분산 시스템에서 가상 체크포인팅을 이용한 비동기화 체크포인팅 알고리즘

김 도 형[†] · 박 창 순[††] · 김 종[†††]

요 약

체크포인팅은 결함을 극복하고, 작업의 빠른 재수행을 위해 많이 사용되는 결함포용 기술 중의 하나이다. 분산 시스템에서의 체크포인팅 알고리즘들은 오랫동안 연구되어 왔다. 현재까지 제시된 체크포인팅 알고리즘들은 크게 동기화 체크포인팅 알고리즘과 비동기화 체크포인팅 알고리즘으로 나눌 수 있다. 본 논문에서는 정기적 체크포인팅 알고리즘과 동일한 최소의 체크포인트 횟수와 결함 발생시 비교적 짧은 롤백 거리를 갖는 비동기화 체크포인팅 알고리즘을 제시하였다. 체크포인트 횟수는 결함이 발생하지 않은 환경에서 작업 완료 시간과 직접적인 관계가 있고, 짧은 롤백 거리는 결함이 발생한 환경에서 작업 완료 시간과 밀접한 관계가 있다. 제안된 체크포인팅 알고리즘을 시뮬레이션을 통해 이전에 제안된 비동기화 체크포인팅 알고리즘들과 비교한 결과, 제안된 체크포인팅 알고리즘이 결함이 발생하지 않은 환경과 결함이 발생한 환경에서 더 좋은 결과를 보여주었다.

An Asynchronous Checkpointing Algorithm Using Virtual Checkpointing On Distributed Systems

Do-Hyung Kim[†] · Chang-Soon Park[††] · Jong Kim[†††]

ABSTRACT

Checkpointing is the one of fault-tolerant techniques to restore faults and to restart job fast. Checkpointing algorithms in distributed systems have been studied for many years. These algorithms can be classified into synchronous Checkpointing algorithms and asynchronous Checkpointing algorithms. In this paper, we propose an independent Checkpointing algorithm that has a minimum Checkpointing counts equal to periodic Checkpointing algorithm, and relatively short rollback distance at faulty situation. Checkpointing count is directly related to task completion time in a fault-free situation and short rollback distance is directly related to task completion time in a faulty situation. The proposed algorithm is compared with the previously proposed asynchronous Checkpointing algorithms using simulation. In the simulation, the proposed Checkpointing algorithm produces better results than other algorithms in terms of task completion time in fault-free as well as faulty situations.

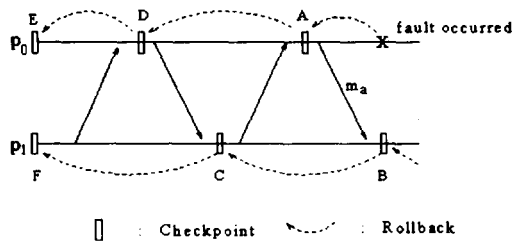
1. 서 론

분산 시스템 설계에서 중요한 주제 중의 하나는 작업 수행 중 결함이 발생하더라도 작업을 처음부터 다시 수행하지 않고, 결함을 복구하여 작업 완료 시간을 줄이는 것이다. 체크포인팅(checkpointing)은 결함을 극복하고, 작업의 빠른 재 수행을 위해 많이 사용되는 결함포용 기술 중의 하나이다[6]. 체크포인팅이란 작업 수행 중에 프로세스의 현 상태를 안정적 저장장치(stable storage)에 기록하는 것을 말한다. 체크포인팅

† 정 회 원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 연구원
†† 정 회 원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 책임기술원
††† 정 회 원 : 포항공과대학교 전산학과 교수
논문접수 : 1998년 8월 2일, 심사완료 : 1999년 3월 11일

에서의 결함복구는 결함이 발생했을 때 가장 최근의 체크포인트(checkpoint) 한 위치로 되돌아가 작업을 재수행함으로써 이루어진다. 체크포인팅은 결함 발생시 결함을 복구하여 작업 완료 시간을 단축시킬 수 있는 방법이지만, 분산 시스템에서의 체크포인팅은 도미노 효과(domino effect)를 발생시킬 수 있다.

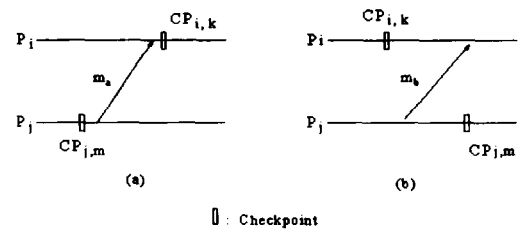
도미노 효과란, 수행 중인 한 프로세스에서 발생한 결함이 다른 프로세스들의 수행을 연속적으로 되돌아가게 함으로써, 최종적으로 모든 프로세스들의 수행이 처음으로 되돌아가는 것을 말한다[1-6]. (그림 1)은 도미노 효과를 보여준다.



(그림 1) 도미노 효과

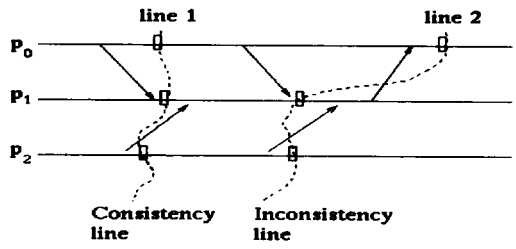
프로세스 P_0는 결함으로 인해 가장 최근의 체크 포인트인 A로 되돌아가게 된다. P_0의 체크포인트(checkpoint) A에는 P_1으로 메시지 ma를 전송하였다는 기록은 존재하지 않는다. 하지만, P_1의 체크포인트 B는 P_0로부터 메시지 ma를 수신하였다고 기록하였다. 즉, P_1의 체크포인트 B는 P_0의 체크포인트 A에서는 보내지도 않은 메시지를 받은 것이다. 결과적으로, P_0와 P_1의 두 체크포인트 A, B에 기록된 내용은 서로 모순된다. 따라서, P_1은 P_0로부터 메시지 ma를 받지 않은 상태, 즉 체크포인트 C로 되돌아간다. 그러면, 같은 상황이 P_0의 체크포인트 A와 P_1의 C 사이에서도 발생하게 된다. 따라서, P_0는 체크포인트 D로 되돌아가게 된다. 결국 P_0와 P_1은 최초의 체크포인트 E와 F로 되돌아가게 된다. 도미노 효과는 결함이 발생했을 때, 작업을 처음부터 다시 수행하여야 하기 때문에 바람직한 현상이 아니다. 따라서, 체크포인팅에서 중요한 문제 중의 하나는 프로세스들의 체크포인트를 이용하여 결함발생시 도미노 효과를 방지할 수 있는 일관성 있는 시스템 상태를 형성하는 것이다.

일관성 있는 시스템 상태를 정의하기 위해서, 지역(local) 체크포인트와 전역(global) 체크포인트를 정의한다[1-5]. 지역 체크포인트는 각 프로세스의 개별적인 체크포인트를 말하고, 전역 체크포인트는 각 프로세스마다 하나의 지역 체크포인트를 모아놓은 체크포인트들의 집합을 말한다. 일관성 있는 시스템 상태를 정의하기 위해서는 프로세스들의 체크포인트 사이에 형성되는 고아(orphan) 메시지와 실종(missing) 메시지를 고려해야 한다. 전송 프로세스의 체크포인트 이후에 전송되고, 수신 프로세스의 체크포인트 이전에 처리된 메시지를 고아 메시지라 한다. 전송 프로세스의 체크포인트 이전에 전송되고 수신 프로세스의 체크포인트 이후에 도달하는 메시지를 실종 메시지라 한다. (그림 2) (a)의 메시지 ma가 고아 메시지이고, (그림 2) (b)의 메시지 mb가 실종 메시지이다. 실종 메시지를 흔히 분실(lost) 메시지, 혹은 채널 상태(channel-state) 메시지라고도 한다.



(그림 2) 체크포인트 일관성 (a) 고아 메시지 (b) 실종 메시지

본 논문에서는 실종 메시지가 항상 수신 프로세스에 전달되어 안정적 저장장치에 기록된다고 가정하기 때문에 (그림 2) (b)의 두 체크포인트 $CP_{i,k}$와 $CP_{j,m}$은 일관성 있는 시스템 상태를 형성한다[1-16]. 하지만, (그림 2) (a)의 두 체크포인트 $CP_{i,k}$와 $CP_{j,m}$는 일관성 있는 시스템 상태를 형성하지 못한다. 따라서, 일관성 있는 시스템 상태란 전역 체크포인트 내의 임의의 두 체크포인트들 사이에 고아 메시지가 존재하지 않는 것을 말한다[1-16]. 일반적으로, 일관성 있는 시스템 상태는 프로세스들의 체크포인트를 연결하는 하나의 라인으로 나타나는데, 이 라인을 복구(recovery) 라인이라 한다. 즉, 결함발생시 각 프로세스는 복구라인에 속한 자신의 체크포인트로부터 작업을 다시 수행하면 된다. (그림 3)은 복구 라인을 보여준다.



(그림 3) 복구 라인

대부분의 체크포인팅 알고리즘들은 어떻게 체크포인트 횟수를 줄이면서, 복구 라인을 최근의 수행지점(결함이 발생하였을 때의 수행지점)에 가깝게 형성할 것인가를 다루고 있다. 체크포인트 횟수가 적다는 것은 체크포인트로 인한 수행지연이 적다는 것이고, 복구 라인이 최근의 수행지점에 가깝다는 것은 결함으로 인한 롤백(rollback) 거리가 작다는 사실과 관계가 있다. 본 논문에서는 각 프로세스들이 최소의 체크포인트 횟수를 가지면서도, 결함발생시 비교적 짧은 롤백 거리를 가지는 체크포인팅 알고리즘을 제시한다. 본 논문은 다음과 같이 구성되어 있다. 2장에서는 이전의 체크포인팅 연구들에 대해 소개하고, 3장에서는 본 논문에서 고려하는 시스템 모델과 용어들에 대해 다룬다. 4장에서는 본 논문에서 제시된 체크포인팅 알고리즘을 소개하고, 5장에서는 본 논문에서 제시된 체크포인팅 알고리즘의 성능을 조사하기 위한 시뮬레이션 방법과 결과를 보여준다. 마지막으로, 6장에서는 본 논문을 요약한다.

2. 이전의 체크포인팅에 관한 연구

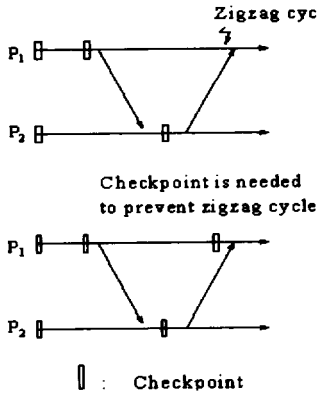
현재까지 다양한 체크포인팅 알고리즘들이 제안되어 왔는데, 체크포인팅 알고리즘들은 크게 동기화(synchronous) 체크포인팅 알고리즘과 비동기화(asynchronous) 체크포인팅 알고리즘으로 나눌 수 있다[1-16]. 동기화 체크포인팅 알고리즘은 공동으로 작업을 수행하는 프로세스들이 체크포인트 시에 서로 협력하는 방식이다. 즉, 한 프로세스가 체크포인트 할 때 다른 프로세스들에게도 체크포인트 하도록 요청한다. 동기화 체크포인팅 알고리즘은 결함 발생시 복구가 빠르고, 도미노 효과를 발생시키지 않는다는 장점이 있다. 하지만, 동기화 체크포인팅 알고리즘은 체크포인트 시에 프로세스들을 동기화 하여야 하기 때문에 동기화로 인한 수행지연이 발생한다는 단점이 있다. 이 단점은 빠른 작업 수행을 요구하는 경우에 심각한 문제가 될 수 있다. 비동기화 체크포인팅 알고리즘은 각 프로세스가 다른 프로세스와 독립적으로 체크포인트 한다. 따라서, 동기화 체크포인팅 알고리즘보다 체크포인트 시에 수행지연이 적다는 장점이 있다. 하지만, 비동기화 체크포인팅 알고리즘은 결함 발생시 결함 복구 시간이 크고, 도미노 효과가 발생할 수 있다는 문제점이 있다.

모든 체크포인팅 알고리즘들은 결함 발생시 프로세스들의 재 수행을 빠르게 하기 위해서 메시지 로깅을 사용한다[1-16]. 메시지 로깅이란 프로세스가 메시지를 결함으로부터 안전한 저장장치에 기록하는 것을 말한다. 메시지 로깅은 크게 비관적(pessimistic) 메시지 로깅과 낙관적(optimistic) 메시지 로깅으로 나눌 수 있다. 비관적 메시지 로깅은 메시지를 안정적 저장장치에 기록할 때까지 프로세스 수행을 정지시키는 방식을 말한다. 낙관적 메시지 로깅은 프로세스 수행이 정지되지 않고, 메시지가 안정적 저장장치에 기록되는 방식을 말한다. 최근에 비관적과 낙관적 메시지 로깅의 장점을 가지는 Causal 메시지 로깅 방식이 제안되기도 하였다[16].

동기화 체크포인팅 알고리즘과 비동기화 체크포인팅 알고리즘에서는 정기적(periodic) 체크포인팅 알고리즘이 기본적으로 사용된다. 정기적 체크포인팅 알고리즘은 각 프로세스가 일정한 시간 간격마다 체크포인트 하는 방식이다. 하지만, 단순한 정기적 체크포인팅 알고리즘의 사용은 많은 수의 고아 메시지를 유발할 수 있다. 따라서, 기존의 체크포인팅 알고리즘들은 정기적 체크포인팅을 기본적으로 사용하면서, 고아 메시지를 방지할 수 있는 부가적인 체크포인팅 기술들을 제공한다. 최근에 메시지를 이용하여 고아 메시지를 제거할 수 있는 비동기화 체크포인팅 알고리즘들이 제안되었다.

Xu등이 제안한 적응적(adaptive) 체크포인팅 알고리즘[2]은 입력 메시지에 의한 지그잭(zigzag) 사이클 존재유무를 이용하여 체크포인트 한다. 제시된 알고리즘은 입력 메시지가 지그잭 사이클을 형성하면, 도미노 효과를 발생시킬 수 있다는 관찰에 기본을 두고 있다. 지그잭 사이클은 지그잭 경로(path)가 사이클을 형성하는 것으로, 지그잭 경로는 사건(event)들 사이에 형

성되는 happend-before relation[4]을 확장한 것이다. 여기서 사건이란, 메시지 전송(send), 수신(receive), 그리고 체크포인트를 말한다. 따라서, 적응적 체크포인팅 알고리즘에서는 입력 메시지가 지그잭 사이클을 형성하면, 수신 프로세스는 메시지를 처리하기 전에 체크포인트하여 지그잭 사이클을 제거한다. (그림 4)는 적응적 체크포인팅 알고리즘을 간략하게 보여준다. 적응적 체크포인팅 알고리즘은 지그잭 사이클을 방지함으로써 일관성 있는 시스템 상태를 형성하지만, 프로세스간 메시지 전송이 많을 때 체크포인트 횟수가 증가하고, 지그잭 사이클을 찾기 위한 부하가 커지게 된다. 그리고, 지그잭 사이클을 완벽하게 발견하지 못하기 때문에 도미노 효과가 발생할 수 있다.



(그림 4) 적응적 체크포인팅 알고리즘

Wang 등의 연구[3]에서는 체크포인트 번호를 이용하는 나태(lazy) 체크포인팅 알고리즘이 제시되었다. 메시지 전송 프로세스는 전송되는 메시지에 자신의 체크포인트 번호를 전송 메시지에 첨가하고, 수신 프로세스는 메시지를 처리하기 전에 메시지에 첨가된 전송 프로세스의 체크포인트 번호와 자신의 체크포인트 번호를 비교한다. 만약 전송 프로세스의 체크포인트 번호가 더 크면, 메시지를 처리하기 전에 체크포인트 한다. 이때, 두 프로세스의 체크포인트 번호는 같게 된다. 나태 체크포인팅 방법은 체크포인트 번호 만을 이용하므로 비교적 간단하고 체크포인팅 시에 부하를 줄일 수 있지만, 프로세스간 수행 속도 차이가 크거나, 메시지 전송이 많을 때 메시지로 인한 체크포인트가 많아지고, 도미노 효과가 발생할 수 있다는 문제점이 있다

본 논문에서는 나태와 적응적 체크포인팅 알고리즘들의 단점을 개선하여 프로세스간 수행 속도 차이와 메시지 전송량에 관계없이 정기적 체크포인팅 알고리즘과 동일한 최소의 체크포인트 횟수를 가지고, 결함 발생시 비교적 짧은 롤백 거리를 가지는 비동기화 체크포인팅 알고리즘을 제시한다.

3. 시스템 모델과 용어

이 절에서는 본 논문에서 가정하고 있는 시스템 모델에 대해 소개하고, 관련 용어들에 대해서 다룬다.

3.1 시스템 모델

본 논문에서 고려하고 있는 시스템에서는 다수의 프로세스들이 하나의 작업을 동시에 수행하고, 프로세스들 사이의 통신은 오로지 메시지를 통해서만 일어난다. 각 프로세스들은 fail-stop 프로세서에서 수행되고, 두 메시지 수신 사이의 프로세스 수행이 비결정적(non-deterministic)이라고 가정한다. 프로세스의 수행이 결정적이라고 가정한 경우에는 프로세스가 결함복구 후 재수행 시에 생성되는 메시지 전달환경이 결함이 발생하기 전과 항상 동일하다고 본다. 하지만, 프로세스의 수행이 결정적이라는 가정은 모든 연산 수행 환경에 적용되지는 못한다. 프로세스들 사이에 전달된 메시지는 손실없이 항상 전달되고, 슬라이딩 윈도우(sliding window)와 같은 통신 프로토콜에 의해 FIFO 형식을 따른다고 가정한다. 또, 시스템은 비동기적(asynchronous)으로 동작한다고 가정한다.

각 프로세스는 메시지 전송 시 자신의 현 체크포인트 번호를 메시지에 첨가하고, 자기의 상태 정보를 메모리와 안정적 저장장치에 기록한다. 메모리에 저장된 프로세스 정보는 결함 발생시에 손실될 수 있지만, 안정적 저장장치에 기록된 정보는 결함에 대해 안전하다고 가정한다. 그리고, 각 프로세스는 수신 메시지 저장을 위해 낙관적 메시지 로깅을 사용한다고 가정한다.

3.2 용어

본 논문에서 사용하는 용어는 다음과 같다. 체크포인트 간격(interval)은 각 프로세스의 연속된 체크포인트 사이의 간격을 말한다. 각 프로세스의 현재(current) 체크포인트 간격이란, 각 프로세스의 가장 최근 체크포인트와 현재의 수행지점 사이의 간격을 말한다.

MAXCPT(Maximum Checkpoint Time)는 두 체크포인트 사이의 최대 시간을 말한다. 즉, 각 프로세스는 MAXCPT 시간마다 정기적으로 체크포인트 한다. 그리고, 메모리에 프로세스 정보를 저장하는 체크포인트를 가상(virtual) 체크포인트라 하고, 안정적 저장장치에 기록하는 체크포인트를 실제(real) 체크포인트라 한다.

4. 제시된 체크포인팅 알고리즘

본 논문에서 제시된 체크포인팅 알고리즘은 정기적 체크포인팅 알고리즘과 같이 일정한 시간 간격(MAXCPT)마다 1번씩 체크포인트 한다. 하지만, 정기적 체크포인팅 알고리즘과 달리 메시지를 통해 발생하는 프로세스간 의존 관계를 가상 체크포인트를 통해 제거하여, 결함발생시 비교적 짧은 롤백 거리를 가진다. 그리고, 제시된 체크포인팅 알고리즘은 기존의 적응적과 나태 체크포인팅 알고리즘과 같이 전송 메시지를 이용하여 프로세스 간의 의존성을 제거한다. 하지만, 프로세스간의 메시지 전송량과 수행속도 차이에 따라 많은 수의 체크포인트를 유발하는 적응적과 나태 체크포인팅 알고리즘의 단점을 개선하여, 각 프로세스의 체크포인트가 일정한 시간 간격마다 1번씩 일어나게 함으로써, 체크포인트로 인한 수행지연을 상당히 감소시킨다. 따라서, 제안된 체크포인팅 알고리즘은 결함이 발생하지 않은 환경과 결함이 발생한 환경에서 빠른 작업 완료 시간을 제공한다. 본 논문에서 제시된 체크포인팅 알고리즘은 (그림 5)와 같다.

제시된 체크포인팅 알고리즘에서 각 프로세스는 일정시간(MAXCPT)마다 정기적으로 체크포인트 한다.

```
/* for process pi ,  0 <= i <  N */
While (pi is not ended) {
        /* normal operation */
    /* pi has to checkpoint */
    If (current_time == Checkpoint_time)  {
            If (virtual checkpoint is done in current interval) {
                    /* write virtual checkpointed contents of memory to stable storage */
                    Take a checkpoint with virtually checkpointed contents;
            }
            else {
                    /* write current process state to stable storage */
                    Take a checkpoint with current process state;
            }
            Increase checkpoint number by 1;
            Checkpoint_time = current_time + MAXCPT ;
            Unset virtual checkpoint exists;
    }

    /* if pi has to process a received message */
    If (senders checkpoint number > current checkpoint number) {
            If (virtual checkpoint is existed in current interval)
                    Set current checkpoint number to senders;
            else {
                    /* write current process state to memory */
                    Take a virtual checkpoint with current process state;
                    Set current checkpoint number to senders;
                    Set virtual checkpoint exists;
            }
    }
    Process a received message;
}
```

(그림 5) 제안된 체크포인팅 알고리즘

이때 각 프로세스는 체크포인트 하기 전에 현 체크포인트 간격 내에 가상 체크포인트가 존재하는 지 조사한다. 만약 가상 체크포인트가 존재한다면, 가상 체크포인트 내용을 안정적 저장장치에 기록한다. 만약 가상 체크포인트가 존재하지 않는다면, 현재의 프로세스 상태 정보를 안정적 저장장치에 기록한다. 체크포인트 후에, 각 프로세스의 체크포인트 번호는 1 만큼 증가되고, 다음 체크포인트 시간도 변경된다(Checkpoint_time = current_time + MAXCPT).

각 프로세스는 다른 프로세스로부터 수신된 메시지를 처리하기 전에 체크포인트 한다. 각 프로세스는 수신 메시지를 처리하기 전에 자신의 현 체크포인트 번호와 메시지 전송 프로세스의 체크포인트 번호를 비교한다. 만약 전송 프로세스의 체크포인트 번호가 더 크다면, 현재 체크포인트 간격 내에 가상 체크포인트가 존재하는지 조사한다. 만약 존재한다면, 자신의 현 체크포인트 번호를 메시지 전송 프로세스의 체크포인트 번호로 일치시키고, 수신 메시지를 처리한다. 만약 가상 체크포인트가 존재하지 않는다면, 메시지를 처리하기 전에 현 프로세스 상태 정보를 메모리에 저장(가상 체크포인트)하고, 체크포인트 번호를 메시지 전송 프로세스의 체크포인트번호와 일치시킨 후에 메시지를 처리한다. 만약 메시지 수신 프로세스의 체크포인트 번호가 전송 프로세스보다 크거나 같다면, 수신 메시지를 바로 처리하게 된다.

본 논문에서 제시된 체크포인팅 알고리즘은 프로세스들간의 수행속도 차이와 메시지 전송 비율에 따라 체크포인트 횟수가 크게 증가하는 적응적 체크포인팅 알고리즘[2]과 나태 체크포인팅 알고리즘[3]의 단점을 개선하여, 체크포인트가 일정한 시간 간격마다 1번씩 일어나도록 하여 정기적 체크포인팅 알고리즘과 동일한 최소의 체크포인트 횟수를 가진다. 일반적으로 결함이 발생하지 않은 환경에서는 체크포인트 횟수가 적을수록 체크포인트로 인한 수행지연이 적기 때문에 빠른 시간 내에 작업을 종료할 수 있다. 물론, 본 논문에서 제안한 체크포인팅 알고리즘은 가상 체크포인트로 인하여 수행지연이 발생할 수 있다. 하지만, 가상 체크포인트로 인한 수행지연은 실제 체크포인트로 인한 수행지연보다 상당히 작기 때문에, 결함이 발생하지 않은 환경에서 비교적 빠른 작업 종료시간을 가지게 된다.

그리고, 본 논문에서 제시한 체크포인팅 알고리즘을 사용할 경우, 안정적 저장장치에는 메시지 수신으로 인한 프로세스간 의존관계가 제거된 프로세스 상태 정보들이 저장된다. 일반적으로 결함발생시 롤백 거리는 프로세스들간의 의존관계에 따라 달라지게 된다. 즉, 메시지로 인한 프로세스간의 의존관계가 클수록 결함발생시 롤백 거리는 증가하고, 의존관계가 작을수록 롤백 거리는 줄어들게 된다. 따라서 본 논문에서 제시된 체크포인팅 알고리즘은 결함발생 시 비교적 짧은 롤백 거리를 가지게 된다. 물론, 메모리에 저장된 가상 체크포인트 정보가 결함으로 인해 손실될 경우에는 결함복구 시 복구라인을 형성할 때 이전의 실제 체크포인트를 이용해야 하므로, 롤백 거리가 나태 체크포인팅 알고리즘보다 커질 수 있다. 하지만, 체크포인트로 인한 수행지연이 매우 적기 때문에 결함발생시 빠른 시간 안에 결함을 복구하여 작업을 완료하게 된다. 본 논문에서 사용하는 결함 복구 알고리즘은 Wang 등이 사용한 결함 복구 알고리즘[3]과 동일하다. 결함 복구 알고리즘은 본 논문을 이해하는데 문제가 되지 않으므로, 자세한 복구 알고리즘은 생략하기로 한다.

5. 성능비교

이 장에서는 본 논문에서 제시된 체크포인팅 알고리즘과 정기적 체크포인팅, 적응적 체크포인팅, 그리고 나태 체크포인팅 알고리즘들을 시뮬레이션을 통해 서로 비교하였다. 정기적 체크포인팅 알고리즘은 각 프로세스가 일정한 시간 간격마다 체크포인트 한다. 적응적 체크포인팅 알고리즘은 각 프로세스가 정기적으로 체크포인트 하고, 수신 메시지가 지그잭 사이클을 형성하게 되면 메시지를 처리하기 전에 체크포인트 한다. 그리고 나태 체크포인팅 알고리즘은 각 프로세스가 정기적으로 체크포인트 하고, 메시지 전송 프로세스의 체크포인트 번호가 더 크면 메시지를 처리하기 전에 체크포인트 한다. 시뮬레이션에서는 각 체크포인트에 일정한 지연시간을 주고, 결함이 발생하지 않았을 때의 평균 작업 완료 시간을 측정하였다. 그리고, 하나의 결함이 발생하였을 때 결함을 극복하고, 작업을 완료하기까지의 시간을 측정하였다. 평균 작업 완료 시간은 실제로 하나의 작업을 수행할 때, 얼마나 빨리 완료될 수 있는가를 보여 줄 수 있기 때문에, 알고리즘들을 비교하는데 적합한 요소이다.

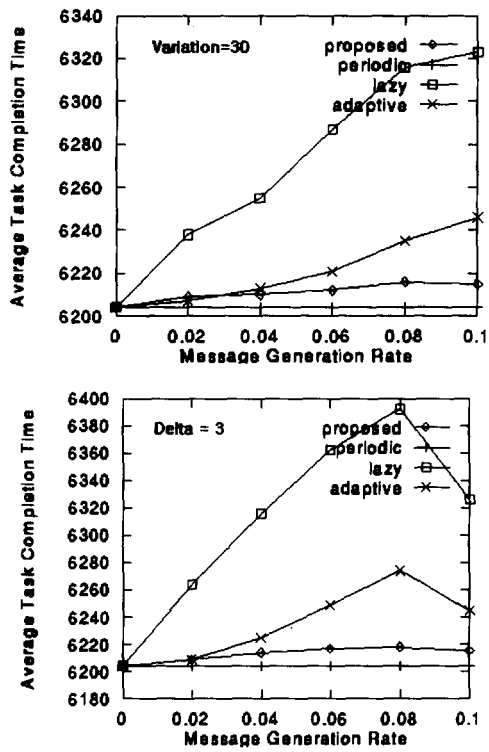
실험방법은 다음과 같다. 각 체크포인팅 알고리즘에 동일한 메시지를 전달하고, 이를 버퍼에 저장한 다음,

각 알고리즘의 특성에 따라 그 메시지를 처리하도록 하였다. 메시지 생성 확률은 0에서 0.1까지 0.02간격으로 변화 시켰다. 실험에서는 각 프로세스가 매 시간마다 주어진 메시지 생성 확률을 이용하여 메시지를 전송할 것인가를 결정하여, 임의의 다른 프로세스에게로 메시지를 전송하게 된다. 실험에서 사용한 프로세스 개수는 총 10개이고, 시뮬레이션에서는 프로세스의 수행 속도 차이를 나타내기 위해서 두 가지 방법들이 사용되었다. 첫 번째는 각 프로세스의 체크포인트 간격이 MAXCPT - 프로세스 번호(0~9) * λ로 이루어진다. 이 방식은 두 프로세스의 속도가 비슷해져도 작업 수행 중에 프로세스 수행속도를 차이나게 할 수 있다[4]. 두 번째 방법에서는 각 프로세스의 체크포인트 간격이 MAXCPT - random()이 되도록 하는 것이다. 여기서, λ와 ε을 체크포인트 보충값(offset)이라 한다. 실험에서는 λ=3, ε=30, 실제 체크포인트 지연 시간 = 10, 가상 체크포인트 지연 시간 = 1, MAXCPT = 300, 작업 시간 = 6000으로 하였다. 실험에서는 프로세스들의 수행속도 차이가 체크포인트 시간(MAXCPT)의 10%내에서 이루어지도록 λ, ε 값을 결정하였다[4]. 가상 체크포인트 시간은 실제 체크포인트 시간에 비해 매우 작지만, 가상 체크포인트로 인한 수행 지연을 실험에 반영하기 위해서 임의로 실제 체크포인트 시간의 10%로 하였다. 체크포인트 인터벌 MAXCPT는 대략 전체 수행 시간의 5%정도로 임으로 결정하였다. 그리고, 실험에서는 각 메시지 전달 확률마다 작업을 10번씩 반복수행하여 평균 작업 완료시간을 측정하였는데, 작업을 반복수행 할 때마다 이전 수행과 다른 메시지 전달환경이 프로세스들 사이에 생성된다. 마지막으로, 실험은 선(Sun) SPARC 20상에서 진행되었고, 전송 메시지에 전송 프로세스의 체크포인트 번호를 첨가하는 지연시간과 지그잭 사이클을 발견하기 위한 지연시간은 실험에서 고려하지 않았다.

5.1 결함이 발생하지 않았을 때의 평균 작업 완료 시간

이 실험은 결함이 발생하지 않았을 때의 각 알고리즘의 평균 작업 완료 시간을 비교하기 위한 것이다. (그림 6)은 실험결과를 나타낸 것이다.

실험결과 결함이 발생하지 않은 환경에서 제시된 체크포인팅 알고리즘과 정기적 체크포인팅 알고리즘이 가장 작은 평균 작업 완료 시간을 가지고, 나태 체크포인팅 알고리즘이 가장 큰 평균 작업 완료 시간을 가짐을



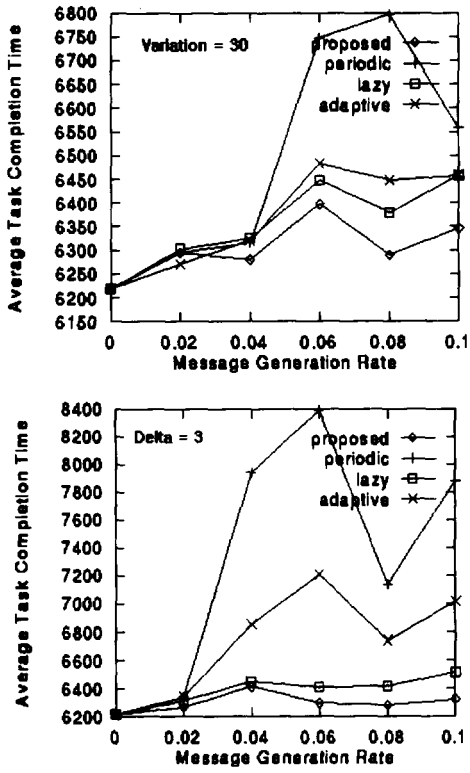
(그림 6) 결함이 발생하지 않았을 때의 평균 작업 시간

알 수 있다. 결함이 발생하지 않았을 때의 평균 작업 완료 시간은 체크포인트 횟수에 크게 의존한다. 제시된 체크포인팅 알고리즘은 정기적 체크포인팅 알고리즘과 동일한 횟수의 실제 체크포인트를 가지지만, 가상 체크포인트로 인한 수행지연 때문에 정기적 체크포인팅 알고리즘보다 더 큰 작업 완료 시간을 가지게 된다. 나태와 적응적 체크포인팅 알고리즘에서는 메시지 전송 비율에 따라 체크포인트 횟수가 많아지기 때문에, 평균 작업 완료 시간이 증가하게 된다. 이 실험결과로부터 결함이 발생하지 않는 환경에서는 정기적 체크포인팅 알고리즘과 제시된 체크포인팅 알고리즘이 유리함을 알 수 있다.

5.2 하나의 결함복구 후의 평균 작업 완료 시간

이 실험은 작업 수행 중에 하나의 결함이 발생하고, 이 결함을 극복한 후에 작업을 완료하기까지의 시간을 비교하기 위한 것이다. (그림 7)은 실험 결과를 나타낸다.

실험결과, 제시된 체크포인팅 알고리즘이 가장 작은 평균 작업 완료 시간을 가지고, 정기적 체크포인팅 알



(그림 7) 하나의 결함 복구 후의 작업 완료 시간

고리즘이 가장 큰 작업 완료 시간을 가짐을 알 수 있다. 정기적 체크포인팅 알고리즘은 순수 체크포인트에 드는 시간은 작지만, 결함발생 시 롤백 거리가 크기 때문에 제일 나쁜 결과를 얻었다. 나태 체크포인팅 알고리즘은 결함발생 시 롤백 거리는 제일 작지만, 많은 체크포인트 횟수로 인한 수행지연 때문에 제시된 알고리즘보다 더 큰 작업 완료 시간을 가지게 된다. 본 논문에서 제시된 체크포인팅 알고리즘은 프로세스 간 메시지 전송으로 인한 의존성을 줄이기 위해서, 메시지 처리 전에 가상 체크포인트를 하게 되고, 체크포인트 번호를 일치시킴으로써 프로세스 간의 수행 속도 차이를 줄이게 된다. 결국, 가상 체크포인트는 결함복구 시 프로세스간의 의존성에 의한 롤백 거리를 줄이는 역할을 수행함으로써, 정기적 체크포인팅과 같은 최소의 체크포인팅 횟수로 보다 좋은 수행 결과를 보이고 있다. 이 실험을 통해서 하나의 결함발생 시 제시된 체크포인팅 알고리즘이 다른 알고리즘들보다 더 작은 작업 완료 시간을 가짐을 알 수 있다.

6. 결론 및 향후 연구

체크포인팅은 결함을 극복하고, 작업의 빠른 재수행을 위해 많이 사용되는 결함포용 기술 중의 하나이다. 분산 시스템에서의 체크포인팅 알고리즘들은 크게 동기화 체크포인팅 알고리즘과 비동기화 체크포인팅 알고리즘으로 나눌 수 있다. 본 논문에서는 프로세스 수행 속도와 메시지 전송 비율에 따라 체크포인트 횟수가 크게 증가하는 적응적 체크포인팅 알고리즘[2]과 나태 체크포인팅 알고리즘[3]의 단점을 개선하여, 체크포인트가 일정한 시간 간격마다 1번씩 일어나도록 하여 정기적 체크포인팅 알고리즘과 동일한 최소의 체크포인트 횟수를 가지는 비동기화 체크포인팅 알고리즘을 제시하였다. 시뮬레이션을 통해 정기적, 적응적, 나태, 그리고, 제시된 알고리즘을 비교한 결과, 제시된 알고리즘이 결함이 발생하지 않은 환경과 결함이 발생한 환경에서 더 짧은 작업 완료 시간을 보여주었다.

현재까지 분산 시스템에서의 체크포인팅 알고리즘들은 많이 연구되어 왔다. 하지만, 이런 연구들은 오히려 다른 체크포인팅 알고리즘들 사이의 비교를 어렵게 하였다. 동기화 체크포인팅 알고리즘들을 비교하기 위한 연구[7]가 일부 진행되었으나, 아직 미비한 실정이다. 이런 상황은 비동기화 체크포인팅 알고리즘들에 대해서도 마찬가지이다. 앞으로 다수의 체크포인팅 알고리즘들을 비교하기 위한 연구가 진행되어야 할 필요가 있다.

참 고 문 헌

[1] R. Koo and S. Toueg, "Checkpointing and Rollback-Recovery for Distributed Systems," IEEE Trans. on Soft. Eng. Vol.SE-13, No.1, pp.23-31, Jan., 1987.

[2] J. Xu and R. H. B. Netzer, "Adaptive Independent Checkpointing for Reducing Rollback Propagation," Proc. of the Fifth IEEE Symp. on Parallel and Distributed Processing, pp.754-761, Dec., 1993.

[3] Y. M. Wang and W. K. Fuchs, "Lazy Checkpoint Coordination for Bounding Rollback Propagation," Technical Report CRHC-92-27, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, Nov., 1992.

[4] Y. M. Wang and W. K. Fuchs, "Scheduling Message Processing for Reducing Rollback Propagation," FTCS-22, pp.204-211, 1992.

[5] J. Xu and R. H. B. Netzer, "Necessary and Sufficient Conditions for Consistent Global Snapshots," Technical Report CS-93-32, Department of Computer Science, Brown University, Sep., 1993.

[6] S. T. Levi and A. K. Agrawal, 'Fault Tolerant System Design,' McGraw-Hill, 1994.

[7] E. N. Elnozahy, D.B. Johnson and W. Zwaenpoel, "The performance of consistent checkpointing," Proc. of the 11th Symp. on Reliable Distributed Systems, pp.39-47, Oct., 1992.

[8] D. B. Johnson and W. Zwaenpoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing," J. of Algorithms, Vol.11, pp.462-491, 1990.

[9] R. E. Storm and S. Yemini, "Optimistic Recovery in Distributed Systems," ACM Trans. on Computer Systems, Vol.3, No.3, pp.204-226, Aug., 1985.

[10] H. V. Leong and D. Agrawal, "Using Message Semantics to Reduce Rollback in Optimistic Message Logging Recovery Schemes," Int. Conf. on Distributed Computing Systems, pp.227-234, 1994.

[11] K. Li, J. F. Naughton and J. S. Plank, "Checkpointing Multicomputer Application," Proc. IEEE Symp. on Reliable Distributed Systems, pp.2-11, 1991.

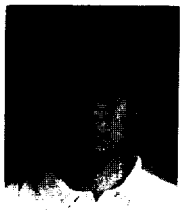
[12] K. Venkatesh, T. Radhakishman and H. F. LI, "Optimal Checkpointing And Local Recording For Domino-Free Rollback Recovery," Information Processing Letter, Vol.25, No.5, pp.295-304, July 1987.

[13] E. N. Elinozahy and W. Zwaenpoel, "On the Use and Implementation of Message Logging," FTCS-23, pp.145-154, 1993.

[14] R. D. Schlichting and F. B. Schneider, "Fail-stop processors : An approach to designing fault-tolerant computing systems," ACM Trans. on Computer Systems, Vol.1, pp.222-238, Aug., 1983.

[15] T. H. Lin and K.G. Shin, "Damage Assessment for Optimal Rollback Recovery," IEEE Trans. on Computers, Vol.47, No.5, pp.603-613, May, 1998.

[16] L. Alvisi and K. Marzullo, "Message Logging : Pessimistic, Optimistic, Causal, and Optimal," IEEE Trans. on Soft. Eng. Vol.24, No.2, pp.149-159, Feb., 1998.



김 도 형

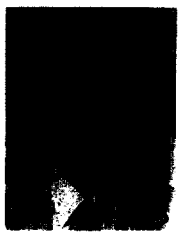
e-mail : dhkim@chinmi.etri.re.kr

1993년 경북대학교 컴퓨터공학과(공학사)

1995년 포항공과대학 전자계산학과(이학석사)

1995년~현재 한국전자통신연구원 컴퓨터·소프트웨어기술연구소 연구원

관심분야 : 결함포용, 실시간시스템, 성능감시, 성능평가



박 창 순

e-mail : cspark@etri.re.kr

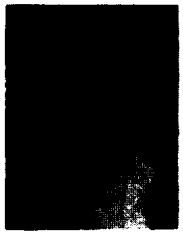
1975년 서울대학교 응용수학과(학사)

1992년 연세대학교 전산학과(석사)

1998년~현재 한국전자통신연구원 컴퓨터소프트웨어기술연구소 책임기술원

1977년~1998년 KIST 연구원, 책임기술원, 시스템공학연구소 네트워크컴퓨팅연구부장

관심분야 : 네트워크컴퓨팅, 실시간시스템



김 종

e-mail : jkim@postech.ac.kr

1981년 한양대학교 전자공학과(학사)

1983년 한국과학기술원 전산학과(석사)

1991년 미국 Penn. State University 전산학(박사)

1983년~1986년 ㈜한국증권전산 시스템 분석가

1992년~현재 포항공과대학교 전산학과 부교수

관심분야 : 결함포용, 병렬처리, 성능평가