A layered framework for decoupling application and checkpointing protocol

Anant Pushkar 11CS10003

Harshit Gupta11CS30014

Manvendra Singh Tomar 11CS30018

Honey Joshi 11CS30046

April 14, 2015

Chapter 1

Introduction

Checkpointing and rollback-recovery are common techniques that allow processes to continue executing in spite of failures. The failures being talked about are transient problems such as hardware errors and transaction aborts, i.e., those that occur only for a short period of time and are unlikely to recur when a process restarts. With this scheme, a process takes a checkpoint regularly from time to time by saving its state on stable storage. When a failure occurs, the process rolls back to its most recent checkpoint, assumes the state saved in that checkpoint, and resumes execution.

Checkpointing and recovery for a single processor system has been studied in detail in [7], [2], [5], [8], etc. However, when we talk of distributed systems, the state of a process is dependent on one another because of inter-process communication. This problem has also been studied in a lot of depth in works like [4], [6], [3], [1], etc.

However, all current approaches require the application developer to code the checkpointing and recovery scheme in the application. This makes his task more complex than it should logically have been. In this report, we propose a framework which alleviates the application developer from the responsibility of implementing the checkpointing and recovering protocol himself. The user shall be given a set of generic function calls using which he would ensure that the application's state is checkpointed and recovery measures are taken. This framework is an extension of the TCP/IP layered architecture.

As an implementation of the framework, we have developed an application using the framework's generic calls for checkpointing and recovery. For the underlying checkpointing protocol, an asynchronous and coordinating protocol [] and an synchronous, uncoordinated protocol has been implemented.

The rest of the report has been structured as follows. Chapter 2 discusses the research works related to this report. Chapter 3 describes the proposed framework in detail. Chapter 4 throws light on the implementation of the proposed framework. Finally, chapter 5 concludes the report and discusses scope of future work.

Chapter 2

Related Work

Several efforts have been directed in the field of checkpointing and recovery in distributed systems.

In [4] , the authors propose an algorithm that takes consistent checkpoints of the entire system and restores the entire system to a consistent state. It can tolerate failures that occur during their executions. Furthermore, when a process takes a checkpoint, a minimal number of additional processes are forced to take checkpoints. Similarly, when a process rolls back and restarts after a failure, a minimal number of additional processes are forced to roll back with it. The proposed algorithm requires each process to keep at most 2 checkpoints in stable storage - thereby making the storage requirement minimal.

In their work [1], the authors propose a checkpoint algorithm which stands on the shoulders of research on concurrency control, commit, and site recovery algorithms in transaction processing. In the proposed approach a number of checkpointing processes, rollback processes, and computations on operational processes can proceed concurrently while tolerating the failure of an arbitrary number of processes. Each process takes checkpoints independently. During recovery after a failure, a process invokes a two-phase rollback algorithm. It collects information about relevant message exchanges in the system in the first phase and uses it in the second phase to determine both the set of processes that must roll back and the set of checkpoints up to which rollback must occur. Concurrent rollbacks are completed in the order of the priorities of the recovering processes. The proposed solution is optimistic in the sense that it does well if failures are infrequent by minimizing overhead during normal processing.

The authors of [3] prove that there is always a unique maximum recoverable system state, regardless of the message logging protocol used for providing fault tolerance. The authors presents a general model for reasoning about recovery in such a system and, based on this model, an efficient algorithm for determining the maximum recoverable system state at any time. The proposed algorithm unifies existing approaches to fault tolerance based on message logging and checkpointing, and improves on existing methods for optimistic recovery in distributed systems.

In [6], a new technique supporting application-independent transparent recovery from processor failures in distributed systems - Optimistic Recovery has been proposed. In optimistic recovery communication, computation and checkpointing proceed asynchronously. Synchronization is replaced by causal depen-

dency tracking, which enables a posteriori reconstruction of a consistent distributed system state following a failure using process rollback and message replay. Because there is no synchronization among computation, communication, and checkpointing, optimistic recovery can tolerate the failure of an arbitrary number of processors and yields better throughput and response time than other general recovery techniques whenever failures are infrequent.

Chapter 3

Proposed Framework

The proposed framework is based on a layered architecture, as shown in figure 3.1. The individual layers of the framework are described as follows.

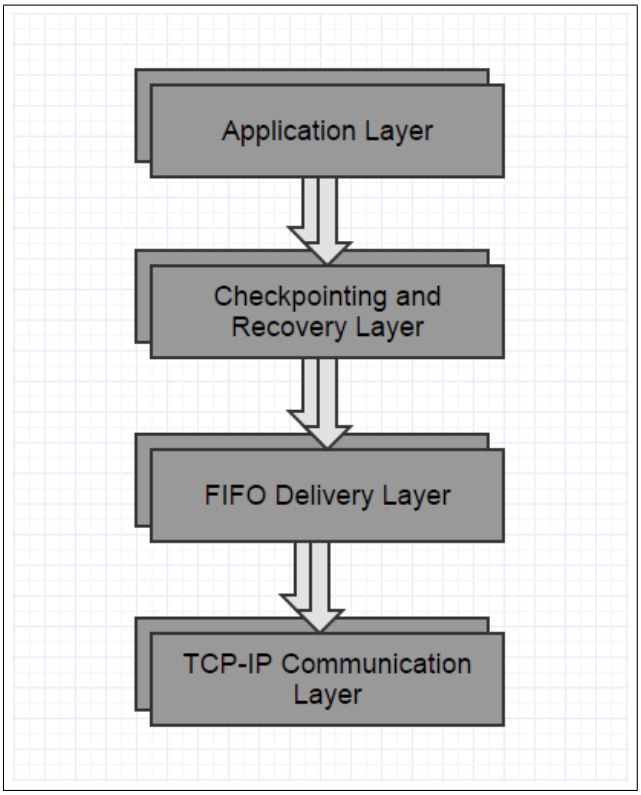

Figure 3.1: Architecture of broker

3.1 Application Layer

The application layer contains the code of the application which will be utilizing the underlying checkpointing and recovery protocol. The application will need to make a set of generic function calls to the checkpointing layer below it in order to avail the services provided by it.

The application layer needs to be aware of the following things :

- The **state of the application** that has to be checkpointed
- **Receive handler** to be called on receiving a message from another process

3.2 Checkpointing and Recovery Layer

This layer contains the code which implements the checkpointing and recovery protocol to be used by the application layer by making generic function calls. This layer exposes the following function calls to the application layer :

- take_tentative_checkpoint() : This function call takes a tentative checkpoint out of the state of the process.
- take_permanent_checkpoint() : This function call makes the last tentative checkpoint of the calling process permanent, and thus available for recovery.
- undo_tentative_checkpoint() : This function call discards the tentative checkpoint last taken.
- get_state() : An abstract function that returns the state of the application process calling it.

3.3 FIFO Delivery Layer

Some checkpointing and recovery protocols require the messages to be delivered in FIFO order. To be able to make this assumption, the proposed framework also has layer which implements FIFO and gives the upper layers an abstraction of FIFO links.

Chapter 4

Implementation

The implementation will be done using Python. Each process will be run on a separate machine. Hence the IP address of each machine can be used as a unique ID for the process running on it. TCP sockets will be used for reliable communication between the nodes. For implementing FIFO delivery of messages, an additional layer above the TCP layer shall be implemented which will make sure that delivery of messages to the processes.

4.1 Assumptions taken

We assume the following points about the underlying network over which this framework will be built. These assumptions are specifically necessary for the checkpointing and recovery algorithms implemented to work and not for the banking system.

- **Reliable communication**. This can be assumed as the implementation will be done over TCP, which is reliable.
- **Completely connected network**. Since each process knows the ID of every other process, it can send a message directly to it.
- **Synchronous system**. Since the processes can crash, an asynchronous system will never be able to detect whether a process has crashed or it is just slow.
- **First-in-first-out delivery of messages required**.
- **Only crash faults** occur. A crash shall be simulated by killing a process.

4.2 Application

We propose to implement a fault tolerant banking system. Each process in this system represents a bank account. Each process holds the following data items :

- amount of money in the account
- it's own ID

- IDs of all the other accounts (assuming no new account joins in between dynamically)

4.2.1 Messages exchanged between processes

Monetary transfers between these accounts is done by passing a message from the giver account to the receiver account. This message is named $SEND$ and carries the ID of the sender and the amount of money to be transferred.

4.2.2 Features

The application will offer a very plain user interface by means of a terminal for each account. This interface would show the amount held by the account and will provide an option of transferring a sum of money to one of the other accounts in the system.

4.2.3 Points to note

The system shall begin with each account having a pre-decided initial sum of money with itself.

The system will be said to be consistent at any time if the sum of amount on all the accounts at that time is equal to the initial sum of amounts.

4.3 Checkpointing and recovery protocols

For implementing the checkpointing and recovery layer, we used two well-known protocols - which are described as follows:

1. Checkpointing and recovery protocol proposed by Koo and Toueg. The protocol is asynchronous and co-ordinated, two-phase checkpointing protocol which takes tentative checkpoint and on decision of the checkpointing cohort decides whether to make it permanent or not. During checkpointing it makes sure that only minimal number of nodes does checkpointing so that there is no orphan message and checkpoint remain consistent. In order to do so each process maintains a cohort of nodes to coordinate with to take tentative checkpoint. When other process receives a message to take tentative checkpoint it checks for whether there is any orphan message as well as will of other nodes in cohort. If each node in cohort is ready to take checkpoint then initiator propagates the decision to take checkpoint else to undo the taken tentative checkpoint.

2. A synchronous and uncoordinated checkpointing protocol. It assumes lossless channel. Whenever the system needs to take checkpoint, all nodes freezes its system messages for a time period large enough to cover transmission delay and the clock-skew and takes the checkpoint. Whenever some node fails, it recovers from that checkpoint and replay received messages from its messages log.

Chapter 5

Conclusion and Scope of Future Work

In this report we have proposed a framework which alleviates the application developer of the responsibility of implementing the checkpointing and recovery protocol. The framework uses a layered approach and hence is logically simple.

Framework could be possibly extended to take a checkpoint whenever the system is about to fail. An external program may continuously monitor the state of the system and compute whether system is about to fail or not and on that basis, the checkpointing protocol will be triggered.

Bibliography

[1] Bharat Bhargava and S-R Lian. Independent checkpointing and concurrent rollback for recovery in distributed systems-an optimistic approach. In *Reliable Distributed Systems, 1988. Proceedings., Seventh Symposium on*, pages 3–12. IEEE, 1988.

[2] A Brock. An analysis of checkpointing. *ICL Technical Journal*, 1(3):211–228, 1979.

[3] David B Johnson and Willy Zwaenepoel. Recovery in distributed systems using asynchronous message logging and checkpointing. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 171–181. ACM, 1988.

[4] Richard Koo and Sam Toueg. Checkpointing and rollback-recovery for distributed systems. *Software Engineering, IEEE Transactions on*, (1):23–31, 1987.

[5] Pierre L'Ecuyer and Jacques Malenfant. Computing optimal checkpointing strategies for rollback and recovery systems. *Computers, IEEE Transactions on*, 37(4):491–496, 1988.

[6] Rob Strom and Shaula Yemini. Optimistic recovery in distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(3):204–226, 1985.

[7] Asser N Tantawi and Manfred Ruschitzka. Performance analysis of checkpointing strategies. *ACM Transactions on Computer Systems (TOCS)*, 2(2):123–144, 1984.

[8] Sam Toueg and Özalp Babaoglu. On the optimum checkpoint selection problem. *SIAM Journal on Computing*, 13(3):630–649, 1984.