An Object-Oriented Testbed for the Evaluation of Checkpointing and Recovery Systems

B. Ramamurthy & S. J. Upadhyaya

Dept. of Electrical & Computer Engineering
State University of New York
Buffalo, NY 14260

R. K. Iyer

Coordinated Sciences Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Abstract

This paper presents the design and development of an object-oriented testbed for simulation and analysis of checkpointing and recovery schemes in distributed systems. An important contribution of the testbed is a unified environment that provides a set of specialized components for easy and detailed simulation of checkpointing and recovery schemes. The testbed allows a designer to mix and match different components either to study the effectiveness of a particular scheme or to freely experiment with hybrid designs before the actual implementation. The testbed also facilitates the evaluation of interdependencies among the various parameters such as communication and application dynamics and their effect on the performance of checkpointing and recovery schemes. The implementation of the testbed as an extension of DEPEND which is an integrated design and fault-injection environment, provides for unique system-level dependability analysis under realistic fault conditions unlike existing simulation tools. We illustrate the versatility of the testbed by using four diverse applications, ranging from the comparison of performances of two checkpointing and recovery schemes to the study of the effect of checkpoint size.

1 Introduction

The advent of object-oriented design (OOD) has led to the proliferation of extensible, reusable and standardized software in computer applications [1]. Computer simulation is one of the prominent areas which took advantage of the success of the object-oriented design [2]. General purpose simulation packages such as CSIM [3] and dependability and reliability analysis tools such as DEPEND [4] are examples of systems that harnessed the strengths of OOD.

Checkpointing and recovery is a growing area of fault tolerance and new schemes [5] are being proposed to suit the complexity and demands of the ever-changing computational and communication models. Evaluation of these schemes through simulation is necessary for their use in real applications. A unified environment for the simulation or implementation of checkpointing and recovery schemes is not currently available. In the few cases where the implementations are presented [6], [7], they are on such diverse platforms that it is difficult to compare their performance in a meaningful way. The effectiveness of a checkpointing and recovery scheme should be studied with realistic fault models within a standard test environment. These issues clearly bring out the need for a comprehensive tool to support the design, development and swift evaluation of rollback and recovery techniques in distributed systems. The Object-oriented Testbed for Evaluation of Checkpointing and recovery (OTEC) that exploits the full potential of OOD is an attempt to fulfill this need. OTEC is composed of a variety of components for error detection, checkpointing, message logging and recovery. Availability of such facilities allows a designer to mix and match different components either to study the effectiveness of a particular scheme or to freely experiment with hybrid designs before actual implementation. OTEC also facilitates the evaluation of interdependencies among the various parameters such as communication and application dynamics and their effect on the performance of checkpointing and recovery schemes.

OTEC saves time by allowing for quick implementation and testing of a scheme. It promotes re-usability; saves hardware, software, time and effort by eliminating several similar implementations by different designers. The uniqueness of this testbed stems from the fact that it is general enough to support a wide variety of schemes but specific to allow customization and detailed simulation of individual schemes.

Section 2 gives the background on the main topic of this paper – checkpointing, message logging and re-

0731-3071/97 $10.00

covery. The objectives of OTEC are listed in Section 3. The principles, design and implementation details of the testbed are discussed in Section 4. The usage of OTEC is illustrated using two schemes – CReP [8] and the optimistic rollback recovery scheme of [9]. The description of the schemes, the details of the experiments, and the results and their significance are provided in Section 5. Section 6 discusses the work in progress and the future plans.

2 Background

Checkpointing and recovery schemes have been developed for database and process control systems as early as the seventies [10], [11]. Today the application domain encompasses a much wider area including memory rejuvenation [12], mobile computing [13] and multi-media. Checkpointing is needed to consistently synchronize different types of multi-media (voice, image and text) data that is transmitted during the processing of video-on-demand and video conferencing applications. When a failure is detected, the system can retransmit from a consistent state with respect to all types of data.

Two major categories of checkpointing in distributed systems are *independent checkpointing* [14], [7] and *consistent checkpointing* [15], [16], [17], [6]. Many variations of these two basic categories are also available. In independent checkpointing, each process periodically saves its state and other relevant information needed for a restart. At the time of recovery, the processes involved spend considerable amount of time coordinating among themselves to decide on a consistent state for recovery. In the case of consistent checkpointing, the processes coordinate or synchronize at the time of checkpointing so that recovery is fast. The desirable features of checkpointing and recovery schemes are: fast recovery from failures, and low overhead during normal operation.

The main problem in independent checkpointing is the *domino effect* [18] which is a cascade of rollback of many dependent processes. This problem can be alleviated by providing a message logging support [19]. *Message logging* is commonly used in combination with checkpointing to minimize the performance overhead of rollback recovery. Message logging is sender-based [20] or receiver-based [8] depending on whether the logging is done at the sender or receiver respectively. If messages are logged on the stable storage as soon as it is available, then the logging is said to be *pessimistic* [21]. This method is commonly used when the message rate is low. Alternatively, it is possible to wait until a set of messages becomes available and then log them all together. This is known as *optimistic* message logging [22]. The information saved at a checkpoint should be sufficient enough to establish a recovery state for the processes involved. Wang and Fuchs [7] implement their scheme on Chare Kernel, while Li et al.'s scheme [6] is implemented on Intel hypercube. A workstation-based platform [19] is available for comparing the various message logging schemes. Even though this work [19] brings out the need for a standard testbed, a hardware-dependent setup such as the one used here restricts its extensibility and portability. Hardware testbeds will require significant modifications to both hardware and software when porting to different architectures. Hardware testbeds also limit the size and the type of the experiments to the available resources. A software testbed is more versatile and portable. OTEC is a fully software-based, portable and extensible testbed.

3 Goals of OTEC

The requirements of the various checkpointing, recovery, message logging and error detection schemes were studied to arrive at the key objectives of OTEC. These objectives are:

1. To provide a testbed for the designers to readily implement and study their checkpointing and recovery schemes. Existence of such a platform encourages the simulation of any new design and the existing ones, and facilitates an objective evaluation of the various attributes of the schemes without incurring much developmental time and effort. Very often, a simulation brings out problems that are not apparent in analytical solutions which generally make simplifying assumptions for tractability.

2. To allow for comparison of different checkpointing and recovery schemes in a common test environment under realistic fault scenarios. No tools based on comprehensive dependability analysis are currently available in existing simulation packages.

3. To enable evaluation of the suitability of a checkpointing scheme for the requirements of an application. For example, the checkpoint size that best suits a particular application can be decided using the testbed.

4. To make it possible to experiment with hybridization. Hybridization is essentially concocting new schemes from compatible components of different designs.

Design decisions in OTEC are primarily based on the above objectives. The design involves (i) simulating a computational system, (ii) support for a variety of error detection, checkpointing, message logging, validation, commit algorithms, and memory (stable storage) requirements, (iii) ability to perform fault injection and dependability analysis, and (iv) providing means for measuring attributes such as time of completion. OTEC is designed as an integrated facility to realize these goals.

4 Design of OTEC

4.1 Design Principles

OTEC is designed using the object-oriented paradigm. Object-oriented design allows for easy integration of existing facilities and packages and extension and modification of the system to satisfy future demands. The design philosophy is to derive or compose the required general purpose facilities (classes) from one or more of the existing simulation packages and to redesign them to suit OTEC's specification. New classes representing checkpointing, error detection and recovery can be easily added to the existing facilities of OTEC. The testbed offers a controlled setup similar to a laboratory associated with scientific experimentation.

4.2 Design Details

In general, concurrent error detection deals with instruction-level operations while recovery deals with both process-level and instruction-level operations. The OTEC simulation package should satisfactorily handle both aspects of system operation. Execution-driven methods [23] are suitable for error detection but not for recovery. Event-driven simulation [24] would suit the recovery aspects but not the error detection. A distribution-driven, process-based simulation is found to satisfy the broader requirements of OTEC. Accordingly, a process-driven simulation tool such as YACSIM [25] or CSIM [3] was our initial choice to implement OTEC. Facilities offered by these simulation packages were studied. In fact, a small scale pilot simulation [26] was carried out using YACSIM which brought out the need for a standardized and realistic fault-injection facility. The general purpose packages such as CSIM or YACSIM do not provide the comprehensive fault-injection capabilities required of OTEC. This led to the examination of the specialized dependability evaluation packages with fault-injection capabilities such as DEPEND [4] to determine if they can provide the basis for OTEC.

DEPEND is a dependability and reliability evaluation tool that can administer and support fault injection at the component level. The object-oriented nature of DEPEND readily satisfied the extensibility requirement of OTEC. Moreover, DEPEND is based on CSIM which was one of the initial choices for providing the basic simulation facility. DEPEND provides CSIM-style process-driven simulation using a library of classes to model the behavior of a system. DEPEND easily yields to other applications, as proven by SIMPAR, an extension of DEPEND implemented at the University of Erlangen-Nurenberg, Germany [27]. SIMPAR is specifically meant for the evaluation of multicomputer systems and massively parallel systems by performing fault-injection. This work is complementary to OTEC and there are plans to integrate SIMPAR into future versions of OTEC. Availability was another factor in the choice of DEPEND to provide the basic infrastructure for the testbed.

The overall design of OTEC exemplifies the object-oriented design by reusing some of the facilities provided by DEPEND and CSIM. Fig. 1 shows the dependency structure of the various software packages involved in the design of OTEC. An application using OTEC has the choice of using facilities provided by any of the packages in the integrated structure. In Fig. 1 interfaces are numbered 1 through 7. The interfaces numbered 1 through 4 indicate that an user application can use the facilities from OTEC, SIMPAR, DEPEND and CSIM respectively. The interfaces 5, 6, and 7 are indicative of the facilities OTEC can inherit.

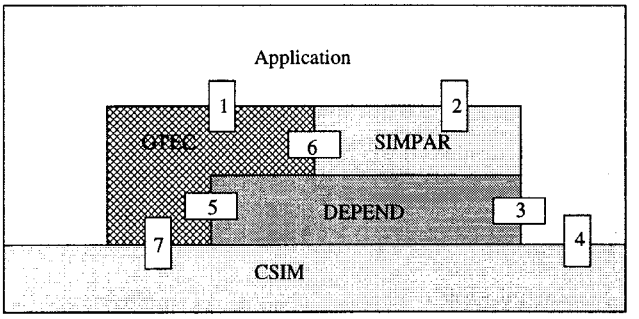

Figure 1: Structural Composition of OTEC

An important capability of interest for OTEC in DEPEND is the set of fault injection-related classes. OTEC inherits and enhances processor facility provided by DEPEND and uses the injection facilities without any modification. What distinguishes OTEC from DEPEND is the provision of a set of classes related to error detection, checkpointing and recovery, message logging and statistical analysis. When using an existing class, OTEC either derives from it or composes it into an OTEC class. In general, complex classes are inherited from and simple classes are

composed into. Virtual functions are used to allow re-definition of methods for fine-tuning. Any class involving memory management is defined as a container class to allow re-usability of standard libraries and to avoid memory leaks. OTEC simplifies the development of checkpointing and recovery systems by providing built-in classes and methods so that various schemes can be evaluated efficiently.

4.3 OTEC Classes

The central component of OTEC is the processing element or the node of a distributed or multicomputer system represented by *FT_node*. The primary purpose of an object of *FT_node* class is to implement a fault-tolerant processor. *FT_node* inherits from *FT_server2* of DEPEND. *FT_node* class currently encompasses both computation and communication aspects of a node. Eventually these two will be broken down into independent classes for modularity and for simulating concurrency between the two. The functions in *FT_node* are broadly classified as below.

1. Execution-related: This comprises the methods for simulating the execution of a test program. The execution resolution is at the machine instruction level. Such finer details as end-of-fetch-cycle are implemented.

2. Communication-related: These simulate operations such as send and receive and the parameters associated with communication.

3. Application-related: These are methods that are specific to the checkpointing and recovery application being simulated. For example, there could be a *commit* algorithm that is specially needed for committing messages in an application.

The basic *FT_node* is augmented with the attributes and methods needed by any application that is using OTEC. OTEC defines other base-classes besides the *FT_node* mentioned above. These OTEC-specific base-classes are: *FT_testpgm*, *FT_msglog*, *FT_watchdog*, *FT_analyst*, *FT_stablestorage*, *FT_commit* and *FT_valid*. *FT_testpgm* represents the distributed application program to be executed by the simulated system. It allows for generation of the program according to specifications, reading in a user program to represent a job, and pre-processing facilities to insert checkpoints and other locations of interest to the user. User can also specify the message rate. The messages are statically inserted into the *FT_testpgm* according to the user specification. Alternatively, the *FT_testpgm* could be an actual distributed application.

A message logging facility is provided by *FT_msglog*. Both sender and receiver logs are available. This class provides a very detailed structure for logging a message. The details that can be logged are receiver sequence number, sender sequence number, timestamp, sender, validity information, and checkpointing information. Both volatile storage and permanent storage are possible. Permanent storage is implemented using yet another class *FT_stablestorage*. This class offers various methods for state saving and checkpointing, and message logging. The application-specific facility *FT_watchdog* provides the watchdog monitoring capabilities and signature generation and comparison functionalities. *FT_analyst* keeps track of the measurement of a user-specified attribute such as completion time, monitors its variance and convergence at a user-specified confidence level, and terminates the simulation accordingly.

The class hierarchy of OTEC is shown in Fig. 2. The reused components of DEPEND are shown by shaded boxes. *FT_node* class includes the basic processing capabilities provided by the *FT_server2* and the additional attributes and methods to implement various OTEC-specific functions. Fault-injection functionality of OTEC is provided by the *FT_injector* class of DEPEND. *FT_injector* offers choices in the types of fault: transient or permanent, duration of the fault, various workload-dependent injection methods, various fault distributions and allows for a user-defined function to define the component where the fault should be injected into. Communication related facilities are derived from *FT_link2* of DEPEND. *FT_analyst* enhances the basic facilities of its base class *DEPEND::Table* by adding convergence monitoring capabilities.

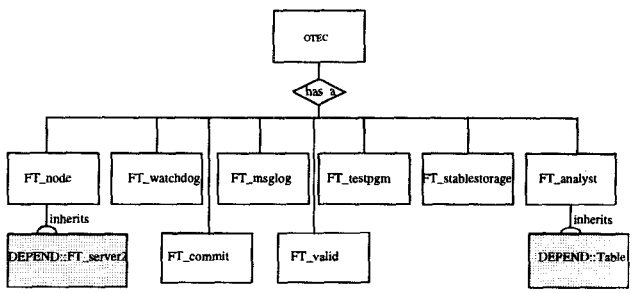

Figure 2: Components of OTEC

5 Experiments with OTEC

Four different cases related to checkpointing and recovery are chosen to illustrate the versatility of OTEC. The first case is a simulation and comparison of two di-

verse checkpointing and recovery schemes. This study shows (i) the degree of reusability of OTEC components and (ii) the ease of setting up experiments. The second case illustrates the behavior of the same two schemes under correlated faults. The third study illustrates the *mix and match* capability of OTEC. The mix and match capability lets a user unplug a certain component of an application and replace it with a compatible component from the testbed. In this experiment a concurrent error detection tool such as a watchdog processor [28] is unplugged and replaced by an ordinary timeout error detection mechanism. The fourth case deals with the effect of checkpoint size on the performance of checkpointing and recovery. For all the experiments a four node configuration is used.

5.1 Benchmarks for the Experiments

Only transient faults are considered in this work, although it is possible to study the behavior under permanent faults and workload-related faults. The magnitude of the error detection latency and the recovery time in our experiments are at the instruction cycle-level granularity. If error detection is done by a watchdog processor, the detection latency is typically in the order of a few instructions. The programs used for simulation are of few kilobytes size. In order to allow for realistic hardware error detection such as signature analysis-based schemes, real code was executed on the simulated machine. Well known benchmarks with realistic instruction mix were examined. The test programs executed were selected from SPEC'92. SPEC'92's *compress.c* yielded five reasonably long traces each with varying interval sizes. Four of these were run on four distributed processes in our experiments. Messages were artificially introduced in the benchmark programs to simulate the interprocess communication at the various rates. The message rates used range from 0 to 10 messages per 100 instructions and the error rates range from 0 to 1 per 1000 instructions. The checkpoint sizes are varied from 1.25% to 20% of the test program size.

5.2 Simulation and Comparison

Two checkpointing and recovery schemes have been chosen for a case study on OTEC: the Comprehensive Recovery Protocol (CReP) [8] and the Optimistic Rollback Recovery (ORR) [9]. These schemes are meant for distributed systems. Both employ optimistic message logging. CReP uses receiver-based message logging while our implementation of ORR uses sender-based logging. The class hierarchies of CReP and ORR are shown in Fig. 3, and Fig. 4 respectively and are described in this section. The classes of OTEC that are reused are shown using shaded boxes in the figures. The degree of reusability of OTEC is apparent from these figures.

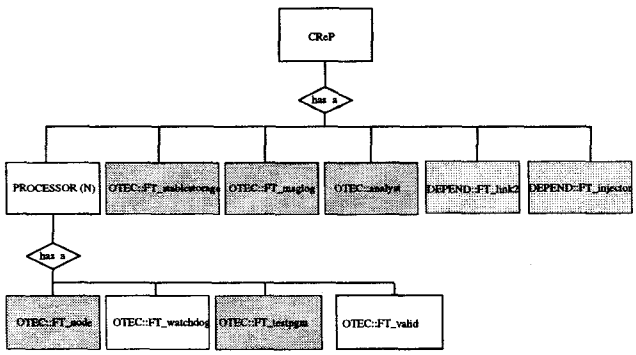

Figure 3: Class Hierarchy of CReP

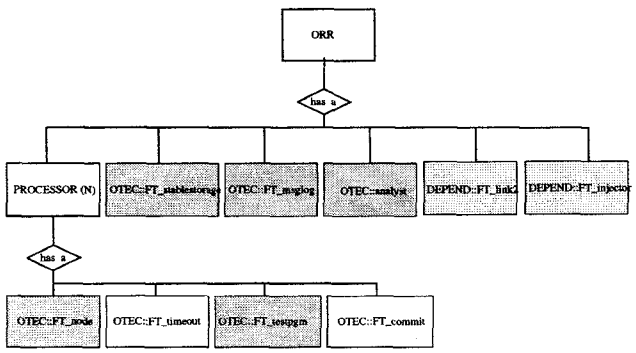

Figure 4: Class Hierarchy of ORR

5.2.1 Description of CReP

CReP employs a hardware-assisted error detection mechanism to achieve fast detection and recovery. In most systems, the error detection is done only at the checkpoints. As a result, after an error occurs and before it is detected, corrupted messages may get transmitted to other processes in the system. All these processes that have received messages between the checkpoints have to be rolled back even though some of the messages may have been transmitted before the error has actually occurred. A simple solution to avoid unnecessary rollbacks is to have more frequent checkpoints. But this proposition is expensive. CReP solves this problem by introducing hardware-assisted error detection points (d_p) which incur very little overhead. The locations along the execution path where a signature generated by instruction code forms an m-out-of-n code are tagged. The sequence of one or more tagged locations form a detection interval. A detection point marks the end of a detection interval and

is tagged during the compilation and the signatures are verified during execution. At any detection point d_p if no error is detected then the detection interval $\{ d_{p-2}, d_{p-1} \}$ is declared valid and all the messages sent in this interval are validated as good. Once the message is validated, the receiver of the message need not be rolled back if an error occurs in the sender of the message. The validation component of CReP is represented by *FT_valid* in the hierarchy chart. Rollback messages are sent only to those processes with outstanding validation messages from the process in error. CReP reduces the recovery time by minimizing the excessive rollback of other processes in case of an error in a given process. More detailed description of CReP can be found in [26].

For CReP, the objects needed for simulation are processing nodes, watchdog processors, a message log, an analyst, a test program, a stable storage, a validation facility, a communication facility, and the fault-injection facility. Among these all except the processor object can be directly instantiated from OTEC's facilities *FT_watchdog*, *FT_msglog*, *FT_analyst*, *FT_stablestorage*, *FT_testpgm*, *FT_valid*, *FT_link2* and *FT_injector* respectively. The class needed for instantiating the processor objects is derived from the *FT_node*.

5.2.2 Description of ORR

Optimistic Rollback Recovery (ORR) is based on a *commit* algorithm to determine a maximal state of the distributed system beyond which the system need not rollback in case of an error. When a process receives a message a new state interval is started for that process. Before committing any message to the outside world, the state interval is committed by calling the commit algorithm. Committing involves sending requests to the dependent processes for their dependency information. These requests may in turn result in one or more dependent processes checkpointing and establishing stable states. Once a state of a process is committed, that process will not rollback beyond this committed point. ORR uses an elaborate commit algorithm, whereas CReP uses a validation mechanism which exploits the error detection characteristics. A major difference between CReP and ORR is in the way recovery is handled. ORR sends rollback message to all processes. It is up to the individual processes to determine if they need to rollback. This is done by using the dependency vectors maintained at each process. CReP propagates error messages only to those processes that need to be rolled back. Detailed description of the ORR scheme can be found in [9].

Like CReP, most objects needed for ORR simulation can be directly instantiated from the OTEC facilities.

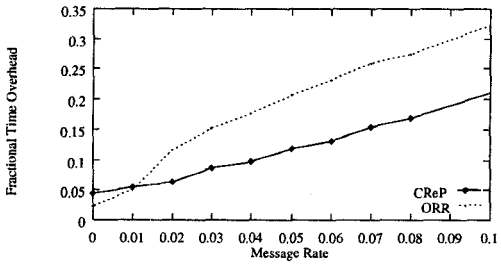

Figure 5: Process Completion Time With Message Rate – Low Error Rate (0.0005)

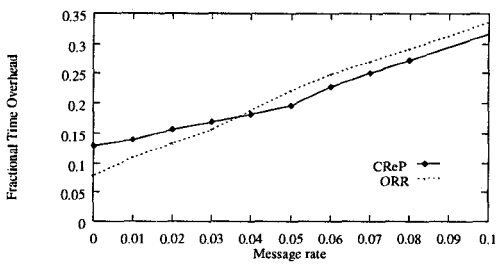

Figure 6: Process Completion Time With Message Rate – High Error Rate (0.001)

5.2.3 The Results

Applications representing CReP and ORR are designed and implemented using the OTEC objects. An OTEC application comprises:

1. def.h : Application-specific class definitions.
2. def.cc : Implementation of the classes specified in def.h.
3. app.cc : The main control function, data, and other user-defined functions.

The three components are compiled and linked with OTEC to prepare the executable. The executable is run with appropriate test data.

Process completion times with respect to error rates and message rates are monitored for CReP and ORR. Fig. 5 and Fig. 6 show their performance at low and high error rates. The Y-axis in the figures represents the percentage increase in the time of completion of a process with respect to its time of completion under error-free condition. For low error rate

(0.0005), ORR has a slight advantage over CReP at lower message rates. But at higher message rates (0.01 and higher), CReP performs consistently better than ORR. For high error rate (0.001), performance of CReP and ORR are almost the same with CReP having a slight advantage.

Consider the process completion time versus error rate at low (0.01) and high (0.1) message rates shown in Fig. 7 and Fig. 8 respectively. With low message rate, the performance of ORR is slightly better than CReP. At the high message rate, clearly CReP has a better performance as seen in Fig. 8. But on closer examination, we realize that ORR's overhead, though much higher than CReP's, rises very slowly for the entire range of error rate. This can be explained as follows: In CReP the error detection latency is very small [8]. Thus the rollback distance following the detection of errors is small. However, detection of an error immediately initiates a recovery. In ORR error detection is not as immediate as CReP. The errors occurring between two checkpoints are detected at the later checkpoint. So, multiple errors between checkpoints will result in only one rollback in ORR whereas in CReP they may result in more than one rollback. As the error rate rises the number of errors between checkpoints may increase resulting in more rollbacks in CReP, but still a single rollback is needed for ORR. This explains the slow rise in the overhead with ORR when the error rate is increased. Such a phenomenon has been made apparent by the OTEC experiment.

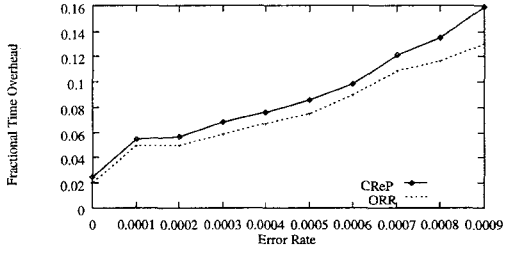

Figure 7: Process Completion Time With Error Rate – Low Message Rate (0.01)

5.3 Simulation with Correlated Faults

This study deals with correlated faults. In this environment, faults occur at the same time in more than one component of the system. Though this phenomenon is not very uncommon, no published work on checkpointing and recovery deals with this. Such experiment can be readily performed using OTEC. OTEC uses the correlated fault injection facility of DEPEND to carry out this unique experiment. The performance of CReP and ORR with correlated faults

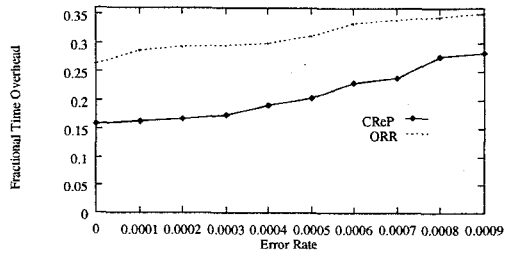

Figure 8: Process Completion Time With Error Rate – High Message Rate (0.1)

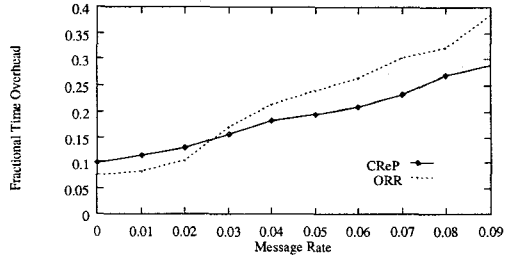

Figure 9: Process Completion Time With Message rate – Percent Correlation: 25

in two components and two correlation factors of 25% and 50% and at error rate 0.0005 are shown in Fig. 9 and Fig. 10 respectively. These values of correlation factor were chosen since smaller values do not produce measurable quantities at the granularity and the time frame of operation of this experiment. Both CReP and ORR are able to handle correlated errors. An observation from the graphs is that correlated errors do not affect CReP performance significantly, whereas it does influence ORR's. This strength of CReP is attributable to the techniques used for error detection and message validation.

5.4 Evaluation of Hardware-Assist in Checkpointing

The purpose of this experiment is to illustrate the mix and match capability of OTEC. The effectiveness of the watchdog processor component of CReP is studied by unplugging the *FT_watchdog* and *FT_valid* objects from CReP implementation and substituting it with a simple user-supplied *FT_ConsistChk* object (called CCK scheme). Fig. 11 shows the performances of CReP and the scheme that uses simple consistency checking at a checkpoint instead of using the hardware-assist. CReP consistently outperforms CCK by 5 – 15%. This experiment brings out the ease with which a simulation of a new scheme can be put together with the support provided by OTEC.

Fig. 12 shows another novel use for the mix and

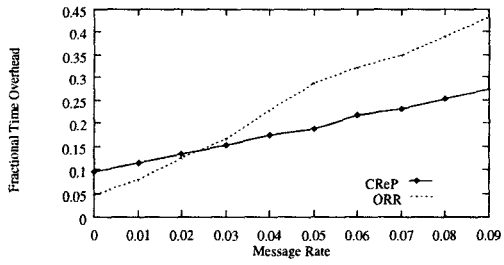

Figure 10: Process Completion Time With Message rate – Percent Correlation: 50

match capability of OTEC. Here we have an application that has extensive message logging. The message logging facility of OTEC may not be sufficient. So *OTEC::FT_msglog* is augmented by defining a new specialized class *USER::recvlog*. *OTEC::FT_timeout* is used for detecting faults and the *PROCESSOR* class has a user-defined consistent recovery class *USER::recovery*. All these components can be linked to the existing facilities of OTEC to setup a simulation for a new hybrid scheme. The effect of the various design decisions such as these can be evaluated before proceeding with the actual implementation.

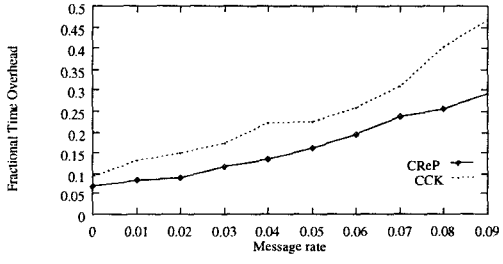

Figure 11: Performance Comparison of CReP and CCK

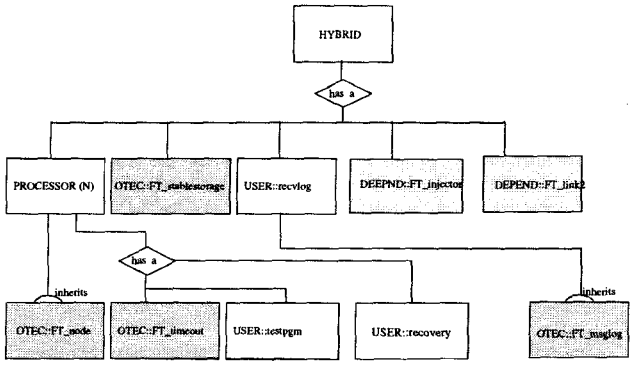

Figure 12: An Example For Hybridization

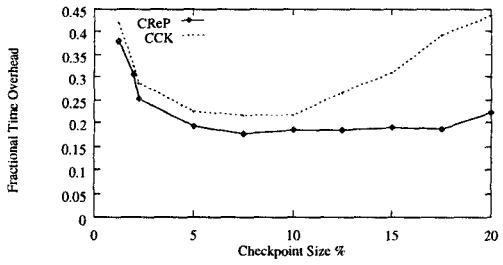

Figure 13: Process Completion Time With Checkpoint Size

5.5 Effect of Checkpoint Size

The checkpoint sizes are varied from 1.25% to 20% of the data size and the effect on the performance was studied for CReP and for the simple checkpointing scheme CCK. Fig. 13 shows the time of completion variation with checkpoint size for a high message rate (0.1) and nominal error rate (0.0005). The overhead of checkpointing comes from the checkpointing time as well as the rollback time. Checkpoint time is same for both CReP and CCK, but the error detection latency is independent of the checkpoint size in CReP because of the hardware-assist. This helps in keeping the overhead almost flat at higher checkpoint sizes. A significant result of this experiment is the observation that the validation scheme of CReP does mitigate the effect of large checkpoint sizes.

6 Discussion

An object-oriented testbed for simulating checkpointing and recovery schemes has been designed, implemented and tested. A concurrent error detection and recovery scheme, CReP was used as a case to study some of the features of the testbed. CReP's performance was compared with that of another scheme, ORR. The performance of both schemes under correlated errors was studied. Effectiveness of the hardware based error detection component of CReP was analyzed by substituting it by a simple timeout mechanism. The effect of checkpoint size was also studied. Our experiments show that communication characteristics and the operating environment strongly influence the choice of a checkpointing and recovery scheme for a distributed environment. For example, results indicate that CReP, the hardware-assisted recovery scheme, needs some modification to make it applicable to environments subject to high error rates.

We have shown how a hybrid system can be designed and simulated out of the facilities provided by OTEC. The checkpointing schemes can be extended to multicomputer systems by using facilities offered

by SIMPAR which has already been integrated into the OTEC environment. With this integrated testbed, checkpointing and recovery schemes for various computational models can be evaluated.

Our future plan includes generalization of the testbed and investigating further the unique hybridization capability offered by OTEC. The experiments reported in this paper can easily scale up to larger distributed systems. The size of the simulated distributed system is determined by the limits imposed by CSIM. The evaluation of scalability and further parameterization of the experiments to include optimal checkpoint interval [29] will be part of our future work.

The scope of OTEC is not limited to just checkpointing and recovery. We have demonstrated a methodology for simulating an application on OTEC. Using this methodology, OTEC can be extended to include standard testing facilities for other techniques of fault-tolerance such as triple modular redundancy.

Acknowledgement

This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract DABT63-94-C-0045. The content of this paper does not necessarily reflect the position or policy of these agencies, and no endorsement should be inferred.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley, 1995.

[2] J. D. Ichbiah and S. P. Morse, "General concepts of the Simula 67 programming language," *Annual Review of Automatic Programming*, vol. 7, no. 1, pp. 65–93, 1972.

[3] H. Schewetman, "CSIM: A C-based process-oriented simulation language," in *Proceedings Winter Simulation Conference*, 1986.

[4] K. K. Goswami, R. K. Iyer, and L. Young, "Depend: A simulation-based environment for system level dependability analysis," *IEEE Transactions on Computers*, vol. 46, pp. 60–74, January 1997.

[5] M. Peercy and P. Banerjee, "Software schemes of reconfiguration and recovery in distributed memory multicomputers using actor model," *Proc. 25th International Symposium on Fault Tolerant Computing (FTCS-25)*, pp. 479–488, 1995.

[6] K. Li, F. Naughton, and J. Plank, "An efficient checkpointing method for multicomputers with wormhole routing," *International Journal of Parallel Programming*, vol. 20, no. 3, pp. 159–180, 1991.

[7] Y. Wang and W. Fuchs, "Scheduling messages for reducing rollback propagation," *Proceedings of IEEE Fault Tolerant Computing Symposium*, pp. 204–211, July 1992.

[8] B. Ramamurthy and S. Upadhyaya, "Hardware-assisted fast recovery in distributed systems," in *Dependable Computing for Critical Applications* (R. K. Iyer, ed.), IEEE Computer Society, 1997, (to appear).

[9] D. Johnson, "Efficient transparent optimistic rollback recovery for distributed application programs," *Proc. 12th Symposium on Reliable Distributed Systems*, pp. 86–95, 1993.

[10] K. Chandy, J. Browne, C. Dissly, and W. Uhrig, "Analytic models for rollback strategies in database systems," *IEEE Transaction on Software Engineering*, vol. SE-1, pp. 100–110, January 1976.

[11] K. Chandy and C. Ramamoorthy, "Rollback and recovery strategies for computer programs," *IEEE Transactions on Computers*, vol. C-21, pp. 546–556, June 1972.

[12] Y. M. Wang, Y. Huang, K. P. Vo, P. Y. Chung, and C. Kintala, "Checkpointing and its applications," *Proc. 25th International Symposium on Fault Tolerant Computing (FTCS-25)*, pp. 22–31, 1995.

[13] R. Prakash and M. Singhal, "Low-cost checkpointing and failure recovery in mobile computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 1035–1048, October 1996.

[14] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback for recovery – An optimistic approach," in *Proceedings of the Seventh Symposium on Reliable Distributed Systems*, pp. 3–12, October 1988.

[15] F. Cristian and F. Jahanian, "A time stamp-based checkpointing protocol for long-lived distributed computations," *Proc. 10th Symposium on Reliable Distributed Systems*, pp. 12–20, 1991.

[16] E. Elnozahy, D. Johnson, and W. Zwaenepoel, "The performance of consistent checkpointing," *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pp. 39–47, October 1992.

[17] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Transactions on Software Engineering*, vol. 13, pp. 23–31, Jan 1987.

[18] B. Randell, "System structure for software fault tolerance," *IEEE Transactions on Software Engineering*, vol. SE-1, pp. 220–230, June 1975.

[19] E. Elnozahy and W. Zwaenepoel, "On use and implementation of message logging," *Digest of Papers, 24th International Symposium on Fault Tolerant Computing (FTCS)*, pp. 298–307, 1994.

[20] D. Johnson and W. Zwaenepoel, "Sender-based message logging," *Proc. 17th Annual Fault Tolerant Computing Symposium*, pp. 14–19, June, 1987.

[21] A. Borg, J. Baumbach, and S. Glazer, "A message system supporting fault tolerance," *Proceedings of 9th ACM Symposium on Operating System Principles*, pp. 90–99, October 1983.

[22] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems," *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pp. 147–154, October 1992.

[23] R. Covington, S. Dwarakadas, J. Jump, and S. Madala, "The efficient simulation of parallel computer systems," *International Journal in Computer Simulation*, vol. 1, pp. 31–58, 1991.

[24] P. Fishwick, *Simpack: C-based simulation seed tool package.* Gainsville, FL: Department of CIS, University of Florida, 1990.

[25] J. R. Jump, *YACSIM Reference Manual.* ECE Dept, Rice University, 1993.

[26] B. Ramamurthy and S. Upadhyaya, "Watchdog processor-assisted fast recovery in distributed systems," *Fifth International Working Conference on Dependable Computing for Critical Applications*, pp. 125–134, September 1995.

[27] A. Hein and K. Bannsch, "Simpar - A simulation environment for performance and dependability analysis of user-defined fault-tolerant parallel systems," tech. rep., University of Erlangen-Nurnberg, Germany, 1995.

[28] S. Upadhyaya and B. Ramamurthy, "Concurrent process monitoring with no reference signatures," *IEEE Transactions on Computers*, vol. 43, pp. 475–480, April 1994.

[29] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, vol. 17, pp. 530–531, September 1974.