INTERACTIVE COMPUTATIONAL STEERING: CONSERVATIVE VS OPTIMISTIC STEERING APPROACHES

by

ARUMUGARAJA SELVARAJ

(Under the direction of Eileen Kraemer)

ABSTRACT

Interactive Computational Steering is the online, interactive allocation and adjustment of system resources and application parameters. Causal consistency is an important feature of interactive steering of distributed computations, as it is often required to maintain the correctness of the computation. However, due to the asynchronous nature of distributed computations, it is difficult to coordinate steering changes across processes to guarantee that the changes are applied consistently at all processes.

Two general approaches exist for achieving interactive computational steering: conservative steering and optimistic steering. In this thesis, we present algorithms for the conservative steering approach. We also compare the performance of the conservative steering approach with that of the optimistic steering approach with regard to perturbation and lag.

INDEX WORDS: Consistent steering, Conservative steering, Optimistic steering, Process synchronization, Process states, Checkpoint, Transaction, Consistency, Message logging, Rollback.

INTERACTIVE COMPUTATIONAL STEERING: CONSERVATIVE VS OPTIMISTIC STEERING APPROACHES

by

ARUMUGARAJA SELVARAJ

B.E., Anna University, India, 2000

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2002

INTERACTIVE COMPUTATIONAL STEERING:
CONSERVATIVE VS OPTIMISTIC STEERING APPROACHES

by

ARUMUGARAJA SELVARAJ

Approved:

Major Professor: Eileen Kraemer

Committee: Thiab Taha
Maria Hybinette

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2002

DEDICATION

To my father Dr.A.Shanmuga SundaraRaj

ACKNOWLEDGEMENTS

I would like to thank Dr.Eileen Kraemer for her continued support and for her guidance throughout my thesis. I would like to thank Dr. Hybinette and Dr.Taha for consenting to be in my committee and for their valuable suggestions. I would like to thank my parents, brother, sister and my family for their continued support and encouragement for the past several years, which has helped me to realize my dream of achieving a Master's degree. I also would like to thank Gayathri and all my friends for providing me the support and the encouragement to keep me charged up whenever I felt let down.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Interactive Computational Steering is the online, interactive allocation and adjustment of system resources and application parameters. Causal consistency is an important feature of interactive steering of distributed computations, as it is often required to maintain the correctness of the computation. However, due to the asynchronous nature of distributed computations, it is difficult to coordinate steering changes across processes to guarantee that the changes are applied consistently at all processes.

Two general approaches exist for achieving interactive computational steering: **conservative steering** and **optimistic steering**. Conservative steering avoids all possible inconsistent steering by strictly adhering to the causality constraint. This typically involves blocking the computation before a consensus decision is made and then the steering changes are applied. Optimistic steering assumes that the next steerable points are consistent, and invokes the steering change at the next steerable point at each involved process without concern for or knowledge of the state of any other process. This eliminates the need for blocking steered processes. However the optimistic steering approach must be able to detect any inconsistent steering transaction and provide a check pointing/rollback mechanism to restore the computation to a correct state.

In this thesis, we present algorithms for the conservative steering approach and compare the performance of the conservative steering approach with that of the optimistic steering approach with regard to perturbation and lag.

1.1 Interactive Computational Steering

Interactive steering of computations permits users to monitor a program's execution and to adjust both application parameters and resource allocation in an online fashion. Interactive computational steering allows users to interpret what is happening to the data during the overall computation and modify application parameters and system resources on the fly. This ability to observe and interact with data enables users to understand the target problem and the dynamics of the execution behavior better. This in turn allows scientists and researchers to experiment with program data and helps them in choosing better default data values and in improving the overall solution to the problem. It also helps them to explore new computational solutions to problems that are not yet well understood and to select ideal parameters for familiar algorithms applied to new data sets or problem areas. Thus, interactive steering is a powerful tool for scientists, researchers, and algorithm developers. Interactive steering can be useful for complex, long running simulations, modeling applications, or control programs executing in parallel or distributed environments.

1.2 Consistency, Perturbation and Lag

The most important concerns in a steering system are **consistency**, **perturbation** and **lag**.

A typical steering system consists of a monitoring component, a visualization or a user interface component and a steering component. The monitoring component monitors the underlying distributed computation and displays only a consistent view of intermediate states of the computation to the user. The steering component allows the user to issue steering commands, instructions to alter the value of a variable at a single

process, or at many processes or at all processes or to change the allocation of resources. The steering component ensures that these steering requests are applied in a way that maintains the consistency of the computation.

Whenever a steering request is applied, local states of one or more processes involved in the transaction are modified. These local changes at the processes are termed as steering events. The local steering events at all the processes involved together constitute a global state change that is termed as a steering transaction. The constraints that steering changes must adhere to, vary considerably from application to application. Some steering changes may be applied at any of the participating processes at any point in the computation, some may be applied only at certain points in the execution of the process and some steering changes may require coordination between the participating processes. However, if a steering transaction is intended to modify or update the critical control parameters that define the global configuration, care should be taken to maintain the causal consistency of the overall computation. That is, all local steering changes must be applied concurrently across all participating processes.

Perturbation is the degree to which the underlying computation is slowed down or affected due to the presence and use of steering components inside the system. Local perturbation describes the effect on a single process. The primary source of local perturbation is the execution of additional instructions for steering changes of the local process state. The overall effect on the application, including the local perturbations, message traffic, and consistency controls constitute the global perturbation.

Latency or lag refers to elapsed time. Presentation lag is the elapsed time between the existence of a state in the program's execution and the presentation of that state to the

viewer. Steering lag refers to the elapsed time between the initiation of a steering command by the user and the application of the associated steering changes at the process of the computation.

An ideal system would feature strong consistency, low latency and low perturbation.

1.3 Conservative and Optimistic Steering

Two general approaches exist for achieving consistent steering: conservative steering and optimistic steering [MGK01].

The conservative approach requires that a computation reach quiescence [DS80] [Lyn96] before a steering change is applied; the computation blocks before a consensus decision is made and steering changes are applied. That is, no two steering events in the steering transaction may be causally dependent. Thus, in conservative steering, steering changes cannot be applied until the next steerable points are confirmed to be concurrent.

In contrast to the conservative approach, the optimistic approach to steering assumes that the next steerable points are consistent, and invokes the steering change at the next steerable point at each involved process without concern for or knowledge of the state of any other process. This eliminates the need for blocking steered processes. However, the optimistic steering approach must be able to detect any inconsistent steering transaction and provide a checkpointing/rollback mechanism to restore the computation to a correct state.

Conservative methods have been found to offer great potential for certain classes of applications, particularly when ample application-specific knowledge of the systems being simulated is available [MRR90]. Optimistic steering approaches are the best ways to simulate large problems if the state saving overhead is kept within manageable level

[Fuj90]. Conservative and optimistic approaches are also used in data replication algorithms. Pessimistic algorithms rely on synchronous replication coordination whereas an optimistic algorithm propagates its updates in the background, discovers conflicts after they happen, and reaches an agreement on the final object contents incrementally [SS02].

1.4 Organization

This thesis is organized as follows. Chapter 2 describes the architecture of the pathfinder system and the underlying transaction model used in the system. In chapter 3 conservative steering algorithms are discussed. Chapter 4 describes the optimistic steering algorithms and in chapter 5 experiments and results are discussed. Chapter 6 discusses the related work and in chapter 7 conclusion and future work are discussed.

CHAPTER 2

PATHFINDER SYSTEM AND TRANSACTION BASED COMPUTATION MODEL

An exploratory visualization system, known as the **Pathfinder** system serves as the base upon which both conservative and optimistic steering are implemented. This system allows a user to pose queries and visualize program data in a real-time fashion. Through this system, the user may monitor attributes and variables of the distributed computation. The monitoring components [HKR97][Har00][VKH00] and the optimistic steering components [MKG01][GKM02] [Guo02] of the Pathfinder system were developed by the monitoring, visualization and steering group. Implementation and evaluation of the conservative steering component is the focus of this thesis.

In this chapter, we give a brief description of the components of the Pathfinder system and their functions, and explain how conservative steering is integrated into these components. Through the integrated system, users may dynamically manipulate program variables and adjust resource allocation, without compromising the correctness of the underlying computation. Section 2.1 describes the monitoring components of our system and section 2.2 describes the steering components. In section 2.3, the underlying model of computation is described.

2.1 Monitoring

The exploratory visualization system focuses on a class of distributed computations that can be abstracted to an interleaving of atomic state changes involving one or more

processes. It is assumed that communication patterns are not affected by steering changes and remain the same during rollback and re-execution. These atomic state changes, or transactions, correspond to logical actions performed by the computation. A transaction boundary is specified explicitly by end-of-transaction (EOT) annotations in an application program [HKR97][Har00][VKH00]. Placement of EOTs controls the granularity of the view of the distributed computation. Annotation may be performed manually or through automated means.

The Pathfinder system consists of three main components: Interaction Managers (IM), a Snapshot Manager (SM), and a User Interface (UI), as seen in Fig. 2.1. The IM is the interface between the application and the Pathfinder system. The responsibility of the IM's is to collect local snapshots (sets of local variable values) and transaction labeling information [VKH00] and forward these to the SM. The transaction labeling information contains information about the processes that participate in each transaction (membership) and the dependence relationships between transactions (ordering).

The SM acts as the central manager and its responsibility is to merge local snapshots from the Interaction Managers and to form consistent global snapshots based on the transaction labeling information. For these global snapshots to accurately reflect the state of the distributed computation, local snapshots must be grouped together and ordered in a manner that does not violate the causal relationships in the distributed computation [Lam78].

Finally, the SM sends the global snapshots to the UI to be visualized. The UI oversees the decoding of global snapshots into animation actions, the control of visualizations, and the interpretation of user interactions with the visualizations into monitoring directives.

Through the UI, the user may also pose queries to the system in order to gather application specific and system specific values to aid in overall understanding.

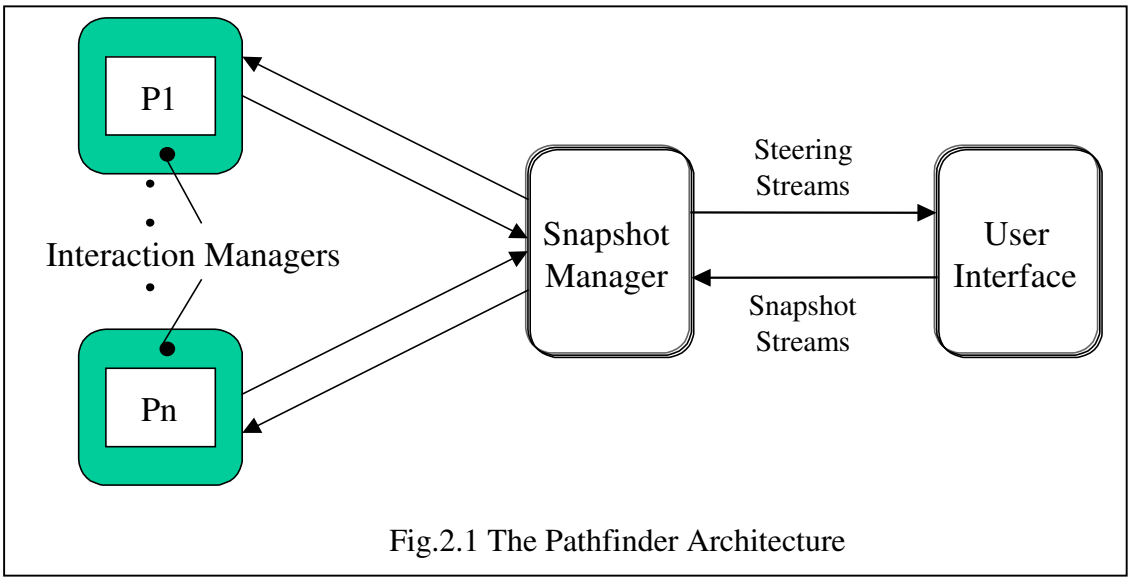

Fig.2.1 The Pathfinder Architecture

Monitoring is explained in detail in [HKR97][Har00][VKH00].

2.2 Steering

The UI provides an interface through which users may issue steering commands to steer the distributed computation at runtime. Once a user issues a steering request, the UI converts this request to a steering directive and sends this to the SM. The SM receives this directive and transforms it into steering commands and forwards these commands to the involved IM's. The direct steering changes, such as manipulating program variables or adjusting resource allocation, are performed by the IM's. The SM coordinates the global steering activities.

For the optimistic steering approach [MKG01][GKM02][Guo02], local checkpointing, message logging, and rollback are performed by the IM's. The detection

of inconsistency, verification of consistency and calculation of the earliest consistent steering time of a steering transaction all require knowledge of all participating processes and the SM performs these operations.

For the conservative steering approach, the global synchronization of steered processes is controlled by the IM's. The SM checks whether or not all steered processes have become passive and whether or not the global transaction in which the steered processes are involved is complete. If so, the SM applies the steering changes.

2.3 Transaction based computational model

In the Pathfinder system, the overall computation is abstracted to an interleaving of atomic state changes involving one or more processes and such state changes are known as transactions. As described in [GKM02], a transaction can be formally defined as

Definition 2.3.1 A transaction relation is an equivalence relation satisfying the following conditions.

1. A **local transaction** is a sequence of events between EOT's (or between the start of the program and the first EOT) of a process.
2. Two local transactions of different processes belong to the same **global transaction** if one local transaction sends at least one message to the other local transaction.
3. A global transaction consists of all local transactions in the same equivalence class.

Fig 2.2 shows an example of a distributed computation consisting of three global transactions. The dark shaded areas are the EOT's and the light shaded areas are the local transactions containing the application events. The application events can be either the send or the receive events. The ellipses are the global transactions comprising local transactions at different processes.

Process P_1 sends a message to process P_3. This send event constitutes the local transaction LT_{11} at process P_1 as this event happens between local transaction boundaries (EOT's). Similarly the receive event at process P_3 constitutes the local transaction LT_{31} at process P_3. Local Transactions T_{11}, T_{31} belong to the same global transaction GT_1 as LT_{11} sends a message to LT_{31}. Global Transactions GT_2 is formed by local transactions LT_{12}, LT_{21} LT_{32} and LT_{41} and similarly local transactions LT_{13}, LT_{22}, and LT_{42} together constitute the global transaction GT_3.

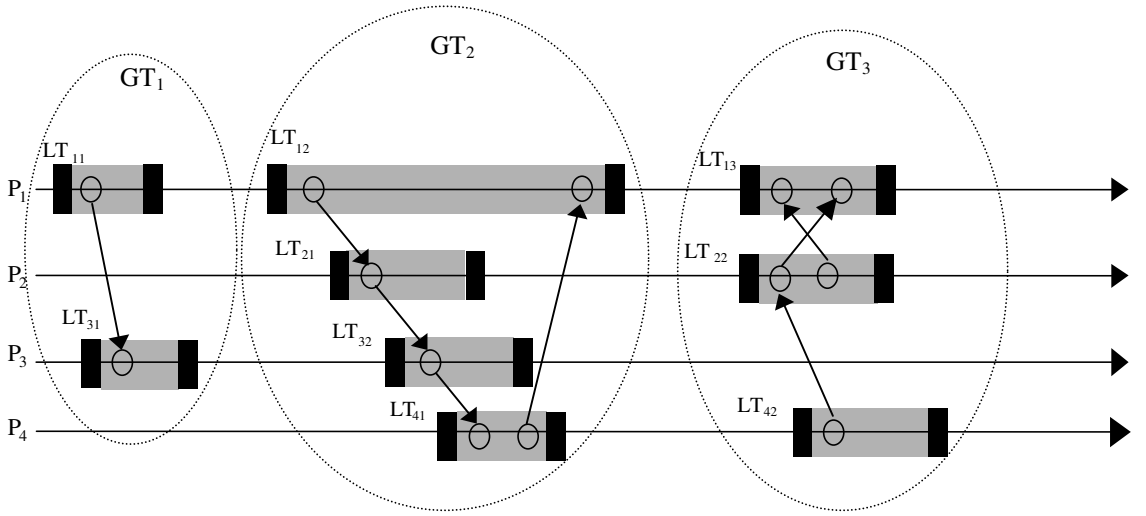

Fig 2.2 A distributed computation with three global transactions. Processes P_1, P_2, P_3 and P_4 are shown as arrows with time increasing from left to right. The dark shaded areas are the EOT's and the light shaded areas are the local transactions containing the application events. The ellipses are the global transactions comprising local transactions at different processes.

The local computation of a process is viewed as a sequence of local transactions and a distributed computation is viewed as a set of partially ordered global transactions. The reason that the model refers to the logical actions taken by the distributed application as transactions is because to the user they appear to exhibit the Atomicity, Consistency,

Isolation and Durability (ACID) properties [GR92]. Computations that satisfy these properties are well-formed. Well-formed computations permit the calculation of equivalence classes, reflecting an ordering of the transactions in the computation.

2.4 Transaction-based Causality and Concurrency Relations

To determine a consistent global snapshot of a distributed computation and transaction ordering, causality and concurrency relations need to be evaluated.

The following definitions are helpful in evaluating these relationships.

Definition 2.4.1 The transaction-based causality relation $\overset{t}{\rightarrow}$ between two transactions T_a and T_b is a transitive relation satisfying the following condition: if there exists a process P_i which participates in both T_a and T_b, and the local transaction of T_a in P_i is before the local transaction of T_b in P_i, then $T_a \overset{t}{\rightarrow} T_b$.

Definition 2.4.2 The transaction-based concurrency relation t‖ between two transactions is defined as: T_a t‖ T_b iff ¬($T_a \overset{t}{\rightarrow} T_b$) and ¬ ($T_b \overset{t}{\rightarrow} T_a$).

The transaction-based causality relation has more constraints than the event-based causality relation. If we consider event-based causality in fig 2.3, $s_1 \parallel s_3$, $s_1 \parallel r_3$, $r_1 \parallel s_3$ and r1 ‖ r3 and hence transaction T_1 would be considered concurrent with transaction T_3. But if transaction-based causality is considered in the same figure, $T_1 \overset{t}{\rightarrow} T_2$, and $T_2 \overset{t}{\rightarrow} T_3$, then $T_1 \overset{t}{\rightarrow} T_3$. Thus, the transaction-based causality relation is stronger because of the view that transactions are logically atomic actions.

A steering transaction is an atomic state change, and hence a steering change cannot be applied in between transactions. For example, in figure 2.2 a steering change cannot be applied in local transaction LT_{11} in process P_1 and in local transaction LT_{21} in process

P_2. This is because local transaction LT_{11} belongs to global transaction GT_1 and local transaction LT_{21} belongs to global transaction GT_2. Thus the steering change in process P_1 has to be delayed till the computation in process moves to local transaction LT_{12} and then the steering change can be applied in both process P_1 and process P_2.

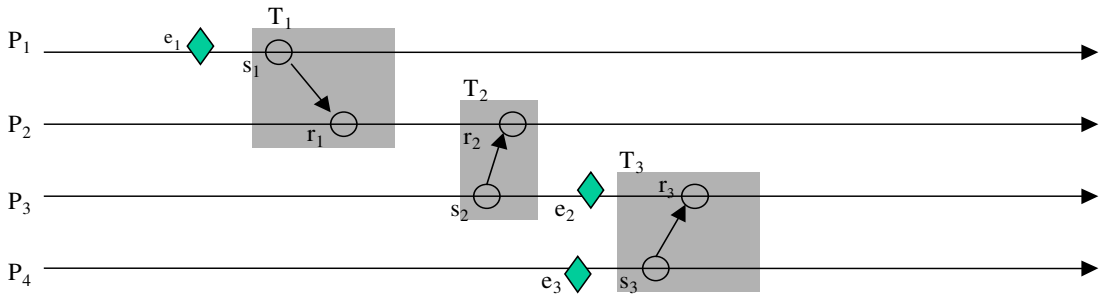

Fig. 2.3 Each global transaction is shown as a shaded area containing application events (circles). The boundaries of the shaded areas represent annotations demarking the transaction boundaries. Messages are shown as arrows between process events. Steering events are shown as diamonds.

CHAPTER 3

CONSERVATIVE STEERING

Conservative steering avoids all possible inconsistent steering by strictly adhering to the causality constraint. The conservative approach requires that a computation reach quiescence [DS80] [Lyn96] before a steering change is applied; the computation blocks before a consensus decision is made and steering changes are applied. That is, no two steering events in a given steering transaction may be causally dependent. Thus, in conservative steering, steering changes cannot be applied until the next steerable points are confirmed to be concurrent.

In this chapter the conservative steering algorithm is discussed in detail. Section 3.1 gives an overview of the conservative steering algorithm, and in section 3.2, the control messages involved in conservative steering are discussed. Section 3.3 discusses about the process states and section 3.4 discusses how consistency is maintained and how process synchronization helps in maintaining consistency in the conservative steering approach.

3.1 Conservative Steering Algorithm Overview

The user interface of the Pathfinder system displays the updates and the changes in the program variables and the system resources to the user. The user can issue a steering request through the user interface to change one or many of the program variables or the system resources at one or many or all of the processes. The UI forwards this steering request to the Snapshot Manager. Once the Snapshot Manager receives the steering

request from the UI, it sends a *prepare-to-steer* message to all processes involved in steering. The processes involved in steering are called steered processes.

A process, upon receiving the *prepare-to-steer* message from the Snapshot Manager becomes passive at the next steerable point (EOT) after sending and receiving all the messages, which it is required to send and receive before the end of transaction (EOT). The process then sends notice of its passive state to the SM and sleeps until it receives a *steering- change* message from the SM or until it receives a *message-request* message from another process. If the process receives the *message-request* message, it becomes active again and continues its execution until it reaches another steerable point. The SM sends the *steering- change* message to all the steered processes after receiving the passive states from all the steered processes and after confirming the completion of the global transactions that involved the steered processes. The steered processes receive this *steering-change* message from the SM and then apply the steering changes and continue their normal execution. Thus, the conservative steering approach requires that all steered processes be synchronized at the consistent steerable point before steering changes are applied [Guo02]. Fig 3.4 gives a detailed description of the conservative steering algorithm.

3.2 Control Messages

There are two kinds of communication messages in the Pathfinder system, application messages and control messages. Messages that are communicated between processes on the application level in the Pathfinder system are considered application messages. In fig 3.1, process P_1 sends an application message to another process P_2. These kinds of

messages that are exchanged between the processes on the application level are termed application messages.

Messages that are communicated between various control modules such as the snapshot manager, interaction manager and user interface and between processes for monitoring and steering are termed control messages. In fig 3.1 the *prepare-to-steer message* is sent by the SM to the processes P_1 and P_2 to initiate steering at P_1 and P_2. The processes send the *steering-confirm* message to the SM after they reach a steerable point.

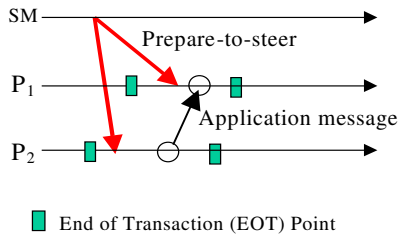

Fig 3.1 The red arrow represents a *prepare-to-steer* control message sent to the processes P_1 and P_2 by the SM. The black arrow is the application message sent by the process P_1 to P_2.

Messages that are used for controlling the various modules and for collecting and transmitting the control information are known as control messages. *Message-send-request, message-receive-request, steering-change* are other control messages used in the conservative steering approach. A *prepare-to-steer* message is sent by the SM to the processes to initiate a steering request. A process sends a *message-send-request* message to another process to request for a particular message. A *message-receive-request* message is sent by a process to another process to inform that it should receive a particular message.

3.3 Process States

In the conservative steering approach, a process can be in a steered state or in an affected state. A process that is involved in a steering transaction and that applies the steering changes is called as the steered process and a process that is involved in a steering transaction but is not involved in actual steering is called an affected process. A process can also be in an active state or in a passive state in the conservative steering approach. An active process is a process that is continuing its execution normally. On the other hand, a passive process is a process that has reached a steerable point (EOT) and is sleeping and is waiting to receive a *message-receive-request* or a *message-send-request* message from another process or a *steering-change* message from the SM. A passive process can still be able to receive control messages and process them even though it is in a passive state. Only then, a passive process will be able to receive a *steering-change* message from the SM and will be able to apply the steering changes. An active process becomes passive after reaching a steerable point. A passive process becomes active when it receives a *message-send-request* or a *message-receive-request* message from another process forcing it to participate in another transaction, as explained in fig 3.2. A passive process can also become active after receiving the *steering-change* message from the SM and after applying the steering changes.

3.4 Consistency Checking and Process Synchronization

The SM is the central manager in the Pathfinder system and is responsible for consistency checking in the conservative steering approach. The SM sends the *steering-change* message to all processes only if: 1) all steered processes have become passive and 2) the

global transaction that each steered process involved in is complete. The first condition ensures that all the steered processes have reached a steerable point ad the second condition ensures that no two steering events can be causally dependent on each other.

Process synchronization plays an important role in maintaining consistency in the conservative steering algorithm. To enable process synchronization, each process maintains four different arrays, each of size N, where N is the number of processes in the system. These arrays are used to keep track of the number of messages being sent and received by a process. These four arrays are: **message-sent** array, **message-received** array, **message-send-request** array and **message-receive-request** array. Each value in an array is a counter. For example, *message-sent[i]* denotes the number of messages sent by the process to the i^{th} process. Whenever a process sends a message to another process, it

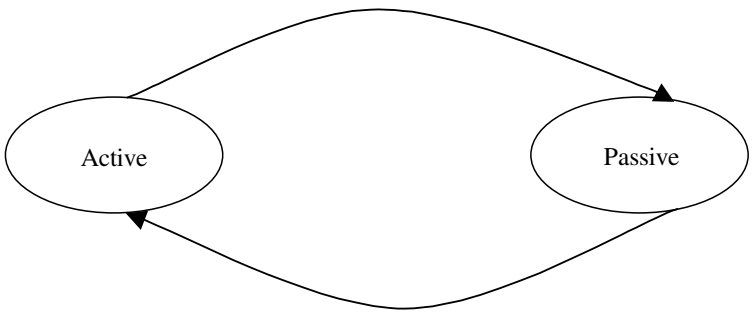

Fig. 3.2 Active and passive process states in conservative steering

increments the *message-sent[i]* counter where i is the process to which the message is sent, and whenever a process receives a message from another process, it increments the *message-received[i]* counter where i is the process from which the message is received.

This enables a process to keep track of the messages being received from and sent to a particular process. Whenever a process is waiting on a message to be received from another process, it sends a *message-send-request* message to that process requesting that it send the message. The *message-send-request* message consists of a message request number that requests the process to send a particular message. The process that receives the message-send-request message checks if it has already sent that requested message. If it has already done so, it ignores the *message-send-request* message. If it has not sent the requested message then it sends the requested message and updates its *message-send-request[i]* counter. Whenever a process becomes passive, it broadcasts a *message-receive-request* message, along with the value of the *message-sent* counter. These counters and the messages help in synchronizing processes and to maintain consistency while applying steering changes.

Process synchronization is explained using fig 3.3(a), where steered processes P, Q and affected processes R and S participate in a global transaction. Process Q has received a *prepare-to-steer* command from the SM and has become passive at EOT_{21}. Process P has also received the *prepare-to-steer* command from SM but it is not able to reach its EOT_{11} as it is waiting for a message to be sent by process Q. Similarly, affected process R is waiting to receive a message from process Q. Since every process is passive or is waiting to receive a message from another process, the whole application would stall. In this case, process Q should become active and send the message to process P to enable the application to start again. To make process Q active, process P sends a *message-send-request* message numbered 2 to process Q. Process Q checks the value of its *message-send-request[P]* counter with the value of *message-sent[P]* counter and finds that it has

not sent the requested message. So, process Q becomes active again, sends the requested message and updates its *message-sent[P]* counter and *message-send-request[P]* counter. Process Q then continues to execute until it reaches the next steerable point.

Suppose that process Q is slow and process P is executing quickly and is waiting for the message from process Q. Then, process P will send the *message-send-request* message to process Q. Process Q receives the *message-send-request* message from process P and will continue to execute until it can send the requested message and reach the next steerable point. This is how processes synchronize themselves to maintain

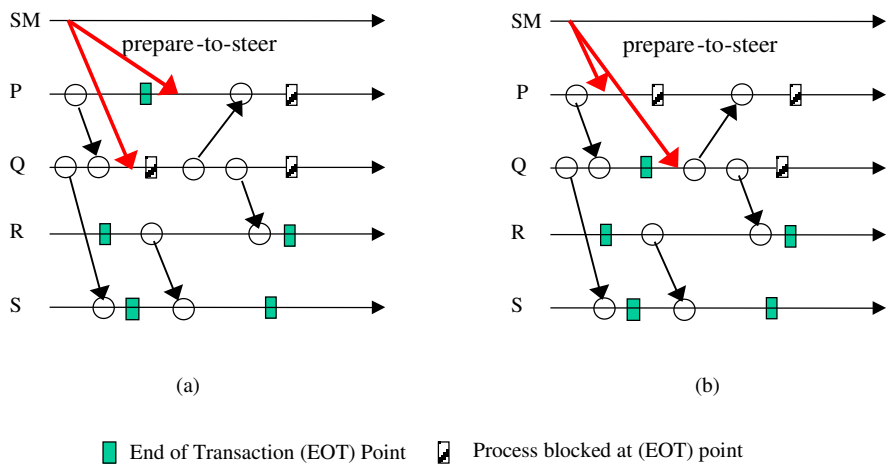

Fig 3.3 Process Synchronization (a). A passive process Q must become active again when it needs to send a message to a steered process. (b). A passive process P must become active again when it needs to receive a message from a steered process.

consistency when a process is not able to receive a message due to another slow process or a passive process.

Suppose that process P receives the *prepare-to-steer* command before EOT_{11} and becomes passive at EOT_{11} and process Q receives the *prepare-to-steer* command after EOT_{21} and has become passive at EOT_{21} as in fig 3.3 (b), the global transaction involving

these local transactions will not be able to complete. In this case, process P must become active and execute forward to receive the message in order to complete the global transaction. To solve this, the *message-receive-request* message is broadcast by process Q to all the processes before it becomes passive at EOT_{22}. Process P receives this message and checks whether it has received this message from Q, and finds that it has not yet received the message. Hence, P becomes active again and receives this message and continues to execute until it reaches the next steerable point.

Once the processes reach a steerable point, they communicate their passive state to the SM. The SM checks the consistency conditions at this point and if it is a consistent point, it issues a *steering-change* command to all steered processes. The processes receive the *steering-change* command from the SM and then apply the steering changes. After applying the steering changes, the processes continue their normal execution.

Snapshot Manager:

```
Send prepare-to-steer to all steered processes;

    IF (all steered processes are passive) AND
          (global transactions that each steered process P_i was involved in at LVT_i are completed)
    Send steering-change to all steered processes;
```

Interaction Manager / User Process:

```
    // Initialization
    state := active;
    steeringFlag := false;

    for i := 1 to N do {          // N = number of processes
        s[i] := 0;                // the number of messages sent to P_i
        r[i] := 0;                // the number of messages received from P_i
        rs[i] := 0;               // P_i requested to send up to rs[i]
        rr[i] :=0;                // P_i requested to receive up to rr[i]
    }
```

When message arrives from the SM:

```
    SWITCH message type

        CASE  prepare-to-steer:
            steeringFlag := true;

            IF the process is waiting for a message from P_i THEN
                k := r[i] + 1;
                send a message-send-request(k) to P_i;
            END IF

       CASE  steering-change:
            apply the steering change;
            steerFlag := false;
            state := active;
            continue execution;

    END SWITCH;
```

EOT:

```
    IF steeringFlag = true THEN
        IF for all i (s[i] >= rs[i]) AND (r[i] >=rr[i]) THEN
            FOR each process P_i DO
                send message-receive-request (s[i]) to P_i;
            END FOR
            state := passive;
            send (passive, LVT_i) to the SM;
            sleep;

        END IF;
```

```
    END IF
```

Message Send to P_i:

```
    s[i]++;
    send the message;
```

Message Receive from P_i:

```
    IF the message is not in the queue THEN
        IF steeringFlag = true THEN
            k := r[i] + 1;
            send a message-send-request(k) to Pi;
        END IF
        wait for the message until it arrives;
    END IF
    r[i]++;
    process the message;
```

When control message arrives from P_i:

```
    SWITCH message type

        CASE  message-send-request(k):
            rs[i] := k;

            IF steeringFlag = false  Then
                steeringFlag := true;
            END IF;

            IF state = passive THEN
                IF s[i] < rs [i] THEN
                    state := active;
                    continue execution;
                END IF
            END IF

        CASE  message-receive-request(k):
            rr[i] := k;

            IF steerFlag = false THEN
                steerFlag := true;
            END IF

            IF (k > r[i]) AND (state = passive) THEN
                state := active;
                continue execution;
            END IF

    END SWITCH;
```

Fig 3.4 Conservative Steering Algorithm

CHAPTER 4

OPTIMISTIC STEERING

The optimistic steering approach to interactive steering assumes that the next steerable points are consistent, and invokes the steering change at the next steerable point at each involved process without concern for or knowledge of the state of any other process. This eliminates the need for blocking steered processes. However, the optimistic steering approach must be able to detect any inconsistent steering transaction and provide a checkpointing/rollback mechanism to restore the computation to a correct state.

In this chapter the optimistic steering algorithm is discussed in detail. Section 4.1 gives an overview of the optimistic steering algorithm, and in section 4.2 the process states and the message types involved in optimistic steering are discussed. Section 4.3 discusses how consistency is checked and section 4.4 discusses how message logging is done in optimistic steering approach. Section 4.5 gives an overview on checkpointing and section 4.6 explains how rollback and re-execution is performed in optimistic steering.

4.1 Optimistic Steering Algorithm Overview

In the optimistic steering approach, the SM receives the steering request from the user interface and broadcasts a *steering-request* message to the processes. Once the processes receive the *steering-request* message from the SM, they checkpoint their local process states and apply the steering changes and then communicate this time to the SM. After receiving the steered time from all the steered processes, the SM checks the

consistency of the steering changes made by the steered processes from the steering times and the TLP information obtained from these processes. If the steering transaction is consistent, the SM sends a *steering-ok* message to all the processes. If the steering transaction is inconsistent, the SM sends a *steering-rollback* message to all the processes along with the earliest consistent steering time. If the steered processes receive a *steering-ok* message, they clear their message logs, remove their checkpoints and continue to execute normally. On the other hand, if the processes receive a *steering-rollback* message, they rollback to their checkpointed states and re-execute till the consistent steering time specified by the SM and apply the steering changes at the consistent point specified by the SM.

The main difference in optimistic steering is that the processes neither need to know the states of other processes before applying the steering changes, nor do they need to be blocked before the steering changes are made. However, additional capabilities are required to provide mechanisms for checkpointing, roll backing and re-executing.

4.2 Process States and Control Messages

The control messages involved in optimistic steering are *steering-request, steering-ok, steering-rollback, and steering-confirm* messages. A *steering-request* message is sent by the SM to the processes to initiate a steering request. The steered processes send a *steering-confirm* message to the SM after applying the steering changes. If the steering transaction is consistent, the SM sends a *steering-ok* message to all the steered processes and if the steering transaction is inconsistent, it sends a *steering-rollback* message to the steered processes.

As seen in fig 4.1, in the optimistic steering approach, a process can be in a **normal** state or in a **tentative** state, or in a **recovering** state. A *normal* state implies that neither the consistency of a steering change is in question nor is the correction of an inconsistency underway [Guo02]. A process enters into a tentative state from a normal state after receiving a *steering-request* message from the SM or after receiving a tainted message from another process. Any message that is sent by a process, which is in a tentative state, is considered to be tainted. A process should checkpoint its state before it

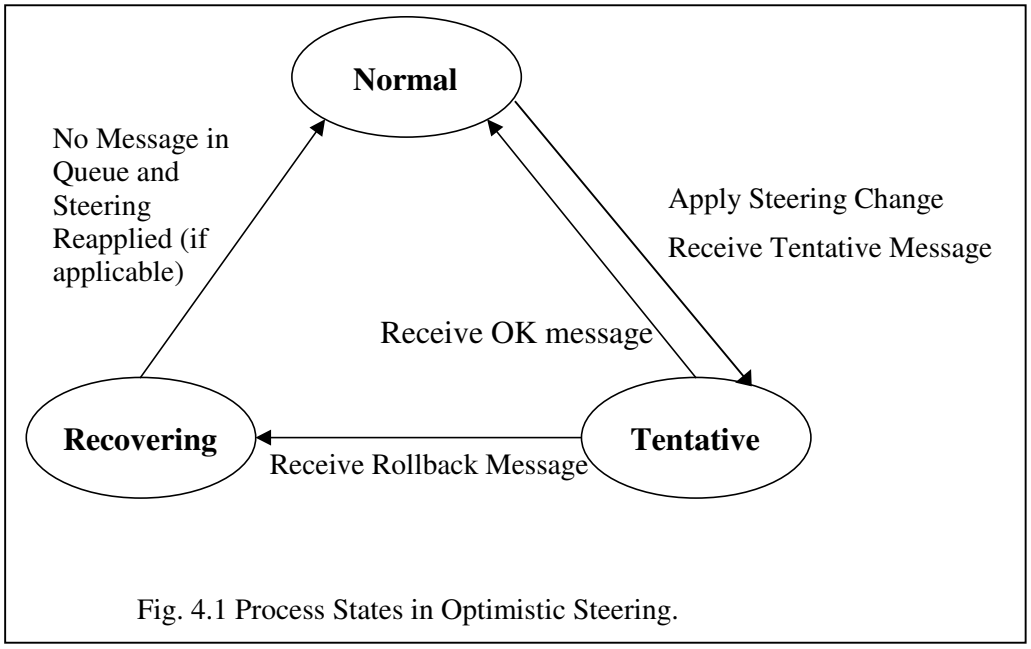

Fig. 4.1 Process States in Optimistic Steering.

enters into the tentative state because the process should be able to rollback to its checkpointed state if the steering transaction is inconsistent. A tentative process executes speculatively, logging all its messages. If the steering transaction is inconsistent, then the process enters into the recovering state from the tentative state. If the steering changes are consistent, then the process enters into the normal state directly from its tentative state. A

process changes its state from recovering to normal only after re-applying the steering changes and after processing all the messages in its queue.

4.3 Consistency Checking

In the optimistic steering approach, the processes apply the steering changes without the knowledge of the states of other processes. So, the steering changes may be applied at a consistent point or in an inconsistent point. If the steering changes are consistent, then the processes continue their normal execution. However, if it is inconsistent, the processes should rollback to their checkpointed states and re-execute until the next consistent point and re-apply the steering changes. Hence, in the optimistic steering approach, whenever processes make steering changes, the consistency of the steering changes should be verified.

We consider a steering transaction as consistent if all the steered processes make the steering changes at the same time, i.e., if all the steering events are concurrent. To determine the concurrency of the steering events, we have developed two algorithms, a **history based** [MGK01] algorithm and a **vector time** [GKM02] based algorithm. These algorithms determine the consistency of the steering transaction. If the steering transaction is inconsistent, they determine the earliest consistent steering point. Section 4.3.1 discusses how the history-based algorithm is used to determine consistency and section 4.3.2 discusses how the vector time based algorithm checks consistency in optimistic steering approach.

4.3.1 History based algorithm

To determine consistency, the history-based algorithm [MGK01], as seen in fig 4.2, maintains a list of vectors representing a partial ordering of the subset of the program

transaction history. This program transaction history contains all transactions that occurred after the earliest steering event and before the latest steering event. To accomplish consistency detection, the algorithm creates a consistency vector representing a consistent cut at which a steering transaction could be applied. The algorithm works backwards, generating vector times for consistent cuts based on comparisons between the program transactions being analyzed and the steering transaction. The algorithm terminates once the present steering transaction is reached or an inconsistency has been detected. If an inconsistency is detected, the last consistent cut stored in the consistency vector contains the earliest consistent times for the steering transaction and represents the point at which the steering changes will be reapplied.

4.3.2 Vector Time Based Algorithm

In the vector time [GKM02] based algorithm, the consistency of a steering transaction is detected by directly comparing the vector timestamps of steering events. If the steering events are concurrent, then the steering transaction is consistent. Otherwise it is inconsistent. Further, the earliest consistent transaction can be obtained by checking the vector time of each steering event in a steering transaction.

In the vector time based algorithm, each process is associated with a vector clock V of size N, where N is the number of processes in the system. Each element in the vector corresponds to a process in the system. The value of V[i] denotes the number of past local transactions at that process as known by this process. The vector time of a steering event in process P_i is the vector time that results from the occurrence of a steering event in process P_i.

If the steering events in a steering transaction are not concurrent, we know that there exists some causal relation between at least two of the steering events. To avoid violating such causal relations, the simplest method is to apply the steering action at the later time of the two. If we can find the latest time for each process involved in the steering transaction and apply the steering at that time, the new steering transaction will be consistent and it will be the earliest consistent transaction, as any steering transaction applied before it is inconsistent.

```
1.  Table TLP /*Table containing transaction history*/
2.      Vector TV /*Vector holding information about present transaction in TLP*/
3.      Vector SV /*Steering Vector containing list of processes involved in steering transaction*/
4.      Vector CV /*Consistent Vector representing when the steering transaction should take place*/
5.      Vector CVtemp /*Temporary vector to hold information until it is verified SV has not made TV
6.          inconsistent*/
7.      Vector Verified /*A Vector of Boolean values set to true when a process listed in SV is at its
8.          earliest logical time to have invoked the steering command*/
9.      Boolean consistent /*Boolean flag to indicate if SV has made TV inconsistent*/
10.
11.     BEGIN
12.
13.     set all elements of Verified to FALSE
14.     set TV equal to last vector of TLP table
15.
16.     WHILE (all processes in SV have not been set to true in Verified)
17.             BEGIN
18.                     consistent set to TRUE
19.
20.                     FOR (compare each corresponding, non-empty cell in TV and SV)
21.                             BEGIN
22.                                     IF(any non-empty cell in TV corresponds to a cell marked TRUE
23.                                     in Verified)
24.                                             THEN mark all cells in Verified corresponding to non-
25.                                              empty cells in TV TRUE
26.                                              set consistent to FALSE
27.                                              break
28.
29.                                     IF(TV is greater than or equal to SV)
30.                                             THEN set corresponding cell of CVtemp to TV
31.                                     ELSE

32.                                             Mark Verified cells corresponding to non-empty TV cells
                                                to TRUE  and mark consistent to FALSE
33.                                     break
34.                                     END
35.
36.                     IF(consistent)
37.                             THEN
38.                                     FOR(each non-empty element of CVtemp)
39.                                             set corresponding cells of CV to CVtemp
40.
41.                     IF(all cells of Verified corresponding to all cells SV are marked true)
42.                             break
43.                     ELSE
44.                             set TV equal to previous vector in TLP
45.             END

46.
47.     IF(SV equals CV)
48.             Return Consistent
49.     ELSE
50.             Return CV
51.
52.     END
```

Figure 4.2 – History-Based Consistency Detection Algorithm

4.4 Message Logging

In the optimistic steering approach, when a steering transaction is inconsistent, the processes have to rollback to their previous checkpointed states and then re-execute from their checkpointed states. When a process re-executes from its checkpointed state, it should be able to send and receive all the messages that were sent and received during its normal execution. So, a process should log all the messages during its normal execution, so that it can replay the messages during its re-execution.

In the optimistic steering approach, a process starts logging its messages after it enters into a tentative state. However, the process need not log all the messages. Only the messages received from a process that is executing in a normal state or in a recovering state need to be logged because a process executing in a normal state or in a recovering state does not have the ability to rollback. The messages that are received from processes that are in tentative states need not be logged because these processes have the ability to re-execute and hence these messages will be re-sent during the re-execution of the processes. Message logging is explained in detail in [Guo02].

4.5 Checkpointing

In the optimistic steering approach, a process should checkpoint its local computational state so that it can rollback to a consistent state and re-execute from this consistent state, in case of an inconsistent steering transaction. This local checkpoint includes the state of all static variables, all dynamic variables explicitly listed for protection by the programmer, the execution stack, and the CPU state, including but not limited to the program counter (PC) and stack pointer (SP). Maintaining the execution

stack and the PC allows the Pathfinder system to seamlessly restart a process's execution at the exact point at which the checkpoint was taken [Guo02].

Each process writes its computational state information into two binary files. The local data and the execution stack information are written into one file and the protected dynamic memory information is written into another file. When dynamic memory needs to be checkpointed, the user can protect this memory using a special protocol. This protocol allows each process to store the base address and the contiguous byte information of the dynamic memory into a data structure. So, when the process takes its checkpoint, the dynamic memory is read from this data structure and written into the second binary file. Checkpointing is a very complex process and is explained in detail in [Guo02] and [Mil02].

4.6 Recovery - Rollback and Re-execution

In the optimistic steering approach, if a steering transaction is inconsistent, the process should not only be able to rollback to its checkpointed state, but also should be able to start its re-execution from the checkpointed state. As described earlier, a process enters into a recovering state from a tentative state upon receiving a *steering-rollback* message from the SM. Once a process enters into a recovering state, it rolls back to its checkpointed state and starts its re-execution from the checkpointed state. During its re-execution the process tries to replay its logged messages from its message queues for each *Receive*. However, the process performs a normal *Receive*, if no logged message exists. When an inconsistent steering transaction is detected, the SM broadcasts the consistent steering time to all the processes along with the *steering-rollback* message. So all recovering processes continue their re-execution until this consistent steering point.

Upon reaching this consistent steering point, the processes reapply the steering changes and clear their message logs, and remove their checkpoints and start executing normally.

On the other hand, if the steering transaction is consistent, the SM sends a *steering-ok* message to all the processes. Upon receiving this message, the processes clear their message logs, remove their checkpoints and start executing normally.

CHAPTER 5

TESTS AND RESULTS

To measure the performance of the conservative steering approach, to compare it with that of the optimistic steering approach, and to measure and compare the overheads associated with conservative steering and optimistic steering, we performed a set of tests and analyzed the results. These tests measured the perturbation and the lag associated with conservative steering and optimistic steering. This chapter describes the tests, which we performed, and analyzes the results obtained from these tests. Section 5.1 describes the parameters considered for the tests and briefly explains how the tests were performed. In section 5.2 the perturbation tests and the results are discussed. Section 5.3 discusses the steering lag tests and the results and section 5.4 summarizes the results.

5.1 Tests and Parameters considered

All the tests were run on a cluster of 4 Pentium II 450 Mhz workstations, each running Red Hat Linux 7.2. Each of these machines had 128 MB of RAM. The application and the control processes, explained in section 3.2, were executed on the workstations and the user interface was run on a separate machine to avoid the workstation load imbalance.

We considered three kinds of **communication patterns** and two kinds of **transaction sizes** for these tests. In the case of the optimistic steering approach we also varied the **size of the checkpoints**. For each test, the **number of processes** involved and the number of processes steered were also varied. The communication patterns we considered were,

1. **Global** communication patterns in which all the processes communicate and synchronize in each transaction.

2. **Mixed** communication patterns in which processes communicate pair-wise in each transaction and for every 10 pair-wise transactions, they perform a single global transaction.

3. **Pair-wise** communication patterns in which processes communicate pair-wise and no global communication or synchronization takes place.

The two kinds of transaction sizes which we considered were: **short transactions**, in which each transaction had a .25 second computation time and **large transactions** in which each transaction had a 2.5 second computation time. In the case of the optimistic steering approach, we used three checkpoint sizes: a **small checkpoint** size, in which the size of the checkpoint was 10 KB, a **medium checkpoint** size, in which the size of the checkpoint was 100 KB and a **large checkpoint** size, in which the size of the checkpoint was 1 MB. We also varied the number of the processes for each test. We used 2 or 4 or 8 processes for each test.

We ran our tests, varying all these parameters. So, for the conservative steering approach, one of the sample tests would be using 2 processes with small transactions and mixed communication pattern. Similarly for optimistic steering approach, it would be using 4 processes with large transactions and global communication pattern with a small checkpoint size.

5.2 Perturbation

Perturbation, as explained earlier, is the degree to which the underlying computation is slowed down or affected due to the presence and use of steering components inside the system. We measure only the execution time overhead. We ran these tests to measure the extent to which the conservative steering components slows down the computation and to compare this with that of the optimistic steering approach and with that of the monitored computations.

Each perturbation test consisted of 150 transactions and 5 steering transactions. Of these 5 steering transactions, 2 steering transactions involved 50 % of the processes, 2 steering transactions involved all the processes, 2 steering transactions involved ¾ of the processes and 1 steering transaction involved half of the processes. For the 2 process tests, all the steering transactions contained both the processes because, steering a single process would ensure consistency and hence such a test would give meaningless performance results. Each of these tests was run 5 times and the execution time was taken as the average execution time of those tests. We ran these tests for an unmonitored distributed computation, monitored distributed computation, optimistic steered computation and conservative steered computation. In the case of optimistic steering approach, we ran these tests with the small, medium and the large checkpoint sizes.

Fig 5.1 and fig 5.2 give the execution times for the unmonitored, monitored, conservative steered and optimistic steered with a small checkpoint size, a medium checkpoint size and large checkpoint size. The x-axis of these graphs denotes the unmonitored, monitored, optimistic steered (10 KB), optimistic steered (100 KB), optimistic steered (1 MB) and conservative steered computations with global, mixed and

pair-wise communication patters for 2, 4 and 8 processes. The y-axis of these graphs denotes the execution time taken for these computations in seconds. In fig 5.1 the execution time (y-axis) starts from 20 seconds rather than from 0 seconds and in fig 5.2, the execution time (y-axis) starts from 360 seconds rather than from 0 seconds.

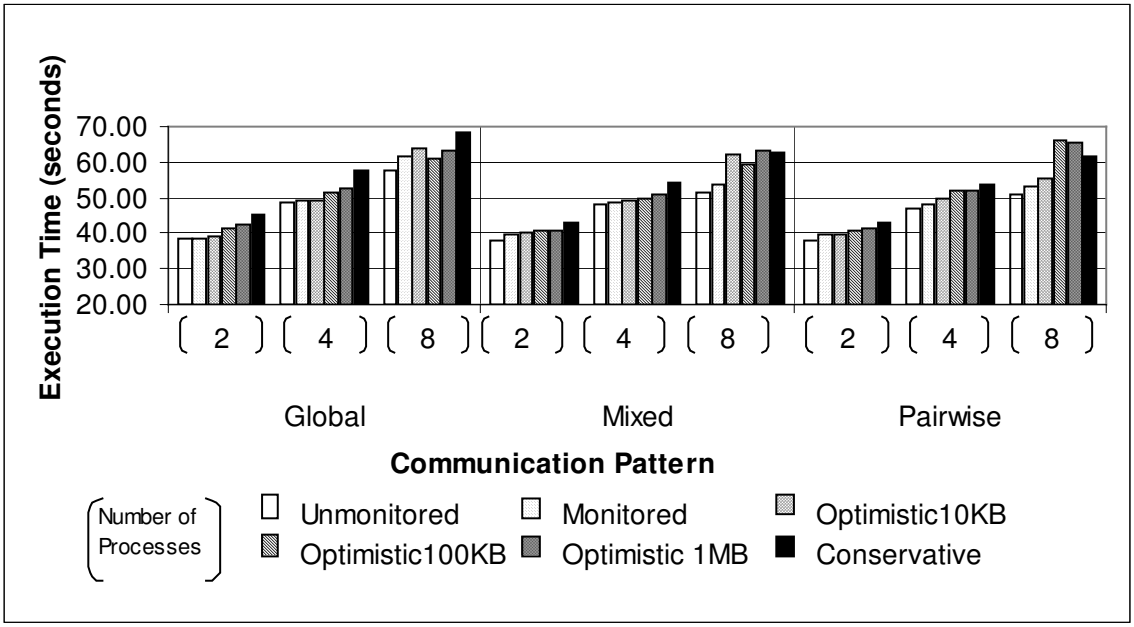

Fig 5.1 Perturbation for Small Transactions

As expected, for the conservative steering approach, the overhead increases with the number of processes. On average, the conservative steering approach introduced a 1.77 % overhead for large transactions and a 13.11% overhead for small transactions. In small transactions, the conservative steering approach has less overhead than the optimistic steering with a medium checkpoint size and a large checkpoint size in the pair wise communication pattern for 8 processes. Similarly, in large transactions, the conservative steering approach has less overhead than the optimistic steering approach in the global communication pattern for 8 processes and in the pair wise communication pattern for 8 processes. In the optimistic steering approach, the overhead increases with the number of the processes and with the size of the checkpoints. From figures 5.1 and 5.2, we can

understand that the conservative steering approach scales better than the optimistic steering approach as the number of processes increase.

On average, for small transactions, the optimistic approach with small checkpoint size (10KB) introduced a 3.39% overhead, the optimistic approach with medium checkpoint size (100KB) introduced a 6.83% overhead, and the optimistic approach with large checkpoint size (1MB) introduced a 9.27% overhead, And for large transactions, the

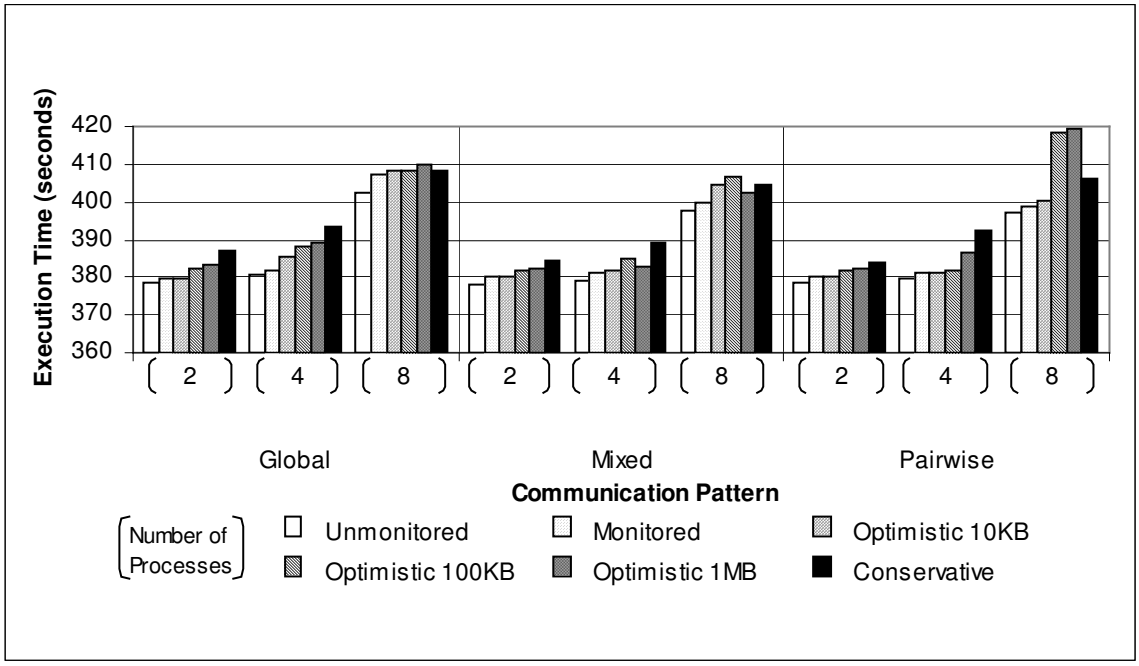

Fig 5.2 Perturbation for Large Transactions

optimistic approach with small checkpoint size (10KB) introduced a 0.41% overhead, the optimistic approach with medium checkpoint size (100KB) introduced a 0.87% overhead, and the optimistic approach with large checkpoint size (1MB) introduced a 1.14% overhead.

We also measured the percentage of steering transactions that were found to be consistent on the first attempt. If the percentage of steering transactions consistent on the first attempt is high, then the optimistic steering approach performs better, because this

ensures that the processes need not rollback and re-execute. Fig 5.3 gives these percentages for global, mixed and pair-wise communication pattern and for small and large transactions. The x-axis denotes the 2, 4 and 8 process computations with global, mixed and pair wise communication patterns for both small and large transactions. The y-axis denotes the percentages of consistency for these computations. From the figure, we understand that for global transactions and for transactions involving only two processes, the percentage of consistency is 100%. The average percentage of consistency for mixed and pair-wise communication patterns is around 70% and 80%. This is due to the fact that in mixed and pair-wise communication patterns, the processes are not all synchronized and faster executing processes have to wait for slower executing processes. This also contributes to the higher percentage of inconsistency for higher number of processes. So the consistency of the steering transactions depends on the number of processes and on the communication pattern.

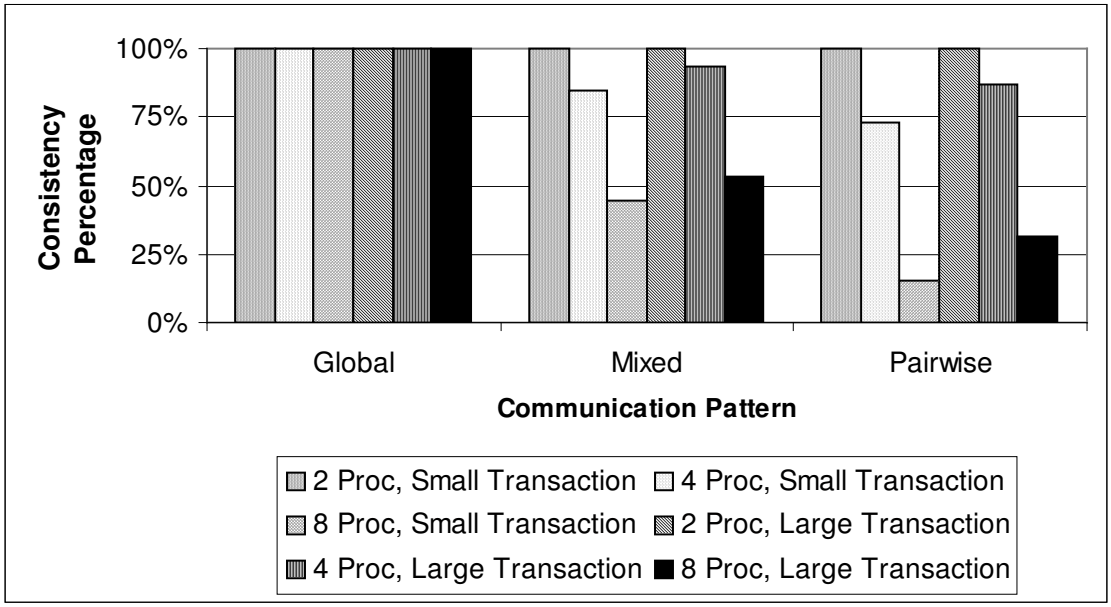

Fig 5.3 Percentages of Steering Transactions consistent on first attempt.

5.3 Steering Lag

Steering lag refers to the elapsed time between the initiation of a steering command by the user and the application of the associated steering changes at the process of the computation. Three kinds of lag were considered and measured: the **IM local lag**, the **IM global lag** and the **SM lag**. The IM local lag refers to the elapsed time between the first EOT event of a process after receiving the steering request and the successful application of the steering change at the process. The IM global lag is the elapsed time between the first EOT event of all processes after receiving the steering requests and the last successful application of the steering changes among all steered processes. The SM lag refers to the elapsed time between the sending out the steering requests by the SM and the last successful application of the steering changes among all steered processes.

Each of the steering lag tests consisted of 150 program transactions and 15 steering transactions. When the tests were run on 2 processes, all steering transactions involved both the processes. When the tests were run on 4 processes, 5 steering transactions involved all the processes and the remaining 10 steering transactions involved only half of the processes. Similarly for 8 processes, during each run, 5 steering transactions involved all the processes, 5 steering transactions involved half of the processes and the remaining 5 steering transaction involved only 2 processes.

The average IM local lag for the optimistic steered approach with a checkpoint size of 100 KB, the optimistic steered approach with a checkpoint size of 1 MB and the conservative steered approach for small and large transactions are given in fig 5.4 and fig 5.5 The x-axis of these graphs denotes the consistent and inconsistent optimistic steered (100 KB and 1 MB) computations and consistent and inconsistent conservative steered

computations for 2, 4 and 8 processes. The y-axis of these graphs denotes the average IM local lag of these computations in microseconds.

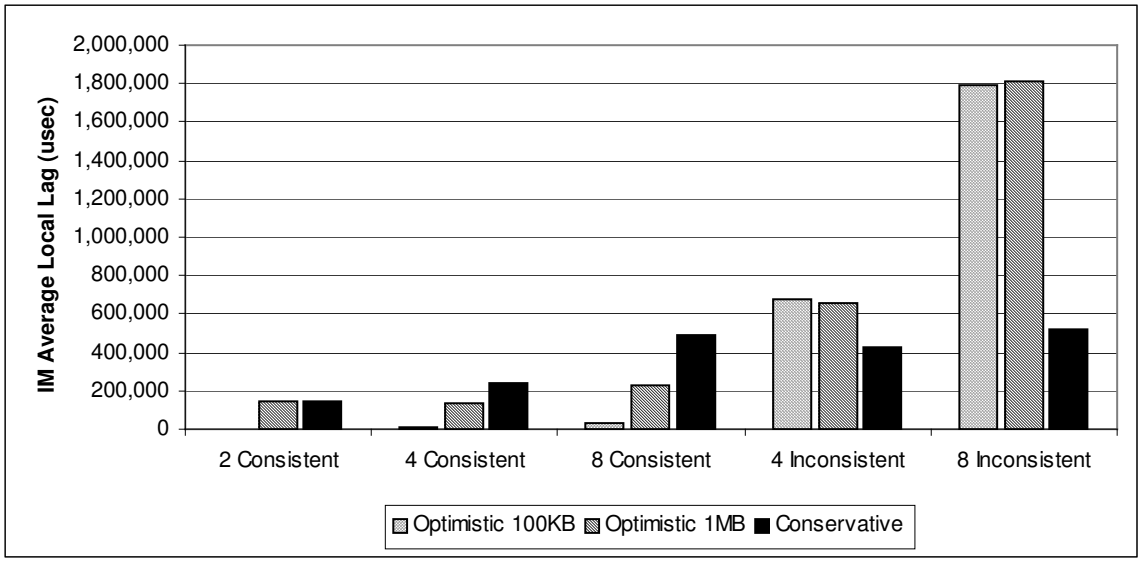

Fig 5.4 Average IM Local Steering Lag for Small Transaction

From these figures we can understand that, in both small and large transactions, when the steering transactions are consistent on the first attempt, the optimistic approach performs better than the conservative approach. However, when the steering transactions are inconsistent, the optimistic approach has a very large steering lag compared to the conservative approach. This is due to the fact that in the optimistic steering approach, when an inconsistent steering transaction is detected, the processes have to rollback and re-execute. This increases the lag significantly, which leads to the large performance difference between the optimistic approach and the conservative approach for inconsistent steering transactions. We also measured the average IM local lag of the optimistic steered approach with a checkpoint size of 10 MB for small transactions and is seen in fig 5.6. The x-axis of these graphs denotes the consistent and inconsistent

optimistic steered (100 KB, 1 MB and 10 MB checkpoint size) computations and

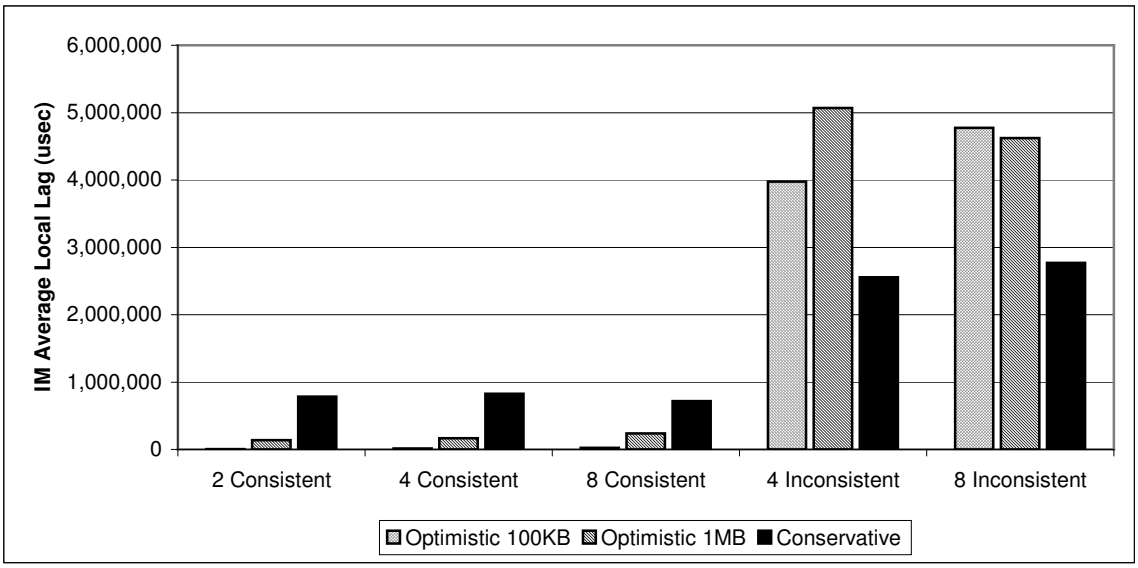

Fig 5.5 Average IM Local Steering Lag for Large Transaction

consistent and inconsistent conservative steered computations for 2, 4 and 8 processes. The y-axis of these graphs denotes the average IM local lag of these computations in microseconds. This figure shows an interesting fact. The IM local lag for the optimistic steered approach with a checkpoint size of 10 MB is much larger than the conservative steered approach even for consistent steering transactions. This is because, in optimistic steering, when the steering transactions are consistent on the first attempt, the IM lag depends mainly on the size of the checkpoints. Hence when the size of the checkpoint is increased to 10MB, the overhead increases dramatically and the lag becomes higher than the conservative steering approach even in the case of consistent steering transactions.

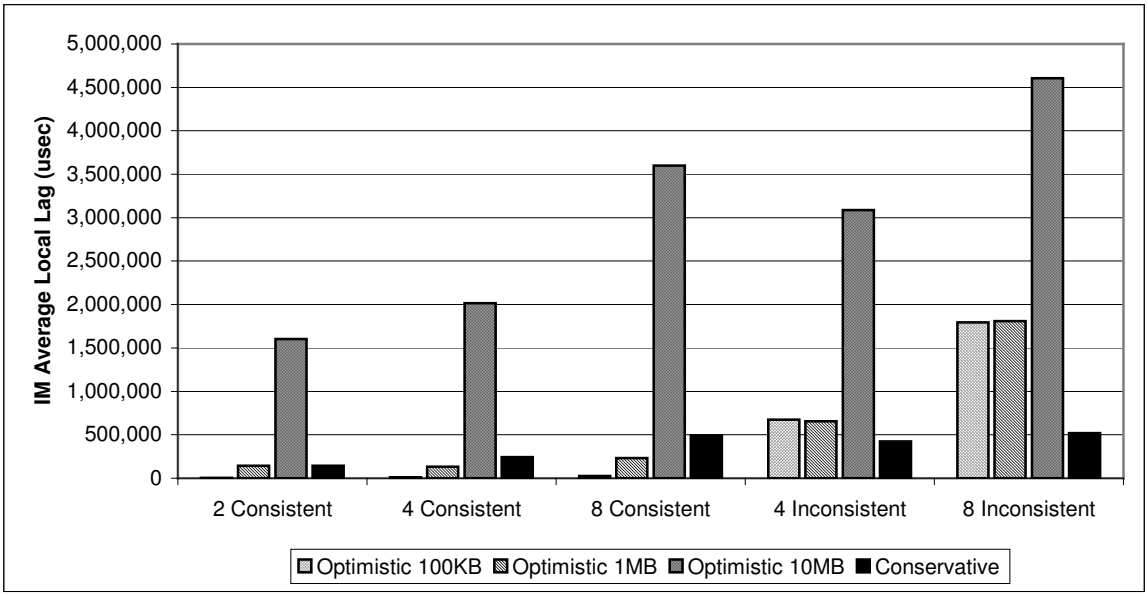

Fig 5.6 Average IM Local Steering Lag for Small Transaction

We also measured the SM lag and compared it with the IM local lag and IM global lag for both small and large transactions. This is seen in fig. 5.7 and fig. 5.8. The x-axes of these graphs denote the consistent and inconsistent optimistic steered (1 MB) computations and consistent and inconsistent conservative steered computations for 8 processes. The y-axes of these graphs denote the average IM local lag, minimum IM local lag, maximum IM local lag, IM global lag and SM lag of these computations in microseconds. From the figure, we understand that, in the optimistic steering approach, when the transaction size is increased, there is a dramatic increase in the IM local lag, IM global lag and in the SM lag. However, in the conservative approach, there is not a significant increase in the lag when the transaction size is increased. This is because, process synchronization and steering changes take roughly the same amount of time for both the small and the large transactions.

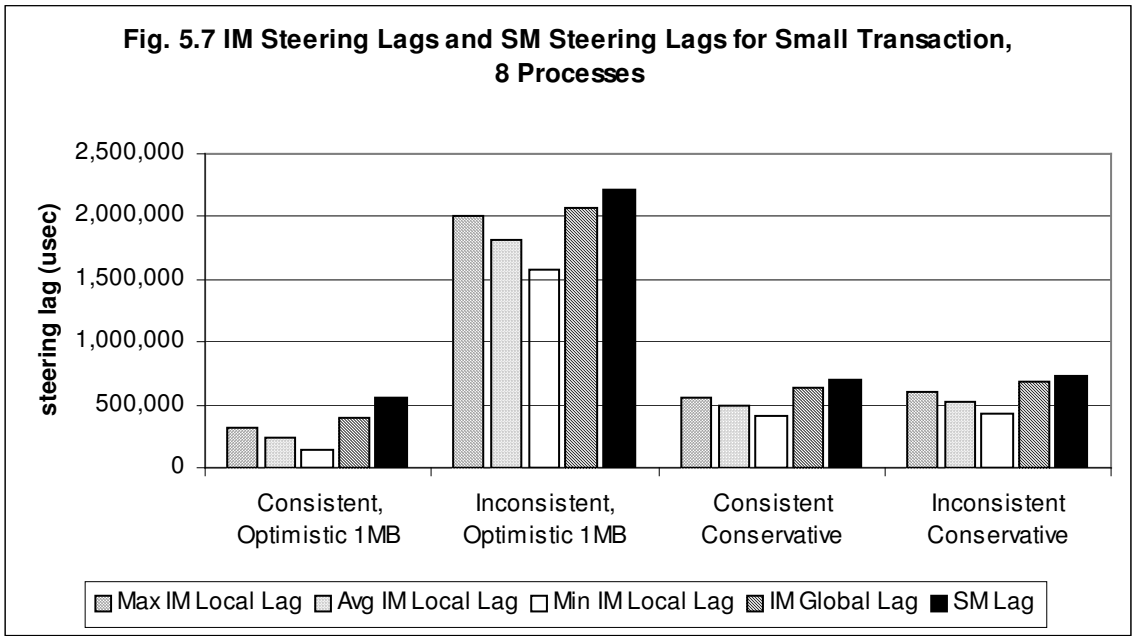

Fig 5.7 IM Steering Lags and SM Steering Lags for Small Transaction 8 Processes

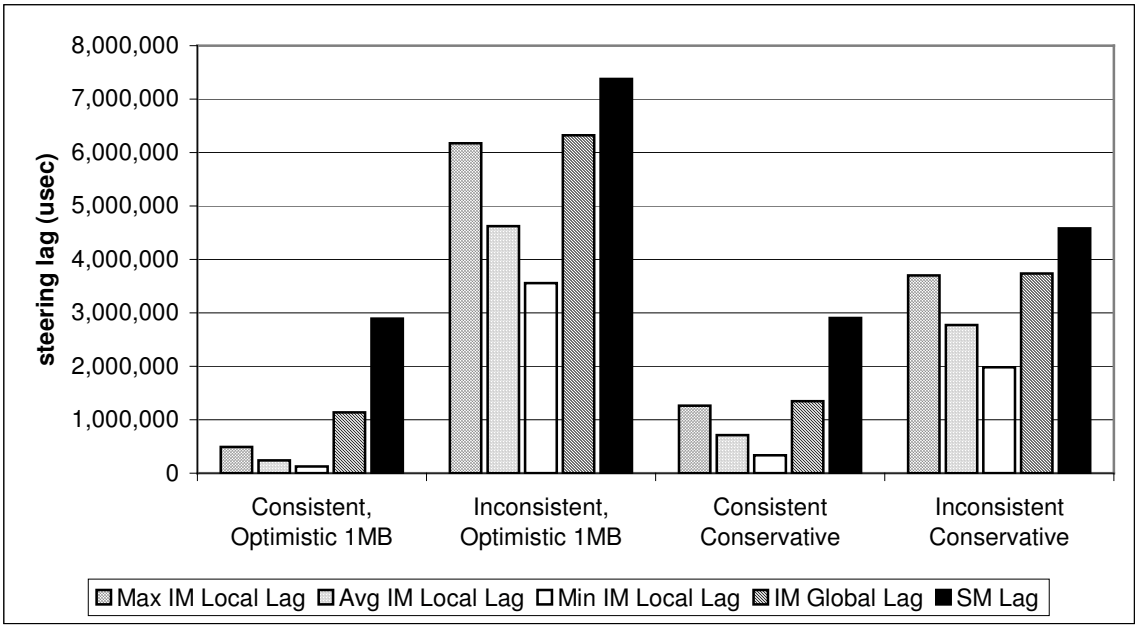

Fig 5.8 IM Steering Lags and SM Steering Lags for Large Transaction 8 Processes

5.4 Summary

From the results we understand that the consistency of steering transactions depend on the number of processes and on the communication pattern. Global communication

patterns have very high consistency while mixed and pair-wise communication patterns have moderate consistency. In the conservative steering approach, the main overhead is the process synchronization, which increases with the number of processes. The main overhead in the optimistic steering approach is the checkpointing process and both the perturbation and lag increase with the size of checkpoints. The optimistic steering approach performs better than the conservative steering approach when the steering transactions are consistent on the first attempt and when the checkpoint size is small (less than or equal to 1 MB). However, if the steering transactions are inconsistent on the first attempt, then conservative steering approach performs far better than the optimistic steering approach. Thus, we believe that the optimistic steering approach can be used in distributed computations where the percentage of consistency on the first attempt is high and the size of checkpoints is not large. Conservative steering is preferred for distributed computations with complex communication patterns and with large checkpoint sizes.

CHAPTER 6

RELATED WORK

In this chapter we describe previous work on computational steering and discuss how our work is different from this previous works.

6.1 Computational Steering Environments

VASE [JBBH93], which stands for **V**isualization and **A**pplication **S**teering **E**nvironment, was developed at University of Illinois and was one of the earliest computational steering environments. VASE provided facilities to alter the values of the key parameters and to add new code at key places in the application. But VASE does not provide any specific support for collaborative steering of multiple processes.

SCIRun [PWJ97][PJ95] stands for Scientific Computing and Imaging Research Group, and was developed at University of Utah. SCIRun is multithreaded and can run on a single multiprocessor system. It provides a visual programming paradigm for the construction of the application and the visualization. But SCIRun does not allow different processes in the steering application to be distributed over multiple machines. And it is cumbersome to integrate existing applications into the system.

Progress [**Prog**ram and **Re**source **S**teering **S**ystem] [VS95] and **Magellan** [VS97] developed at Georgia Institute of Technology are client server steering models. Progress and Magellan were developed to run on multiprocessor systems. But Progress does not support collaborative steering of multiple processes. Magellan, extended from Progress

supports steering of multiple processes but requires consistency points to be placed inside the application by the user.

Another computational steering system **CUMULVS** [GKP97], was developed at Oakridge National Laboratories. In CUMULVS, multiple viewers can start up, interactively "attach" and independently view different fields of regions of the same parallel computation. These same independent viewers can also coordinate updates to steering parameters. To prevent incoherent parameter updates, CUMULVS requires viewers to acquire a token before modifying an application parameter. Hence, CUMULVS uses a token mechanism to maintain consistency, which is different from the way consistency is maintained in Pathfinder.

VIPER [RL97] stands for **VI**sualization of **P**arallel numerical simulation algorithms for **E**xtended **R**esearch and was developed at Technical University of Munich, Germany. VIPER follows a dual server model, which is based on client/server/client architecture. One client is the parallel simulation algorithm; the other client is the visualization system. The dual server extracts the data, transfers and hands it out to the visualization unit. The application program is annotated to identify the data and input parameters as objects. Each object is associated with a synchronization point. When the application encounters such a synchronization point, the dual server is notified which in turn implements the steering changes. VIPER does not provide any support for steering of multiple processes.

Another computational steering system is **CSE** [WL97]. CSE stands for Computational Steering Environment and was developed at the center for Mathematics and Computer Science in Amsterdam, Netherlands. In CSE multiple existing applications can be integrated into the environment by annotation of the application source codes. The

applications and multiple user interface processes can be distributed over different platforms. No strict distinction is made between application developers and end-users. Although the integration of an application requires source code annotation, the construction of the final steering application can be performed by the end-users [MWL99].

Pathfinder system differs from all the above-mentioned systems in the following ways. Pathfinder allows simultaneous steering of multiple processes and Pathfinder has capabilities to use either the optimistic approach or the conservative approach for steering. This allows the user to choose either of the approaches based on the complexity of the computation and to employ the most favorable steering approach for that computation.

In general conservative methods have been found to offer great potential for certain classes of applications, particularly when ample application-specific knowledge of the systems being simulated is available [MRR90]. And optimistic steering approach is better for general-purpose simulations if the state saving overhead is kept within manageable level [Fuj90]. Conservative and optimistic approaches are also used in data replication algorithms. Pessimistic algorithms rely on synchronous replica coordination whereas an optimistic algorithm propagates its updates in the background, discovers conflicts after they happen, and reaches an agreement on the final object contents incrementally [SS02].

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this chapter we summarize our research and discuss about the possible future directions of our research.

7.1 Conclusion

Interactive computational steering provides users with the opportunity to tackle new problems in a way that helps them to learn about the computation in a highly engaging, interactive, visual environment. Consistency guarantees that displays presented to the viewer represent some valid state of the computation and that steering operations are applied in a way that maintains the correctness of the computation. Although numerous interactive computational steering systems exist, only a few address the problem of consistency and only a few support simultaneous steering of multiple processes. Our computational steering system, Pathfinder is unique in that it has both conservative and optimistic steering modules integrated and it also allows simultaneous steering of multiple processes.

In this research, we have developed an algorithm for conservative steering. We have also compared the performance of the conservative approach with that of the optimistic steering approach with regard to perturbation and lag. From the results, we understand that the conservative steering approach is preferred for distributed computations with complex communication patterns and large checkpoint sizes. The optimistic steering

approach is preferred in distributed computations with a smaller checkpoint size and a higher consistency percentage.

7.2 Future Work

At present, the Pathfinder system does not have the potential to handle multiple steering requests being issued at the same time. Handling steering requests one at a time, considering in the order they were issued would be a possible solution to this problem.

Another interesting work would be to reduce the overhead involved with the checkpointing process in the optimistic steering approach. Improving the I/O would be one of the possible solutions to this problem.

REFERENCES

[DS80] Dijkstra and B.P. Scholten, "Termination Detections for Diffusing Computations," *Information Processing Letters*, 1980, 11(1): 1-4.

[Fuj90] R. M. Fujimoto, "Parallel discrete event simulation," *Communication of the ACM*, 33(10): 30-53, October 1990.

[GKM02] J. Guo, E. Kraemer and D.W. Miller, "Consistency Detection in a Transaction-Based Interactive Steering System", *The 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, in submission.

[GKP96] G. A. Geist, J. A. Kohl, and P. M. Papadopoulos, "CUMULVS: Providing fault tolerance, visualization, and steering of parallel applications," *SIAM*, Aug. 1996.

[GR92] J. Gray and A. Reuter, "Transaction Processing: Techniques and Concepts," Morgan Kaufmann, San Mateo, CA, 1992.

[Guo02] J. Guo, "Consistent, interactive steering of distributed computations: algorithms and implementation," Ph.D. thesis, Department of Computer Science, The University of Georgia, August 2002.

[Har00] D. Hart, "Supporting Exploratory Visualization of Distributed Computations," *Ph.D. dissertation*, Department of Computer Science, Washington University, Aug. 2000.

[HKR97] D. Hart, E. Kraemer, and G.-C. Roman, "Interactive Visual Exploration of Distributed Computations," *in Proceedings, 11th International Parallel Processing Symposium,* Geneva, Switzerland, 121-127, Apr. 1997.

[JBBH93] D.J. Jablonowski, J.D. Bruner, B. Bliss and R.B. Haber, "VASE: The Visualisation and Application Steering Environment," *in Proceeding, Super Computing*, 560-69, 1993.

[Lam78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, 558-565, 1978.

[Lyn96] N. Lynch, "Distributed Algorithms", Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[MGK01] D.W. Miller, J. Guo, E. Kraemer and Y. Xiong, "On-the-Fly Calculation and Verification of Consistent Steering Transactions," *Supercomputing Conference* (SC2001), Denver, CO.

[MRR90] B. Merrifield, S. Richardson, and J. Roberts, "Quantitative studies of discrete event simulation modeling of road traffic," *Proceedings of the SCS MultiConference on Distributed Simulation*, 22(1):188-193, 1990.

[MWL99] J. Mulder, J. van Wijk, and R. van Liere, "A Survey of Computational Steering Environments," *Future Generation Computer Systems*, 15(2):91-102, 1999.

[PJ95] S. Parker and C. Johnson, "SCIRun: A Scientific Programming Environment for Computational Steering." *In Proceedings of SuperComputing '95*, 1995.

[PWJ97] S.G. Parker, D.M. Weinstein, and C.R. Johnson, "The SCIRun Computational Steering Software System." In E. Arge, A.M. Bruaset, and H.P. Langtangen, editors, *Modern Software Tools for Scientific Computing*, pages 1-40. Birkauser Verlag AG, Switzerland, 1997.

[RL97] S. Rathmayer and M. Lenke, "A Tool for On-line Visualization and Interactive Steering of Parallel HPC Applications." *In Proceedings of the 11th International Parallel Processing Symposium, IPPS 97*, pages 181-186, 1997.

[SS02] Yasushi Saito and March Shapiro, "Replication: Optimistic Approaches, *Technical Report,* HP Labs, 2002.

[VKD00] H. Vuppula, E. Kraemer, and D. Hart, "Algorithms for Collection of Global Snapshots: An Empirical Evaluation," *ISCA Conference on Parallel and Distributed Computing*, Las Vegas, NV, 197-204, Aug. 2000.

[VS95] J. Vetter and K. Schwan, "PROGRESS: A Toolkit for Interactive Program Steering," *in Proc. 24th Ann. Conf. on Parallel Processing,* 1995.

[VS97] J. Vetter and K. Schwan, "High performance computational steering of physical simulations," *in Proceedings of the 11th International Parallel Processing Symposium*, Geneva, Switzerland, Apr. 1997.

[WL97] J.J. van Wijk and R. van Liere, "An Environment for Computational Steering." In G.M. Nielson, H. Muller, and H. Hagen, editors, *Scientific Visualization: Overviews, Methodologies, and Techniques,* pages 89-110. Computer Society Press, 1997.