CONF-9509217--1

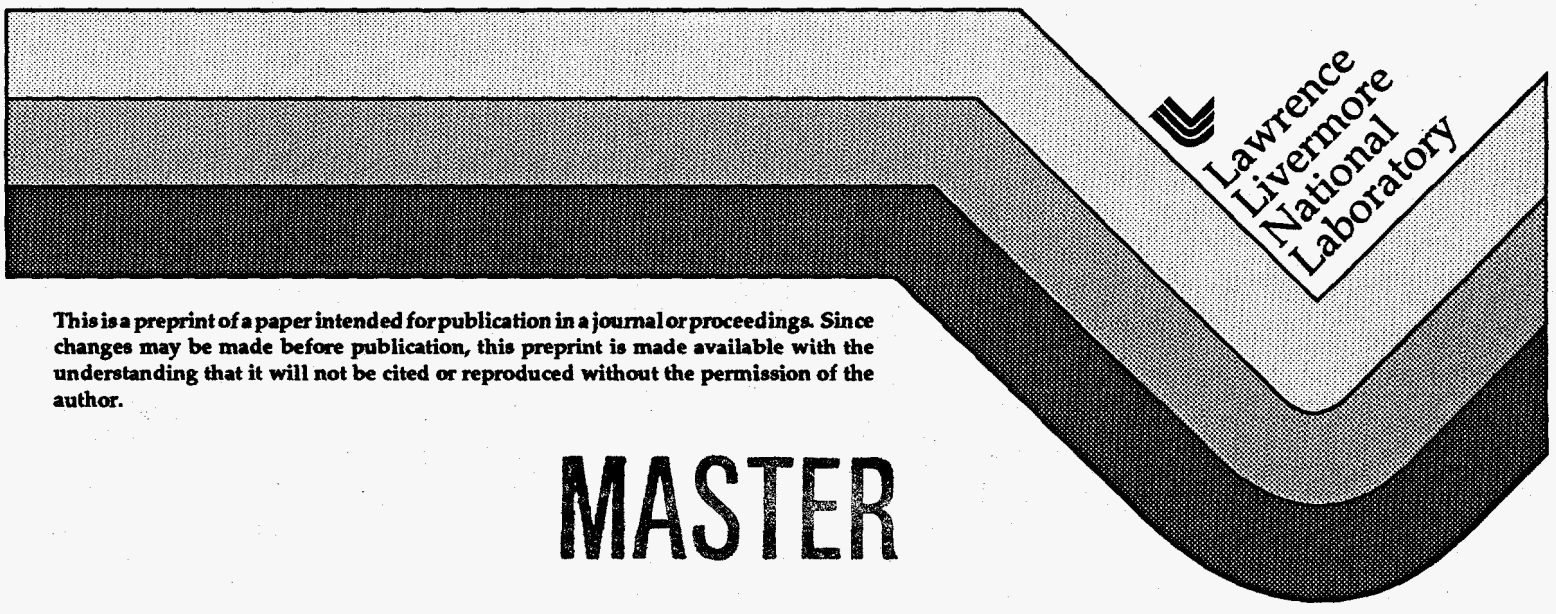
UCRL-JC-121507
PREPRINT

Maintenance Simulation: Software Issues

C. H. Luk
M. A. Jette

This paper was prepared for submittal to the
Eurosim Simulation Congress 95
Vienna, Austria
September 11-15, 1995

July 1995



 85

Maintenance Simulation: Software Issues[1]

Clement H. Luk and Morris A. Jette

National Energy Research Supercomputer Center, Lawrence Livermore National Laboratory, University of California, P. O. Box 5509, L-431, Livermore, California 94551, U. S. A.

Abstract

The maintenance of a distributed software system in a production environment involves: 1. maintaining software integrity, 2. maintaining and database integrity, 3. adding new features, and 4. adding new systems. These issues will be discussed in general: what they are and how they are handled.

This paper will present our experience with a distributed resource management system that accounts for resources consumed, in real-time, on a network of heterogenous computers. The simulated environments to maintain this system will be presented relate to the four maintenance areas.

1 Introduction

The maintenance cost for a software system can range from 40% to 80% of the system's outlay [BUCK89, EDEL92, SCHN89]. Informally, software maintenance is generally effected with repetition of review/walkthrough [OSBO90]. More formally, software metrics is used to ensure the quality of software [GIBS89, LI93, STAR94]. While the need to simulate actions and objects (e.g. databases) should be natural, there appear to be very little published materials on maintenance simulation[2].

In this paper, we briefly describe a distributed resource management system for an energy research supercomputer center. The maintenance issues are then presented to provide discussions on maintenance simulation.

1.1 The National Energy Research Supercomputer Center (NERSC)

NERSC is funded by the United States Department of Energy (DOE). The user community consists of about 4,800 researchers worldwide. The user community is divided into about 600 project groups. The available compute resources include a massively parallel Cray T3D with 256 processing element and 16GB memory, one 16 CPU Cray C90 with 2GB memory, two older multiCPU Cray2s each with 1GB memory, SUNs, HPs and an assortment of workstations. Other resources include an international network, a Common File System (CFS) with 12 terabytes of data etc.

1.2 Unified Production Environment (UPE)

Through UPE, NERSC integrates services within a given computer and across computers on a Wide Area Network (WAN). One important service is a highly reliable resource allocation and accounting system, called CUB (Centralized User Banking) [JETT91]. CUB (FIG. I) has been designed to provide resource management service in an environment of heterogeneous, UNIX

---

1. This work has been supported by the US Department of Energy under contract W-7405-Eng-48 by LLNL.

2. One study by Gibson & Senn[GIBS89] addressed an experiment to help programmers visualize maintenance complexity with simulation, as part of their investigation on maintainability and system structure,

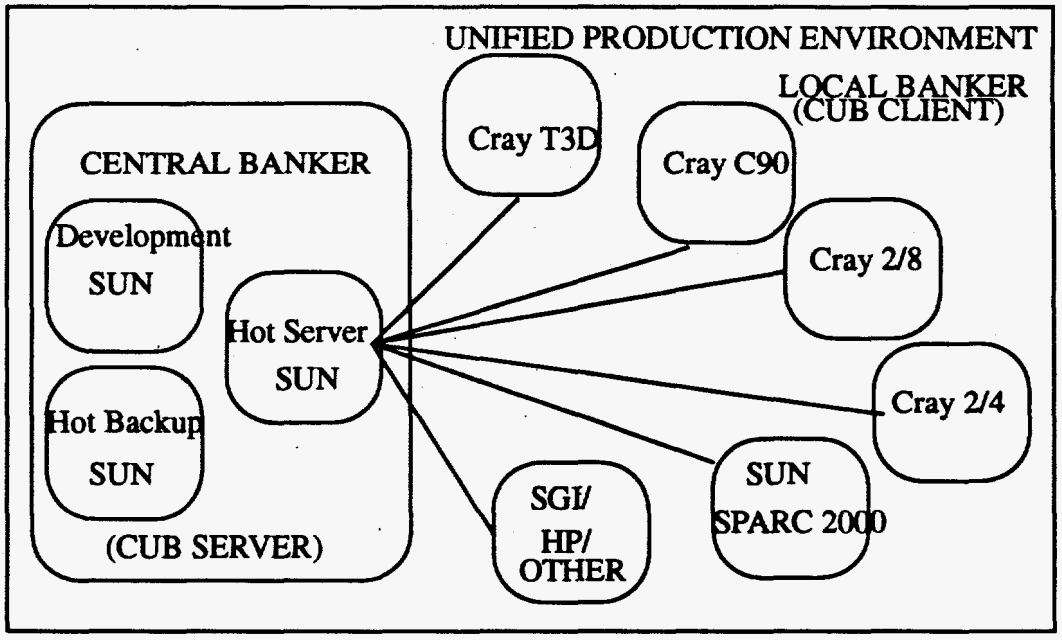


FIGURE I. UPE & CUB

based computers ranging from mid-sized workstations to the largest of supercomputers. CUB is also integrated with the batch system (Network Queuing System, NQS) to control resource use in as unobtrusive a fashion as possible.[1]

On most UNIX computers, resource allocation is lacking and resource accounting is based upon timecards generated when processes complete execution. Many processes run on NERSC's supercomputers for hundreds of hours, giving timecard based resource allocation very poor accuracy[2]. CUB accounting and resource allocation is done in near real-time. Daemon processes monitor resource use every ten seconds and report this use to a centralized CUB server at about the same interval. The centralized CUB server reports back the resources remaining by user and group. Once the resources allocated to an individual or his group have been exhausted, executing batch and interactive jobs are suspended. Only limited work may be performed until additional resources are made available, at which time suspended jobs resume execution.

Communications are UDP/IP[3] based, for the flexibility in addressing time-outs, retransmissions, etc. Transactions are protected by a crypto-checksum to insure security.

To insure reliability of CUB, three machines are available as central bankers: a production machine, a "hot backup", and a development platform. The central banker runs on the production machine. A backup program runs on the "hot backup" machine and maintains database transaction records that will be only a few minutes older than the database transaction records of the pro-

1. For example, a program using all the resources on the T3D for one hour probably will take a few days on the 6 CPU SUN Sparc 2000. Resource management also has to provide operational facilities to move resources from computer to computer in real time.
2. Waiting for timecards to be generated on process completion, as in standard UNIX accounting, is not considered acceptable. With some processes running for hundreds of CPU hours, monitoring timecards would occasionally result in substantially more resources being consumed than authorized. Additionally, many users have discovered that they can avoid charges at some computer centers by preventing their jobs from terminating. Their jobs can complete useful work and sleep until the computer crashes, avoiding the generation of a timecard. [JETT94]
3. UDP (User Datagram Protocol) is a simple datagram protocol which is layered directly above the Internet Protocol (IP).

duction machine. The "hot backup" machine can be made into the production machine within about 30 minutes, if needed. The development platform provides "warm backup". All CUB database and software are available on this machine. The database does not necessary contain the latest information. However, should the production machine fail and the "hot backup" is called to duty, the development platform can become the "hot backup". The platform is normally used for development, and provides an environment for maintenance simulation for the central banker.

1.3 Centralized User Banking (CUB)

FIG II shows a simplified structure of CUB. On each user production machine, e.g. Cray C90,

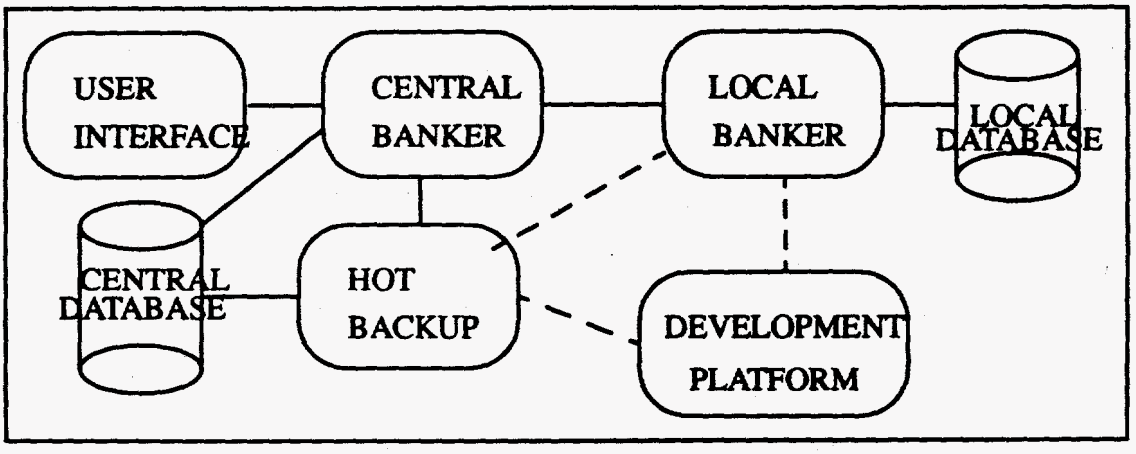


FIG II. SIMPLIFIED STRUCTURE OF CUB

a local banker (daemon) gathers information on CPU time and other resources consumed on that machine for each user and each project group at intervals of ten seconds[1]. The information is recorded in local databases. Changes in resource consumption such as CPU time used are noted. These changes are transferred for all active users and project groups to the central banker (CUB Server) about once every 10 seconds unless the central banker is unavailable. The central banker updates its central (ORACLE) database then transfers updated records of resources remaining by user and project group back to the local banker. The local banker updates it local database. This way the local and the central databases are synchronized.

The local bankers can operate for an extended period without the support of the central banker. The lack of an operating central banker merely prevents the local bankers from synchronizing their databases and prevents correct updates to database on other machines. Once communications are reestablished, the databases are synchronized. Database alterations (other than consumption of resources) are recorded in a journal to insure persistence across machine restarts.

The primary user interface to CUB communicates directly with the central banker. Resource reallocation by users will be communicated by the central banker to the local banker as described above.

2 Software Maintenance Issues

Software maintenance represents efforts to ensure that the software operates as expected on its database(s), even as the operating environment changes. For a system with distributed databases and operations, software maintenance will involve the following:

---

1. The resources consumed by the local banker itself are only 6.92 CPU minutes per day on a Cray Y-MP C90.

2.1 Software Integrity

When software does not perform as expected, it is commonly referred to as having bugs. This is frequently caused by the software engineer's failure to anticipate or visualize some possible operational requirements. Software maintenance to ensure integrity requires recognition of the problem (bug), followed by design and engineering of a solution.

2.2 Database Integrity

Data entry can be made by users and operators or support personnel. Improved user interface will help minimize problem potentials. There is still no guarantee that error entries will not be made. Maintenance will require the ability to provide playback [OSBO90] or the ability to backup the databases.

2.3 New Feature Addition

As a system evolves, new features are frequently needed to support changes in the operational environment. These changes may or may not conflict with original system requirements. They may also be incompatible with original design and implementation assumptions.

2.4 New System Addition

When a new system is added, the most common problem is incompatibility. With CUB, the local banker code required major rewrite to handle the addition of the Cray T3D Massively Parallel System.

3 Maintenance Simulation

The burden (cost) of software maintenance is significantly affected by software complexity [BANK93]. For distributed systems, software complexity and therefore maintenance cost will be extremely high. For our system, maintenance is categorized based on the four areas described above.

3.1 Software Integrity

Since integrity is based on whether software performs as expected, performance monitoring will be the first step. Problems discovered by the monitor are analyzed. The analysis frequently requires the ability to recreate (playback) the situation so as to "cause the problem to occur again".

3.1.1 Performance Monitor

With only a limited operations staff at NERSC, a Sun workstation is used to monitor all key systems. This monitors messages sent to each computer's operator console and checks key subsystems with daemons on these computers. Staff members are alerted to notable events by e-mail and/or audible message reporting the problem and computer(s) effected. The operations monitor is configured to note CUB communications failures, lack of space in key CUB server file systems, abnormally large or small resource rates, and restart of CUB components. Operations staff are capable of investigating failures, restarting CUB clients, or contacting CUB developers, as required.

3.1.2 Playback

Playback involves event recreation to help analyze the environment and cause(s) of the problem. Obviously, if playback is successful, the problem will occur again, possibly bringing down the system again. Playback must therefore be conducted in a simulated environment. With CUB,

transactions are recorded in such a way that their processing daemons can "playback" or re-act on previous transactions.

3.2 Database Integrity

The local banker's database is synchronized with the central banker's database each time the local banker is (re)started to insure current information. Alteration of this database would presumably be accompanied by an alteration in the communications protocol. To accomplish such a transition, we would operate parallel banking systems for a time. Independent sets of local bankers would communicate with a simulated central banker and results would be compared.

We could have simulated the databases. However, such an endeavor is not practical. With distributed database systems, the central database can act as backup to ensure integrity, provided databases are synchronized frequently. In our case, synchronization happens every 10 seconds. In the event of prolonged synchronization failure, maintaining information on changes in resource usage enable us to "automatically" monitor the local bankers' database integrity[1].

On the other hand, the central banker database cannot be easily monitored predicated on remote local databases without running a potential risk of deadlock. With CUB, the central database is "backed up" every 30 minutes to the "hot backup" machine. It is possible to reconstruct the central database from the "hot backup" with a loss of 30 minutes, or with a loss of about 10 seconds if we choose to modify our backup procedure to include "input" from local bankers.

For maintenance purpose, the two databases can be compared and any discrepancy of over 30 minutes indicates potential problems. At this point, maintenance can be performed on the development computer with a copy of the backup database. Real or experimental versions of the central banker and/or local bankers can be executed to simulate the events and its effects on the simulated database.

3.3 New Feature Addition

When new features are added the local banker, it is built on the platform (e.g. C90) upon which it will execute. Testing is initially performed on a limited subset of users with a simulated central banker. Jobs are execute at various nice values (priorities), batch and interactive, single and multi-threaded to validate a substantial portion of the local banker's code. CUB accounting information is compared with the UNIX accounting records generated upon process completion to assure accuracy. Typical tests (e.g. boundary conditions of user account exhaustion for either the user or the project group) are also performed.

After this limited testing, a local banker is built to account for all resources used by all users. This versions of the banker is prevented from changing user shells or suspending/resuming jobs, but does account for resource use. This banker is run in parallel with the production version for a period of days to assure its integrity and consistency with the version used in current production. Upon successful completion of this test period, the banker is rebuilt to perform all functions, communicate with the CUB server used for production, and replaces the former banker code.

The central banker with new features is built on the development platform. The actual central banker database is "copied" on the development computer. Simulated local bankers are started on production machines with simulated user account information. When these local bankers start, it will synchronize its local "new" databases from the simulated central database. Monitoring programs will report on local banker transactions, transmission log, and central banker transactions. By reviewing the events live, the software engineer can determine if the new features have been

1. For example, a process cannot use more than 10 seconds of one CPU in less than 10 seconds. The maximum amount of time deliverable to users for a 256 CPU T3D is 2,560 CPU seconds over a 10 second period. Operations staff are alerted to abnormally large or small resource use rates.

implemented correctly. As part of maintenance simulation, the central database can be "backed up" locally on the development machine and the databases can be compared, as described before.

3.4 New System Addition

For a heterogenous distributed system, the addition of a new system requires consideration of both the new system features (e.g. multiple CPUs) and normalization of measures for resource utilization [JETT94]. The approaches discussed above will apply. However, it will be necessary to create initial databases for the new system and careful evaluation that the databases are correct. Maintenance simulation can then be employed to ensured integrity of all software components and the databases.

4 Concluding Remarks

While there are calls for "A standard for software maintenance..." [EDEL92], [SCHN89]. the need for maintenance simulation has not been widely addressed. In this paper, we present the practical issues for software maintenance. From these, maintenance simulation is probably the most effective way to provide visualization for the support a complex system such as the one at NERSC.

Acknowledgment

Many people contributed to the CUB project. We would like to especially acknowledge Patty Clemo, Harry Massaro, John Reynolds (currently at Sequent Corp.) and Suzanne Smith for their software efforts and their helpful advice to the authors on this paper.

Bibliography

BANK93 Banker, Rajiv D. et al, "Software Complexity and Maintenance Costs", Communications of the ACM, Vol 36, #11 (November, 1993)

BUCK89 Buckley, Fletcher J., "Some Standards for Software Maintenance", Computer, Vol 22, #11 (November, 1989)

EDEL92 Edelstein, D. Vera & Mamone, Salvatore, "A Framework for Managing and Executing Software Maintenance Activities", Computer, Vol 25, #6 (June, 1992)

GIBS89 Gibson, Virginia R. & Senn, James A., "System Structure and Software Maintenance Performance", Communications of the ACM, Vol 32, #3 (March, 1989)

JETT91 Jette, Morris A. et al, "Report of the ERSUG Charging and Scheduling Algorithms Working Group", Lawrence Livermore National Laboratory, Report UCRL-ID-106965 (March, 1991)

JETT94 Jette, Morris A. & Reynolds, John J., "Centralized User Banking and User Administration on UNICOS", Cray User Group Meeting, Denver CO (March, 1994)

LI93 Li, Wei & Henry, Sallie, "Object-Oriented Metrics That Predict Maintainability", Journal of Systems and Software, Vol 23, #2 (November, 1993)

OSBO90 Osborne, Wilma M. & Chikofsky, Elliot J., "Fitting Pieces to the Maintenance Puzzle", IEEE Software, Vol 7, #1 (January, 1990)

SCHN89 Schneidewind, Norman F., "Software Maintenance: The Need for Standardization", Proceedings of the IEEE, Vol 77, #4 (April, 1989)

STAR94 Stark, George E., Kern, Louise C. & Vowell, C.W, "A Software Metric Set for Program Maintenance Management", Journal of Systems and Software, Vol 24, #3 (March, 1994)