# Distributed Operating Systems

Suresh Sridharan
*CS 739: Distributed Systems*
*University of Wisconsin, Madison*

$25^{th}$ January, 2006
Spring 2006

## 1 Overview

The paper aims at providing a complete survey of contemporary distributed systems of that time. The authors **define** a distributed operating system distinguishing it from networked systems. The paper endeavours to also explain in detail the key design issues involved in the building of such systems. A few examples of research projects are considered in light of the issues discussed.

A distributed system is defined to be one that looks to its users like an ordinary centralized operating system but runs on multiple, independent CPUs. In other words, a distributed operating system appears to users as a single coherent system. In a networked system, users are aware that they are using a specific system for a given service, with each computer running its own private operating system, with little fault tolerance. The distinguishing characteristic is hence that of transparency.*It can be argued as to how transparent the system needs to be. A utility, to display information of where processes are getting executed, or where files are stored, does not necessarily have to be enough to brand the system as non-transparent*.

## 2 Goals and Assumptions

Distributed Operating systems tend to leverage the availability of cheap microprocessor technology to achieve performance similar to their more expensive counterparts. Obtaining computing power proportionate to processors added to the system, reliability and availability in the face of failure of certain system components are other key advantages of distributed operating systems.

Some of the characteristics of the system are listed below:

| | |
|---|---|
| Model | Client/Server |
| Workload | *General purpose* |
| Membership | Static |
| Scale | 10's of machines |
| Network | Local-area |
| Homogenity/Heterogenity of machines | *Clients/servers heterogeneous*<br>*Clients homogeneous* |

*The attributes listed in the above table differ from those of P2P distributed systems in that, in P2P systems, each host is both a client and a server, workload is application specific, membership is dynamic, scale is much larger (millions), WAN is usually involved, and resources at hosts are heterogeneous*.

## 3 Design Issues

### 3.1 Communication Primitives

Due to the availability of high bandwidth network links, the price (computationally) for the ISO model is too prohibitive to be used in distributed systems. *The use of standard protocols like TCP and UDP was probably avoided due to the same reason (for having to go through the protocol stack), and perhaps because of the addition of a larger number of bytes as headers or checksum data.* The paper discusses the fundamental tradeoffs between reliable vs unreliable primitives, and blocking vs non-blocking primitives. *Not all operations in a system are idempotent, and hence can cause problems with the se-*

*mantics that can be guaranteed by RPC* (since exactly once semantics is almost impossible to achieve). In order to make message passing efficient, RPC needs to avoid copying data if possible between different levels. Messages need to be made longer to amortize overhead. Allowing ACKs to be replaced by subsequent request/reply messages (higher-level knowledge) saves on the number of messages sent over the network. *Hence, the RPC implementation avoids using a reliable connection oriented protocol like TCP.*

## 3.2 Naming and Protection

The problem of naming is to associate logical names to actual physical storage locations in the system (the physical names). The simplest model of having a centralized name server creates a bottleneck in the case of systems of a larger scale.

Problems arise in the domain of protection because names/identifiers are not derived from a global name space. This allows for the possibility of having more than one entity with the same identifying attributes, making it difficult to enforce protection.

## 3.3 Resource Management

This mostly deals with the capability to balance load/execute processes by picking appropriate processor to execute processes on. The objective could be either to minimize communication costs (requiring knowledge of future behaviour of processes) or load balancing assuming that nothing about the future behaviour of processes are unknown. *Jobs need to be typically run on the local machine if interactive (attributes being fast response time, and short completion times). Detection of an interactive job could be based on statistics measured about the process like whether it is CPU bound or I/O bound (interactive).*

**Migration vs Remote Execution** is a tradeoff that needs to be explored. *It is harder to do migration. History needs to be maintained and analyzed before deciding whether a job is short or not.*

To perform processor allocation suitably, **Load Information** needs to be maintained/propagated. *Due to the constantly changing nature of the system information, information tends to be inaccurate*. Ways such as averaging number of runnable processes, computing residual running time provide fair estimates of the load. Information can be propageted through:

- Frequent broadcasts - This takes up too much bandwidth, a processor may become heavily loaded at once.
- Diffusion (Pairwise Exchange) - Processors pick a machine at random and share load information.
- Polling - Requesting load information from another processor.

Events such as **Distributed deadlocks** cannot be detected readily due to the absence of centralized tables giving the status of all resources.

The problem of **Scheduling** is more complex due to possible dependencies between the processes on different machines which communicate to continue doing useful work. Coscheduling works by having communicating processes being allocated the same time slices (the same row), such that one process will not be blocked waiting on another that does not have a time slice at that instant. This requires:

- *Synchronized time slices. This in turn requires that there is a master timer, which becomes a single point of failure.*
- *It is also important that the latency is small compared to the time-slice for scheduling. Since time slices are in the range 20-500 ms for typical systems, this is usually the case.*

## 3.4 File System

The distributed file system involves the decision of whether to keep the file service stateless or virtual-circuit oriented. Having the server connection oriented causes clients to be susceptible to failure in case of a server failure. However, in the stateless case, though each message needs to contain more information, this kind of a failure is accounted for. *Here, Idempotency of read and write operations is utilized. The fundamental trade-off between stateless vs connection oriented is that of robustness vs performance.*

# 4 Research System Implementations

The most interesting features of the described systems are described below. All systems use a mechanism similar to RPC for communication. They all use their own protocols rather than established protocols like TCP/UDP. Performance was the primary goal of all systems.
**Cambridge**:

- Naming done using centralized name server. Protection by active name table.

**Amoeba**:

- Naming and protection achieved through capabilities. Access rights to an object and the checksum constant are encrypted using random keys from an internal table. Hence, the kernel need not be trusted to establish protection. *Replay attacks using obtained capabilities are possible, but new capabilities/rights cannot be created using the obtained capability.*
- Allows for dynamic allocation of processors from pool.
- Allows servers to charge for services (bank account scheme) and limit resource usage.

**V**:

- Does not address fault tolerance.

**Eden**:

- Protection through capabilities in unencrypted form.
- Provides most reliability among all systems. Complete objects are checkpointed from time to time. *Incremental checkpointing of objects would have been more efficient.*

*The usage of capabilities, etc. represents components of centralized OS research in distributed systems research. The applications for all these distributed systems has been limited to parallelized compilation. A parallelized version of the travelling salesman problem could be implemented using Amoeba.*

*The Cambridge distributed system project was the most practical in that it accomodated the most number of users for the system, while creating a fairly stable system.*

*The communication primitives used made the systems potentially capable of achieving reliability, but the systems themselves did not address reliability to a very large extent.*

# 5 Summary

The paper presents a comprehensive summary of the ideals of a distributed operating system. The ideas of fault tolerance have received little attention in the systems described. The systems also implement only basic versions of required features like naming and protection.

*The first half of the paper was about general objectives/guidelines for distributed systems. Not all systems described in the paper actually addressed the range of issues discussed in the first part.*