CMPSCI 677 Operating Systems **Spring 2013**

Lecture 1: January 23

Lecturer: Prashant Shenoy *Scribe: Demetre Lavigne*

1.1 Course Introduction

The class name is Operating Systems but it will deal mostly with distributed computer systems. This class is taught by professor Prashant Shenoy, in CMPSCI 140, every Monday and Wednesday from 9:05am to 10:20am. Professor Shenoy has office hours on Mondays from 10:20am to 11:30pm in his office (CS336). The recommended but *not* required, textbook for this course is *Distributed Systems*, 2nd edition, by (AST) Tannenbaum and Van Steen (Prentice Hall 2007). For all course information (syllabus, lecture video, notes, etc) visit `http://lass.cs.umass.edu/~shenoy/courses/677/`.

1.1.1 Grading

Student's grades will consist of 4-5 Homeworks (15%), approximately 3 programming assignments (40%), 1 mid-term and a final exam (a total of 40%), and class participation and quizzes (a total of 5%). The final exam will be a take-home exam.

It is expected that students already have a *good* set of programming skills in some high-level programming language.

1.2 Introduction to Distributed Systems

A distributed system can be defined as any computer system where multiple CPUs are connected and working together. This can be extended to include any group of independent computers that appear to users as a single system. This is a fairly broad definition that includes both parallel machines and collections of networked machines.

Many systems that are used on a daily basis around the world are distributed. These include anything on the world wide web such as Google or Amazon.com. Other examples include SETI@Home, grid, cluster computing, etc.

1.2.1 Advantages

There are many benefits that come from the naturally connected state of distributed systems. This connected nature enables *communication* both for users (high-level) and for systems (low-level). With this ease of communication comes the ability to *share resources* such as data, memory, computational resources, etc. One very common example of this kind of sharing is network file systems which allows distributed systems to store and retrieve files.

Another main area of benefits results from having many independent systems working together. Having multiple systems all working with a single purpose allows for greater *reliability* and *scalability* if the distributed

system is designed well. Being well designed means that failures are expected and redundancy is built in. A single failure shouldn't result in a failure of the distributed system as a whole. Failures are more common because the probability of a single machine failing is less than the probability that one of many machines will fail. A good distributed system design also entails being able to use more machines than are currently available for the same application. This allows your system to *grow incrementally* and add capacity as is needed.

The final primary benefit to distributed systems is *economies of scale*. This means that it is generally cheaper to buy a bunch of smaller machines rather than a single equivalent machine. This combines with incremental growth so that instead of replacing a single large machine with a new even larger machine it is more economical to add a bunch of small machines to what you currently have.

1.2.2 Disadvantages

The main drawback to distributed systems is the added *complexity* versus a single centralized machine. First it requires network connectivity by its very nature. The operating systems and applications themselves must also be distribution-aware. *Security* and *privacy* are also concerns because of the ease of communication and resource sharing as mentioned in the advantages.

1.2.3 Transparency

Transparency in distributed systems refers to what is hidden from the user of the system. It is desirable to have transparency in a distributed system because the more transparent the system is the easier it is for end user to use. There are many forms of transparency:

Access
The difference between the actual data and how it is accessed is hidden from the user.

Location
The user doesn't know where a resource is located.

Migration
The user doesn't know when a resource may move or is moved.

Relocation
A resource may move to another location while being used and the user doesn't know.

Replication/Concurrency
A resource may be shared by multiple competitive users and each user isn't aware of the others.

Failure
The user isn't aware of when failure and recovery occurs.

Persistence
A (software) resource may be in memory or on the disk and the user doesn't know.

1.2.4 Open Distributed Systems

Systems which offer services in a known and predefined way are *Open*. The syntax and semantics of open systems are published protocols that can be used via an interface (API). This allows open systems to be

built upon and extended to include new functionality. Open systems promote interoperability and portability because they are intended to be used by a wide variety of users.

1.2.5 Scalability

A goal of most distributed systems is to *scale* well. This means that given a current distributed system and application, the same application will be able to run on a larger distributed system and scale well. A system that scales well should ideally offer a linear improvement in performance as the number of machines added increases. Problems that prevent good scaling include *centralized data*. This means that all of the data that is needed for a problem exists in a centralized source. The problem of *centralized services* is similar. For example, you might have a centralized system for processing but with a distributed database. Another similar problem is that of *centralized algorithms*. This means that a computation requires complete information and so it can only be computed by a centralized system.

To avoid problems with centralized algorithms there is a variety of principles to use. For example, no machine in the system should be expected to have complete state and each machine should be able to make decisions based on local information. As previously stated, it should also be the case that a single failure doesn't cause a total system failure. Finally, using a global clock (such as time stamps) to keep machines synchronized should be avoided because it is problematic. In general, a good distributed system will use asynchronous communication, caching, and replication.

1.3 A Quick History

The first distributed systems used early dialup communication (before the Internet). These systems were the *minicomputer model* where each user had a local machine for processing but remote data could be fetched. This led to the first E-Mail systems. Next the *workstation* model was developed in the 1990's to allow processing to also migrate (not just data). An example of this was the Sprite system which we will discuss later. Following the workstation model is the *client-server* model which is still very much used today. Users have local workstations and they connect to powerful workstations that serve as servers. These servers can be file, print, database, WWW servers (and more).

The *processor pool* model has developed more recently with systems such as Amoeba and Plan 9. In this model there are terminals (Xterms or diskless terminals) and a pool of backend processors that are available to perform processing. The processor pool could be made up of one or many individual servers. This is similar to the *cluster computing* used in data centers. Cluster computing is a local area network (LAN) with lots of servers and storage. Grid computing systems is essentially the same a cluster computing, but the machines are connected over a WAN such as the Internet. An example of Grid computing is SETI@home. This combined with virtualization and distributed data centers have led to the recent rise of cloud computing.

1.3.1 Emerging Models

Currently still developing is the idea of distributed pervasive systems which are made up of very small computing devices with networking capabilities. These devices include: smart phones, TiVO, Windows Media Center, car-based PCs, sensor networks, etc. The main idea is that computing is available everywhere.

1.4 Uniprocessor Operating Systems

This is a quick review of how a single processor OS is structured. The OS manages the system resources such as the CPU, memory, and I/O devices. It determines which programs will access the resources and how via an interface that is simpler than using the raw hardware. There are a variety of ways to structure such an OS:

Monolithic There is one massive kernel that handles everything that the OS needs to do. This was used in early UNIX systems and MS-DOS.

Layered The required functionality is broken up into N layers. Each layer can only use services provided by the layer below it (N-1) to implement new services for the layer above it (N+1).

Microkernel The kernel is kept as small as possible and implement additional functionality as user-level servers. This ensures protection by keeping most of the components of the OS independent and outside the kernel. User-level servers requires the use of interprocess communication rather than simple system calls because the user-level servers are separate programs. The main drawback of this is decreased performance due to additional communication.