"Lessons learned at Project Athena"

Notes for talk at Amsterdam SIGOPS Workshop, Sept. 8-10, 1986
J. H. Saltzer
Version of September 2, 1986

Background:

Project Athena is finishing the third year of a five-year cooperative project of M.I.T., IBM, and Digital Equipment Corporation, to explore the potential impact on engineering education of using networked workstations with good graphics. Its scale is large: 1500 workstations plus 50 network servers for 4000 undergraduates. The project is structured in two major phases, with the first, now installed, phase being an approximation of the future environment. The approximation consists of 50 Digital VAX 11/750 time-sharing systems and 150 IBM PC/AT's linked with local area networks. allowing development and initial use while waiting for MicroVAX II and RT PC class machines and their successors to become available. The project is now entering its second phase, a client-server model based on so-called 3M (1 Mips, 1 Megapel, and 1--actually 2 or 3--Megabyte) workstations. At this time, about 150 3M workstations are deployed, and the project is installing 3/day. This is therefore an interim report.

Although not intended as a distributed systems research project, the design, implementation, and deployment of the Athena workstation-server model is bringing into sharp focus some sometimes-fuzzy concepts current in the distributed systems research community, and at an operating scale that forces a very candid appraisal of both the real value and the state of readiness of each proposed idea. As one might expect, some mundane but real problems of large-scale distributed systems are showing up.

Some of these things will strike certain listeners as "obvious," because those listeners have been (correctly) predicting them. The interest in repeating them here is the confirmation from the field that the lessons are for real.

I have three observations about networks, and five about systems.

I. Network lessons.

I.1. Network firewalls are important. Products such as the DEC LAN Bridge (which allow several Ethernets to be linked together and act as one) can be a mistake because they allow a machine whose software is running out of control to bring down a larger region of the network. Things that link LAN's must not forward trouble.

I.2. Broadcast protocols are like the plague; avoid them because errors lead to floods of (response) packets when you least expect them. (Examples: the broadcast packet whose content appears to a low level in the recipient's software to require forwarding or a rerouting suggestion; the damaged broadcast packet that triggers a flood of maintenance responses or resend requests; etc.) [In connection with I.1, note that one way to avoid forwarding trouble is to never forward broadcasts.]

I.3. Ethernet hardware quality is not uniformly high. We trip over bugs and misfeatures in most hardware. (Examples: interfaces that wedge under massive collisions, repeaters that lock up on physical disconnect and require pressing a reset button, multiple buffer interfaces that lose track of packets then deliver them later; interfaces that interfere with other bus devices; transceivers that can talk to one another but not to those of other vendors; transceivers that demand much more electric power than the standard specifies.)

II. System design lessons

II.1. The biggest system design problem: human engineering. It is time to place much more emphasis on the human interface to the operating system, under the assumption that the only person who will ever interact with it is a non-expert. Care and feeding of a one-user operating system isn't polished yet. MSDOS and the Macintosh come closest, but other OS designers haven't yet picked up the ball. For example, UNIX with a MAC-like user interface would still require a wizard to recover from power failures that damage the file system. From the point of view of a first-year college student trying to uncrate it, the personal computer together with its operating system must be a low-technology item. A 3M workstation linked in a distributed system to an array of interdependent services does not currently have this essential property.

II.2. Update isn't so hard, even when there are lots of copies around. Update for most software is easier than usually assumed, because changes happen slowly, and don't have to be propagated instantly or atomically. The important thing is to include in the design some kind of mechanics that make update inevitable. Then relax and wait.

II.3. System designers have very different requirements from students. So they don't design the right systems. (Example: we rarely ask students to work together, but many education applications can benefit from 2-D, 3-D, and image graphics. System designers usually implement by working in teams, but hardly ever use graphics above the level of multiple windows for text. So the system designers spend all their time developing tools for working together, and don't notice the rough edges in the display packages.)

II.4. Easy licensability of a software package is more important than better features or higher quality. We have had to discontinue consideration of several good third-party packages because of unacceptable licensing conditions. (E.g., Physical copy protection; legal copyright liability; non-disclosure clauses; inability to license to more than one CPU type; per-copy identification requirements; inability to cope with network distribution.)

II.5. Coherence works. We have had excellent results by defining a programming interface that consists of the C, Fortran, and LISP languages plus the 4.2 UNIX (a trademark of ATT Bell Labs) operating system calls plus the X window package. Applications written for the VAX usually run on the RT just by recompiling. Modest size systems often move just by copying the sources and typing "make". Bigger ones have minor problems the first time; modest tinkering usually leaves them workable in both environments. (Example: gnuemacs.) The same programs then run on the Sun, not an intentional target. [But VAX/VMS programs are a major project to move to VAX/UNIX, RT/AIX programs require substantial work to get into the RT/UNIX environment, and 68000/MAC programs are hopeless despite using the same engine as the SUN. (Because of the dramatically different display interface.) Lesson: what matters are standard OS and display interfaces, not a standard instruction set.]

II.6. The most useful client-server example we have so far encountered is a division of a windowing display manager into a device driver service and an application client library separated by a network connection. This architecture allows one to "pop up a window across the net," a surprisingly useful ability, not just for remote login to old-fashioned systems, but for cooperation among workstation users.

II.7. Naming? Despite all the discussion about naming in the operating systems literature over the last few years, it still isn't apparent just how much (or how little) function is really appropriate to glue a particular community of users and services together. One reason seems to be that the boundary between naming and service management isn't clear. The naming system we have specified seems to be nothing more than a specialized inquiry interface on the side of our central service management system, which is the real holder of fundamental knowledge about configuration information.

II.8. Storage models: what files should be on a workstation and what should be on a central file store? The right compromise is very hard to find.

1. There isn't room on a student-affordable disk for the complete library of goodies (we have 150 Megabytes now and it is growing). And placing things on a central file store allows much easier update.

2. If the student owns the workstation, he or she wants to be able to take it home over the summer, and to move to an off-campus apartment, where connection to a central file store is slow or non-existent.

3. CD ROM's are big enough to hold the library but have awful performance.

4. The student's own files need to be available for use not only from a workstation in the dormitory or fraternity, but also at the library or in a laboratory.

5. Most candidates for distributed file systems don't perform well enough to make deployment with 1500 clients economically feasible.

6. The most effective way for a student to participate in the economic decision about how many personal files to retain is for the student to own the storage medium.

These considerations seem to add up to two different configurations:

- (short term) A public workstation, with all local disk storage used as a cache, importing all its software libraries from a central server and importing the personal files of its users either on removable media or via the net from central lockers.

- (longer term) A privately owned workstation, with a range of disk size options; software that allows export of the private file system via the network to public workstations. Libraries may be imported via the net or copied via the net to the workstation, at the student's choice.