A. Arpaci-Dusseau
CS739: Distributed Systems

Department of Computer Science
University of Wisconsin, Madison

Survey: Distributed Operating Systems

Introduction

- What is a distributed operating system?
- What were the goals of distributed systems (in 1980s)?
- What were the defined challenges of distributed systems (in 1980s)?
- What were the (unstated) assumptions made about the computing environment?

	1980s	Modern and P2P
Scale:		
Node similarity:		
Network:		
Membership:		

Discuss the following issues in terms of how distributed systems will have to adapt when scale, similarity, network, and membership change in "modern" systems.

1 Network Operating Systems

- How is a network operating system different than a distributed OS?
- How are network OSes criticized here?
- Are the criticisms of network systems more or less applicable given "modern" environments?

2 Design Issues

2.1 Communication Primitives

- **Message Passing:** Early distributed systems built specialized communication primitives; for example, one could choose between reliable vs. unreliable and blocking vs. nonblocking and buffered vs. unbuffered. Why? Should modern systems make a different choice?
- Interesting example of specialization presented in Case Studies: Amoeba uses unbuffered messages; message dropped if receiver is not ready to receive; sender retries message later if not acknolwedged. Why might this be okay between a client and a multi-threaded server? Would this be more or less reasonable in a modern system?
- **Error Handling:** The ability to handle client and server crashes is important for availability and reliability. What is nice about protocols with idempotent operations?
- **Implementation Issues:** When implementing systems, it has been found to be important to avoid copies, amortize startup costs of sending small messages, avoid extra round-trips. Any change for modern systems?

2.2 Naming and Protection

- **Name Servers:** Why is a name server needed in a distributed system? This paper suggests a number of approaches in which a name server can be structured. What are they? How well will these approaches work in a modern system?

2.3 Resource Management

- **Processor Allocation:** Nodes can be managed in a hierarchical k-way tree. Why does this work well for a classic distributed system? How well does this approach work for a modern system?
- **Scheduling:** When time-sharing communicating processes across multiple nodes, it is important to coschedule the communicating processes. What happens if processes are not coscheduled? In a classic system, how would this coordinated context-switch occur? How well will this work in a modern system? Is this as important of a problem?
- **Load Balancing:** How processes are allocated to nodes depends upon one's metrics. Why might one want communicating processes to be placed on the same node? Why on different nodes? What considerations are most important currently?
- **Heuristic Load Balancing:** After a process computes its own load, how can the others find out this load? Which works in modern systems?
- **Heuristic Load Balancing:** Why is it so much harder to migrate a running job, compared to perform remote execution of a newly creately job? Why would someone prefer migration?

2.4 Fault Tolerance

- What is the difference between reliability and availability?
- **Redundancy Techniques:** One way to make a system reliable is through redundancy, in which multiple nodes perform the same computation after ensuring that each has received the same message; if a node crashes, the others can continue. What are the drawbacks of this approach?
- **Redundancy Techniques:** A variant of this approach performs a periodic checkpoint of each process, records each message on another node, and then plays back the messages against the checkpoint to recover a failed process. What are the pros and cons of this approach relative to the previous one?
- **Redundancy Techniques:** N-version programming helps avoid software bugs (in addition to hardware faults) by running different versions of the program on different nodes. Will this work better or worse now?

2.5 Services

- **Server Structure:** What is the problem with a main loop of a server looking as follows:

```
while (1) {
  getRequest();
  doRequest();
  sendReply();
}
```

  What is the proposed solution? Will it work for modern systems?

- **Protection:** Why is authentication more complex in a distributed system than on a single machine? How can encryption be used to make a capability?
- **File Service Caching:** Caching on clients greatly improves performance, but what problem does it raise? What are some of the different approaches for (at least partially) fixing this problem?
- **File System Reliability:** Data can be made more reliable by making muliple copies of it on different disks. For what type of workloads can this improve performance? For type of workload does it raise new complexities? How do the issues change for modern systems?
- **Time Service:** Why is it impossible to have a single global time in a distributed system?