The Workstation as Terminal

Andrew P. Black
Digital Equipment Corporation

Models of Operating System

There are a number of different models of distributed operating system. One point of reference is "clustered" systems, where a distributed OS Kernel runs on a collection of (usually homogeneous) machines, with distributed algorithms implementing at least some OS functions. Examples are LOCUS, VAXcluster systems, the Eden System and the Clouds system. At the other extreme, from the point of view distribution, is a networked group of independent machines, each running its own operating system and capable of standing alone. The network is used explicitly by specially written distributed applications, such as mail delivery and remote conferencing or bulletin board service. However, any distributed application must itself be aware of the network.

These two models can be contrasted according to the degree of autonomy that each machine exhibits. In the clustered arrangement, each machine depends for its correct operation on the operation of all the others, and the machines mutually depend on the communications medium. They conspire to present the fiction that the several machines really represent a single large time-shared computer. Moreover, the machines use a significant proportion of their resources on maintaining this fiction. The independent machines model illustrates complete autonomy: the unavailability of a "foreign" host may impact a particular distributed application that uses it, but not the operation of the system as a whole.

Neither of these models is appropriate for the extended, geographically distributed community of users that now exists in many companies and research environments. On the one hand, a single world-wide cluster is unmanageable from an administrative point of view, even if the implementation can be scaled. Moreover, researchers want autonomy so that they can run experimental software on their machines – they do not want to be constrained to conform to the cluster protocols. They want to be able to turn their machines off, and they may sometimes want to take a machine on an aeroplane or a train. At the same time, they do not want to spend their time managing those aspects of their systems that are standard. In addition, heterogeneity is a fact of life; users with different requirements and budgets will choose systems from different vendors with different architectures and interfaces, and it is unrealistic to expect to bring all such systems into a single cluster. On the other hand, a group of independent machines poses an unacceptable system management burden: all those machines must be administered. Engineers, graphic artists and secretaries should not have to be systems managers. Moreover, writing distributed applications in an environment where the network is visible only as a transport service is just too much work.

I wish to discuss a third point of view: the workstation as terminal. This does not fall within the spectrum defined by the above extremes. Instead, the workstation runs a minimal operating system and obtains most services from "The Network". Of course, the services really come from servers on the network, but this is a detail that most users will choose to ignore. I think that this change in viewpoint is significant: to the user the network is a pool of services, not a communications medium. The servers themselves may be clustered or autonomous – but that is not the topic of this workshop.

Author's Address: Dr A.P. Black, Digital Equipment Corporation, 550 King Street, LKG1-2/A19 Littleton, MA 01460-1289.
Electronic Address: Black@hobart.dec.com

The Workstation as Terminal

The workstation is like a terminal in the sense that its main purpose is provide the interface between the user sitting in front of it and the computing resources sitting behind it. The display protocol might be Postscript or X-windows rather than a character-cell protocol, but the functionality is in any case broadly similar. The computing resources may be distributed over the network rather than concentrated in a single mainframe, but this will usually not concern the user. Both terminals and workstations use remote services to support themselves – down-line loading of fonts is an obvious example. Both terminals and workstations use remote services to support their user: in the case of a terminal, almost every service the user sees beyond simple editing is remote, but in the case of a workstation, quite significant services can be provided locally. "Large grain" services, like filing, naming, and mail are commonly regarded as best obtained from the network. How many other services can be removed from the workstation and provided by the network? Possible candidates are authentication, loading, dumping, error-logging or perhaps swapping or paging.

The main difference between a terminal and a workstation is that the manager of a workstation has, at least potentially, much greater freedom in deciding which services should be local and which remote. If the interfaces to the various network services are both well defined and location independent, it is possible to (1) reimplement any component to suit a special purpose, and (2) to relocate any desired service to the workstation. Two reasons for doing this are customization (of the computing environment) and portability (of the workstation – the ability to operate it on a train).

Location Independence

Conventionally, an operating system's services are accessed by hardware traps. Essentially, the function of a trap is to transfer control from an executing thread in the user's address space to the system's address space. In distributed systems, remote procedure call (RPC) provides another mechanism for a thread to cross from one address space to another: although we conventionally imagine the address spaces to be physically remote, they need not be. Remote procedure calls can be used to access system services in a uniform manner, whether they are resident on the same machine as the caller or are on a remote server machine. There are two caveats: performance, and security.

Accessing a remote service will normally incur a larger communications overhead than accessing a local service. This overhead can be eliminated by suitably slowing down the *local* (i.e., on machine) RPC transport, but this can hardly be recommended. The communications overhead may sometimes be offset by the availability of more and/or faster computing resources on the network, but it is clear nevertheless that locating a service correctly can have a major impact on performance.

It is also true that the naive remoting of a service can create a security hole. A trivial example is a routine that takes a password and principal name and checks that the password is indeed that of the principal. This may be an appropriate interface for a local kernel, but it is not adequate for a network service without adding significantly more machinery. For example, one needs to ensure that the server cannot be spoofed, and that the password cannot be read in the clear from the network.

Security becomes more important in a networked environment because of the greater potential for damage, and because users' expectations are raised. Most applications should be ubiquitous, i.e. remote access should be available despite administrative, managerial and cluster boundaries. That is not to say that any user of the internetwork must be able to access the police force's database, or someone else's tax return. What it means is that a properly authorized principal should be able to do these things, regardless of her geographic or administrative location in the network. This implies the availability of access controls that are easy to use, trusted and deserving of that trust.

Remote Access and Distributed Implementation

Despite this requirement for ubiquitous remote access, when *implementing* a distributed application it may be reasonable to require that all of the machines that form part of the implementation are closely integrated, homogeneous, part of the same security domain, and so on. This introduces an important (although not sharp) distinction, and one that is often neglected. A **distributed application** is one whose

implementation is distributed. This should be compared to a centralized application, which is implemented on a single machine. Either kind of application may provide **remote access**.

The most primitive form of remote access is explicit remote login and file transfer, typified by telnet and ftp. Because a conventional operating system allows both applications and system services to be accessed either from a terminal or as a file, telnet and ftp can be used to access almost anything – but at a considerable cost in convenience. Running an application in a remote X-window is the workstation equivalent of remote login: it is more convenient than telnet, but often unresponsive.

A more sophisticated form of remote access can be provided at the system call level. In the Newcastle Connection and in the VMS operating system's Record Management Services System (RMS), the same call can be used to open a file whether it is local or remote. This often means that an application can be run unchanged on remote files. However, in both of these systems the name of the file betrays its location; if the file is moved, its name must change. The most complete and transparent form of remote access occurs when naming transparency is also provided, for example by indirecting through a name server. In this case, inspection of a file name says nothing about its location: a file name can be changed without moving the corresponding file, and a file can be moved without changing its name.

A true distributed system is characterized by the lack of centralized state. Instead it exhibits distributed state, usually with imperfect knowledge. In the case of a local area network, the usual reason for designing such a system is to obtain fast response and loose consistency. (If tight consistency and and slower response are required, it is usually easier to use centralized information and remote access – the extreme case of non-distribution.) Other reasons for distributing an application are to increase availability, for scalability, and to mirror the structure of an organization.

Summary

I have advocated a view of the workstation as a means of accessing network resources rather than as a computing engine. Its operating system should be modular, and OS services should be accessed by remote procedure call, in order that individual system services can be relocated at will. For this to be feasible, reliable authentication and authorization mechanism must be provided.

Some system services will be implemented in a distributed fashion in the network, while others will be centralized. These implementation decisions will be hidden from the user, since all will provide a similar interface for remote access.

The views expressed herein are those of the author, and should not be construed as those of Digital Equipment Corporation.

VAXcluster, VMS and VAX are trademarks of Digital Equipment Corporation. LOCUS is a trademark of Locus Computing Corporation.