1984

The TILDE Project

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

John T. Korb
Purdue University, jtk@cs.purdue.edu

Thomas Murtagh

Walter Tichy

Report Number:
84-500

The TILDE Project

Tilde Report CSD-TR-500

November 23, 1984

Douglas Comer
John T. Korb
Thomas Murtagh
Walter Tichy

Department of Computer Science
Purdue University
West Lafayette, IN 47907

ABSTRACT

The TILDE project investigates general-purpose computing systems that run on a multi-machine computing engine. A multi-machine computing engine is a cluster of heterogeneous processors loosely coupled with a high-speed local area network. The goal of the project is to explore computing systems in which the user interface hides details of the underlying architecture, making the distributed computing engine appear to be a single, large time-sharing system.

This paper presents project plans and status. It describes the architecture of the TILDE computing engine as well as research in the areas of operating systems, user interfaces, and computational services. The novel computational services planned for TILDE include high-level electronic mail, a timed event service, satellite bulletin-board broadcast, supercomputer access, and others.

This project is supported in part by grants from the National Science Foundation (MCS-8219178) and SUN Microsystems Incorporated.

1. Introduction

Recent advances in silicon technology have made it economical to produce a variety of processors ranging from single-chip microcomputers to special purpose, high-speed array processors. Advances in network technology have made it possible to communicate among such processors over high-speed, low-cost local area networks. Furthermore, research in protocol software has resulted in uniform transport-level protocols that work over physically large distances (long-haul networks) as well as over small physical distances (local area networks).

With the processor and network technology available, it is possible to build a computing system that employs a set of heterogeneous processors to carry out computation. The uniform protocol interface can hide machine differences, making communication among the processors easy. Most importantly, processors can be selected that are well-suited for the computation they perform. For example, microcomputers could be used to perform trivial tasks like capturing keystrokes, while large, CPU intensive programs execute on more powerful machines. Such use would be economically wise because microprocessors carry out trivial tasks at less cost per cycle than large mainframe processors.

The TILDE project (Transparent Integrated Local and Distributed Environment) investigates computing systems that run on a network of heterogeneous processors. The goal of the project is to explore general-purpose computing systems in which the user interface hides details of the underlying architecture, making the network appear to be a single, large time-sharing system.

2. The TILDE Architecture

Many projects have considered the economic viability of using multiple, inexpensive processors to perform computation (e.g., COCANET [24], EDEN [17], LOCUS [20], UNIX UNITED [3], the V-Kernel [5,16], and project Athena [10]). In most cases, emphasis has been placed on ways to provide a general purpose computing environment by interconnecting conventional processors. We take a broader view. First, we envision a computing environment in which users access high-level computational services on both local and distant computing systems. For example, a user might access the electronic mail service on a local machine, and the linear equation solver service that only runs on a distant supercomputer. To make services easy to obtain, we provide a homogeneous interface to the local and distant environments. Second, we acknowledge that computing needs grow over time, and provide for incremental expansion of the local computing environment. In our system, machines are loosely coupled, and intelligent terminals identify services dynamically, making it possible to add additional processors without recompiling the operating system. Third, because we permit the local environment to contain heterogeneous processors, processors can be selected that are best suited for the services that they provide.

Figure 1 shows the architecture of a TILDE computing system, which we call a *Multi-machine Computing Engine*. The computing engine consists of a cluster of heterogeneous machines loosely coupled with two high-speed local area networks. User interface machines, called *intelligent terminals*, access the services of the computing engine over one of the networks. One or more machines provide a gateway

This project is supported in part by grants from the National Science Foundation (MCS-8219178) and SUN Microsystems Incorporated.

service, connecting the computing engine to the rest of the world via an internet.

At the center of the computing engine is a high-speed file transfer network. Later we will describe how the file transfer network is used to move single blocks of files (pages) or entire files among processors. Because file transfer places high demand on the central network, that net must behave well under heavy load. We are currently using a 10Mbit per second token passing ring which we plan to upgrade to 80Mbit capacity.

The other local area network in the computing engine connects users' intelligent terminals to the processors that supply services. Because users need to search for services among all machines, the interface network needs broadcast or, preferably, multicast addressing. The prototype uses 10 Mbit per second CSMA/CD technology for the interface network.

Computing engines connect to other computing engines across an internet. Thus, at least one of the machines in a given computing engine serves as a gateway that provides a path between the local system and the rest of the world. We assume that although services may be available over the internet that are not available locally, the time required to access such services may be significantly longer than delays within the computing engine.

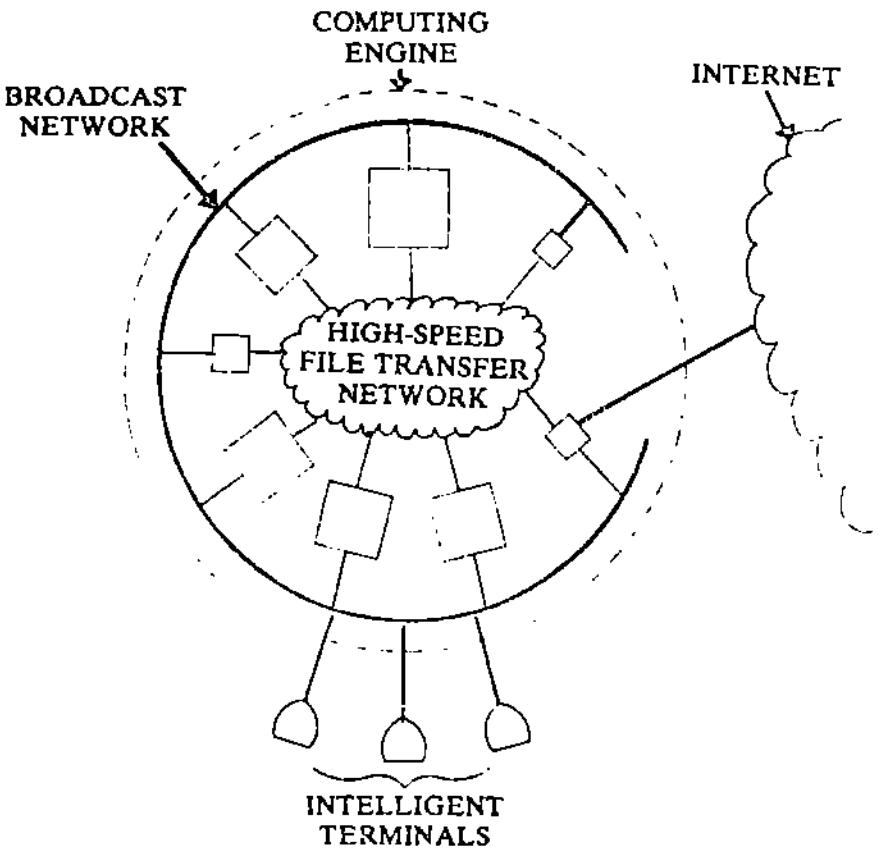

Figure 1. The Architecture of a TILDE Computing Engine.

Multi-machine computing engines differ from conventional architectures in significant ways. They have more power than systems of personal workstations because most processing is performed on powerful back-end processors in the computing engine. They have greater flexibility than conventional time-sharing systems because new services and more computational power can be added to the system by adding individual machines to the engine. Most importantly, it is possible to choose hardware appropriate to the computation. For example, CPU-intensive services can be performed on large mainframe machines, while services like electronic mail can be provided by inexpensive minicomputers.

3. Project Goals and Areas of Research

The goal of the TILDE project is to explore general-purpose computing systems that run on the multi-machine computing engine. We focus on location-transparent interfaces, distributed file systems, and high-level computational services. More specifically, the project explores computing systems in which the user interface hides details of the underlying architecture, making the multi-machine computing engine appear to be a single, large time-sharing system. To achieve this goal, we are investigating the following three primary research areas:

a) **Operating systems.** The two major problems in the operating systems area are the file and directory system and the name binding mechanism. The file and directory system must provide efficient access to local and distant files. A new name binding mechanism simplifies the naming of files in a large name space and eliminates location dependencies.

   A centralized file system would form a serious bottleneck in the computing engine. Any node in the engine may therefore have secondary storage capability. In addition, one or more nodes in the net are file servers. A location transparent, distributed file and directory system spans all machines in the computing engine. Location transparency means that it is impossible to derive the location of a file from its name. Since the placement of files is the responsibility of the operating system, data can be migrated to the point of most frequent access.

   The new name binding mechanism allows more flexibility in using incomplete file names. Rather than always interpreting incomplete names relative to a current directory, it permits each user to specify how incomplete names should be resolved relative to several entry points in the directory. In addition, the naming mechanism provides location transparency for access to files outside the computing engine.

   Ultimately, we would like to design a new operating systems kernel that supports efficient communication between the intelligent terminal and services, as well as an efficient and reliable distributed file system that allows files to be shared among all services. However, a complete operating system rewrite would be both unrealistic and premature. Instead of designing from scratch, we are modifying UNIX† [22,23]. We adopted UNIX because we have considerable expertise and experience with it, it is easy to modify, and it presents a programming environment that faculty and research staff find attractive.

---

†UNIX is a trademark of Bell Laboratories.

To adapt Unix for the computing engine, we have implemented a package for accessing remote files and have begun experimenting with a new directory system. We plan to retain the process manager, the device manager, and the communication systems. Like UNIX, the new TILDE system does not attempt to migrate processes across machine boundaries. Instead, it provides a novel name mapping and file access scheme that permits a user to access data independent of the machine on which computation is performed.

b) **User interfaces.** The intelligent terminal provides a homogeneous user interface for the TILDE computational services. It is a crucial part of the project. We have identified the desirable characteristics of an intelligent terminal, and chosen a model for such a terminal. Implementation of a prototype intelligent terminal has begun using a SUN workstation.

   We plan to complete the prototype and experiment with it, refining both the user interface and the underlying mechanisms. Of particular interest is the network protocol software used to communicate between the intelligent terminal and service processors. Although the prototype uses a conventional internet protocol (TCP/IP), we eventually expect to build a more efficient protocol for communication with the service processors.

c) **Computational services.** We have identified several specific high-level computational services that TILDE will provide: a high-level electronic mail service, a timed event service, a satellite-based bulletin-board broadcast service. supercomputer access via satellite links, a version control service, a parallel computer access service, and a distributed data base. As the system becomes available for departmental use, we expect faculty to add more services to the environment based on their research interests.

The next three sections describe each of these three areas of research, giving a detailed description of the research and project status in each area.

4. Operating Systems Research (detailed description).

Naming. Computing environments provide access to objects such as processes, users, devices, files, and services. In distributed environments, the question of naming is central to system design because names can only be exchanged freely among mechanisms that dereference them the same way. For example, if each machine in the computing engine maintains its own file name space, a given file name may not refer to the same file when interpreted on two different machines.

A naming mechanism is *transparent* if all names are known globally (i.e., if a given name refers to exactly the same object independent of the context in which it is interpreted). Name transparency is desirable because it allows users to exchange objects like programs that reference data by name. Ultimately, name transparency can only be achieved if names are unique. Such uniqueness requires either a central authority to designate all names, or an agreement by which individual "sites" (e.g., a computing engine) can assign names. Consulting a central authority is too inefficient, so the question becomes how to assign transparent names in a distributed environment.

The major complication with distributed name assignment is that most schemes introduce location dependencies. For example, some distributed systems prepend machine names onto file names, making names take the form *machinefile* [18]. We reject this approach because it implies that the name used to reference a file must

change whenever the file moves from one machine to another.

Another naming consideration concerns name length. Users prefer to *shorten* names, using abbreviations in place of longer names whenever the correct name can be deduced from the context in which the name is resolved. For example, file names in UNIX are specified either as full path names or shortened names (which are interpreted with respect to the "current" directory). Allowing users to substitute short names for longer ones makes naming convenient, but increases the probability that names conflict.

TILDE solves the problem of naming in a distributed environment in a novel way: it replaces a per-machine name binding mechanism with a per-user name binding mechanism. Thus, names are relative to the individual, not to the machine on which the computation is performed. In a local computing engine, users can refer to a file by name without knowing the file's physical location or the location of the computational service they invoke. Furthermore, a set of default name bindings allows users on a single computing engine to refer to each other's files in much the same way as they would on a conventional single-processor system. More importantly, TILDE makes local and distant file names transparent, allowing transparent access to files on distant computing engines.

In general, TILDE provides transparency only for those objects which users routinely manipulate. Primarily, this means that in a given computing engine, TILDE interprets file names and user login names consistently. We expect, however, that other identifiers may not be known globally. For example, process identifiers may remain local to the machine that executes the process. Keeping some identifiers local helps make the system more efficient, but it introduces nontransparent names. We seek a compromise that adopts nontransparent names for efficiency, but hides them from the user. One of the important issues is whether such partial transparency provides sufficient flexibility to enable us to make names used by higher levels of the software transparent.

File System Design. Having transparent file names is especially important because users reference data and programs by naming the files in which they reside. The file system can be partitioned into two parts, one that deals with the binding of file names to files, and another that provides access to a file. The first part, usually called the *directory system*, implements the naming scheme and the binding of names to objects. The second part, the underlying *file system*, provides access to a file object once its location is known.

Because users interact with more than one machine in the computing engine at a given time, the file system must provide efficient access to all files. On one hand, centralized file systems are less efficient than distributed systems because the central file server forms a bottleneck in the system. On the other hand, consistency and synchronization are difficult in distributed systems. We are interested in a compromise between centralized file systems and completely distributed ones. Our compromise considers file systems which keep data "close" to the most frequent point of access, and migrate files slowly as the focus of access changes.

We have completed the implementation of a file system that allows block-level access to all files in a network. The package provides access transparency in that all files are accessed in the same way, regardless of whether they are remote or local. The package is implemented with the Berkeley Unix IPC mechanism, and is based on the transport-level protocol TCP/IP [21]. It can thus be used in both local and long-

haul networks. Performance over a 10 Mbit local area network is surprisingly good. For instance, listing a remote directory is instantaneous, without any noticeable delay compared to listing a local directory. The file system also implements a remote current working directory and remote symbolic links. Remote symbolic links can be used to compose a directory spanning several nodes, providing a degree of location transparency. Remote execution is not currently supported. More information about the file system can be found in [27].

We are now building a location transparent directory system. Location transparency means that, within a given computing engine, the physical location of a file cannot be determined from its name. A preliminary design of the directory system appears in [27]. It employs a primary-copy strategy and is carefully tuned to avoid overhead for operations on local files. For efficient read access, the directory system implements demand replication. Demand replication automatically replicates files and directories that are read by several hosts. The update protocol is such that frequently written files experience a low level of replication. File migration, finally, improves write access by moving files to sites where they are written. We plan to investigate several migration and replication heuristics.

Protocols. Purdue has a history of protocol research. Currently, we are involved in work on protocols for local area networks, long-haul networks, and satellite-based broadcast networks. All three will be important in TILDE. Work on satellite protocols will contribute directly to the satellite broadcast and supercomputer access services described below. Work on local area network protocols will be important in the local computing engine. Work on long-haul network protocols will contribute to our understanding of gateways.

In the area of satellite protocols, two schemes currently exist. Random channel access protocols allow multiplexing of traffic on demand to keep the utilization high; the disadvantage of such schemes is that more overhead is introduced to identify traffic. By contrast, reservation schemes divide the bandwidth into reserved slots; the disadvantage is that unused reservations result in underutilization of the bandwidth. We plan to concentrate on hybrid protocols that combine random access with reservation strategies to achieve the best of both schemes.

Protocol designers choose a tradeoff between efficiency and generality. For example, Popek [20] insists that efficient, special-purpose protocols may be needed to obtain satisfactory performance in a distributed system. However, DARPA has recently developed a general-purpose transport-level protocol, TCP/IP, that provides for internetwork connections. Although we have adopted TCP/IP for the TILDE prototype, we plan to conduct experiments measuring its cost, and to consider possible alternatives. Of special interest here is the protocol used to ship files among back-end machines. Because the file transfer network has large capacity, we expect that network protocols may limit performance. Thus, both simulation and experimental measurements will be important in helping us assess the protocol overhead and tune the system.

5. User Interface Research (detailed description).

Each individual machine in a computing engine provides a set of *services* such as electronic mail, general-purpose command interpretation, hardcopy printing, or high-speed vector processing. Heavily used services may be duplicated on several machines to improve performance. Less heavily used services, or services that require

special-purpose hardware may be available on only one machine in each computing engine, or may only be available on remote computing engines.

Conceptually, a user has access to all possible services simultaneously, whether they are supplied by the local computing engine or a remote one. The user interface that supports such access consists of an *intelligent terminal* that communicates with the computing engine over the interface network. More powerful than conventional terminals, but less powerful than independent "workstations", intelligent terminals provide three important mechanisms [14].

1. They request service from individual machines of the computing engine on behalf of the user. The intelligent terminal is responsible for locating special-purpose or lightly-loaded processors, as well as handling the details of authorization (by logging in for the user) and resource requesting.
2. They provide limited local computing power for special-purpose interactive tasks (e.g., front-ends of editors [11]). By handling all interactive processing such as keyboard and mouse input at the intelligent terminal, the host processors that make up the computing engine are relieved of the context switching burden required to implement highly interactive programs. This layering of resource utilization not only reduces the load on the computing engine, but also improves response to the user.
3. They allow users to customize interactive interfaces to suit their own personal needs or tastes [4]. Much intelligent terminal work has been done to define *Virtual Terminal Protocols* [7,15] that allow programs to access the intelligent terminal in a machine-independent way, and into building special-purpose systems for a single application area [8,19]. These approaches force the host program to operate at a low-level, worrying about the details of window management and user interface styles. Our approach is to maintain a high-level, style-independent interface between host programs and the intelligent terminal, and use the intelligent terminal to implement a particular interaction style. Thus, we preserve the standard UNIX programming paradigm of easily-written filters and pipes, and allow the intelligent terminal to provide a screen-oriented interface to these line-oriented programs. Decisions about key bindings, window organization, menus, error handling, and so on can be handled uniformly by the intelligent terminal (and can be studied as a separate area of research). These decisions can also be customized to the user's preference, without modifying the programs running on the computing engine.

6. High-Level Services Research (detailed description).

A major part of TILDE concentrates on designing high-level services that the computing engine provides. Work has started on a high-level mail service, a timed event service, satellite bulletin-board broadcast, and a version control service. A supercomputer access service, a simulation service, parallel computer access, and a distributed data base are in the planning stages. Short descriptions of these services follow.

Electronic Mail. Electronic mail is usually based on independent memos. However, subscribers frequently use mail systems to transport files and to conduct conferences among groups of individuals. Research on mail explores a higher-level mail environment based on conversations rather than memos. The notion of

"conversation" encompasses traditional memo systems, bulletin-boards, and conferencing systems. Our conversation-based mail allows users to "attach" files to a message in such a way that the recipient can save the attached files without reading them. It also keeps the history of an exchange, sorts incoming mail so that all messages pertaining to a given conversation are kept together, and stamps an expiration date on messages so messages like "system going down at 10AM" will disappear automatically at the appropriate time. More details of the conversation-based mail service are given in [6].

Timed Event Service. TILDE will provide a service that handles scheduled events. Examples of such events include: asking for a reminder message at a given time, scheduling computations to be performed at a given time (or on a periodic basis), keeping an appointment calendar, and providing a time-stamp for distributed algorithms. We plan to construct a timed-event service that collects together all mechanisms that deal with time, and provides a single, uniform interface for accessing them. The event service must be reliable (in the sense that system crashes will not cause loss of events), and efficient (in the sense that scheduling an event is fast). Low-overhead protocols will achieve efficiency, and a special processor will provide high reliability.

Satellite Bulletin-Board Broadcast. Satellites offer a convenient way to distribute large volumes of broadcast data over wide geographic areas. They can deliver data with less cost than systems that route bulletin board information from host to host (e.g., USENET). We plan to expand and apply on-going research in the area of satellite-based protocols by creating a bulletin-board broadcast service. The premise of the satellite delivery system is that while every site wishes to receive the entire bulletin board, a given site contributes only a small amount of material. Thus, slower, land-based connections are sufficient for traffic that flows to the bulletin-board broadcast site; satellites are used only to deliver the bulk data. The economic justification is obvious: while a satellite transmission station costs approximately $250K, a receiving station can be established for under $5K. The project has two parts: We plan to continue our simulation studies of satellite based protocols for one year, and undertake an experimental study of new protocols using a subchannel on an existing satellite in successive years.

Supercomputer Access Service. With the satellite mechanism in place, we will turn attention to a related question -- that of how to deliver output from a supercomputer to remote users. Purdue has a Cyber 205 supercomputer that scientists use in their work. Our model of user interaction with the supercomputer is based on the following observation: Many scientific users follow a simple cycle of editing a program (or data file) by hand, running the program on the supercomputer, and analyzing the large volume of output produced. We will therefore focus on software that follows what we call the *shadow paradigm.* The shadow paradigm uses local computing power to capture incremental changes to programs and data, and to communicate these changes to a "shadow" file located on the remote supercomputer. For example, a *shadow editor* consists of two parts. When a scientist edits a file, the local part captures keystrokes makes the specified modifications, displays the new data instantly, and sends a record of the changes to the shadow on the remote supercomputer. The shadow part applies the changes to a remote copy of the file. Since keystrokes result in low-volume traffic, they can easily be handled by land-based networks. When the scientist executes a program on the supercomputer, the system verifies that the remote file is the same as the local one, and then uses the remote copy. Because

checksum techniques can verify a file without sending its contents across the network, the scientist will perceive little or no delay before execution begins. Because data comes back over high-speed satellite connections, the scientist will receive it almost immediately. Finally, because the shadow editor updates the display based on local information, the scientist does not observe network delays while editing. The effect is almost instantaneous response even though heavily CPU oriented processing is performed on a remote supercomputer.

Version Control Service. A major problem in programming environments is how to keep a constantly changing software system well organized. Constant change is unavoidable because any large system requires correction, adaptation, and extension throughout its entire life. Constant change creates multiple versions of all system components. Old versions cannot normally be discarded, because they may be in use, they may be required for backup, and they are needed for understanding the development history.

The TILDE version control service is intended to manage the multitude of versions in large systems. We have already developed an important building block for this service, the Revision Control System (RCS) [26,28]. RCS manages revisions of individual components by organizing them into groups and tracking their development history. RCS does not deal adequately with configurations and derived versions (i.e., compiled or linked programs). A combination with MAKE has proven to be unsatisfactory, since MAKE's concept of multiple versions is primitive. We therefore plan to extend RCS into a full version control system. New results concerning the minimization of recompilations after changes will be incorporated [29]. The version control service helps eliminate configuration errors during development and maintenance, as well as speed up the regeneration of configurations after changes.

Simulation service. Faculty working in the area of simulation and modeling are planning to expand existing simulation software, and incorporate it into a general purpose simulation facility under TILDE. Although software exists to solve various queueing problems, there is no uniform user interface, and the selection of appropriate routines is a manual process. We plan to unify the user interface and automate the choice of simulation software appropriate for a given problem. Such a facility will be important to our work with protocols because the early research on broadcast protocols (one-many communication) will use simulation models to help develop and assess satellite protocols.

Parallel computer access. As part of the parallel and vector computation project, faculty members in the department have built a prototype configurable parallel processor called the PR II [9,13]. Currently accessible only through a special-purpose microcomputer, the PR II has 100 processors with programmable switches connecting them. We plan to attach the PR II support processor to the TILDE prototype system, and provide a parallel machine access service. The service will consist of two parts: a program preparation facility (that currently runs on a VAX using a bit-mapped display), and a program execution facility that will download the PR II and control execution. We plan to work primarily on the program execution facility, making it easy to transfer data between the PR II and the TILDE file system.

Distributed Database. Faculty in the database area plan to develop a distributed database system for the TILDE architecture. The system will provide (1) concurrency control and failure/recovery control transparency, and (2) an intelligent query interface. Although the database is physically distributed, the user can assume

that it is logically integrated. The system hides the details of concurrency, allowing a user to assume that only one transaction is present in the system. The system enforces failure recovery in the sense that transactions are atomic (each transaction is either processed completely or not processed at all).

Although we plan to provide a working database service, the service is not an end in itself. We expect to use the opportunity to study a variety of fault tolerant and concurrency control mechanisms, including locking, optimistic, and version-based schemes. In particular, we will measure system overhead, recovery costs, and lost processing costs [2]. To make such experiments possible, we will implement the system with generic primitives, using table-driven processing at the detailed level.

Queries will be expressed in a high level language such as QUEL of INGRES [12,25]. The query processor will attempt to eliminate ambiguities, and provide an optimal access path through the physical database before executing the user's request. In case a request cannot be processed, the query processor will provide an explanation of the problem based on the contents of the database and problems encountered during access. If a request can be satisfied, the system will cache in the working buffer related information that is available at a low cost. For example, because it is possible to transmit an entire block of data across a high-speed network in roughly the same time as it takes to transmit a partial block (assuming that the network is not fully utilized), the system can optimize response time for sequential access by caching entire blocks whenever partial blocks are transmitted [1].

7. The TILDE Prototype

Our immediate project objective is to produce an experimental prototype TILDE system. The prototype will serve as the basis for further experiments with system design, and as the foundation on which new high-level services are built. The model we have in mind is a UNIX-like system with the syntactic parts of the interactive shell (command interpreter) resident in the intelligent terminal, and the command set augmented with new commands that implement high-level services. We envision a prototype system that will:

- provide roughly the same mix of general-purpose computing as a single-machine running the UNIX timesharing system.
- be easily expandable. In particular, it will be possible to increase the power of the system by adding more individual machines to the computing engine; it will be possible to increase the variety of services provided by introducing new user-level software without rebinding the kernel.
- have roughly the same performance characteristics as a single-machine UNIX system (i.e., the user will perceive no gross difference in performance).
- provide each user access to all services.
- provide new, high-level services.
- integrate communication/access among local and remote computing engines.

We plan to design and build a multi-machine computing engine prototype during the coming year, and refine it during the second year. During these two years, the necessary operating system primitives and the intelligent terminal concept will be developed, and a few high-level services will be implemented. The following years will see a refinement of the prototype, additional services, and an expansion of the

computing engine with more processors and intelligent terminals, to provide a general departmental facility.

References

1. B. Bhargava, Design of intelligent query systems for large databases, in *Pictorial Information Sytems*, Springer Verlag, May 1980, 431-445.
2. B. Bhargava, Performance evaluation of reliability control algorithms for distributed database systems, *Journal of Systems and Software Vol. 3*(July 1984), 239-264.
3. D. R. Brownbridge, L. F. Marshall and B. Randell, The Newcastle Connection or UNIXes of the world unite!, *Software—Practice & Experience 12*(1982), 1147-1162.
4. T. T. Carey, A workstation for interaction styles, *IEEE COMPSAC Proc.*, 1982, 303-307.
5. D. R. Cheriton and W. Zwaenepoel, The distributed V kernel and its performance for diskless workstations, in *Proc. of the 9th Symp. on Operating System Prin.*, June 1983, 129-140.
6. D. E. Comer and L. L. Peterson, DRAGONMAIL: A prototype conversation-based mail system, in *Proc. of the Summer USENIX Conf.*, June 1984, 42-51.
7. J. D. Day, Terminal protocols, *IEEE Transactions on Communications COM-28*,4 (Apr. 1980), 585-593.
8. J. D. Foley, A tutorial on satellite graphics systems, *Computer 9*,8 (Aug. 1976), 14-21.
9. D. B. Gannon, Pringle: An experimental 64 processor parallel computer, in *Proc. International Conf. on Parallel Processing*, 1984, 1-3.
10. J. Gettys, Project Athena, in *Proc. of the Summer USENIX Conf.*, June 1984, 72-77.
11. R. N. Goldberg, Software design issues in the architecture and implementation of distributed text editors, Tech. Rep. DCS-TR-110, Rutgers Univ., Jan. 1982.
12. G. D. Held, M. R. Stonebraker and E. Wong, INGRES – A relational data base system, in *Proc. of the AFIPS National Computer Conf.*, AFIPS, 1975, 409-416.
13. A. Kapauan, J. T. Field, D. B. Gannon and L. Snyder, The Pringle parallel computer, in *Proc. 11th Annual International Symp. on Computer Architecture*, 1984, 12-20.
14. J. T. Korb, An overview of the DASH intelligent terminal project, Tech. Rep. CSD-TR-492, Purdue Univ., Sep. 1984.
15. K. A. Lantz and R. F. Rashid, Virtual terminal management in a multiple process environment, *Proc. of the 7th Symp. on Operating System Prin.*, Dec. 1979, 86-97.
16. K. A. Lantz, W. I. Nowicki and M. M. Theimer, Factors affecting the performance of distributed applications, *Computer Communication Review 14*,2 (June 1984), 117-123.

17. E. D. Lazowska, H. M. Levy, G. T. Almes, M. J. Fischer, R. J. Fowler and S. C. Vestal, The architecture of the Eden system, in *Proc. of the 8th Symp. on Operating System Prin.*, Dec. 1981, 148-159.

18. S. J. Leffler, W. N. Joy and M. K. McKusick, *UNIX programmer's manual for 4.2BSD*, The Univ. of California at Berkeley, Aug. 1983.

19. J. Michel and A. van Dam, Experience with distributed processing on a host/satellite graphics system, *Computer Graphics 10*,2 (July 1976), 190-195.

20. G. Popek, B. Walker, R. English, C. Kline and G. Thiel, The LOCUS distributed operating system, in *Proc. of the 9th Symp. on Operating System Prin.*, Oct. 1983, 49-70.

21. J. Postel, Internet program protocol specifications, RFCs 790, 791, 792, 793, 794, 795, 796, USC Information Sciences Institute, Marina del Ray, 1981.

22. D. M. Ritchie and K. Thompson, The UNIX time-sharing system, *Comm. ACM 17*,7 (July 1974), 365-375.

23. D. M. Ritchie, The UNIX time-sharing system: A retrospective, *Bell System Tech. J. 57*,6 (1978), 1947-1969.

24. L. A. Rowe and K. P. Birman, A local network based on the UNIX operating system, *IEEE Transactions on Software Engineering SE-8*,2 (Mar. 1982).

25. M. Stonebraker and D. R. Ries, Effects of locking granularity in a database management system, *Transactions on Database Systems* 2,3 (Sep. 1977), 233-246.

26. W. F. Tichy, The string-to-string correction problem with block moves, *ACM Transactions on Computer Systems* 2,4 (Nov. 1984).

27. W. F. Tichy and Z. Ruan, Towards a distributed file system, *Proc. of the Summer USENIX Conf.*, June 1984, 87-97.

28. W. F. Tichy, RCS – A System for Version Control, *Software – Practice and Experience 15*(1985). (to appear).

29. W. F. Tichy and M. C. Baker, Smart Recompilation, in *12th Annual Symposium on Principles of Programming Languages*, New Orleans, LA, Jan. 1985.