Parallel & Distributed System

Distributed Systems Architecture-01

Md. Biplob Hosen

Lecturer, IIT-JU

Email: biplob.hosen@juniv.edu

Contents

- Architectural Styles
- Layered Architecture
- Layered Communication Protocols
- Application Layering

Reference Books

- Distributed Systems: Principles and Paradigms, 3^{rd} Edition by Andrew S. Tanenbaum & Maarten van Steen, Publisher: Pearson Prentice Hall.

Overview

- The organization of distributed systems is mostly about the **software components** that constitute the system.
- These software architectures tell us how the various software components are to be organized and how they should interact.
- An important goal of distributed systems is to separate applications from underlying platforms by providing a middleware layer.
- The actual realization of a distributed system requires that we instantiate and place software components on real machines.
- Initially, we will look into traditional centralized architectures in which a single server implements most of the software components (and thus functionality), while remote clients can access that server using simple communication means.
- In addition, we consider decentralized peer-to-peer architectures in which all nodes more or less play equal roles.
- Many real-world distributed systems are often organized in a hybrid fashion, combining elements from centralized and decentralized architectures.

Architectural Styles

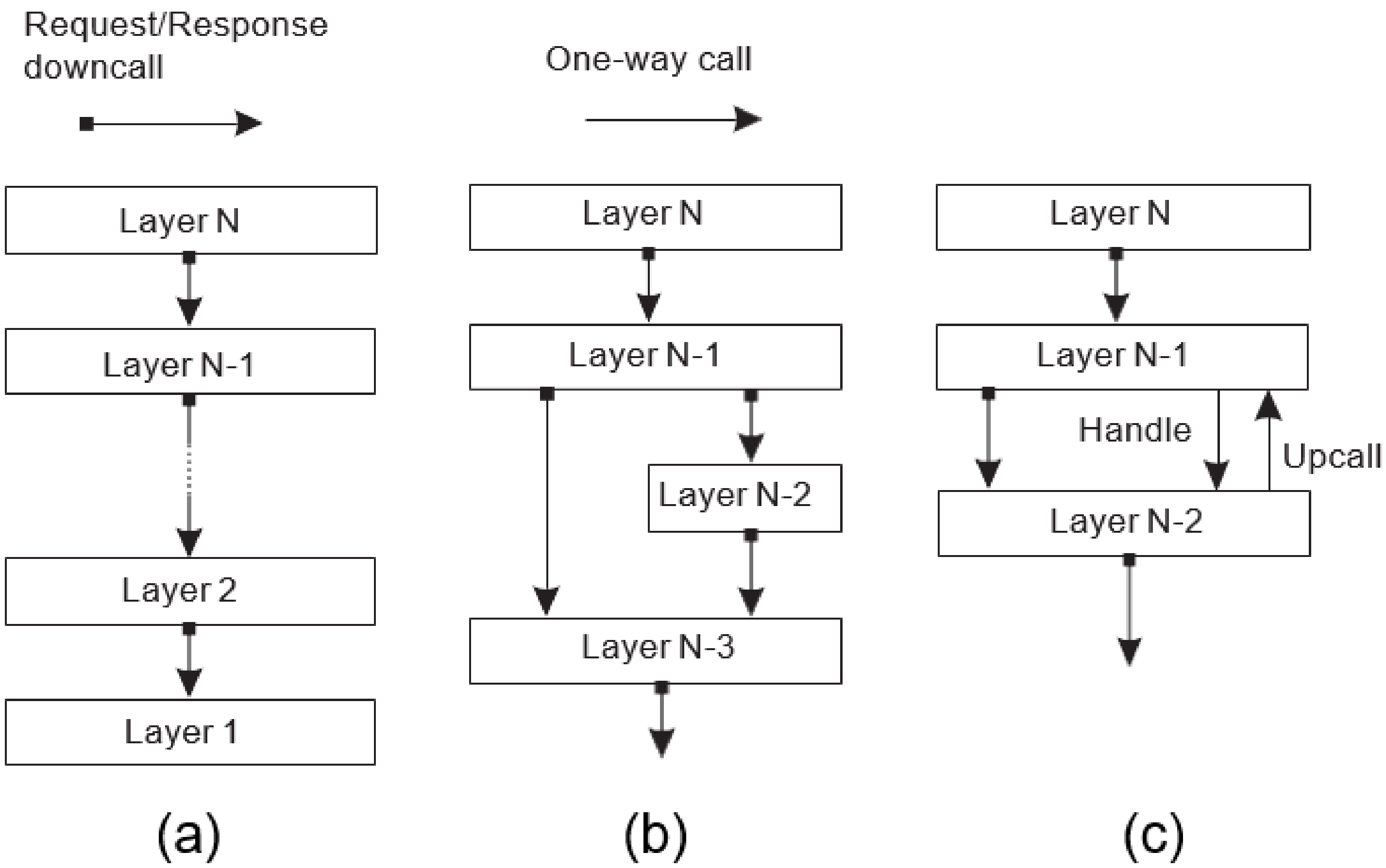
- **Software architecture** is the logical organization of a distributed system into software components.
- Designing or adopting an architecture is crucial for the successful development of large software systems.
- An **architectural style** is formulated in terms of-
  - Replaceable **components** with well-defined interfaces.
  - The way that components are connected to each other.
  - The data exchanged between components.
  - How these components and connectors are jointly configured into a system.
- That a component can be replaced, and, in particular, while a system continues to operate, is important.
- This is due to the fact that in many cases, it is not an option to shut down a system for maintenance. At best, only parts of it may be put temporarily out of order.
- Replacing a component can be done only if its interfaces remain untouched.

Continue…

- **Connector:** A mechanism that mediates communication, coordination, or cooperation among components.
- A connector allows for the flow of control and data between components.
- Example: facilities for (remote) procedure call, messaging, or streaming.
- Using components and connectors, we can come to various configurations, which, in turn, have been classified into architectural styles.
- Several styles have by now been identified, of which the most important ones for distributed systems are:
  - Layered architectures
  - Object-based architectures
  - Resource-centered architectures
  - Event-based architectures

Layered Architecture

- The basic idea for the layered style is simple: components are organized in a layered fashion where a component at layer L_j can make a downcall to a component at a lower-level layer L_i (with $i < j$) and generally expects a response.
- Only in exceptional cases will an upcall be made to a higher-level component.
- The three common cases are shown in the figure.



Continue…

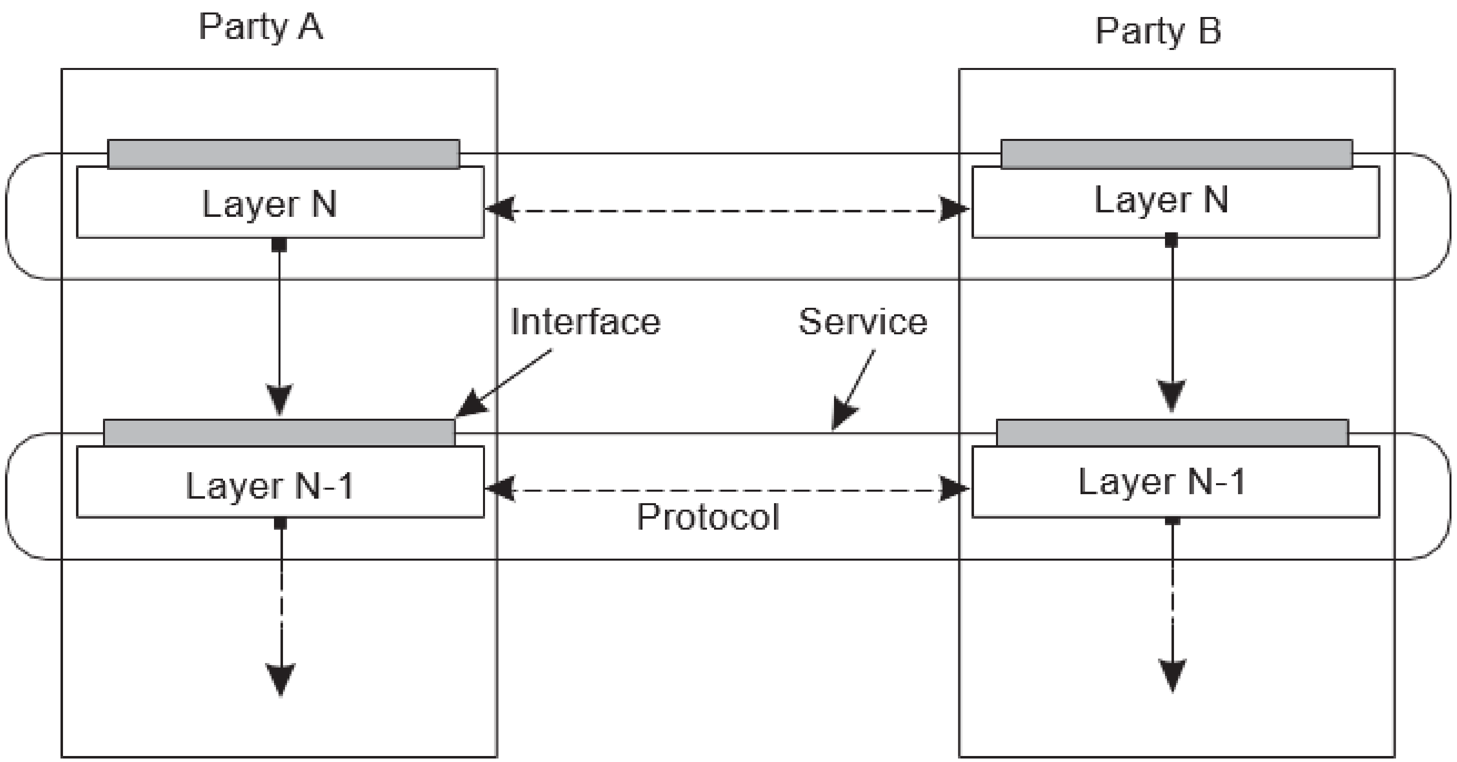
- Figure (a) shows a standard organization in which only downcalls to the next lower layer are made. This organization is commonly deployed in the case of network communication.
- In many situations we also encounter the organization shown in Figure (b). Consider, for example, an application A that makes use of a library L_{OS} to interface to an operating system. At the same time, the application uses a specialized mathematical library L_{math} that has been implemented by also making use of L_{OS}. In this case, referring to Figure (b), A is implemented at layer N−1, L_{math} at layer N−2, and L_{OS} which is common to both of them, at layer N−3.
- Finally, a special situation is shown in Figure (c). In some cases, it is convenient to have a lower layer do an upcall to its next higher layer. A typical example is when an operating system signals the occurrence of an event, to which end it calls a user-defined operation for which an application had previously passed a reference.

Layered Communication Protocols

- A well-known and ubiquitously applied layered architecture is that of so called communication-protocol stacks.
- In communication-protocol stacks, each layer implements one or several communication services allowing data to be sent from a destination to one or several targets.
- To this end, each layer offers an interface specifying the functions that can be called.
- In principle, the interface should completely hide the actual implementation of a service.
- Another important concept in the case of communication is that of a (communication) protocol, which describes the rules that parties will follow in order to exchange information.
- It is important to understand the difference between a **service** offered by a layer, the **interface** by which that service is made available, and the **protocol** that a layer implements to establish communication.

Continue…

- This distinction is shown in the figure.
- To make this distinction clear, consider a reliable, connection-oriented service, which is provided by many communication systems.



- In this case, a communicating party first needs to set up a connection to another party before the two can send and receive messages.
- Being reliable means that strong guarantees will be given that sent messages will indeed be delivered to the other side, even when there is a high risk that messages may be lost (as, for example, may be the case when using a wireless medium).

Continue…

- In addition, such services generally also ensure that messages are delivered in the same order as that they were sent.
- This kind of service is realized in the Internet by means of the Transmission Control Protocol (TCP).
- The protocol specifies which messages are to be exchanged for setting up or tearing down a connection, what needs to be done to preserve the ordering of transferred data, and what both parties need to do to detect and correct data that was lost during transmission.
- The service is made available in the form of a relatively simple programming interface, containing calls to set up a connection, send and receive messages, and to tear down the connection again.
- In fact, there are different interfaces available, often dependent on operating system or programming language used.
- Likewise, there are many different implementations of the protocol and its interfaces.

Example: Two-Parties Communication

- To make this distinction between service, interface, and protocol more concrete, consider the following two communicating parties, also known as a client and a server, respectively, expressed in Python.

Server

```
from socket  import *
s = socket(AF_INET, SOCK_STREAM)
(conn, addr) = s.accept()  # returns new socket and addr. client
while True:                # forever
   data = conn.recv(1024)  # receive data from client
   if not data: break      # stop if client stopped
   conn.send(str(data)+"*") # return sent data plus an "*"
conn.close()               # close the connection
```

Client

```
from socket  import *
s = socket(AF_INET, SOCK_STREAM)
s.connect((HOST, PORT)) # connect to server (block until accepted)
s.send('Hello, world')  # send some data
data = s.recv(1024)     # receive the response
print data              # print the result
s.close()               # close the connection
```

Continue…

- In this example, a server is created that makes use of a connection-oriented service as offered by the socket library available in Python.
- This service allows two communicating parties to reliably send and receive data over a connection.
- The main functions available in its interface are:
    - socket(): to create an object representing the connection.
    - accept(): a blocking call to wait for incoming connection requests; if successful, the call returns a new socket for a separate connection.
    - connect(): to set up a connection to a specified party.
    - close(): to tear down a connection.
    - send(), recv(): to send and receive data over a connection, respectively.

Continue…

- The combination of constants AF_INET and SOCK_STREAM is used to specify that the TCP protocol should be used in the communication between the two parties.
- These two constants can be seen as part of the interface, whereas making use of TCP is part of the offered service.
- How TCP is implemented, or for that matter any part of the communication service is hidden completely from the applications.
- Finally, also note that these two programs implicitly adhere to an application-level protocol: apparently, if the client sends some data, the server will return it.
- Indeed, it operates as an echo server where the server adds an asterisk to the data sent by the client.

Application Layering

- Let us now turn our attention to the logical layering of applications.
- Considering that a large class of distributed applications is targeted toward supporting user or application access to databases, many people have advocated a distinction between three logical levels, essentially following a layered architectural style:
  - The application-interface level
  - The processing level
  - The data level
- In line with this layering, we see that applications can often be constructed from roughly three different pieces:

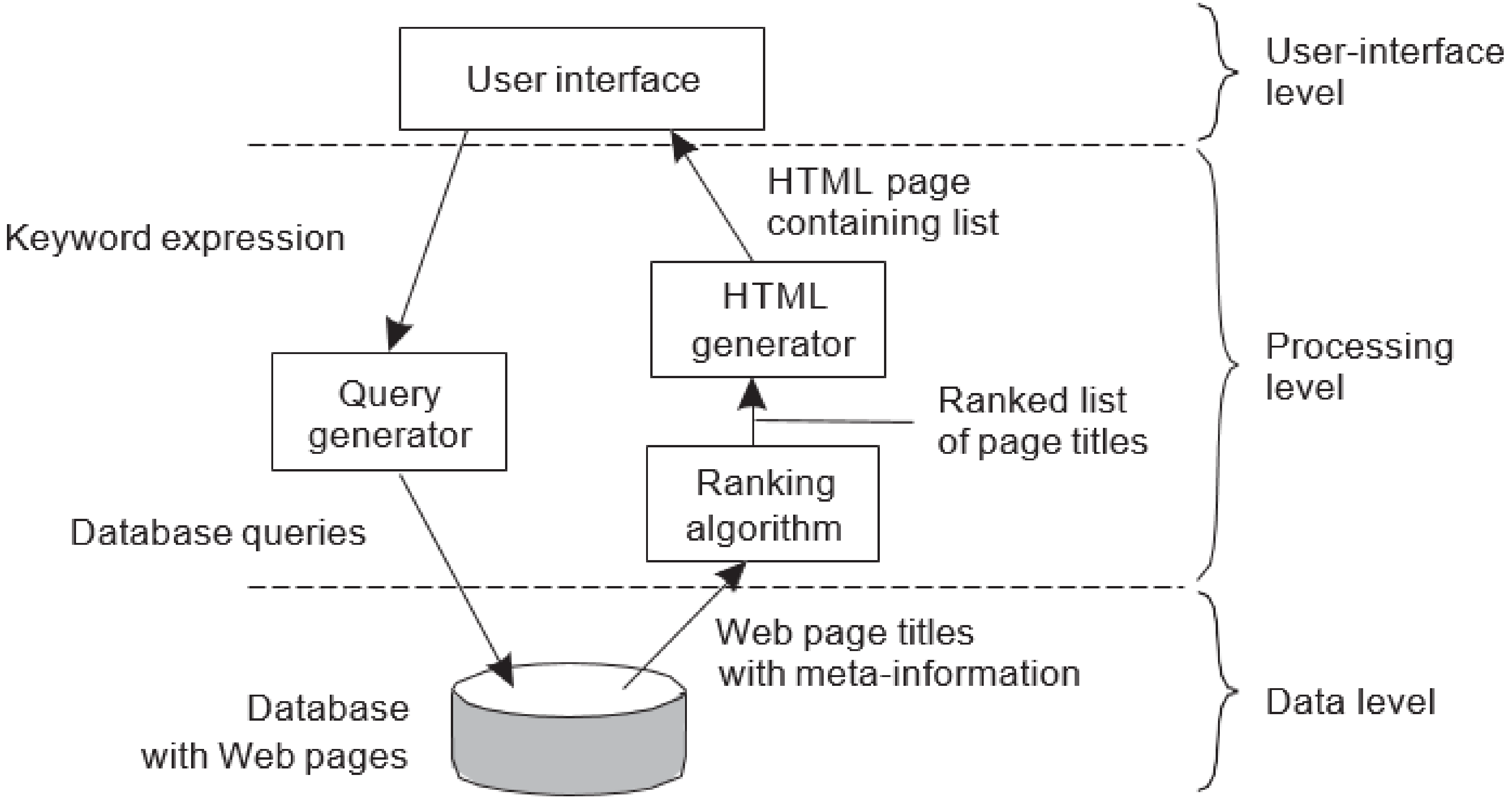
i) a part that handles interaction with a user or some external application,
ii) a part that operates on a database or file system,
iii) a middle part that generally contains the core functionality of the application.

- This middle part is logically placed at the processing level.
- In contrast to user interfaces and databases, there are not many aspects common to the processing level.
- Therefore, we shall give a number of examples to make this level clearer.

Application Layering (Example-01)

- Consider an Internet search engine.
- Ignoring all the animated banners, images, and other fancy window dressing, the user interface of a search engine can be very simple: a user types in a string of keywords and is subsequently presented with a list of titles of Web pages.
- The back end is formed by a huge database of Web pages that have been pre-fetched and indexed.
- The core of the search engine is a program that transforms the user’s string of keywords into one or more database queries.
- It subsequently ranks the results into a list, and transforms that list into a series of HTML pages.
- This information retrieval part is typically placed at the processing level.
- Following figure shows this organization.

Continue…



User interface
User-interface level
HTML page containing list
Keyword expression
HTML generator
Query generator
Ranked list of page titles
Processing level
Ranking algorithm
Database queries
Web page titles with meta-information
Database with Web pages
Data level

Application Layering (Example-02)

- As a second example, consider a decision support system for stock brokerage.
- Analogous to a search engine, such a system can be divided into the following three layers:

1) A front end implementing the user interface or offering a programming interface to external applications.
2) A back end for accessing a database with the financial data.
3) The analysis programs between these two.

- Analysis of financial data may require sophisticated methods and techniques from statistics and artificial intelligence.
- In some cases, the core of a financial decision support system may even need to be executed on high performance computers in order to achieve the throughput and responsiveness that is expected from its users.

Application Layering (Example-03)

- As a last example, consider a typical desktop package, consisting of a word processor, a spreadsheet application, communication facilities, and so on.
- Such “office” suites are generally integrated through a common user interface that supports integrated document management, and operates on files from the user’s home directory.
- (In an office environment, this home directory is often placed on a remote file server.)
- In this example, the processing level consists of a relatively large collection of programs, each having rather simple processing capabilities.
- The data level contains the programs that maintain the actual data on which the applications operate.
- An important property of this level is that data are often persistent, that is, even if no application is running, data will be stored somewhere for next use.

Continue…

- In its simplest form, the data level consists of a file system, but it is also common to use a full-fledged database.
- Besides merely storing data, the data level is generally also responsible for keeping data consistent across different applications.
- When databases are being used, maintaining consistency means that metadata such as table descriptions, entry constraints and application-specific metadata are also stored at this level.
- For example, in the case of a bank, we may want to generate a notification when a customer’s credit card debt reaches a certain value.
- This type of information can be maintained through a database trigger that activates a handler for that trigger at the appropriate moment.

Thank You ☺