실시간 시스템에서의 효과적인 Checkpointing Interval에 대한 연구

°변계섭, 김재훈
아주대학교 정보통신전문대학원

A Study of Optimal Checkpointin g Interval in Real-Time Systems

°Gye-Sub Byeon, Jai-Hoon Kim
College of Information and Communication
Ajou University

요 약

실시간 시스템에서 예상치 못한 오류 발생은 성능에 악영향을 미친다. 이를 예방하기 위하여 체크포인팅이라는 후방 에러복구기법을 이용하여 오류 발생시에도 예측 가능한 결과를 보장할 수 있다. 실시간 시스템에서의 체크포인팅은 비실시간 시스템과는 달리 시간제약성을 만족시켜야 하기 때문에 비실시간에서 최적인 체크포인팅 간격과는 다르게 고려 되어야 한다. 이런 체크포인트 간격에 따른 성능의 차이를 시뮬레이션을 통하여 확인하였고 결과를 분석하였다.

1. 서 론

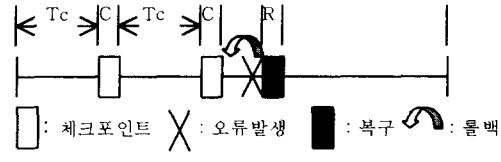
실시간 컴퓨팅에서 가장 필요한 요소중의 하나는 데드라인을 지키는 것에 있다. 이를 위해서는 컴퓨팅 중에 일어나는 오류를 감지하여 데드라인안에 작업을 끝낼 수 있도록 하여야 한다. 이를 위해서 후방 에러 복구기법(backward error recovery)으로 알려진 체크포인팅과 롤백을 이용하여 데드라인 안에 작업을 종료할 수 가능성을 높인다. 체크포인트와 롤백은 오류가 일어 났을 때 컴퓨팅의 손실을 최소화할 수 있는 기술이다. 체크포인트는 안정된 상태에서 주기적으로 어플리케이션 타스크의 상태를 저장한다. 타스크는 체크포인트를 이용하여 주기적으로 자신의 상태를 저장하고, 오류가 발생할 때 가장 최근의 체크포인트로 롤백하여 타스크를 회복시킬 수 있다[1,5].

본 논문에서는 실시간 시스템에서 최적의 체크포인트 간격이 일반 시스템과 다르며 이 차이를 시뮬레이션을 통하여 확인하고 분석하였다. 프로그램 수행 도중 오류가 발생되면 가장 최근의 체크포인트로 롤백하여 다시 진행하기 때문에 체크포인팅 간격을 좁게 하면 롤백 간격을 좁힐 수 있지만, 체크포인팅을 하는 과정에서 적지않은 비용이 소요되어 전체적인 성능을 저해하는 요소로 작용할 수 있다. 이를 위해서 적절한 간격이 요구된다. 최적의 체크포인팅 간격을 결정하는데 체크포인팅에 드는 오버헤드와 오류발생율을 고려해야 한다[5,7]. 비실시간 시스템에서는 고려되지 않는 시간제약을 고려하여 실시간 시스템에서는 어떻게 간격을 두어야 하는가를 고려하고 성능 평가를 하였다.

2. 관련연구

기존의 비실시간 시스템에서의 오류 발생율대 최적의 체크포인팅 간격에 관한 연구는 많이 이루어져 왔다[3,5,7].

오류가 발생하였을 때 영향을 최소화 하기 위해서 수행도중 일정한 주기로 체크포인팅을 하고 현재의 상태를 안전한 저장장치에 저장한다. 오류가 발생하면 가장 최근에 저장된 체크포인트로 롤백하여 그 시점부터 다시 시작하게 된다.



[그림 1] 체크포인트와 롤백 기법

[그림1]은 오류가 발생하였을 때 성능향상을 위해 수행되는 체크포인팅 과정을 나타낸다. 전체 실행시간을 T, 롤백 시간을 R, 체크포인팅 오버헤드를 C라 하고, 오류발생율을 λ라 할 때, 최적의 체크포인트 간격 T_{opt}는 아래와 같다.

$$T_{opt} = \sqrt{\frac{2C}{\lambda}} \quad [5,7]$$

위 식에서 알 수 있듯이 체크 포인팅 코스트가 일정하며, λ가 증가함에 따라 체크포인팅을 자주해 주어야 함을 알 수 있다. λ와 T_{opt}는 제곱근에 반비례적인 관계이고 C 와는 제곱근에 비례함을 알 수 있다.

결함허용 실시간 시스템을 위하여 중복 기법에 관한 연구가 이루어졌다[6]. 많은 실시간 시스템에서는 성능과 신뢰성을 요구하게 된다. 성능은 한번에 실행할 수 있는 태스크의 수나 양으로 결정된다. 신뢰성은 하드웨어나 소프트웨어의 결함모델에 의해 검증된다. 여러 상황에서 태스크에 대한 성능이나 신뢰성이 상충되는 경우가 많다. 실시간 결함허용 시스템의 검증을 위해서는 성능-신뢰성의 상충에 대한 수학적 평가가 필요하다. 만약 신뢰성 향상을 위해서 태스크를 수행시킬 때 똑같은 카피를 두는 방법을 사용한다면, 그 카피의 정도에 대한 결정이 필요하다.

3. 성능 평가

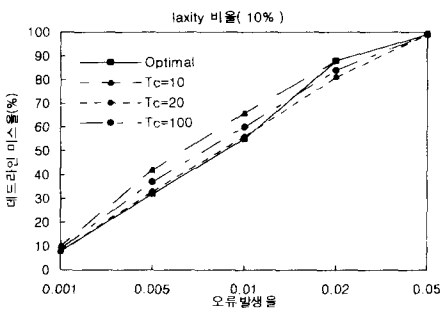
실시간 시스템에서는 비실시간 시스템에서와는 다른 최적의 체크포인트 간격이 존재한다. 이는 실시간 시스템에서의 시간 제약성에 기반한다. 체크포인트 간격을 설정함에 있어서

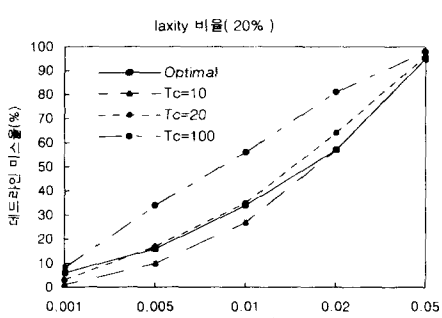
실시간 시스템에서는 오류발생율이 높아질수록 체크포인트 간격을 자주해야함을 나타내며 이는 실시간 시스템에서는 주어진 시간 안에 작업을 끝내는 것이 중요하므로 비실시간 시스템에서의 체크포인트 간격보다 더 자주 발생시킬 수 있음을 의미한다. 또한 데드라인이 길어질수록 더 체크포인트 간격을 좁게하여 성능을 향상시킬 수 있다. 실시간 시스템에서의 최적 체크포인팅 간격을 결정하기 위해 아래와 같이 시뮬레이션 변수를 가정하였다. 애플리케이션에서 하나의 프로세스가 작업을 한다고 가정한다. 실시간 시스템에서 성능 평가 기준은 여러가지가 있을 수 있지만(데드라인 준수율, 데드라인을 지나친 정도 등) 본 논문에서는 데드라인 준수율을 성능 평가의 기준으로 삼았다. 성능평가에 사용된 환경 변수는 다음과 같다.

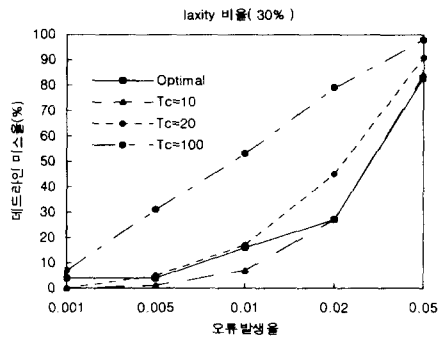
- R = 1 (롤백 오버헤드)
- C = 1 (체크포인팅 오버헤드)
- 수행시간 : 100
- 오류발생율 : { 0.001, 0.005, 0.01, 0.02, 0.05 }
  (Poisson process)
- Tc (체크포인트 간격)
  (Tc = 100은 체크포인트 않음을 나타냄)
- laxity 비율 : { 10%, 20%, 30%, 40%, 50%}
  ((수행시간 + laxity)/수행시간 x 100)%)

오류발생율과 체크포인팅 간격, laxity 비율을 변화하면서 시뮬레이션을 통하여 성능을 측정하였다. 전체 실험 횟수는 100,000번으로 데드라인 미스율을 구하였다.

- **오류발생율에 대한 데드라인 미스율**

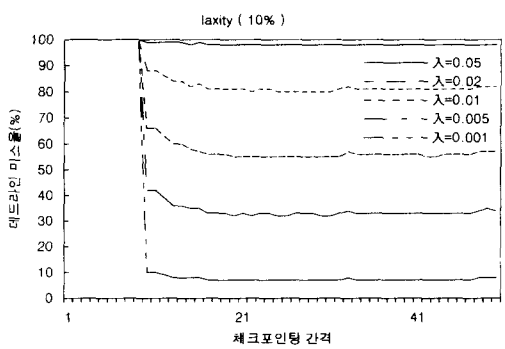


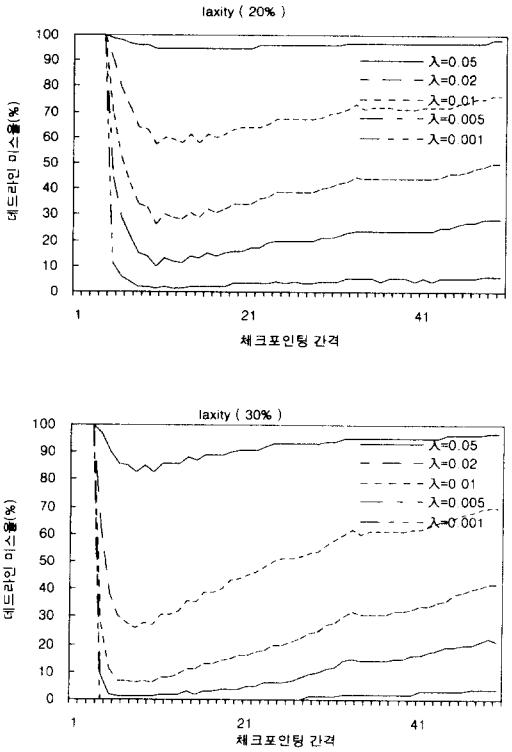




[그림 2] 오류발생율에 따른 데드라인 미스율

[그림2]에서 나타난 것과 같이 비실시간 시스템에서 최적인 체크포인트 간격(optimal)이 실시간 시스템에서는 적용되지 않는다. 예를들면, laxity가 20% 일 때의 체크포인트 간격을 보면 최적인 것보다 Tc=10일 때에 더 좋은 성능을 보이고 있다. 또한 30% 일 때에는 더 많은 성능의 차이를 보이고 있다. 이는 실시간 시스템에서는 비실시간 시스템과는 달리 데드라인이 존재하므로 평균 수행시간을 줄이는 것보다 그 데드라인을 준수하는 것이 중요하기 때문에 체크포인팅을 자주해주면 상대적으로 예측할 수 있는 시간에 타스크가 종료되므로 성능향상에 도움이 된다는 것을 알 수 있다. 반면에 체크포인팅을 적게 할 경우, 오류가 발생한 지점에서 이전 체크포이트한 지점으로 다시 되돌아가는데 걸리는 오버헤드가 많아져서 종료시간을 예측하기 힘들기 때문에 실시간 시스템에서 성능향상을 기대할 수 없다. 그러나 laxity가 적어(10%인 경우) 체크포인트 오버헤드가 부담이 되거나 오류발생율이 매우 낮을 때는 체크포인팅을 자주하는 것이 성능향상에 도움이 되지 못한다.





[그림 3] 체크포인팅 간격대 데드라인 미스율(%)

[그림3]에서는 체크포인트 간격에 대한 데드라인 미스율을 나타내고 있다. 간격이 아주 작거나 클 때에는 전체적인 성능의 향상을 기대할 수 없다. 그림에서도 알 수 있듯이 최적의 체크포인트 간격이 존재한다. 실시간 시스템에서의 최대 성능을 나타내는 체크포인트 간격은 비실시간 시스템에서의 간격과 일치하지 않는다. 그러므로 실시간 시스템에서는 비실시간 시스템에서의 최적의 체크포인트 간격과 다른 값을 사용해야 한다.

[표 1] 비실시간대 실시간 시스템의 최적 체크포인트 간격

오류발생율	0.001	0.005	0.01	0.02	0.05
비실시간	44	20	14	10	6
실시간(10%)	17~41	20, 30	20~32	25~31	17~96
실시간(20%)	10, 12	13	13	13	15
실시간(30%)	4~47	6~10	6, 7	8	8,10

[표1]에서 나타난 값들은 오류발생율에 따른 비실시간과 실시간 시스템에서의 최적 또는 최적에 가까운(sub-optimal)체크포인팅 간격을 나타낸다.

4. 결론 및 향후 과제

시뮬레이션을 통하여 비실시간 시스템과 실시간 시스템에서 최적의 체크포이팅 간격이 서로 다르게 나타남을 알 수 있다. 일반적으로 laxity가 늘어나게 되면서 체크포인팅 간격이 좁아짐을 알 수 있다. 일반적으로 laxity에 여유가 있다면 좀 더 자주 체크포인팅을 할 경우 상대적으로 예측 가능한 타스크 종료 시간을 보장받을 수 있다. 향후 실시간 시스템에서 최적의 체크포인팅 간격을 수학적으로 정리하는 연구를 진행할 예정이다.

참 고 문 헌

[1] K. Chandy and L. Lamport, "Distributed snapshots: Determining global states in distributed systems," *IEEE Transactions on Computer systems*, vol.3, pp.63-75, Feb.1985.

[2] E. Elnozah y, D. Johnson, and W. Zwaenepoel, "Measured performance of consistent checkpointing," in *Proc. of the Eleventh Symposium on Reliable Distributed Systems*, pp. 33 -47, Oct. 1992.

[3] Jai-Hoon Kim and Nitin H. Vaidya, "Analysis of one-level and two-level failure recovery schemes for distributed shared memory system," IEE Proceedings-Computers and Digital Techniques, vol. 146, Issue 3, pp125-130, May 1999.

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

[5] N. H Vaidya, "On Checkpoint latency," in *Proc. Of the 1995 pacific Rim International Symposium on Fault-Tolerant Systems*, pp60-65, Dec. 1995.

[6] F. Wang, K. Ramamritham and J. A. Stankovic: Determining Redundancy Levels for Fault Tolerant Real-Time Systems, Special Issue of IEEE Transactions on Computers on Fault Tolerant Computing, Vol. 44, No. 2, February 1995, pp. 292-301.

[7] J. Young, "A first order approximation to the optimal checkpoint interval," Communication of the ACM, Vol.17 pp530-531, Sept, 1974.