Dr Markus Hagenbuchner
markus@uow.edu.au

CSCI319

Distributed Systems
Chapter 2 - Architectures
ARCHITECTURES
Lecture notes based on the textbook by Tannenbaum

Study objectives:

1. Understand important system architectures and architectural styles in DS

2. Explain centralized and decentralized architectures.

3. Explain structured, unstructured, and hybrid structures.

4. Understand the differences in peer-to-peer system structures.

5. Explain system adaptation in DS.
In this section we will:

• Give an overview of common architectural styles
• Give an overview of common distributed system architectures
• Introduce to peer-to-peer systems.
• Introduce principles of adaptive systems…
• On a number of examples
Architectural Styles (1)

The **architectural style** defines how the elements of a DS communicate.

Important styles of architecture for DS:
Ordered from most tightly coupled to most loosely coupled:

1. Layered architectures
2. Object-based architectures
3. Data-centered architectures
4. Event-based architectures
Architectural Styles (2)

The layered architectural style. Each layer can be a process, resource, or a system. The layers are said to be tightly coupled.

Example: Any multi-tiered DS.
Architectural Styles (3)

The object-based architectural style. Note that the topology needs to be defined.

Example: Most cluster computer systems.
Architectural Styles (4)

With event-based architectural styles, the components are loosely coupled and choose message which is of use to the component.

Example: Publish/subscribe systems (i.e. most social networks)
Architectural Styles (5)

The shared data-space architectural style. Here the components are even more loosely coupled. Main challenge here: synchronization.

Example: Distributed DB systems, many cloud computing systems
System architectures

The DS architecture defines how resources (data, software, memory, etc.) in a DS are distributed.

Important architectures for distributed systems:

- Centralized architectures
- Decentralized architectures
- Hybrid architectures
Application Layering in centralized architectures (1)

Centralized architectures often use a layered architectural style.

Example: Layers of the centralized architecture:

- The user-interface level (i.e. the X-windows interface, a keyboard, mouse)
- The processing level (i.e. applications that provide processing methods)
- The data level (i.e. a DB, disk, storage)

Layering can help achieve transparency, and results in modularization.
Application Layering (2)

Example: The organization of an Internet search engine into three different layers can be illustrated as follows:

- **User interface level**
  - User interface
  - Keyword expression
  - Query generator
  - Database queries
  - Database with Web pages

- **Processing level**
  - HTML page containing list
  - HTML generator
  - Ranked list of page titles

- **Data level**
  - Web page titles with meta-information
  - Ranking algorithm
We have already learned that a Internet search engine makes use of distributed file systems (i.e., GFS). So why are internet search engines said to be centralized data systems”? Note, there exist decentralized search engines.
We have already learned that an Internet search engine makes use of a **distributed** file systems (i.e., GFS). So why are internet search engines said to be **centralized data systems**”?

The search engine is a client of the distributed file system (DFS). The DFS provides a single system view as per definition of DS. Hence, the search engine is centralized, while the DFS itself is decentralized.

Note, there are decentralized search engines in existence which are normally confided to Intranet applications.
Multitiered Architectures (1)

If a layer is implemented on a dedicated machine then this is called a \textit{tier}. A layered architectural style in which the layers are implemented on dedicated machines is called a \textit{multi-tiered} architectural style.

The simplest organization is to have only two types of machines:

- A client machine (tier 1) containing the programs implementing (possibly only parts of) the user-interface level
- A server machine (tier 2) containing the rest (such as the programs implementing the processing and data level)
Multitiered Architectures (2)

Options available in a 2-tiered client-server architecture:

Thin client ←  Fat client
Multitiered Architectures (3)

In multi-tiered systems it is common that a server also acts as a client. An example of a server acting as client.

Note that latency increases with the number of tiers. Hence, in practice it is rare to have more than 4 tiers.
Decentralized Architectures

- Resource may be distributed across a number of computing nodes.
- Structure of the underlying (communication) architecture becomes very important.

- Example: Peer to peer networks
  - Can be structured or unstructured
Example: The mapping of data items onto nodes in Chord. Each node is responsible for a hash-key which identifies a resource. Index nodes are responsible for a number of predecessors. A search starts at any index node.
Structured de-centralized Peer-to-Peer Architectures (2)

The nodes in the Chord system:

- Each refers to a data item mapped to its ID
- Are organized in a ring structure ordered by ID
- Index nodes “know” about some other index nodes (more on this later in chapter Naming).
- A lookup of a node returns the ID and its associated data keys.

The Chord system can find a data item in \( \log(N) \) time. This is similar to binary search, and is equivalent to a distributed hashtable where the ID refers to the hash value.
Structured Peer-to-Peer Architectures (2)

Example 2a: Content Addressable Network (CAN). A CAN is a structured peer-to-peer system. The mapping of data items onto nodes in CAN is by an m-dimensional address. Each node in the CAN is responsible for a sub-address space. Think about how a data item can be inserted/found in CAN.
Example 2b: As with any P2P, nodes can be added/removed during runtime. With CAN, a region is split when a node joins the CAN.
Structured Peer-to-Peer Architectures (4)

Joining regions when a node leaves.

a.) What will happen when node (0.2,0.8) leaves?

b.) What will happen when node (0.6,0.7) leaves?

Think about what implication the joining of regions has on the algorithm that inserts/finds data in the CAN.
What does CAN stand for?

How does CAN work?

Of how many dimensions can a CAN be?

What is the main disadvantage of CAN?
What does CAN stand for? Content Addressable Network

How does CAN work? It realizes a tree structure through a geometric decomposition of the address space. The efficiency is drawn from the symmetric decomposition of the address space.

Of how many dimensions can a CAN be? Any natural number.

What is the main disadvantage of CAN? Segments can become irregular shaped when nodes leave. This can impact efficiency, and may require the re-building of the segmentation -> overhead.
Unstructured Peer-to-Peer (1)

- Does not assume an overall structure amongst peers.
- Peers are connected at random. Thus, this produces a random graph.
- Peers have a partial view of neighbors which creates the topology of the graph.
- Challenges:
  - Requires increased robustness.
  - How to locate a data item?
Unstructured Peer-to-Peer (2)

continued:

• A random graph in which nodes have a partial view of “neighbors” this creates an overlay network (which defines the topology of the graph).

• Nodes in a P2P are assumed to join and leave at any time.

• A strategy to propagate and maintain partial views is as follows:
A strategy to propagate a partial view: The steps taken by the active thread (the hub which initiates exchange of partial views):

**Actions by active thread (periodically repeated):**

select a peer P from the current partial view;
if PUSH_MODE {
    mybuffer = [(MyAddress, 0)];
    permute partial view;
    move H oldest entries to the end;
    append first c/2 entries to mybuffer;
    send mybuffer to P;
} else {
    send trigger to P;
}
if PULL_MODE {
    receive P’s buffer;
}
construct a new partial view from the current one and P’s buffer;
increment the age of every entry in the new partial view;

(a)
The steps take by the passive thread (a leaf):

**Actions by passive thread:**

- receive buffer from any process Q;
- if PULL_MODE {
  - mybuffer = [(MyAddress, 0)];
  - permute partial view;
  - move H oldest entries to the end;
  - append first c/2 entries to mybuffer;
  - send mybuffer to P;
}
- construct a new partial view from the current one and P’s buffer;
- increment the age of every entry in the new partial view;

(b)

Note that this causes a hub to exchange views with another hub.
Interactive slide

Which of the following two terms “static” “dynamic” is correct in the following sentence?
In unstructured peer-to-peer architectures, the topology is ____________.

Give the main advantage and disadvantage to unstructured peer-to-peer architectures:

•

•

•
Which of the following two terms “static” “dynamic” is correct in the following sentence?
In unstructured peer-to-peer architectures, the topology is dynamic.

Give the main advantage and disadvantage to unstructured peer-to-peer architectures:
• Is robust against failure of a node due to redundancies.
• Produces an increased network overhead.
• Need to flood the network to locate an item/resource.
Interactive slide: Unstructured vs structured Peer-to-Peer

- Advantages of structured P2P:
  - Can locate a data item quickly
  - Less network overhead

- Advantages of unstructured P2P:
  - Robust against failures
  - Efficient in very dynamic environments

- How can we combine the advantages?
  - Semi-structured P2P
  - Topology management
Example of a two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems.
Topology Management of Overlay Networks (2)

An overlay network creates a 2\textsuperscript{nd} view (in addition to the random partial view). The aim of the 2\textsuperscript{nd} view is to obtain a partially structured view so that search requests can be directed.

An overlay network reduces the communication overhead of search requests because the requests can be “directed”.

A neighborhood relationship needs to be defined (i.e., based on address space, geographic location of peers, etc.)
Semi-structured P2P

An alternative solution: Superpeers. For example: A hierarchical organization of nodes into a superpeer network.

Superpeers split a large P2P into smaller subsets thus reducing communication overhead.
Interactive slide

What main issues does a superpeer network address?

Main disadvantage of a superpeer network:
What main issues does a superpeer network address?

- Helps to locate a data item more quickly by segmenting the network into sub-structures which are rooted by a (super-)peer.
- Helps to introduce (some) structure into an otherwise unstructured P2P.

Main disadvantage of a superpeer network:
- Requires superpeers (reliable and long lived peers).
Hybrid structures

- Elements of structured and unstructured architectures found in the same system.
- Some examples are:
  - edge server (servers are located at end-points, the edge, in a DS)
  - many collaborative systems
Edge-Server Systems

The Internet can be seen as consisting of a collection of edge servers.
Collaborative Distributed Systems (1)

BitTorrent uses another hybrid architecture known as a collaborative system. The principal working of BitTorrent is as follows:

The nodes are partially redundant.
Collaborative Distributed Systems (2)

BitTorrent is said to be of a hybrid structure because:

- The overlay structure between nodes and client is not well defined (unstructured component).
  - Connections between a node and clients is not fixed but established dynamically as needed.
- The network between web server, file server, and tracker is well defined (structured component).
System Adaptability

Consider a file server which is overwhelmed by client requests. How should the server react?

- Ignore requests until load reduces (not a good solution), or
- Respond with an error, or
- Relay requests to secondary servers?

But how can the later two be achieved?
Interceptors

One solution to system adaptability is through interceptors which handle remote-object invocations. Example: Client side interceptors.
More General Approaches to Adaptive Systems

Three basic approaches to realize adaptive systems:

• Separation of concerns (task specific)
• Computational reflection (monitor and adapt itself i.e. feedback-control in Globule)
• Component-based design (atomic decomposition of system components i.e. libraries which are called as needed)
The Feedback Control Model

Example of a self-management system. The logical organization of a feedback control system can be illustrated as follows.
Example: Cluster System Monitoring with Astrolabe, PBS, or similar (1)

<table>
<thead>
<tr>
<th>Machine A</th>
<th>Machine B</th>
<th>Machine C</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>avg_load</th>
<th>avg_mem</th>
<th>avg_procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.06</td>
<td>0.55</td>
<td>47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IP-addr</th>
<th>load</th>
<th>mem</th>
<th>procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2</td>
<td>0.03</td>
<td>0.80</td>
<td>43</td>
</tr>
<tr>
<td>192.168.1.3</td>
<td>0.05</td>
<td>0.50</td>
<td>20</td>
</tr>
<tr>
<td>192.168.1.4</td>
<td>0.10</td>
<td>0.35</td>
<td>78</td>
</tr>
</tbody>
</table>
Example: Systems Monitoring with Astrolabe, PBS, or similar (2)

The system monitors load, memory utilization, number of processes. If access for resources is requested, then the best node (i.e. lowest load) is assigned.

This approach can achieve either:

• Load balancing across all nodes in the DS.
• Maximise usage of individual nodes.
Adaptation through prediction (1)

Adaption can also be achieved through prediction. This is particularly important for Web-based DS.

But how can this be achieved?

I.e. analyze past access patterns in order to obtain conclusions about likely future occurrences. For example, if in the past a high load is typically observed at noon, then it is likely to observe a high load at noon in the future. The system can be designed to account for such knowledge.
Adaptation through prediction (2)

But how far into the past should one go? This depends on the circumstances. If we go too far into the past then

- Too many patterns may need to be analyzed
- Access patterns may have changed over time making a prediction inaccurate or even impossible.

![Graph showing the relationship between error in prediction and trace length used for selecting next policy.](image)
Summary

This chapter provided:

• an overview of common architecture styles
• an overview of common distributed architectures
• An overview of organization of communication lines (the topology)
• An overview of organization of system components on a number of examples.

We find that a good choice of a system architecture can aid the design of scalable and fault tolerant systems.