1.7 Design issues and challenges

Distributed systems challenges from a system perspective

a. **Communication:** This task involves designing appropriate mechanisms for communication among the processes in the network. Some example mechanisms are: remote procedure call (RPC), remote object invocation cation (ROI), message-oriented communication versus stream-oriented communication.
b. **Processes:** Some of the issues involved are: management of processes and threads at clients/servers; code migration; and the design of software and mobile agents.
c. **Naming:** Devising easy to use and robust schemes for names, identifiers, and addresses is essential for locating resources and processes in a transparent and scalable manner. Naming in mobile systems provides additional challenges because naming cannot easily be tied to any static geographical topology.
d. **Synchronization**: Mechanisms for synchronization or coordination among the processes are essential. Mutual exclusion is the classical example of synchronization, but many other forms of synchronization, such as leader election are also needed.
e. **Data storage and access:** Schemes for data storage, and implicitly for accessing the data in a fast and scalable manner across the network are important for efficiency.
f. **Consistency and replication**: To avoid bottlenecks, to provide fast access to data, and to provide scalability, replication of data objects is highly desirable.
g. **Fault tolerance:** Fault tolerance requires maintaining correct and efficient operation in spite of any failures of links, nodes, and processes.
h. **Security:** Distributed systems security involves various aspects of cryptography, secure channels, access control, key management – generation and distribution, authorization, and secure group management.
i. **Applications Programming Interface (API) and transparency**: The API for communication and other specialized services is important for the ease of use and wider adoption of the distributed systems services by non-technical users.

Transparency deals with hiding the implementation policies from the user, and can be classified as follows.

a. Access Transparency hides differences in data representation on different systems and provides uniform operations to access system resources.
b. Location transparency makes the locations of resources transparent to the users.
c. Migration transparency allows relocating resources without changing names.
d. The ability to relocate the resources as they are being accessed is relocation transparency.
e. Replication transparency does not let the user become aware of any replication.
f. Concurrency transparency deals with masking the concurrent use of shared resources for the user.
g. Failure transparency refers to the system being reliable and fault-tolerant.

j. **Scalability and modularity:** The algorithms, data (objects), and services must be as distributed as possible. Various techniques such as replication, caching and cache management, and asynchronous processing help to achieve scalability.

Algorithmic challenges in distributed computing

a. Designing useful execution models and frameworks

The interleaving model and partial order model are two widely adopted models of distributed system executions. They have proved to be particularly useful for operational reasoning and the design of distributed algorithms.

b. Dynamic distributed graph algorithms and distributed routing algorithms

The distributed system is modeled as a distributed graph, and the graph algorithms form the building blocks for a large number of higher level communication, data dissemination, object location, and object search functions.

c. Time and global state in a distributed system

The processes in the system are spread across three-dimensional physical space. Another dimension, time, has to be superimposed uniformly across space. The challenges pertain to providing accurate physical time, and to providing a variant of time, called logical time.

d. Synchronization/coordination mechanisms

The processes must be allowed to execute concurrently, except when they need to synchronize to exchange information, i.e., communicate about shared data. Synchronization is essential for the distributed processes to overcome the limited observation of the system state from the viewpoint of any one process. Here are some

examples of problems requiring synchronization. They are Physical clock synchronization, Leader election, Mutual exclusion, Deadlock detection and resolution, Termination detection and Garbage collection.

e. Group communication, multicast, and ordered message delivery

A group is a collection of processes that share a common context and collaborate on a common task within an application domain. Specific algorithms need to be designed to enable efficient group communication and group management wherein processes can join and leave groups dynamically, or even fail.

f. Monitoring distributed events and predicates

Predicates defined on program variables that are local to different processes are used for specifying conditions on the global system state, and are useful for applications such as debugging, sensing the environment, and in industrial process control.

g. Distributed program design and verification tools

Methodically designed and verifiably correct programs can greatly reduce the overhead of software design, debugging, and engineering. Designing mechanisms to achieve these design and verification goals is a challenge

h. Debugging distributed programs

Debugging sequential programs is hard; debugging distributed programs is that much harder because of the concurrency in actions and the ensuing uncertainty due to the large number of possible executions defined by the interleaved concurrent actions.

i. Data replication, consistency models, and caching

Fast access to data and other resources requires them to be replicated in the distributed system. Managing such replicas in the face of updates introduces the problems of ensuring consistency among the replicas and cached copies.

j. World Wide Web design – caching, searching, scheduling

The Web is an example of a widespread distributed system with a direct interface to the end user, wherein the operations are predominantly read-intensive on most objects.

k. Distributed shared memory abstraction

A shared memory abstraction simplifies the task of the programmer because he or she has to deal only with read and write operations, and no message communication primitives. However, under the covers in the middleware layer, the abstraction of a shared address

space has to be implemented by using message-passing. Hence, in terms of overheads, the shared memory abstraction is not less expensive.

l. Reliable and fault-tolerant distributed systems

A reliable and fault-tolerant environment has multiple requirements and aspects, and these can be addressed using various strategies. They are Consensus algorithms, Replication and replica management, Voting and quorum systems, Distributed databases and distributed commit, Self-stabilizing systems, Checkpointing and recovery algorithms, Failure detectors.

m. Load balancing

The goal of load balancing is to gain higher throughput, and reduce the userperceived latency. The following are some forms of load balancing: Data migration, Computation migration and Distributed scheduling.

n. Real-time scheduling

Real-time scheduling is important for mission-critical applications, to accomplish the task execution on schedule. The problem becomes more challenging in a distributed system where a global view of the system state is absent. On-line or dynamic changes to the schedule are also harder to make without a global view of the state.

o. Performance

Although high throughput is not the primary goal of using a distributed system, achieving good performance is important. The following are some example issues arise in determining the performance: Metrics and Measurement methods/tools

Applications of distributed computing and newer challenges

- **Mobile systems**

Mobile systems typically use wireless communication which is based on electromagnetic waves and utilizes a shared broadcast medium. Hence, the characteristics of communication are different; many issues such as range of transmission and power of transmission come into play, besides various engineering issues such as battery power conservation, interfacing with the wired Internet, signal processing and interference

- **Sensor networks**

  A sensor is a processor with an electro-mechanical interface that is capable of sensing physical parameters, such as temperature, velocity, pressure, humidity, and chemicals. Recent developments in cost-effective hardware technology have made it possible to deploy very large (of the order of 106 or higher) low-cost sensors.

- **Ubiquitous or pervasive computing**

  Ubiquitous systems represent a class of computing where the processors embedded in and seamlessly pervading through the environment perform application functions in the background, much like in sci-fi movies. The intelligent home, and the smart workplace are some example of ubiquitous environments currently under intense research and development.

- **Peer-to-peer computing**

  Peer-to-peer (P2P) computing represents computing over an application layer network wherein all interactions among the processors are at a "peer" level, without any hierarchy among the processors. Thus, all processors are equal and play a symmetric role in the computation.

- **Publish-subscribe, content distribution, and multimedia**

  In a dynamic environment where the information constantly fluctuates (varying stock prices is a typical example), there needs to be:

  (i) an efficient mechanism for distributing this information (publish),

  (ii) an efficient mechanism to allow end users to indicate interest in receiving specific kinds of information (subscribe)

  (iii) an efficient mechanism for aggregating large volumes of published information and filtering it as per the user's subscription filter

- **Distributed agents**

  Agents are software processes or robots that can move around the system to do specific tasks for which they are specially programmed. The name "agent" derives from the fact that the agents do work on behalf of some broader objective.

- **Distributed data mining**

  Data mining algorithms examine large amounts of data to detect patterns and trends in the data, to mine or extract useful information. A traditional example is: examining the purchasing patterns of customers in order to profile the customers and enhance the efficacy of directed marketing schemes.

- **Grid computing**

  Many challenges in making grid computing a reality include: scheduling jobs in such a distributed environment, a framework for implementing quality of service and real-time guarantees, and, of course, security of individual machines as well as of jobs being executed in this setting.

- **Security in distributed systems**

  The traditional challenges of security in a distributed setting include: confidentiality (ensuring that only authorized processes can access certain information),
  authentication (ensuring the source of received information and the identity of the sending process), and availability (maintaining allowed access to services despite malicious actions). The goal is to meet these challenges with efficient and scalable solutions.