Simple Object Access Protocol	Representational State Transfer
Design	Standardized protocol with pre-defined rules to follow.	Architectural style with loose guidelines and recommendations.
Approach	Function-driven (data available as services, e.g.: “getUser”)	Data-driven (data available as resources, e.g. “user”).
Statefulness	Stateless by default, but it’s possible to make a SOAP API stateful.	Stateless (no server-side sessions).
Caching	API calls cannot be cached.	API calls can be cached.
Security	WS-Security with SSL support. Built-in ACID compliance.	Supports HTTPS and SSL.
Performance	Requires more bandwidth and computing power.	Requires fewer resources.
Message format	Only XML.	Plain text, HTML, XML, JSON, YAML, and others.
Transfer protocol(s)	HTTP, SMTP, UDP, and others.	Only HTTP
Recommended for	Enterprise apps, high-security apps, distributed environment, financial services, payment gateways, telecommunication services.	Public APIs for web services, mobile services, social networks.
Advantages	High security, standardized, extensibility.	Scalability, better performance, browser-friendliness, flexibility.
Disadvantages	Poorer performance, more complexity, less flexibility.	Less security, not suitable for distributed environments.
	SOAP can't use REST because it is a protocol.	REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.

- ✓

10. a. Describe the current limitations and Strengths of leading distributed computing platforms with proper references.

Computing Platforms are software client applications that you can run on your computer and that host various, often unrelated, project applications.

1. The Berkeley Open Infrastructure for Network Computing (BOINC)

It is an open source client-server middleware system created to allow projects with large computational requirements, usually in scientific domain, to utilize a technically unlimited number of volunteer machines distributed over large physical distances.

Strength □

- Used to share resources among independent and autonomous projects □
- BOINC provides HTTP-based remote job submission APIs so that projects can submit and manage jobs using web-based interfaces or via existing systems such as HTCondor □
- Supports diverse applications by providing flexible and scalable mechanism for distributing data, and its scheduling algorithms intelligently match requirements with resources.

Limitation □

- Project developers are required to port their application to every target machine architecture. □
- Project developers need to provide application-level check pointing to ensure job progress is not lost upon host termination or failures. □
- Project developers are limited to creating applications that have no dependencies.
- Users of BOINC must trust that project servers they attach to, will not distribute malicious or untrustworthy applications. [18][19]

2. World Community Grid (WCG)

It is a project which aims at creating the world's largest public computing grid. This project gains its computing power and the most of its "infrastructure" from individuals who volunteer 31 to contribute by sharing the computing power and storage of their own devices. WCG is a distributed computing platform that support multiple computing projects.

Strength □

- WCG is platform independent it's currently available for Windows, Linux, macOS, and Android operating systems.

Limitation

- The WCG software increases CPU usage by consuming unused processing time. With modern CPUs, where dynamic frequency scaling is prevalent, increased usage makes the processor run at higher frequency, increasing power usage and heating counter to power management. Additionally, because of an increasing focus on power performance, connecting old computers to the grid will increase the total/average power required to complete the same calculations. [23][24]

b. Explain the major strengths and weakness of the SOA.

SOA: Major Strength

SOA offers better opportunities for enterprise application integration with the additional advantages of reduction in costs, ease of maintenance and improvement in flexibility and scalability. SOA allows enterprises and their IT systems to be more agile to the changes in the business and the environment.

It provides an opportunity to achieve broad-scale interoperability while offering flexibility to adapt to changing technologies. If implemented correctly, its technical characteristics translate directly into real bottom-line benefits. The main characteristics include:

- ✓ Loosely coupled architecture
- ✓ Modular approach
- ✓ Non-intrusive nature
- ✓ Universality of standards.

Various benefits derived from the above can be summarized as follows :

- ✓ Loosely coupled applications and location transparency – this allows enterprises to plug in new services or upgrade existing services with relative ease.
- ✓ Seamless connectivity of applications and interoperability – this provides opportunities to increase business agility and the ability to respond on demand.
- ✓ Alignment of IT around the needs of the business – this results in IT as the enabling technology to provide added value to business operations.
- ✓ Increased business agility, capturing new channels of business – this provides flexibility, ease of access and increased customer satisfaction.
- ✓ Enhanced reuse of existing assets and applications – this helps to reduced costs, reduced development time and a reduction of time to market
- ✓ Relatively easy integration of legacy systems – this promotes interoperability and efficient use of resources and existing facilities.
- ✓ Process-centric architecture and flexibility of approach – this allows a process driven approach, which is a worthwhile aim to achieve.
- ✓ Parallel and independent applications development – this is due to the reuse of services and ability to withdraw or plug in new developments with considerable ease.
- ✓ Better scalability and graceful evolutionary changes - this is due to the reuse of services and ability to develop applications independently and in parallel.
- ✓ Reduced costs of integration and change management.
- ✓ Automatic derivation of process metrics and hence reduction of process metrics costs.
- ✓ Easier and more effective systems maintenance.

- ✓ Reduced business risk and exposure in spite of increased business visibility.
- ✓ Reduced vendor lock-in.
- ✓ Maintenance is Easy: Editing and updating any service implemented under SOA architecture is easy. You don't need to update your system. A third party maintains the service, and any amendment in this service won't have an effect on your system. In most cases, previous API work because it is functioning before.
- ✓ Quality of Code Improved: As services run independent of our system, they have their own variety(style) of code; therefore, our code is prevented from redundancy. Also, our code becomes error-free.
- ✓ Platform Independence: Services communicate with alternative applications through a common language, which implies it is independent of the platform on that application is running. Services can provide API in different languages.
- ✓ Scalable: If any service obtains several users, it is often simply scalable by attaching additional servers. This will create service out there all time to the users.
- ✓ Reliable: Services square measure typically tiny size as compared to the full-fledged application. So it's easier to correct and check the independent services.
- ✓ Same Directory Structure: Services have an equivalent directory structure so customers can access the service information from an equivalent directory on every occasion. If any service has modified its location, then the additional directory remains the same. This is very helpful for consumers.
- ✓ Independent of Other Services: Services generated using SOA principles are independent of each other. So services are often utilized by multiple applications simultaneously.
- ✓ Prevent reinventing the wheel: The company can use pre-build service and don't have to rebuild the same functionality from scratch. This will also increase the productivity of the company. Now the company can only focus on its own website or software without worrying about external component integration.
- ✓ Independent location. It doesn't really matter where the services are located. They can be published on one server or several different ones. Consumer requests would still work fine.
- ✓ High reusability. Services can be reused regardless of their earlier interactions with other services. This reusability is possible due to the SOA applications infrastructure — a combination of small self-sufficient functions.
- ✓ Improved scalability. You can easily scale up the system since multiple layers of a single service can run simultaneously on different servers.
- ✓ Parallel development opportunities. Thanks to the layer-based architecture, developers can work on independent services and have them both delivered fast. This not only increases productivity but also enables businesses to reduce costs.

The SOA represents a new way of developing systems - it promotes a shift from writing software to assembling and integrating services. A natural consequence is that systems are no longer implemented as single monolithic structures but as a series of component parts that work together to deliver whatever functionality is required. It is an effective strategy for integrating enterprise–wide applications and legacy systems. Underlying platform implementation becomes irrelevant as standard interfaces and message exchange patterns provide integration, both within and across enterprises. The goal is to build a flexible infrastructure to abridge applications more easily and to solve business challenges quickly. However, to support the goal of SOA, the infrastructure must support flexibility, heterogeneity, distributed development and management.

SOA: Major Weakness

SOA is a much better architecture as opposed to distributed client server architecture and it can bring huge benefits in the form of code reuse, better integration and improved responsiveness to business needs. However, the eternal battle between flexibility and efficiency exists in just the same way as it always has. Obviously, it is desirable to develop architectures that allow easy integration of the existing and new enterprise applications. However, the integration technology solutions are often proprietary which present issues of inoperability.

He mentions the following problems with respect to the existing distributed architectures:

- ✓ Vendor lock-ins: as many architectures are based on proprietary protocols and implementations.
- ✓ Tight coupling: as distributed architectures typically link components directly to one another.
- ✓ Complexity: as the interactions between objects are often rich and complex.
- ✓ Connectivity: as majority of distributed architectures do not work over wide-area, intermittent networks.

SOA also requires a large upfront investment by way of technology, development and staff deployment. It may cost a great deal and the Return on Investment (ROI) could take a long time to materialize. Overall the downsides to SOA:

- ✓ Since services can invoke other services, each service needs to validate completely every input parameter. This has negative implications by way of response time and overall machine load
- ✓ It is entirely possible, at times, that a bug or corruption introduced in a well-used service takes out the entire system, not just a single application.

Managing services metadata is another obvious challenge. As services keep exchanging messages to perform tasks, number of these messages can go into millions even for a single application.

Technology risk of SOA is particularly challenging due to the following factors:

- ✓ Early adoption – especially when the technology is new and evolving,
- ✓ Organizational changes within the enterprise – to accommodate a new way of thinking,
- ✓ Architecture – this is enterprise wide and thus encompasses heterogeneous systems,
- ✓ Quality assurance – this is particularly difficult as services are distributed and ownership is often unclear.

Although, Web Services provide a sensible implementation platform, many infrastructure services (eg security, systems management, interface contracts) are not yet fully defined. Finding a service that is at the right level of abstraction is also a challenge. Issues, inherent due to the very nature of serviceorientation, can be summarized as follows:

- ✓ Coarse granularity: This may mean that

  1) testing and validating every combination of every condition in a complex service may well become humanly impossible;

  2) one service trying to serve a dozen masters may lead to spaghetti code and therefore introduce massive inefficiency and

  3) a generic service, because of its coarse granularity, cannot be easily optimized for efficiency.

- ✓ Loose coupling: It is an architect's dream but making a system distributed adds a new level of complexity and therefore, it can become a developer's nightmare
- ✓ Integration of services: This can be a complex task especially when there is a lack of skilled people to work in a SOA based environment
- ✓ Service interoperability: When web services are used to exchange SOAP messages over HTTP, encapsulating XML data, integration of services in heterogeneous environment can become a serious issue
- ✓ Evolutionary development: Building and updating services is fine. However, if applications continually require additional functionality, and these requests are granted, the entire system may become unstable.

❖ Other challenges and limitations can be summarized as follows:

- ✓ WS standards: These standards are open and amorphous. Many are still working drafts. Higherlevel services and security have not been standardised at all. This could result in a rework of existing code to conform to new and evolving standards.
- ✓ Internet protocols: They are not designed for reliability, guaranteed delivery or order of delivery so the service or the consumer needs to ensure that messages has been delivered/received in a timely manner.

- ✓ Interoperability: Common implementations of SOA use web services to exchange messages over the Internet. These encapsulate XML data. Problems may be encountered when integrating these services in heterogeneous environments.
- ✓ Development tools: Vendors are producing tools to enable organizations to reap the benefits of SOA but a majority of these are early releases and based on evolving standards. This may also result in rework of existing code once the standards are agreed upon.
- ✓ Network connections: If the architecture and services are highly interconnected then even a partial network failure may create a huge problem for the relevant organizations. Currently, although the required technology is available, there is no guarantee that infrastructures will be robust with enough redundancy to cope with the system downtimes.
- ✓ Security: When using open standards, a service is much more open to other services and applications than a monolithic application and thus security becomes an issue. Internet protocols also lack reliability. Although, WS-Security addresses such issues, there is a considerable amount of work that still needs to be done.
- ✓ Application ownership: SOA blurs the boundaries of application ownership so who owns what becomes an issue. Thus, service management becomes a real issue.
- ✓ Choosing a vendor: Knorr and Rist recommend a multi-vendor solution to fully benefit from the flexible nature of SOA.
- ✓ Training: It takes time to learn and use new technologies. There are too many technologies and not enough expertise available in the market. For an organization adopting this architecture, thorough understanding of the underlying technologies remains critical.
- ✓ Adoption: staff's reluctance to embrace a new technology is always an issue that management needs to consider carefully.
- ✓ Governance: UK firm Gartner warns that SOA projects will fail unless they are tightly managed and audited. Since SOA is a new paradigm, governance issues are not properly understood.
- ✓ Large upfront investment. SOA architecture is a great choice for further business development. It enables your team to work on several applications simultaneously at a smaller cost. However, its implementation is usually a pricey endeavor.
- ✓ Greater load and increased response time. In SOA, each interaction between services is followed by a full validation of all input parameters. As a result, the load gets extremely heavy, which in turn prolongs the response time.
- ✓ Vast variety of services. SOA has a very special approach to operation. Services constantly exchange messages while executing tasks. Consequently, the number of those messages gets overwhelming, and this peculiarity makes it difficult to ensure decent service management.
- ✓ Exponential increase in connections Increases burden on sever and management overhead due to transmission control protocol.

- ✓ Due to multiple points of failure, it is hard to determine the reliability of each point. Software debugging can affect both client and server side.
- ✓ High Bandwidth Server: As web service sends and receives messages and information frequently so it easily reaches a million requests per day. So it involves a high-speed server with a lot of data bandwidth to run a web service.
- ✓ Extra Overload In SOA, all inputs square measures its validity before it's sent to the service. If you are victimization multiple services, then it'll overload your system with extra computation.
- ✓ High Cost: It is expensive in terms of human resource, development, and technology.
- ✓ XML format messaging: Messages are large in size due to which more bandwidth and resources are consumed.
- ✓ SOA is not suitable for deployment when an application is a stand alone entity, or does not require call based or message passing mechanism.
- ✓ SOA is not suitable for an application which does not provide full functionality or does not work as a complete system instead serve as a component and have limited scope.
- ✓ SOA does not support application that are tightly coupled
- ✓ It is not recommended to use SOA in a homogenous application environment.
- ✓ SOA face the problem of lack of a uniform testing framework. There are no tools that provide the required features for testing services in a service-oriented architecture.
- ✓ SOA is not suitable for an application that has short living time span
- ✓ Service oriented architecture is not suitable for applications which are based on heavy data exchange.
- ✓ Not scalable; It's hard to add a new application or require additional development.

References

1. Tanenbaum, Andrew S.; Steen, Maarten van (2002). Distributed systems: principles and paradigms. Upper Saddle River, NJ: Pearson Prentice Hall. ISBN 0-13-088893-1.

2. https://www.ibm.com/docs/de/SSAL2T_8.1.0/com.ibm.cics.tx.doc/concepts/c_wht_is_distd_comptg.html

3. https://blog.stackpath.com/distributed-system/

4. https://www.techopedia.com/definition/7/distributed-computing-system

5. Vasil s.Denckel and Gopel Pandurangan "Distributed quantum computing" sep 2008

6. Nakhun Chumpolsathien, "Microservices: the Future of Distributed System" School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China 3820181012@bit.edu.cn , Jan 2019

7. A. Goyal, "9to5Cloud," Endive Software, 1 2019.

8. "The future of distributed system with cloud, edge and colocation" August 8, 2018

9. Kavita Nikhil Ahire, Distributed Computing-Future and Applications

10. https://www.researchgate.net/publication/301691260_Limitations_of_Service_Oriented_Architecture_and_its_Combination_with_Cloud_Computing

11. https://www.educba.com/what-is-soa/

12. https://newizze.com/soa-vs-microservices-comparison-for-2019/

13. Z. Mahmood, „The Promise and Limitations of Service Oriented Architecture", in NAUN International Journal of Comuputers Vol. 1 (3) 2007,

14. http://www.myreadingroom.co.in/notes-and-studymaterial/65-dbms/539-two-phase-locking.html

15. https://www.investopedia.com/terms/c/cloud-computing.asp

16. https://www.techopedia.com/definition/87/gridcomputing

17. https://en.wikipedia.org/wiki/Ubiquitous_computing

18. Gary Andrew McGilvary, Adam Barker, Ashley Lloyd and Malcolm Atkinson Jun 2013 "V-BOINC: The Virtualization of BOINC"

19. David P. Anderson October 2019 "BOINC: A Platform for Volunteer Computing"

20. Silas De Munck, Kurt Vanmechelen, and Jan Broeckhove University of Antwerp, Department of Computer Science and Mathematics, Middelheimlaan 1, 2020 Antwerp, Belgium "Improving the Scalability of SimGrid Using Dynamic Routing"

21. Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose and Rajkumar Buyya August 2010 "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms"

22. Neha N. Dalwadi1 and Dr. Mamta C. Padole2 12The Maharaja Sayajirao University, Vadodara, Gujarat, India Nov 2017 “Comparative Study of Clock Synchronization Algorithms in Distributed Systems”

23. https://en.wikipedia.org/wiki/World_Community_Grid

24. Mirela Riveni, Muhammad Zuhri Catur Candra PhD School of Informatics Institute of Information Systems, Distributed Systems Group Vienna University of Technology Vienna, 2012 “Extracting and Analyzing Network Data about World Community Grid”

25. https://aws.amazon.com/builders-library/leader-election-in-distributed-systems/

26. Seema Balhara et al, International Journal of Computer Science and Mobile Computing, Vol.3 Issue.6, June- 2014, pg. 374-379

27. https://www.sciencedirect.com/topics/computer-science/distributed-file-systems

28. Neha N. Dalwadi1 and Dr. Mamta C. Padole,”Comparative Study of Clock Synchronization Algorithms in Distributed Systems”