프로세스 수행 시간의 비용 분석에 기반을 둔 페이지 단위 점진적 검사점의 작성 시점 결정 기법

이 상 호[†] · 허 준 영[†] · 홍 지 만[††]

요 약

검사점 기법은 시스템이 장애를 내재한 경우에 효과적으로 프로세스가 장애 지점으로부터 다시 시작 할 수 있게 하는 결함 허용 방법이다. 특히, 페이지 단위 점진적 검사점 기법은 검사점 사이에서 변경된 페이지 데이터만을 저장함으로써 검사점 기록 오버헤드를 감소시킨다. 이 기법은 매 검사점 사이에서 변화하는 데이터의 크기가 가변적이므로 검사점 수행 시간도 매번 변하는 성질을 갖고 있다. 기존의 연구로 고정적인 검사점 수행 시간을 갖는 경우에 대한 효율적인 검사점 작성 시점 결정 방법이 제시된 바 있다. 그러나 매 검사점 마다 가변적인 시간을 필요로 하는 페이지 단위 점진적 검사점 기법에 대한 효율적인 작성 시점 결정 방법은 아직 연구되지 않은 분야이다. 본 논문에서는 효율적이고 적응성 있는 검사점 작성 시점 결정 방법을 제안하고, 이 방법에 기반을 둔 적응성 있는 페이지 단위 점진적 검사점 기법을 보인다. 여러 가지 응용 프로그램의 실험 결과를 통하여, 제안한 방법을 사용하는 것이 기존의 고정적인 인터벌을 갖는 페이지 단위 점진적 검사점을 사용하는 경우보다 프로세스의 평균 수행 시간을 현저히 줄임을 알 수 있다.

키워드 : 검사점 및 복원, 점진적 검사점, 결함 허용, 비용 분석

Taking Point Decision Mechanism of Page-level Incremental Checkpointing based on Cost Analysis of Process Execution Time

Sangho Yi[†] · Junyoung Heo[†] · Jiman Hong[††]

ABSTRACT

Checkpointing is an effective mechanism that allows a process to resume its execution that was discontinued by a system failure without having to restart from the beginning. Especially, page-level incremental checkpointing saves only the modified pages of a process to minimize the checkpointing overhead. This means that in incremental checkpointing, the time consumed for checkpointing varies according to the amount of modified pages. Thus, the efficient interval of checkpointing must be determined on run-time of the process. In this paper, we present an efficient and adaptive page-level incremental checkpointing facility that is based on the cost analysis of process execution time. In our simulation, results show that the proposed mechanism significantly reduced the average process execution time compared with existing fixed-interval based page-level incremental checkpointing.

Key Words : Checkpointing and Recovery, Incremental Checkpointing, Fault-tolerance, Cost Analysis

1. 서 론

검사점 기법은 프로세스의 수행 중에 프로세스의 상태를 주기적으로 안정한 저장장치에 저장하여 시스템 장애가 발생했을 때 마지막으로 저장된 상태부터 프로세스가 다시 시작할 수 있게 한다[1, 2]. 따라서 검사점 기법을 사용하게 되면 마지막 검사점에서 다시 시작함으로써 검사점 이전의 수행을 다시 할 필요가 없게 되므로, 결함이 발생할 가능성이 있는 시스템에서는 전체 수행 시간을 단축시킬 수 있다. 그러나 이러한 검사점 기법에는 시간적으로 또는 공간적으로 검사점을 안정한 저장장치에 기록하는 오버헤드가 따르게 된다[3].

이러한 오버헤드를 줄이기 위하여 다양한 방법들[2, 4-8]이 제안되었다. 이 방법들은 크게 두 가지로 분류할 수 있다[2]. 첫째로, 검사점을 작성할 때 발생하는 지연을 감추기 위한 방법으로 디스크 없이 검사점 작성[7], 포크 검사점[5], 압축 검사점[5, 6] 같은 것이 있다. 둘째로, 검사점을 작성할 대상을 줄이는 기법으로 메모리 배제 기법[2]과 점진적 검사점 기법[3] 등이 있다. 대상을 줄이는 기법의 핵심은 마지막 검사점에서 변경된 데이터만을 기록하는 것이다. 예를 들자면, 변경 자체가 일어나지 않는 읽기

† 준 회 원 : 서울대학교 컴퓨터공학부 박사과정
†† 종신회원 : 광운대학교 컴퓨터공학부 조교수(교신저자)
논문접수 : 2006년 4월 27일, 심사완료 : 2006년 7월 5일

전용의 데이터의 경우에는 이미 이전의 검사점에 기록이 되어있기 때문에, 다음의 검사점에서는 기록할 필요가 없으므로 배제시키게 된다. 위와 같이 다양한 검사점 방법들 중에서는 점진적 검사점 기법이 다양한 실제 시스템 환경에서 사용되고 있다. 특히, 페이지 폴트 하드웨어를 사용하여 변경된 페이지만을 저장하는 페이지 단위 점진적 검사점이 널리 쓰이고 있다.

그러나 위의 점진적 검사점 기법에는 효율적인 검사점의 주기를 계산 및 결정하는 방법에 대하여는 아직까지 연구되지 않았다. Duda는 [10]에서 검사점 수행 시간이 고정적일 때에, 오프라인에서 최적의 검사점 작성 시점을 계산해내는 방법을 증명하였다. 그러나 점진적 검사점 기법에서는 검사점 수행 시간이 검사점 사이에서 변경된 페이지 데이터의 크기에 영향을 받아 변한다. 이러한 페이지 변경 패턴은 결정적이지 않으므로 우리는 검사점의 수행 시간을 예측할 수 없다. 따라서 증명된 최적의 검사점 작성 시점 계산 방법은 점진적 검사점 기법에 적용하는데에 무리가 있다.

본 논문에서는 효율적이고 적응성 있는 페이지 단위 점진적 검사점 기법을 보인다. 이것 역시 페이지 폴트 하드웨어 메커니즘을 사용하여 변경된 페이지만을 기록한다. 또한 이것은 프로세스의 예상 복원 시간에 대한 비용 분석에 기반을 둔 주기 결정 방법을 사용하여 효율적인 검사점 작성 시점을 보장한다. 또한 다양한 응용 프로그램들에 대한 실험 결과를 통하여 본 논문에서 제안한 방법을 사용한 경우에, 기존의 고정적 인터벌을 갖는 페이지 단위 점진적 검사점을 수행한 경우보다 프로세스의 평균 수행 시간이 현저히 감소함을 볼 수 있다.

본 논문의 구성은 다음과 같다. 2절에서 관련 연구들을 설명하고, 3절에서는 페이지 단위 점진적 검사점 기법에 대한 설명과 함께, 검사점 기법을 사용한 경우와 그렇지 않은 경우에 대한 프로세스의 예상 수행 시간을 도출한다. 4절에서는 비용 분석에 기반한 적응성 있는 검사점 작성 시점 계산 방법을 설명하고, 5절에서는 본 논문에서 제안한 검사점 기법과 기존의 기법의 성능을 비교한다. 마지막으로 6절에서 결론을 맺고 앞으로의 연구 방향에 대하여 보인다.

2. 관련 연구

본 절에서는, 기존의 잘 알려진 검사점 기법의 설계 및 구현과 관련된 연구 및 검사점 기법의 비용 분석에 대한 몇 가지 연구를 소개한다. 먼저, 실제 시스템 환경에서 사용할 수 있는 검사점 기법의 설계 및 구현 방법에 대한 여러 가지 연구가 제안되었고[2, 4-8], 제안된 검사점 기법에 대한 이론적인 분석도 여러 연구에서 이루어졌다[1, 10].

Duda 는 결함이 포아송 분포로 발생하고, 검사점을 수행하는 동안에는 결함이 발생하지 않으며, 모든 검사점은 고정적인 수행 시간을 갖는다는 가정 하에서, 최적의 검사점 작성 시점에 대한 계산을 증명하였다[11]. 또한 시스템이 장애를 가질 경우, 검사점 기법을 사용하는 것이 그렇지 않은 것보다 더 나은 성능을 보임을 수학적인 계산을 통하여 보였다.

Hong 등은 검사점 기법을 사용하는 경우와 그렇지 않은 경우에 대하여 프로세스의 예상 수행 시간을 비교하여 포크 검사점의 비용을 분석하였다[1]. 이 연구에서는 결함이 검사점의 작성 중에도 나타날 수 있다고 가정하였다.

Beck 등은 컴파일러의 도움을 받는 메모리 배제 검사점 방법을 제안하였다[9]. 이는 데이터 흐름 분석을 통하여 읽기전용 메모리와 쓰기전용 메모리를 구분함으로써 저장하지 않아도 될 메모리를 배제시켜 검사점의 오버헤드를 줄이는 방법이다.

Plank 등은 유닉스에서 동작하는 사용자 수준 검사점 기법인 Libckpt를 제안하였다[5]. 이는 사용자에게 투명한 기법으로 '쓰기 시 복사' 방법을 사용한 점진적 검사점 기법이다. 그러나 Libckpt를 사용하기 위해서는 사용자의 소스코드에 일부의 수정을 가해야 한다. 예를 들자면, main() 함수는 ckpt_target()으로 이름을 변경해야 한다.

Heo 등은 공간 효율적인 점진적 검사점 기법을 제안하였다[11]. 이것의 주 목적은 여러 가지 버전을 관리를 해야 하는 점진적 검사점의 특성에서 기인한, 디스크 공간 낭비를 최소화 하는 것이다. 이 기법은 이전 버전의 검사점에서 사용되지 않을 페이지 데이터들을 수집하여 제거하는 기법을 통하여 보다 공간 효율적으로 동작하게 한다. 이 연구에서 그들은 페이지 버전 정보 및 그림자 복사본을 사용하여 위의 방법을 구현하였다.

3. 페이지 단위 점진적 검사점의 비용 분석

본 절에서는 페이지 단위 점진적 검사점 기법에 대한 간략한 소개를 하고, 이 검사점 기법을 사용한 경우와 그렇지 않은 경우에 대한 프로세스의 예상 수행 시간을 도출한다.

3.1 시스템 모델 정의 및 가정

다음의 <표 1>에는 앞으로 검사점 기법에 대한 비용 분석에서 사용할 몇 가지 기호 및 정의를 보인다.

<표 1> 본 논문에서 사용하는 기호 및 정의

t	프로세스의 전체 필요 수행 시간
t_i	i-번째 구간 동안의 프로세스의 필요 수행 시간
r	프로세스 복원 비용
c_i	i-번째 구간의 검사점 기록 비용
λ	결함 발생률
m	검사점 구간 안에서 변경된 페이지 수
c_p	한 페이지의 검사점 기록 비용
c_s	프로세스 스트럭처의 검사점 기록 비용
$F(k)$	k 시간 동안의 결함 발생 누적 밀도 함수(CDF)
$f(k)$	k 시간 동안의 결함 발생 확률 밀도 함수(PDF)
$T(t)$	검사점을 사용하지 않을 경우의 예상 수행 시간
$T_c(t,c')$	점진적 검사점을 사용할 경우의 예상 수행 시간
$R_{skip}(t)$	검사점을 작성하지 않을 경우의 예상 복원 시간
$R_{take}(t)$	점진적 검사점을 작성할 경우의 예상 복원 시간

본 논문의 예상 수행 시간 및 예상 복원 시간의 비용 분석 및 성능 평가를 위한 실험에서 사용한 시스템 모델은 다음과 같다. 먼저, 프로세스의 전체 예상 수행 시간은 프로세스의 수행 시작 시각부터 프로세스가 일을 끝마치는 종료시간까지의 프로세스 수행 시간으로 정의한다. $T(t)$는 프로세스의 예상 수행 시간, t는 프로세스의 필요 수행 시간으로 정의한다. 만약, 결함이 전혀 없는 시스템이라면, 다음과 같은 식이 성립한다.

$$T(t) = t$$

또한 페이지 단위 점진적 검사점을 사용하는 경우의 프로세스의 예상 수행 시간은 $T_c(t,c)$로 정의한다. 이것의 전체 수행 시간은 N개의 구간으로 나뉘어 지고, 각 구간의 끝에서 점진적 검사점이 수행된다. 여기에서, 각 검사점의 구간은 인접한 두 검사점의 시간 차이로 나타낼 수 있고, 이것은 하나의 검사점이 완료된 이후부터, 다음 검사점이 완료되는 시점 까지를 가리킨다.

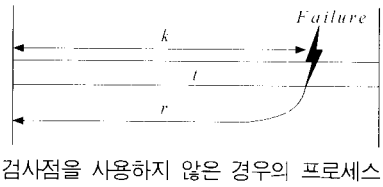
$T_c(t_i,c_i)$는 점진적 검사점을 사용하는 경우의 i 번째 구간의 프로세스 예상 수행 시간을 나타낸다. 따라서 위의 두 정의를 식으로 관계를 표현하면, 결함을 내재한 시스템에서의 프로세스 예상 수행 시간에 대한 관계식으로 다음과 같은 결과를 얻을 수 있다.

$$\sum_{i=1}^{N} T_c(t_i,c_i) = T_c(t,c)$$

또한 본 논문에서 사용하는 몇 가지 가정은 다음과 같다. 먼저, 시스템에 내재된 결함은 프로세스의 수행 또는 검사점의 수행 시에도 나타날 수 있다고 가정한다. 또한 이러한 결함은 결함 발생률 λ의 포아송 분포를 갖는다고 가정한다. 이러한 가정은 결함의 특성상 언제 어디에서나 발생할 수 있으므로, 일반적으로 받아들일 수 있는 것들이다.

3.2 검사점 기법을 사용하지 않을 경우의 예상 수행 시간

앞서 보인바와 같이, 시스템에 결함이 발생하지 않는다면 프로세스의 예상 수행 시간인 $T(t)$는 수행 필요 시간인 t와 같다. 그러나 시스템에 결함이 내재되어있고 이것이 프로세스의 수행 중에 발생한다면, 프로세스 복원 비용인 r이 발생하게 되고, 해당 프로세스는 처음부터 다시 시작해야 한다. (그림 1)은 검사점을 사용하지 않은 프로세스의 경우에 결함이 발생하였을 때의 예제 상황을 보여준다.



(그림 1) 검사점을 사용하지 않은 경우의 프로세스 실행 예제

(그림 1)과 같이, 결함이 프로세스의 종료 전에 발생하였다면 $(k < t)$, 검사점을 사용하지 않은 경우의 프로세스의 예상 수행 시간인 $T(t)$는 $k, r, T(t)$에 의하여 재귀적으로 계산할 수 있다. 위의 (그림 1)의 경우, 프로세스는 반드시 처음부터 다시 시작해야 하기 때문에, 남아있는 수행 필요 시간은 t가 되며, 따라서 예상 수행 시간은 $T(t)$가 된다.

우리는 결함을 결함 발생률 λ의 포아송 분포를 갖는다고 가정하였으므로, 따라서 결함 발생에 대한 누적 밀도 함수는 $F(k) = 1 - e^{-\lambda k}$가 되고, 이것을 확률 밀도 함수의 식으로 나타낼 경우에는 $f(k) = \lambda e^{-\lambda k}$가 된다. 이 때에, 예상 수행 시간은 다음의 정리 1과 같이 표현된다[1].

[정리 1] 검사점 기법을 사용하지 않는 경우의 프로세스의 예상 수행 시간 $T(t)$는 다음의 식과 같다.

$$T(t) = \frac{(e^{\lambda t} - 1)(1 + \lambda r)}{\lambda}$$

[증명] *예상 수행 시간을 다음과 같이 조건 별로 나눌 수 있다.*

$$T(t) = \begin{cases} t & \text{if } k \geq t \\ k + r + T(t) & otherwise \end{cases}$$

조건별 수행 시간을 바탕으로 전체 예상 수행 시간을 계산하면,

$$T(t) = \int_t^{\infty} t f(k)dk + \int_0^t (k + r + T(t)) f(k)dk$$

위의 식을 계산하면,

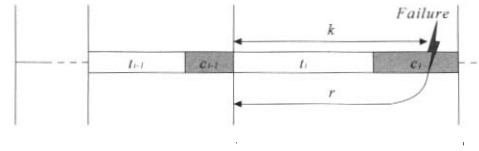
$$T(t) = \frac{t + \int_0^t (k + r - t) f(k)dk}{1 - \int_0^t f(k)dk}$$

마지막으로, $f(k) = \lambda e^{-\lambda k}$를 사용하여 식을 풀면, 다음의 식을 얻는다.

$$T(t) = \frac{(e^{\lambda t} - 1)(1 + \lambda r)}{\lambda}$$

3.3 페이지 단위 점진적 검사점 기법을 사용할 경우의 예상 수행 시간

(그림 2)는 페이지 단위 점진적 검사점을 사용하는 경우에 대한 프로세스 실행 예제를 보여준다. 프로세스의 전체 수행 시간은 N개의 구간으로 나뉘어 지고, 각각의 구간은 t_i로 표현되는 프로세스 수행 시간과 c_i로 표현되는 각 구간의 검사점 기록 시간으로 구성된다. 어떠한 구간 안에서 시스템 결함이 발생할 경우, 프로세스는 해당 구간의 첫 부분으로 복원 한 후에 프로세스를 다시 시작해야 한다. 만약 i번째 구간 안에서 결함이 발생하지 않는다면, 이 구간의 예상 수행 시간은 $(t_i + c_i)$와 같게 된다. 그러나 i번째 구간 안에서 결함이 발생한다면, 이 구간의 처음으로의 복원 비용인 r이 발생한다. 따라서 i번째 구간의 종료 이전에 결함이 발생한다면 $(k < (t_i + c_i))$, 페이지 단위 점진적 검사점 기법을 사용하는 경우의 프로세스의 예상 수행 시간인 $T_c(t_i,c_i)$는 다음의 정리 2와 같이 표현된다[1].



(그림 2) 점진적 검사점을 사용한 경우의 프로세스 실행 예제

[정리 2] 페이지 단위 점진적 검사점 기법을 사용하는 경우에서 i번째 구간의 프로세스 수행 시간인 $T_c(t_i,c_i)$는 다음과 식과 같다.

$$T_c(t_i,c_i) = \frac{(e^{\lambda(t_i + c_i)} - 1)(1 + \lambda r)}{\lambda}$$
$$where \quad c_i = m \cdot c_p + c_s$$

[증명] *위의 정리 1에서, t를 $(t_i + c_i)$로 대체하면, 정리 2의 식을 얻을 수 있다.*

또한, 위의 정리 2를 사용하여, 전체 예상 수행 시간 $T_c(t,c)$는 각 구간의 합으로 계산된다.

$$T_c(t_i,c_i) = \sum_{i}^{n} \frac{(e^{\lambda(t_i+c_i)}-1)(1+\lambda r)}{\lambda}$$

만약 모든 검사점들의 구간이 동일하고, 검사점 기록 비용이 동일한 상수인 c로 표현이 된다면, 위의 수식들은 계산하기에 훨씬 쉬워질 수 있고 수학적으로 최적의 구간을 증명하는 것도 무리가 아닐 것이다[10]. 그러나 이러한 가정은 페이지 단위 점진적 검사점의 실제 특성을 반영하지 못한다. 점진적 검사점 기법에서는 검사점 기록 비용인 c_i가 가변적이고, 더 나아가서 검사점들의 구간도 동일해서는 효율성을 갖기 어렵다. 예를 들자면 c_i는 프로세스의 메모리 페이지의 변경 패턴에 의해 영향을 받게 되고, 이러한 변경 패턴은 결정적으로 알기 힘들다. 따라서 위의 프로세스의 예상 수행 시간에 대한 비용 분석만으로는 효율적인 검사점의 작성 시점 계산을 수행할 적합한 방법을 찾을 수 없다.

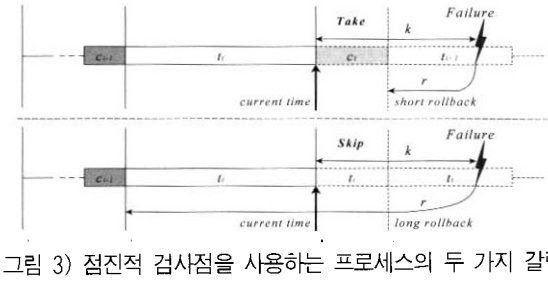
4. 적응성 있는 검사점 작성 시점 결정 방법

본 절에서는 본 논문의 핵심 내용이 되는 페이지 단위 점진적 검사점 기법을 위한 효율적이고 적응성 있는 검사점 작성 시점 결정 방법을 제안한다. 이 결정 방법은 프로세스의 예상 복원 시간의 비용 분석에 기반을 둔 것으로, 이를 통하여 가변적인 검사점 기록 비용을 갖는 점진적 검사점 기법에 대한 효율적인 검사점 작성 시점을 결정해줄 수 있게 된다.

4.1 예상 복원 시간의 비용 분석

(그림 3)은 페이지 단위 점진적 검사점을 사용하는 프로세스의 예제 상황에서 두 가지 갈림길을 보여주고 있다. c_{i-1}은 $(i-1)$번째 구간의 검사점 기록 시간이고, t_i는 i 번째 구간의 프로세스 수행 시간, 그리고 c_i는 현 시점에서 예상되는 프로세스의 점진적 검사점의 기록 시간이다. 이 상황에서, 우리는 두 가지 선택, 즉, 점진적 검사점을 수행(*Take*)하거나, 무시(*Skip*)하고 진행할 수 있다.

현 시점에서 프로세스가 점진적 검사점을 수행한다면, 이 프로세스는 c_i의 점진적 검사점 기록 시간동안 작업을 중단시키고 기다려야 한다. 만약 현 시점 이후로 시스템 결함이 발생하지 않는다면, 프로세스는 c_i의 시간만큼 작업이 지연되어, 전체 수행 시간이 그만큼 연장될 것이다. 그러나 현 시점 이후로 시스템 결함이 발생한다면 예상 복원 시간이 현 시점에서 수행한 검사점에 의하여 현저히 감소할 것이다. 그러므로 우리는 점진적 검사점을 사용하는 경우에는 작업의 수행 중에, 비용 효율성의 분석을 통하여 검사점을 현재 수행할 것인지, 혹은 무시할 것인지를 결정해야 할 것이다.



(그림 3) 점진적 검사점을 사용하는 프로세스의 두 가지 갈림길

[검사점 무시(Skip)] 이 때의 예상 복원 시간은 다음의 식과 같다.

$$R_{skip}(t_i,c_i) = \int_0^{\infty} (k+r+T(t_i))f(k)dk$$

[증명] *정리 1과 같은 방법으로 전체 예상 수행 시간을 계산하면, 위의 식을 얻을 수 있다.*

[검사점 수행(Take)] 이 때의 예상 복원 시간은 다음의 식과 같다.

$$R_{take}(t_i,c_i) = \int_0^{c_i} (k+r+T(t_i))f(k)dk + \int_{c_i}^{\infty} (k+r)f(k)dk$$

[증명] *조건별 예상 수행 시간을 나타내면 다음과 같다.*

$$R_{take}(t_i,c_i) = \begin{cases} k+r+T(t_i+c_i) & \text{if } k < c_i \\ k+r & otherwise \end{cases}$$

위의 식에 대하여, 전체 예상 수행 시간을 계산하면, 위의 식을 얻을 수 있다.

위의 두 가지 상황에 대한 수식들을 사용하여, $R_{take}(t_i,c_i) - R_{skip}(t_i,c_i)$를 검사점 수행을 결정하는 판별식인 $D(t_i,c_i)$로 나타내었고, 아래의 식은 $f(k) = \lambda e^{-\lambda k}$를 대입하여 계산한 결과이다.

$$D(t_i,c_i) = (1-e^{-\lambda c_i})T(t_i+c_i) - T(t_i)$$

위의 판별식 $D(t_i,c_i)$의 계산을 통하여, 점진적 검사점 기법을 사용할 경우에 프로세스가 검사점을 수행해야 하는지, 그렇지 아니한지를 알아낼 수 있다. 만약 계산된 $D(t_i,c_i)$가 양수의 값을 갖는다면, $R_{take}(t_i,c_i)$가 $R_{skip}(t_i,c_i)$보다 큰 값을 가진 것이므로, 우리는 현재 검사점을 수행하지 말아야 할 것이다. 그 반대로 $D(t_i,c_i)$가 음수의 값을 갖는다면, 검사점을 수행하는 것이 더 좋을 것이다. 예를 들자면, 만약 t_i가 거의 0에 가까운 상황, 즉 검사점을 수행한 이후 시간이 거의 흐르지 않은 상황이라면, $e^{-\lambda c_i} < 1$이기 때문에, $D(0,c_i)$는 양수 값을 가질 것이다. 또 다른 예로, 만약 t_i가 무한대의 값을 갖는 상황, 즉 검사점을 수행한 이후 상당한 시간이 지난 상황이라면, $e^{-\lambda c_i} > 0$이기 때문에, $D(\infty,c_i)$는 음수의 값을 가질 것이고, 따라서 이 상황에서는 검사점을 수행하는 것이 유리할 것이다.

4.2 검사점 작성 시점 결정 방법

$D(t_i,c_i)$는 t_i와 c_i의 값에 따라 변화하는 식이다. 따라서 t_i와 c_i의 값이 변화할 때에 $D(t_i,c_i)$를 계산해야한다. i 번째 구간의 검사점 기록 시간인 c_i는 페이지가 변경될 때에 해당 페이지를 안정 저장장치로 기록하는 시간만큼 증가한다. 만약 오랜 시간동안 변경되는 페이지 데이터가 없다면 c_i는 변하지 않을 것이다. 하지만 시간은 계속 흐르기 때문에, 변하는 시간에 대한 새로운 값을 얻기 위해서는 $D(t_i,c_i)$를 지속적으로 계산해야할 필요가 있다. 그러나 매 시간마다 새로이 계산하는 것은 계산비용의 낭비를 가져올

수 있으므로, 다음의 식인 $D(t_i+\alpha,c_i)=0$를 만족시키는 $\alpha(t_i,c_i)$를 단 한번 계산함으로써 처리할 수 있다.

$$\alpha(t_i,c_i)=\frac{1}{\lambda}ln(\frac{e^{\lambda c_i}}{2-e^{\lambda c_i}})-t_i$$

여기에서, $\alpha(t_i,c_i)$는 i 번째 구간의 검사점 수행 위치를 나타낸다. 따라서 이 식을 계산하면 점진적 검사점의 작성 시점을 비용 분석으로부터 계산된 효율적인 위치로 결정할 수 있게 된다. 이 계산 과정을 효율적으로 수행하기 위하여 우리의 구현에서는 $D(t_i,c_i)$와 $\alpha(t_i,c_i)$의 계산을 각각 페이지 폴트 핸들러와 타이머 루틴에 등록하였다. 아래의 알고리즘 1은 페이지 폴트 핸들러와 타이머 루틴에 추가된 검사점 작성 시점 결정 방법을 보인다.

알고리즘 1에서 m은 두 검사점 사이에서 변경된 페이지의 개수를 의미하고, $\alpha(t_i,c_i)$의 값은 $D(t_i+\alpha,c_i)=0$을 만족시키는 α값이다. 위의 알고리즘에서, 타이머는 $\alpha(t_i,c_i)$의 계산된 시간에 의하여 세팅되고, 해당 타이머가 호출되면 점진적 검사점을 수행하게 된다. 이렇게 함으로써, 페이지 단위 점진적 검사점 기법을 사용하는 경우, 비용 효율적으로 검사점의 작성 시점을 결정할 수 있게 된다.

알고리즘 1. 검사점 작성 시점 결정 방법

```
-- 페이지 폴트 핸들러에서
IF 페이지 쓰기 폴트 발생 THEN
   m := m + 1
   D(t_i,c_i) 값 계산
   IF D(t_i,c_i) < 0 THEN
      점진적 검사점 수행
   ELSE
      새로운 α(t_i,c_i) 값 계산
      α(t_i,c_i) 값으로 타이머 설정
   END IF
END IF

-- 타이머 루틴에서 ---
IF α(t_i,c_i)로 설정한 타이머 종료 THEN
   새로운 α(t_i,c_i) 값 계산
   α(t_i,c_i) 값으로 타이머 설정
END IF
```

5. 성능 평가

본 절에서는 본 논문에서 제안한 검사점 작성 시점 계산 방법을 사용한 페이지 단위 점진적 검사점 기법의 기존의 고정적인 인터벌을 갖는 점진적 검사점 기법에 대한 상대적 성능을 보인다. 제안한 방법의 구현은 리눅스 커널 2.4.20 버전에서 이루어졌다[1].

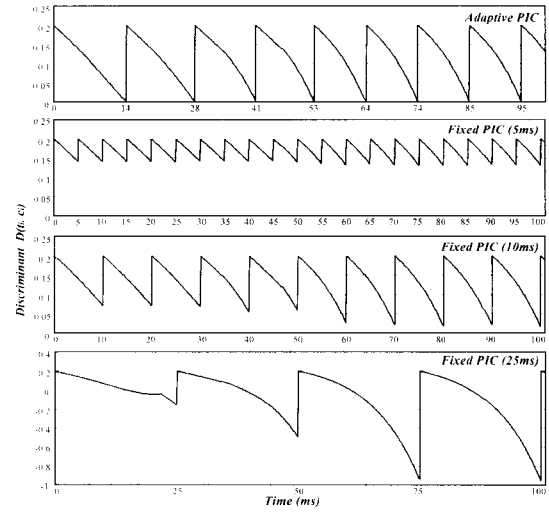
본 논문에서는 성능 측정을 위하여 산술적인 계산을 수행하는 다음의 4가지 응용 프로그램들을 사용하였다. 행렬 곱셈 프로그램(*MATMUL*), JPEG 인코딩 프로그램(*JPEG*), 퀵 정렬 프로그램(*QSORT*), 신경망을 통한 문자 인식 프로그램(*NN*)의 네 가지 프로그램으로, 각 응용 프로그램의 동작 방식 및 특성, 그리고 메모리 사용에 관한 세부 정보는 다음과 같다.

◇ **MATMUL** : 4개의 행렬들을 순차적으로 곱하는 예제이다. 행렬의 크기는 100x100 이고, 4바이트의 정수로 구성되어있다. 이 예제는 행렬로 선언한 메모리 영역에 대하여 인접한 쓰기 연산이 빈번하게 발생할 것이고, 이 결과로 페이지 폴트의 공간적 지역성이 매우 크게 나타날 것이다.

◇ **JPEG** : 256x256 크기의 로우 이미지 데이터를 JPEG 방식으로 인코딩하는 프로그램이다. 이 예제는 앞의 MATMUL과 비슷한 공간적 지역성을 갖는 쓰기 연산(혹은 페이지 폴트)이 발생하지만, 보다 시간을 오래 소요하는 DCT 혹은 IDCT 등의 변환 함수를 호출하기 때문에 보다 오랜 산술 계산 작업을 수행할 것이다.

◇ **QSORT** : 이는 50,000개의 4바이트의 정수를 정렬하는 프로그램이다. 초기에 랜덤 값으로 설정된 값을 퀵 정렬 알고리즘을 사용하여 정렬한다. 다른 응용 프로그램에 비하여 메모리 공간에 대한 쓰기 연산(혹은 페이지 폴트)의 지역성이 적게 나타나는 특성이 존재한다.

◇ **NN** : 신경망 알고리즘을 사용하여 문자 인식을 수행하는 예제이다. 이 예제는 10x10, 20x15, 15x10 크기의 4바이트의 부동소수점 변수를 쓰기연산에서 사용하며, 다른 3가지의 예제들에 비하여 보다 오랜 산술 계산을 수행한다. 메모리 공간에 대한 쓰기 연산(혹은 페이지 폴트)의 지역성은 크지만, 시간상의 지역성은 적게 나타날 것이다.

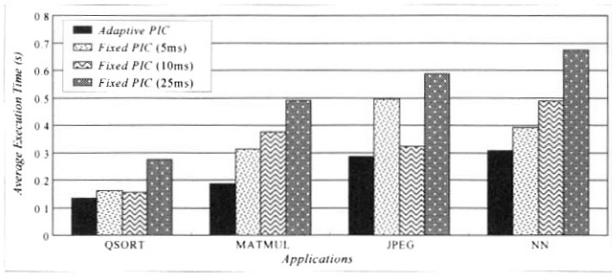


(그림 4) QSORT 응용 예제의 검사점의 위치와 $D(t_i,c_i)$의 관계

이 네 가지의 응용 프로그램들은 본 논문에서 제안한 점진적 검사점 기법을 추가한 리눅스 커널에서 컴파일 되었다. 검사점의 오버헤드를 비교하기 위하여, 본 논문의 실험에서는 각각의 응용 프로그램에 대한 평균 수행 시간을 비교 측정하였다.

(그림 4)는 QSORT 응용 예제에 대한 판별식 $D(t_i,c_i)$의 변화 양상 및 검사점의 위치를 표기하였다. 본 논문에서 제안한 기법을 사용한 적응성 있는 점진적 검사점 기법(*Adaptive PIC*)과 기존의 고정적인 인터벌을 갖는 점진적 검사점 기법(*Fixed PIC*)을 사용한 경우, 검사점의 위치가 상당히 달라짐을 볼 수 있다. 제안한 방법을 사용할 경우, 판별식 $D(t_i,c_i)$의 계산 결과에 따른, 응용 예제에 맞는 적응성 있는 작성 시점이 계산되어 검사점의 위치가 나타남을 볼 수 있고, 고정적 인터벌을 갖는 경우에는 각각의 *5ms, 10ms, 25ms* 의 고정적인 주기로 검사점이 나타남을 볼 수 있다. 이 결과를 통하여, $D(t_i,c_i)$의 변화 양상을 바탕으로, 고정

1) 구현한 모든 소스코드 및 커널 패치는 다음의 URL: http://ssrnet.snu.ac.kr/~shyi/ 에서 다운로드 할 수 있다.



(그림 5) 응용 예제들의 평균 수행 시간

인터벌을 갖는 점진적 검사점 보다는 제안한 방법이 더 나은 검사점 작성 시점을 결정할 수 있음을 볼 수 있다.

(그림 5)는 네 가지 응용 예제들에 대하여 본 논문에서 제안한 기법과 기존의 고정 인터벌을 갖는 점진적 검사점 기법을 사용하였을 때의 평균 수행 시간을 보인다. 이 결과를 통하여 네 가지 응용 프로그램 예제의 모든 경우에 대하여 본 논문에서 제안한 적응성 있는 점진적 검사점 기법을 사용한 경우가 기존의 방식을 사용한 경우보다 평균 수행 시간이 더 짧은 것을 볼 수 있다. 따라서 본 논문에서 제안한 검사점의 위치 결정 방법을 사용하면 응용 예제의 성격에 영향 받지 아니하고 적응성 있게 동작하므로, 이 방법을 점진적 검사점 기법에서의 검사점 작성 시점 결정에 사용하는 것이 좋을 것이다.

6. 결론 및 앞으로의 연구 방향

점진적 검사점 기법은 검사점 사이에서 변경된 페이지만을 저장함으로써 검사점의 오버헤드를 감소시키는 기법으로, 실제 시스템에도 상당히 많이 쓰이는 검사점 기법이다. 본 논문에서는 페이지 단위 점진적 검사점 기법에 대한 약간의 설명과 함께, 점진적 검사점 기법에서의 효율적이고 적응성 있는 검사점의 작성 시점 결정 방법을 제안하였다. 페이지 단위 점진적 검사점 기법을 사용하는 프로세스의 예상 수행 시간과 예상 복원 시간에 대한 비용 분석을 통하여, 비용 효율적인 검사점의 위치를 결정하는 알고리즘을 제시하였고, 여러 응용 예제를 통한 실험을 수행하였다. 실험 결과를 통하여, 본 논문에서 제안한 방법을 사용한 것이, 기존의 고정 인터벌을 갖는 점진적 검사점 기법을 사용하는 것보다 나은 성능을 보임을 확인하였다.

앞으로는 본 논문에서 제안한 방법을 사용하는 점진적 검사점 기법을 모바일 환경 및 분산 환경에서 효율적으로 접목시키고, 다중 처리기 환경에서의 검사점 비용의 효율성에 대하여 연구할 것이다. 또한 경성 실시간 시스템의 마감시한을 지키면서 검사점을 수행해야 할 경우, 어떻게 스케줄 가능성을 보장할 수 있는가에 대한 연구도 수행할 것이다.

참 고 문 헌

[1] J. Hong, S. Kim and Y. Cho, Cost Analysis of Optimistic Recovery Model for Forked Checkpointing, IEICE Transactions on Information and Systems, Vol.E86-D, No.9, pp.1534-1541, Sep., 2003.

[2] J. Plank, M. Beck and G. Kingsley, Compiler-Assisted Memory Exclusion for Fast Checkpointing, IEEE Technical Committee on Operating Systems and Application Environments, Special Issue on Fault-Tolerance, pp.62-67, Dec., 1995.

[3] A. Ziv and J. Bruck, An On-Line Algorithm for Checkpoint Placement, IEEE Transactions on Computers, Vol.46, No.9, pp. 976-985, Sep., 1997.

[4] J. Lawall and G. Muller, Efficient Incremental Checkpointing of Java Programs, IEEE Proceedings of the International Conference on Dependable Systems and Networks, pp.61-70, Jun., 2000.

[5] J. Plank, M. Beck and G. Kingsley, and K. Li, Libckpt:Transparent Checkpointing under Unix, Usenix Winter Technical Conference, pp.213-223, Jan., 1995.

[6] J. Plank, J. Xu, and R. Netzer, Compressed differences: An algorithm for fast incremental checkpointing, Technical Report CS-95-302, Aug., 1995.

[7] J. Plank, K. Li and M. Puening, Diskless Checkpointing, IEEE Transactions on Parallel and Distributed Systems, Vol.9, No. 10, pp.303-308, Oct., 1998.

[8] J. Plank, Y. Chen, K. Li, M. Beck and G. Kingsley, Memory exclusion: optimizing the performance of checkpointing systems, Software Practice and Experience, Vol.29, No.2, pp.125-142, Feb., 1999.

[9] M. Beck, J. S. Plank and G. Kingsley, Compiler-Assisted Checkpointing, Technical Report of University of Tennessee, UT-CS-94-269, 1994.

[10] A. Duda, The effects of checkpointing on program execution time. Information Processing Letters 16, pp.221-229, 1983.

[11] J. Heo, S. Yi, Y. Cho, J. Hong, S.Y. Shin, Space-efficient Page-level Incremental Checkpointing. Proceedings of the 2005 ACM symposium on Applied computing, pp.1558-1562, 2005.

이 상 호

e-mail : shyi@ssrnet.snu.ac.kr

2003년 고려대학교 전기전자전파공학부(학사)

2003년~현재 서울대학교 컴퓨터공학부 박사과정(박사수료)

관심분야 : 시스템 및 운영체제, 무선 센서 네트워크, 센서 운영체제, 결함 허용 시스템 등

허 준 영

e-mail : jyheo@ssrnet.snu.ac.kr

1998년 서울대학교 컴퓨터공학과(학사)

2002년~현재 서울대학교 컴퓨터공학부 박사과정(박사수료)

관심분야 : 시스템 및 운영체제, 무선 센서 네트워크, 결함 허용 시스템, 임베디드 시스템 보안 등

홍 지 만

e-mail : gman@daisy.kw.ac.kr

2004년 서울대학교 컴퓨터공학부(공학박사)

2004년~현재 광운대학교 컴퓨터공학부 조교수

관심분야 : 운영체제 및 임베디드 시스템, 임베디드 운영체제, 결함 허용 컴퓨팅, 분산 시스템, 무선 센서 네트워크 등