

AD-A186 557

THE STARLITE PROJECT(U) VIRGINIA UNIV CHARLOTTESVILLE
DEPT OF COMPUTER SCIENCE R P COOK ET AL OCT 87
UVA/525410/CS88/101 N00044-86-K-0245

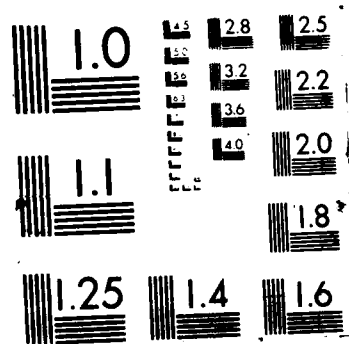
1/1

UNCLASSIFIED

F/G 12/5

NL





12

A Semi-Annual Progress Report
Contract No. N00014-86-K-0245
March 1, 1987 - September 1, 1987

DTIC FILE COPY

THE STARLITE PROJECT

Applied Math and Computer Science
Dr. James G. Smith, Program Manager, Code 1211

Computer Science Division
Dr. Andre van Tilborg, Scientific Officer, Code 1133

Submitted to:

Director
Naval Research Laboratory
Washington, D.C. 20375

Attention: Code 2627

Submitted by:

R. P. Cook
Associate Professor and Chairman

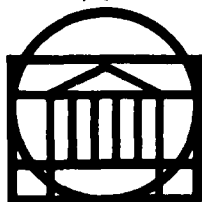
S. H. Son
Assistant Professor

Report No. UVA/525410/CS88 101

September 1987

DTIC
FILED
OCT 1 1987
S H

AD-A186 557



SCHOOL OF ENGINEERING AND
APPLIED SCIENCE

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA 22901

87 10 9 009

A Semi-Annual Progress Report
Contract No. N00014-86-K-0245
March 1, 1987 - September 1, 1987

THE STARLITE PROJECT

Applied Math and Computer Science
Dr. James G. Smith, Program Manager, Code 1211

Computer Science Division
Dr. Andre van Tilborg, Scientific Officer, Code 1133

Submitted to:

Director
Naval Research Laboratory
Washington, D.C. 20375

Attention: Code 2627

Submitted by:

R. P. Cook
Associate Professor and Chairman

S. H. Son
Assistant Professor

Department of Computer Science
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Report No. UVA/525410/CS88/101
September 1987

Copy No. 101

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKING None	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution Unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UVA/525410/CS88/101			5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION University of Virginia Department of Computer Science		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research Resident Representative	
6c. ADDRESS (City, State, and ZIP Code) Thornton Hall Charlottesville, VA 22901			7b. ADDRESS (City, State, and ZIP Code) 818 Connecticut Ave., N.W., Eighth Floor Washington, DC 20006	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research		8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0245	
8c. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, Virginia 22217-5000			10. SOURCE OF FUNDING NUMBERS	
			PROGRAM ELEMENT NO.	PROJECT NO.
			TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) The Starlite Project				
12. PERSONAL AUTHOR(S) R. P. Cook; S. H. Son				
13a. TYPE OF REPORT Semi-Annual Progress		13b. TIME COVERED FROM 3/1/87 TO 9/1/87	14. DATE OF REPORT (Year, Month, Day) October 1987	15. PAGE COUNT
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
The StarLite Project has the goal of constructing a program library for real-time applications. The initial focus of the project is on operating system and database support. The project also involves the construction of a prototyping environment that supports experimentation with concurrent and distributed algorithms in a host environment before down-loading to a target system for performance testing. The components of the project include a Modula-2 compiler, a symbolic Modula-2 debugger, an interpreter/runtime package, the Phoenix operating system, the meta-file system, a visual simulation package, a database system, and documentation.				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. James G. Smith			22b. TELEPHONE (Include Area Code) 202-696-4713	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR

33 APR edition may be used until exhausted

All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

TABLE OF CONTENTS

	<u>Page</u>
1. Introduction to the StarLite Project	1
2. Student Participation	2
3. Publications	2
4. Operating Systems	5
5. The StarLite Programming Environment	6
6. The Meta-File System	8
7. Database Systems	9
7.1 Algorithms for Synchronization and Recovery	9
7.2 Development of a Message-Based Simulator	11

1. Introduction to the StarLite Project

It seems improbable that a single operating system will suffice to solve all the application problems that are likely to arise in future real-time systems. A much more likely scenario is that future engineers, with support from a programming environment, will select and adapt modules from program libraries. The selected modules must have proven operating characteristics and the domain over which they are applicable must be well-defined.

The StarLite Project, supported by the Office of Naval Research, has the goal of constructing such a program library for real-time applications. The initial focus of the project is on operating system and database support. The project also involves the construction of a prototyping environment that supports experimentation with concurrent and distributed algorithms in a host environment before down-loading to a target system for performance testing.

The components of the project include a Modula-2 compiler, a symbolic Modula-2 debugger, an interpreter/runtime package, the Phoenix operating system, the meta-file system, a visual simulation package, a database system, and documentation. The database system is the responsibility of Professor Son. The remaining components are the responsibility of Professor Cook. Figure 1 gives a historical review of progress to date on each component of the StarLite project and also outlines our future plans. Each component is discussed in more detail in later Sections.

The module libraries for the database system and Phoenix are being implemented in Modula-2. The target hardware is currently PC-compatibles, but the library will also be ported to other architectures, including the Motorola 68000 architecture. The prototyping environment is being developed for PCs and Sun workstations. While Modula-2 has been chosen as the implementation language for our experiments, there are no Modula-2 dependencies in the program library. It is intended that the library be easily portable to other languages.

The research includes a systematic analysis of the functional and operational characteristics of the library modules. Such an analysis often pays extra dividends by exposing operational environments in

which none of the extant algorithms work; thus, the result is research focused towards the solution of particular problems. This can be contrasted with the popular approach of selecting problems that can be solved, but which are not useful.

2. Student Participation

The following students participated in the StarLite project over the last six months. Some were supported by the contract, others contributed as the result of class projects.

Jim Brown(B.S. student), visual simulation package

Richard Crowe(M.S. student), file system development

Bill Dixon(B.S. student), symbolic debugger

Lori Fitch(M.S. student), compiler development

David Kaminsky(B.S. student), graphics routines

Yumi Kim(M.S. student), multi-version database synchronization

Jeremiah Ratner(M.S. student), database development

Richard Testardi(Ph.D. student), compiler and operating system development

Jenona Whitlach(Ph.D. student), Meta-File System Development

Nancy Yeager(M.S. student), Meta-File System Development

3. Publications

• Refereed Journal Publications

- (1) Son, S.H., Synchronization of Replicated Data in Distributed Systems, *Information Systems*, Vol. 12, No. 2, 1987, pp 191-202.
- (2) Son, S.H., On Multiversion Replication Control in Distributed Systems, *Computer Systems Science and Engineering*, Vol. 2, No. 2, April 1987, pp 76-84.
- (3) Son, S.H., Using Replication to Improve Reliability in Distributed Information Systems, *Information and Software Technology*, October 1987 (to appear).
- (4) Cook, R.P., An Empirical Analysis of the Lilith Instruction Set, *IEEE Transactions on Computers*, (to appear).

Figure 1. PROGRESS TO DATE FOR EACH COMPONENT OF STARLITE AND FUTURE PLANS

COMPILER	3/1/87 - Lilith one-pass compiler port to a PC started	5/1/87 - Lilith one-pass compiler running on a PC using the Logitech Modula-2 system	7/1/87 - One-pass compiler converted to a 32-bit machine model
	7/1/87 - Lilith symbolic debugger port started	1/1/88 - Estimated completion date for the symbolic debugger running on a PC. Implementing windows first is the hard part	6/1/88 - Extend debugger to support view definitions of user objects
INTERPRETER	4/1/87 - Interpreter for M-code started on the PC	5/1/87 - Interpreter for M-code completed. S-code version started	7/1/87 - Interpreter for 32-bit S-code completed
PHOENIX OS	6/1/86 - Phoenix operating system implementation started on the Lilith	12/1/86 - Phoenix operating system implementation on the Lilith completed	9/1/87 - First version of Phoenix operating system implementation runs on the 32-bit PC environment
META-FILE SYSTEM	1/1/86 - Meta-File system implementation started	12/1/86 - Meta-File system implementation on the Lilith completed	7/1/87 - Meta-File system ported to the PCs for experimentation with a UNIX interface
SIMULATION PACKAGE	6/1/86 - Visual simulation package on Lilith under development	12/1/86 - Simulation package on the Lilith completed	7/1/87 - Visual simulation package completed on PCs; simulation package ported to SUN-3
DATABASE SYSTEM	6/1/87 - Port of concurrent programming modules to SUN-3 begun	9/1/87 - Transaction manager, lock manager, message server completed on Sun-3	1/1/88 - Time-stamp synchronization, deadlock detection, user interface to be completed. Start performance modelling of synchronization algorithms
DOCUMENTATION	6/1/86 - Draft of "Introduction to Modula-2 for Pascal Users"	1/1/88 - Estimated completion date of the first StarLite User's Manual	

Figure 1. PROGRESS TO DATE (continued)

COMPILER	1/1/88 - Convert one-pass compiler to C for portability	6/1/88 - C compiler completed plus LONGREAL/CARD support
DEBUGGER		
INTERPRETER	1/1/88 - Extend S-code for LONG REAL/CARD support	6/1/88 - Add additional machine models to runtime support
PHOENIX OS	1/1/88 - Extend functionality of Phoenix and do performance testing	6/1/88 - First release of StarLie with Phoenix OS
META-FILE SYSTEM	1/1/88 - Do performance testing and fine tune design	6/1/88 - First release of meta-file system standard proposal
SIMULATION PACKAGE	1/1/88 - Add additional network devices	6/1/88 - First release of the simulation package
DATABASE SYSTEM	6/1/88 - Complete performance testing. First release of prototype	
DOCUMENTATION		

• Conference Publications

- (5) Cook, R.P. and R.J. Auletta, StarLite, A Visual Simulation Package for Software Prototyping, *ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, (Dec. 1986); also *SIGPLAN Notices* 22, 1(Jan. 1987).
- (6) Cook, R.P., StarLite, A Network-Software Prototyping Environment, *ACM/IEEE Symposium on the Simulation of Computer Networks*, (August 1987).
- (7) Son, S.H., Reliable Distributed Database Systems, *Proceedings of ACM Computer Science Conference*, St. Louis, Missouri, February 1987.
- (8) Son, S.H., Replication Control and Recovery in Distributed Systems, *Proceedings of IEEE Southeastcon*, Tampa, Florida, April 1987.
- (9) Son, S.H., A Synchronization Scheme for Replicated Data in Distributed Information Systems, *Proceedings of IEEE Symposium on Office Automation*, Gaithersburg, Maryland, April 1987.
- (10) Cook, R.P. and S. H. Son, The StarLite Project, *Proceedings of Fourth IEEE Workshop on Real-Time Operating Systems*, Cambridge, Massachusetts, July 1987.
- (11) Son, S.H., A Recovery Scheme for Database Systems with Large Main Memory, *Proceedings of 11th International Computer Software and Applications Conference (COMPSAC 87)*, Tokyo, Japan, October 1987 (to appear).
- (12) Son, S.H., Using Replication for High Performance Database Support in Distributed Real-Time Systems, *8th Real-Time Systems Symposium*, San Jose, California, December 1987 (to appear).
- (13) Son, S.H., An Adaptive Scheme for Checkpointing and Recovery in Distributed Databases with Mixed Types of Transactions, *Fourth International Conference on Data Engineering*, Los Angeles, California, February 1988 (to appear).

• Technical Reports

- (14) Son, S.H., Using Replication to Improve Reliability in Distributed Information Systems, *Technical Report TR-87-16*, Dept. of Computer Science, University of Virginia, August 1987.
- (15) Son, S.H., Using Replication for High Performance Database Support in Distributed Real-Time Systems, *Technical Report TR-87-17*, Dept. of Computer Science, University of Virginia, August 1987.

4. Operating Systems

There are two views of operating systems. The traditional view treats an operating system as a monolithic entity that owns all low-level resources, such as memory, devices, CPUs. In the second view, an operating system is simply a collection of modules with certain properties. The modules are "bound" to the application program to form a single, larger program that solves a particular problem. We favor the latter view as it is consistent with our modular programming approach to real-time systems. It is also consistent with the notion of single-language programming environments, such as those being developed for Ada.

In a modular operating system, the implementation of a module can be replaced without affecting the module's interface. Thus, it should be possible to construct an operating systems generator, which when given a requirements specification and a target machine description, could produce a set of modules meeting the specification. Judging from experience in the compiler area, such a generator will only be possible if a set of suitable modules can be agreed upon.

Another area of investigation concerns real-time, full-function operating systems. Implementation technology needs to be "pushed" to determine the costs of providing additional functionality in terms of the potential effects on real-time guarantees. In some cases, an engineer might opt for increased functionality, even if response time were degraded or could only be expressed in probabilistic terms. The StarLite module library currently includes a partial UNIX implementation, Phoenix, which we are currently testing to determine its real-time performance limits. Figure 2 illustrates the current module dependency matrix for the operating system.

The dependency matrix indicates the hierarchical nature of the module library. For example, the Atomic module is used to build SEMAPHORE, SEMAPHORE is used to build MONITOR and CONDITION, and then they are used to build Phoenix. The module names "in caps" are type modules. A hierarchical structure is desirable because it enhances the reusability of the component modules.

In addition to having a hierarchical organization, the Phoenix system has a number of other interesting properties. First, the Atomic module contains the only code in the system that disables interrupts. Furthermore, the worst-case disable time is only a few instructions. Thus, the response time to interrupts is fixed and very small. This can be contrasted to the majority of UNIX implementations in which disabled sections of code can run for many milliseconds. Another attribute of Phoenix is that it is designed to operate in either a multi- or uni-processor environment. For example, the use of the Monitor data type is particularized to data instances, not code modules. As a result, it is possible for an arbitrary number of processes to execute kernel code simultaneously. Again, this is quite different from most UNIX implementations. As a final point, the Phoenix implementation is object-based. The main advantage is that the User structure, which represents user-level processes, can be modified arbitrarily without necessitating a recompilation of the entire operating system. Again, this is an improvement over existing implementations.

5. The StarLite Programming Environment

The StarLite programming environment consists of a Modula-2 one-pass compiler, a symbolic debugger, interpreter, and simulation package. The goal is to produce a portable, standard environment with which researchers can develop real-time concurrent and distributed software. Furthermore, by using the environment, researchers at one university will be able to validate the experiments performed at other universities. This Section describes in more detail the progress-to-date of each component (refer to Figure 1).

The compiler supports the Revised Modula-2 Language Definition, except for the LONGREAL/CARD types; LONGINT is supported. Its compilation speed is twice as fast as the Logitech 286 compiler and five times as fast as the SUN-3 Modula-2 compiler. It also compiles faster than either the MicroSoft C compiler on a PC286 or the SUN-3 C compiler. Fast compilation has been rated as essential to the success of a programming environment (see, for example, Xerox CSL-80-10).

The compiler is currently implemented in Modula-2, but we are also creating a version in C so that the entire environment can be easily ported to new architectures. The generated code is for a 32-bit virtual architecture(S-code) that is designed to be extremely space efficient. For example, the object code sizes for a program consisting of 1,000 assignment statements was SUN-Modula(130K), SUN-C(65K), PC286-C(35K), PC286-StarLite(11K). Compact code has a significant effect on the speed with which the environment can load both system components and user-level programs that might run on those components. Code generators for a number of target environments are planned for the future.

Figure 2. MODULE DEPENDENCY MATRIX

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1 Storage																		
2 OBJECT																		
3 Errors																		
4 COROUTINE																		
5 Traps	X																	
6 Atomic			X	X	X													
7 QUEUE						X												
8 PROCESS	X	X	X	X	X	X	X		X									
9 Low		X		X		X	X											
10 SEMAPHORE			X			X		X	X									
11 COUNTINGSEM			X			X		X	X									
12 Priority								X	X									
13 MONITORTYPE										X	X							
14 DISK																		
15 MONITOR			X							X			X					
16 CONDITION			X					X		X			X		X			
17 LOWUSER			X	X				X	X						X	X		
18 Phoenix	X	X	X			X		X	X						X	X	X	

The interpreter/runtime system for StarLite is unique in a number of respects. First, it supports dynamic linking; that is, modules are loaded at the point that one of their procedures is called. Thus, a large software system begins execution very quickly and then loads only the modules that are actually tested. At the current time, a linker is superfluous; as soon as a module is compiled, it may be executed. The second feature of the interpreter is that it maximizes sharing. There will be only one copy of shared code no matter how many times it is used at either the user or operating system levels. Next, the clocks on the interpreter's virtual machines are driven by the number of S-code instructions executed. Thus, timings for the StarLite host environment can be used to approximate those in a target environment by varying the ratio of S-code instructions necessary for a clock tick. Finally, the interpreter is designed to support a number of different execution models. The present model is a traditional uniprocessor, but we have plans for a shared-memory multiprocessor model and a distributed systems model. For example, an experiment might consist of a file server, name server, and client node all running on top of the base interpreter system. If all three nodes were implemented using the same base operating system, the environment would store only one copy of the code; however, each node would require its own data area.

The visual simulation package[5,6] incorporates many of the features of the GPSS simulation language. The traditional "delay" function is provided, as well as the Store and Table simulation types that are used for statistics gathering. Typically, the presence of simulation code is isolated at the lowest levels of a system. By keeping interfaces compatible, a simulation module can be replaced by a module for the target machine. Thus, the higher levels of the software hierarchy remain unchanged when moving code from the host environment to a target.

In addition to the simulation entities, a window interface is provided to allow a user in the host environment to "view" the state of the system dynamically. The current device library includes EtherNet, clock, and disk modules. We hope to add additional modules for optical disks, token rings and token busses. Another goal for the future is to support hybrid execution in which parts of a system run on the target and parts run in the host environment.

The visual simulation capability is also important in the area of software documentation. For example, by recording window activity, the software developer can produce an animated document of important system attributes.

The final component of the StarLite environment is the symbolic debugger. It is being developed so that it can be used as either an interactive or post-mortem debugger. Also, it has the capability to examine multiple threads of control. Eventually, we will add support for user-defined "views" of data abstractions and the ability to view data other than program images. For example, the debugger could be used to examine, or modify, a file that was described in the Modula-2 Interface Definition Language, which is supported by the compiler.

In summary, the StarLite environment is designed to maximize productivity. Therefore, it accelerates a researcher's ability to conduct experiments, which advances the state-of-the-art.

6. The Meta-File System The meta-file system is designed to be a national standard. It has two properties that support that goal. First, it is extensible; new access methods can be defined dynamically. Access methods are bound to the meta-file system kernel by means of dynamic linking and up-level calls. Secondly, it is *Parnas transparent*; that is, the states and efficiency of the underlying disk mechanism are made available to the application programmer. This can be contrasted with UNIX, which is extensible, but new access methods cannot manipulate the disk directly. For example, a user is not allowed to create an index block.

Obviously, before proposing any kind of standard, we intend to perform a significant amount of experimentation to determine the suitability of our design. We are currently implementing a UNIX interface that runs with the meta-file system kernel. Once it is complete, we will do performance testing to validate our "execution transparency" claim.

7. Database Systems

Distributed systems with real-time constraints must maintain high reliability, which is the ability of the system to maintain consistency and to provide continued service in spite of failures. Several actions may result in inconsistent system states. Incorrect synchronization and system failures are two of them. It is rather straightforward to achieve only the requirement of consistency, for example, by shutting down the entire system when a failure occurs. However, this approach is not acceptable for distributed real-time systems that require high reliability as well as high availability. Therefore, when developing a distributed system, all important factors, such as performance and resiliency, must be taken into consideration. One of the primary reasons for the difficulty in successfully designing and evaluating a distributed system is that it is very complicated and it takes a long time and huge effort to actually develop a system before it can be evaluated. In many cases, developing a distributed real-time system requires a concerted effort for solving problems associated with many different components of the system, including process scheduling, synchronization, and failure recovery.

High reliability is not the only requirement for real-time systems. Since real-time systems typically require access to large volumes of data within a certain time limit (deadline), developing a highly reliable real-time system needs to have very powerful access and processing mechanisms for large volumes of data. Therefore, it is clear that the database systems for such real-time applications must provide a very high throughput. A relatively low throughput provided by the database systems based on conventional control mechanisms may not be able to support such a high performance requirement.

In a distributed real-time system, data can be stored at several sites. The proliferation of workstations and personal computers makes data distribution and replication attractive because one way of improving the availability of data in a system with unreliable sites is to replicate the data and store it at multiple sites. A single site failure does not make replicated data inaccessible; the system can access the data in the presence of failures, even if some of the redundant copies are not available. In addition to improved availability, data distribution and replication also enhance performance by placing the data closer to the process that requires it. For example, queries initiated at sites where the data are stored can be processed locally without incurring communication delays, and the workload of queries can be distributed to several sites where the subtasks of a query can be processed concurrently. However, the benefits of data distribution and replication must be balanced against the additional cost and complexities introduced for the synchronization of replicated and distributed data.

Our research effort during February 1987 to August 1987 was concentrated in two areas: designing algorithms for synchronization and recovery, and developing a message-based, discrete-event simulator for evaluating database support mechanisms. The research has resulted in the development of a set of database support algorithms for synchronization, replication control, and system recovery. Simulator development was initiated in June, and most of the basic procedures have been implemented.

7.1. Algorithms for Synchronization and Recovery

An obvious approach to improve reliability of critical data in a distributed environment is to keep replicated copies of data at several sites. A major restriction in using replication is that replicated copies must behave like a single copy, i.e., *mutual consistency* of a replicated data must be preserved. The principal goal of a replication control mechanism is to guarantee that all updates are applied to copies of replicated data in a way that preserves mutual consistency. The task of synchronization in a distributed environment is more complicated than that in a centralized environment mainly because the information used to make scheduling decisions is itself distributed, and it must be managed properly to make correct decisions.

Partial operation policies for replicated data are critical in maintaining correctness and achieving the high reliability of the system. To achieve high reliability, the system must allow user requests to be processed even when network partitioning occurs. Two alternatives are possible when a partition occurs:

- (1) pessimistic: allow at most one group to process transactions,

- (2) optimistic: allow each group to process transactions.

Neither of the two alternatives is superior to the other. An optimistic approach may achieve high availability while partition failures exist. However, it may be penalized during recovery by the overhead in merging different execution orders of updates committed at several partitions into a correct schedule. Moreover, the system predictability would be severely impaired if some transactions should be backed out which violate consistency constraints. In each partition, different algorithms may be used for replication control and synchronization. Two major techniques for managing replicated data are *voting* and *special copy*. In voting-based schemes, each copy has a number of votes, and a predetermined number of votes is necessary to perform a desired operation. In the special copy approach, availability of a special copy (or any copy) enables an operation on the data object. Each partial operation policy and replication control technique has its own benefits and costs. We have classified different replication control techniques by their underlying mechanisms and the type of information they use in ordering the transactions[1]. The trade-offs between performance and reliability of several database support mechanisms have been evaluated[8]. We will expand our work by including predictability as another important measure for the evaluation of database support mechanisms. The results of this study will provide a clear understanding of different partial operation policies with their costs and benefits in quantitative measures, which would be very helpful for real-time system designers.

In addition, we have developed database support mechanisms for integrated concurrency and replication control. Our mechanisms can be considered as a compromise between voting and special copy approaches[9]. The mechanisms are extended either by using multiple versions of data[2], or by using the semantic information of transactions[12]. The preliminary results indicate that these mechanisms are promising for distributed real-time systems because they increase the degree of concurrency by exploiting the *old-values*, that are maintained in the system for recovery reasons. This idea of integrated concurrency and replication control in distributed environments will be further studied in this project to verify their correctness and to evaluate their performance/reliability characteristics.

Recovery is another critical requirement for real-time systems. Database recovery essentially consists of two parts: preparation for recovery by saving necessary information during normal operation of the system, and actual recovery after failure. We have developed a recovery scheme using a non-interfering checkpointing mechanism[11]. Instead of waiting for a consistent state to occur, the non-interfering checkpointing approach constructs a state that would result by completing the transactions that are in progress when the global checkpoint begins. Users are allowed to submit transactions while the checkpointing is in progress, and the transactions are executed in the system concurrently with the checkpointing process. Two main properties of this checkpointing algorithm are global consistency and reduced interference, both of which are essential for achieving high availability.

Non-interfering checkpointing mechanisms, however, may suffer from the fact that the diverged computation needs to be maintained by the system until all of the transactions that are in progress when the checkpoint begins, come to completion. For database systems with many long-lived transactions that need long execution time, this requirement of maintaining diverged computation may make non-interfering checkpointing not practical. We have developed a checkpointing algorithm that is non-interfering with transaction processing, and efficiently generates globally consistent checkpoints[13]. The algorithm also prevents the well-known "domino effect", and saves intermediate results of the transaction, in an adaptive manner.

Several problems related to the idea of a non-interfering checkpoint and associated backward recovery mechanisms have not been investigated in depth so far. One of them is the robustness of the mechanisms to failures during checkpointing or the recovery process, and to severe failures like network partitioning. Most checkpointing and recovery mechanisms proposed in the literature assume no failures during the execution of the mechanisms. This assumption is based on the expected low probability of failures during the execution. However, an inconsistent state would result if a failure occurs at some unfortunate instant of time, unless the mechanisms are designed to manage such failures.

Other problems of checkpointing and recovery mechanisms are related to the efficient implementation and performance characterization. The practicality of non-interfering checkpointing depends partially on the amount of extra workload incurred by the checkpointing mechanism. We will evaluate the real-time performance characteristics of checkpointing and recovery mechanisms using the message-based simulation software described in the next section.

7.2. Development of A Message-Based Simulator

Although there already exist numerous tools for system development and analysis, only a few of them are really useful for system designers. If the designer must deal with message-passing distributed systems and tight real-time constraints, it is essential to have an appropriate simulation software for the success in the design and analysis tasks.

Message-based simulations, in which events are message-communications, do not provide additional expressive power over standard simulation languages; message-passing can be simulated in many discrete-event simulation languages including SIMSCRIPT and GPSS. However, a message-based simulation can be used as an effective tool for developing a distributed system because the simulation "looks" like a distributed program, while a simulation program written in a traditional simulation language is inherently a sequential program. Furthermore, if the simulation program is developed in a systematic way such that the principles of modularity and information hiding are observed, most of the simulation code can be used in the actual system, resulting in a reduced cost for system development and evaluation. The few primitives required for message-based simulations are constructs to create and terminate processes, to send and receive messages between processes, and to block a process for messages and/or simulation time to elapse. In our simulation software, these primitives are based on the concurrent programming kernel developed as part of the StarLite operating system. The simulation is implemented in Modula-2 on a SUN-3 workstation.

For a message-based simulation, the process view of simulation has been adopted. A distributed system consists of a number of *processes* that interact with others at discrete instants of time. Processes are basic building blocks of a simulation program. A process is an independent, dynamic entity that manipulates *resources* to achieve its objectives. A *resource* is a passive object and may be represented by a simple variable or a complex data structure. A simulation program models the dynamic behavior of processes, resources, and their interactions as they evolve in time. Each physical operation of the system is simulated by a process, and the process interactions are called *events*.

We use the client/server paradigm for process interaction in our model. The system consists of a set of clients and servers, which are processes that cooperate for the purpose of transaction processing. Each server provides a service to the clients of the system, where a client can request a service by sending a request message (a message of type *request*) to the corresponding server. The computation structure of the system to be modeled can be characterized by the way clients and servers are mapped to processes. For example, a server might consist of a fixed number of processes, each of which might execute the requests of every transaction, or it might consist of a varying number of processes, each of which executes on behalf of exactly one transaction.

The internal actions of a process, i.e., actions that do not involve interactions with other processes in the system, are modeled either by the passage of simulation time or by the execution of sequential statements within the process. We use a simulator clock to represent the passage of time in a simulation. The simulator clock advances in discrete steps, where each step simulates the passage of time between two events in the system.

In a physical system, each process makes independent progress in time, and many processes execute in parallel. In a simulation, the multiple processes of a physical system must be executed simultaneously on one processor. This simultaneity is achieved by interleaving the execution of different processes and executing them in a quasi-parallel fashion. A scheduling primitive, **PauseProcess**, is provided to guarantee quasi-parallel processing and finite progress for all active processes.

In message-based simulations, the communication of a message takes zero units of simulation time. For instance, in transaction execution we may assume that the time taken for sending a message to a destination process is insignificant; thus, in the simulation, the transmission time for the message that models this event can be zero. However, nonzero transmission delays exist in distributed systems, and they can be modeled by causing the process sending (or receiving) a message to wait for a certain time corresponding to the message-transmission time before (or after) sending (or receiving) the message.

Performance comparisons of synchronization algorithms have been carried out by several researchers using simulation and analytical models. However, there are important issues that have received little consideration in previous studies:

- (1) Most of the previous modeling studies considered only the performance of synchronization algorithms in a single processor system. Few studies have addressed distributed environments.
- (2) Since synchronization algorithms must work correctly even with subsystem failures, performance and reliability characteristics of synchronization algorithms in degraded circumstances need to be studied.
- (3) Although message-based simulations appear to be more natural for simulating distributed systems, a message-based approach to discrete-event simulation of distributed systems has not been fully developed.

An evaluation based on a combination of performance characterization and modeling studies seems necessary in order to understand the impact of synchronization algorithms on the performance of distributed systems. In fact, performance measurement or a detailed simulation is the only way to obtain values for many of the parameters used in analytic modeling studies.

Although our primary goal in using the StarLite environment is the design and evaluation of synchronization algorithms and recovery mechanisms for distributed systems, the development of network operating system kernels or other application software, such as network mail servers, could also benefit from StarLite.

DISTRIBUTION LIST

Copy No.

1 - 6 Director
 Naval Research Laboratory
 Washington, D.C. 20375
 Attention: Code 2627

7 Dr. James G. Smith, Code 1211
 Applied Math and Computer Science Division
 800 N. Quincy Street
 Arlington, VA 2217-5000

8 - 19 Defense Technical Information Center, S47031
 Bldg. 5, Cameron Station
 Alexandria, VA 22314

20 - 22 R. P. Cook, CS

23 - 24 S. H. Son, CS

25 - 26 E. H. Pancake, Clark Hall

27 SEAS Publications Files

* Office of Naval Research Resident
 Representative
 818 Connecticut Ave., N.W.
 Eighth Floor
 Washington, D.C. 20006
 Attention: Mr. Michael McCracken
 Administrative Contracting
 Officer

* Send Copy of Cover Letter Only

J0#0366: jlb

UNIVERSITY OF VIRGINIA
School of Engineering and Applied Science

The University of Virginia's School of Engineering and Applied Science has an undergraduate enrollment of approximately 1,500 students with a graduate enrollment of approximately 560. There are 150 faculty members, a majority of whom conduct research in addition to teaching.

Research is a vital part of the educational program and interests parallel academic specialties. These range from the classical engineering disciplines of Chemical, Civil, Electrical, and Mechanical and Aerospace to newer, more specialized fields of Biomedical Engineering, Systems Engineering, Materials Science, Nuclear Engineering and Engineering Physics, Applied Mathematics and Computer Science. Within these disciplines there are well equipped laboratories for conducting highly specialized research. All departments offer the doctorate; Biomedical and Materials Science grant only graduate degrees. In addition, courses in the humanities are offered within the School.

The University of Virginia (which includes approximately 2,000 faculty and a total of full-time student enrollment of about 16,400), also offers professional degrees under the schools of Architecture, Law, Medicine, Nursing, Commerce, Business Administration, and Education. In addition, the College of Arts and Sciences houses departments of Mathematics, Physics, Chemistry and others relevant to the engineering research program. The School of Engineering and Applied Science is an integral part of this University community which provides opportunities for interdisciplinary work in pursuit of the basic goals of education, research, and public service.

END

DATE

FILM

JAN
1988