CS 454/654 Notes
Khuzaima Daudjee

Introduction to Distributed Systems

This module focuses on the fundamentals of distributed systems.
Slide 2:
The general definition of a distributed system is that it is a collection of components (elements) that are independent and autonomous, i.e., each component will perform a set of tasks and make decisions independently of what other components will do. Collectively, the components comprise the distributed system.
By component, we will typically mean a software component or process such as a scheduler in a server.
A key concept of a distributed systems is its ability to provide a single system image, i.e., the collection of software components working together that make up the distributed system appear as if it is a single system (or server) to the user or the outside world. This is a level of abstraction that a distributed system provides.
Autonomous nodes need to collaborate to get tasks done. How this collaboration is established is the heart of distributed systems.
Slide 3: One aspect of being autonomous is that no one process has exactly the same notion of time. This means that each process is likely to be either behind or ahead of true time (global notion of time). This can create problems such as if a process P1 requests service from a server S before a process P2 but the timestamping scheme that S uses to order messages assigns a lower timestamp to P2's message than to P1's, which process should get service first? We'll address time synchronization later in the course.
It may be necessary to know which nodes are part of your distributed system (group membership) as nodes may have left or new ones may have joined.
Communicating with an authorized node – leads to security concerns. To address these concerns, may need authentication mechanism as well as privacy when communicating (this is CS 458/658 material that we won't get into).
Slide 4:
Sometimes, a distributed system can be described as an overlay of network of components, i.e., software components that talk (communicate) with each other through the (physical) network. In an overlay network, nodes (or servers) can join, leave or fail at any time.
To communicate with its peers (these are other nodes in the system/network), each node in an overlay network needs to find them.
Structured overlay networks (or systems) use a data structure to record, and find, peers.
In unstructured overlay networks/systems, a node typically knows which peer nodes are in its neighbourhood – these neighbours are peers that are are directly connected (via a network) to the node. Thus, exploration of the distributed system (or peer-to-peer network) is done via contacting

a neighbouring peer and that peer then in turn contacts its neighbour and so on until the target or destination peer is reached.

Slide 5:

A key aspect of a coherent distributed system is offering a single-system image that makes the distributed system *appear* as if the functionality offered by all components that make up the distributed system is offered by a single component. An example of a single-system view or image is `gmail.com`; when you interact with this e-mail service, you enter a single URL and the service makes you feel and think that it is a single server you are interacting with that is offering your e-mail service. `gmail.com` does not show/reveal to a user all of the components that are interacting with each other behind the scenes, i.e., in a data center possibly in a different geographic zone, to respond to your requests from your (e-mail) client. There are many distributed components that are working together *coherently* to provide this e-mail service: machines, server software, networks, storage devices, operating systems, and so on. There are many popular examples of services that provide a single-system view/image, e.g., `google.com, amazon.ca, quest.uwaterloo.ca`. In fact, most web-based services strive to provide you with this distribution *transparency* where you don't see what's happening behind the scenes in providing the user with the distributed service.

Slide 6:

Middleware (MW) acts as a glue for components to work with each other in a distributed system. MW provides abstractions for APIs and their services. For example, an application is offered the same interface on every machine. MW is also a uniform layer that can provide services such as for interprocess communication or security to application (clients). MW allows applications to (i) seamlessly talk to each other (ii) hides differences in operating systems (OSs) and hardware (H/W) from each application (iii) allows applications to share and deploy resources across the network.

Examples of MW services:

– communication: remote procecure call (RPC): allows invocation of remote function as if available locally

– transaction manager: a service that can provide the all-or-nothing effect for execution of requests as transactions

– service composition: web services/pages that combine and aggregate data from different sources

– reliability: group communication service that allows message delivery to all servers in a group