Distributed Systems CH1 Notes

Keh-Harng Feng

August 23, 2017

Contents

1 Characterization of Distributed Systems

1.1 Introduction

Distributed System: A system in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages. Can be spatially separated by any distance. Why? **To share resources.**

Resources:

Hardware: disk drives, printers etc.

Software: files, databases, data objects etc.

Consequences from the definition:

- *Concurrency*: work can be done simultaneously by different components, sharing resources as necssary.
- *No global clock*: when programs need to cooperate the coordination is done by exchanging messages (no single clock synchronized for all components of the system).
- *Independent failures*: any of the component can fail but still leave the other components running. The failure of a component or unexpected termination of a program is not immediately made known to the other components with which it communicates.

1.2 Examples of Distributed Systems:

- Web Search: need to search and index an enormous amount of data. Google's solution involves very large numbers of networked computers in data centers, a distributed file and storage system, lock service (?) and a programming model that supports the management large parallel and distributed computations.
- Massive Multiplayer Online Games (MMOGs): very large numbers of users interact through the Internet in a persistent world. Some approaches are:

1. Client/Server: A central server with a single copy of game state distributed to clients. May use cluster of nodes to increase reliability.
2. Partition the universe to a number of servers (ie: based on geographical location). Easy to extend by adding more servers.

3. P2P technology (each client contributes resources such as storage and processing to acoomodate the game). See Chapters 2 and 10.

- Financial Trading: need to have real-time access to a wide range of data (ie: current share prices, trends, economic/political developments etc).The emphasis is to deliver events such as a drop in a share price reliably and quickly to very large numbers of clients. Client/Server doesn't work! Solution: *distributed event-based systems* (see Chapter 6).

1.3 Trends in Distributed Systems

Changes caused by:

- Pervasiveness of networking technology. For example, the Internet.
- Ubiquitous computing computing + desire for mobile distributed computing.

Ubiquitous computing: constantly harnessing many small, cheap computational devices in users' physical environments to the point of them not being noticed.Devices are generally fixed to their environments.

Mobile computing: laptop computers, smart phones, GPS, etc.Users can take the devie with them.

spontaenous interoperation: allowing visitor devices to quickly communicate with others on the host network.

service discovery: associating visitor devices with suitable local services

- Increasing demand for multimedia services. The system should support storage, transmission and presentation of a range of media types.

discrete media: text messages, pictures

continuous media: audio, videos (has a temporal dimension) -> needs to preserve a thoroughput such as frames per second & maximum delay/latency.

- The view that distributed system is a utility. Physical resources such as storage and processing can be made available to networked computers. Software services can also be made available.

cloud computing: computing as a utility. A cloud is a set of Internet-based application, storage and computing services for most users' needs.Generally implemented on cluster computers to provide the necessary scale and performance.

cluster computer: a set of interconnected computers that cooperate closely to provide a single, integrated high-performance computing capability.

blade servers: minimal computational elements containing processing & storage capabilities.

1.4 Focus on Resource Sharing

service: a part of a computer system that manages a collection of related resources and presents their functionality to users and applications. ie: file service (provides read, write & delete etc).

server: a running program/process on a networked computer that accepts requests to perform service. -> passively respond to clients. Continuously running.

client: the requesting process to a server.-> actively send requests to servers. Only a client as long as the process that sends requests is running.

Technically, client/server only refer to the processes.

1.5 Challenges

1.5.1 Heterogeneity

Differences in networks, computer hardware, operating systems, programming languages and implementations by different developers etc.

Middleware: a software layer that provides a programming abstraction and also to mask the heterogeneity of the underlying components. ie: Common Object Request Broker(CORBA), Java Remote Method Invocation (RMI) etc.

Mobile code: program code that can be trasnferred from one computer to another and run at the destination. ie: Java applets. Javascript programs for browsers.

virtual machine: allows the compiler of a particular language to generate code for a virtual machine instead of a particular hardware. ie: Java compiler for JVM.

1.5.2 Openness

The characterstic that determines whether the system can be extended and reimplemented in various ways. For a distributed system, the most important factor is the degree to which new resource-sharing services can be added and made available.

To achieve openness: require specification and documentation of key software interfaces to be made available to software developers. ex: "Requests For Comments (RFCs)" for the Internet protocols.

To create open systems:

- key interfaces are published
- based on a uniform communication mechanism and published interfaces for access to shared resources.
- can be constructed from heterogenous hardware and software.

1.5.3 Security

Three components:

- Confidentiality (protection against disclosure to unauthorized individuals)
- Integrity (protection against alteration or corruption)
- Availability (protection against interference with access)

1.5.4 Scalability

A system is scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

Challenges:

Controlling the cost of physical resources: extending the system should be at a reasonable cost. In general, for a system with n users to be scalabe, the quantity of physical resources required to support them should be at most O(n).

Controlling the performance loss: for a system to be scalable, the maximum performance loss should be no worse than O(log n).

Preventing software resources running out: a counter example is the IP address running out due to using 32bits.

Avoiding performance bottlenecks: algorithms should be decentralized to avoid having performance bottlenecks. ie: predecessaor of DNS had one central computer with the entire domain name table -> bottleneck! Current solution: partition the tables to different DNS servers throughout the Internet.

1.5.5 Failure handling

Failures in a distributed system are partial -> some components fail, others keep working. Must be handled appropriately!

Challenges:

Detecting failures: ex: using checksums to detect corrupted data in a message or a file. Must manage the presence of failures that cannot be detected but may be suspected.

Masking failures: some detected failures can be hidden or made less severe. Ex:

1. Messages can be retransmitted if they fail to arrive.
2. Data can be backed up.

Tolerating failures: Ex: web browser cannot contact a web server -> inform problem then stop trying.

Recovery from failures: design that allows data to be recovered or rolled back after failure.

Redudancy: services that can tolerate failures by having backup plans. ie: at least two different routes between any two routers, DNS replicated in at least two different servers, database replicated in several servers to ensure accessbility after failure of one server.

A distributed system should provide a high degree of availability in the face of hardware faults.

availability: a measure of the proportion of time the system is available for use.

1.5.6 Concurrency

If a shared resource only allows one client request at a time it will limit throughput. Solution: allow multiple client requests to be processed concurrently. Must ensure operations on a shared resource operate correctly in a concurrent environment (ie: data remains consistent)!

1.5.7 Transparency

Conceal from the user and the application programmer of the components in a distributed system, so that the system can be seen as a whole entity as opposed to a collection of components.

Eight forms of transparency:

Access transparency: allows local and remote resources to be accessed using identical operations. <- **one of the most important!**

Location transparency: allows resources to be accessed without knowledge of their physical or network location (ie: not knowing their building or IP address). <- **one of the most important!**

These two significantly affect the utilization of distributed resources. Sometimes bundled together as *network transparency*.

Concurrency transparency: allows several processes to operate concurrently using shared resources without interference between them.

Replication transparency: allows multiple instances of resources to be used to increase reliability under the hood.

Failure transparency: enables concleament of faults to allow completiont of tasks despite failures.

Mobility transparency: allows the movement of resources and clients without affecting the operations.

Performamnce transparency: allows the system to be dynamically reconfigured based on the load.

Scaling transparency: allows the system and application to expand in scale without changing system structure or algorithm.

1.5.8 Quality of service

Four measures: *reliability*, *security*, *performamnce*, *adaptability*.