실시간 시스템에서의 효과적인 체크포인트 간격에 대한 연구

정회원 변 계 섭*, 김 재 훈*

A Study on Optimal Checkpointing Interval in Real-Time Systems

Kyue-Sup Byun*, Jai-Hoon Kim* *Regular Members*

요 약

실시간 시스템에서 예상치 못한 오류 발생은 성능에 악영향을 미친다. 이를 예방하기 위하여 체크포인팅(checkpointing)이라는 후방 에러복구기법을 이용하여 오류 발생시에도 예측 가능한 결과를 보장할 수 있다. 실시간 시스템에서의 체크포인팅은 비실시간 시스템과는 달리 시간제약성을 만족시켜야 하기 때문에 비실시간에서 최적인 체크포인팅간격과는 다르게 고려되어야 한다. 본 논문에서는 체크포인트 간격에 따른 실시간 시스템과 비실시간 시스템간의 성능의 차이를 시뮬레이션을 통하여 확인하였고 결과를 분석하였다.

ABSTRACT

Checkpoint and rollback recovery scheme can be used to reduce the loss of computation upon failures. Many researchers presented methods to compute optimal checkpoint intervals to minimize expected execution time. In real-time computation with time constraints, optimal checkpoint interval is different from those in previously proposed schemes to reduce average computation time. By the simulation results, we find that optimal checkpoint intervals to maximize the probability of task's meeting the deadline depend on failure rates and laxity ratios. We can choose optimal checkpoint intervals to maximize the ratio of meeting the deadline in real-time computations.

I. 서 론

실시간 컴퓨팅에서 가장 필요한 요소중의 하나는 컴퓨팅 결과의 정확성 이외에 시간의 제약성을 만족시키는 것이다. 이를 위해서는 컴퓨팅 중에 일어나는 시스템의 장애를 감지하여 데드라인안에 작업을 끝낼 수 있도록 하여야 한다. 이를 위해서 후방 에러 복구기법(backward error recovery)으로 알려진 체크포인팅(checkpointing)과 롤백(rollback)을 이용하여 데드라인(deadline)안에 작업을 종료할 수 있는 가능성을 높일 수 있다.

체크포인트와 롤백은 시스템의 장애가 발생했을 때, 컴퓨팅의 손실을 최소화할 수 있는 기법이다. 체크포인트는 어플리케이션 태스크의 상태를 안정된 저장장치(예:하드디스크)에 저장한다. 태스크는 체크포인트를 이용하여 주기적으로 자신의 상태를 저장하고, 장애가 발생할 때 가장 최근의 체크포인트(checkpoint)로 롤백하여 타스크(task)를 회복시킬 수 있다[1,5]. 프로그램 수행 도중 장애가 발생되면 가장 최근의 체크포인트로 롤백하여 다시 진행하기 때문에 체크포인팅 간격을 좁게 하면 롤백 간격(소요 비용)을 좁힐 수 있지만, 체크포인팅을 하는 과정에서 적지않은 비용이 소요되어 전체적인 성능을 저해하는 요소로 작용할 수 있다. 이를 위해서 적절한 체크포인팅 간격이 요구된다. 최적의 체크포인팅 간격을 결정하는데 체크포인팅에 드는 오버헤드와 오류발생율을 고려해야 한다[5,7]. 비실시간 시스템과는 달리 시간제약이 존재하는 실시간 시스템에서는

* 아주대학교 정보통신전문대학원({ksbyun,jaikim}@madang.ajou.ac.kr)
논문번호 : K01112-0403, 접수일자 : 2001년 4월 3일
※ 이 논문은 2000년도 학술진흥재단의 지원에 의하여 연구되었음(KRF-2000-041-E00293)

어떻게 체크포인팅 간격을 조정해야 하는가를 고려하여야 한다. 체크포인팅 간격을 작게 하는 경우, 시스템 장애 발생시 컴퓨팅 시간 낭비가 적어 상대적으로 예측 가능한 태스크 종료 시점을 얻을 수 있지만, 체크포인팅 비용이 증가하므로 시간 제약이 내에 태스크 종료를 보장 받지 못할 경우가 있다.

본 논문에서는 실시간 시스템에서 최적의 체크포인트 간격이 일반 시스템과 다르며 이 차이를 시뮬레이션을 통하여 확인하고 분석하였다.

Ⅱ. 체크포인팅 간격

기존의 비실시간 시스템에서의 오류 발생율에 따른 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.

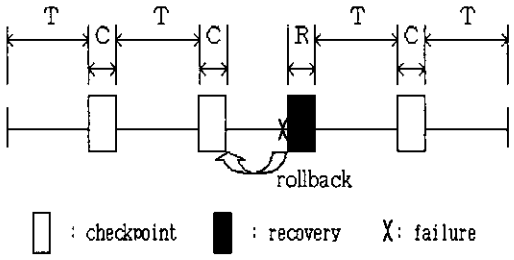


그림 1. 체크포인트(checkpointing)와 롤백(rollback) 기법

그림1은 오류가 발생하였을 때, 성능향상을 위해 수행되는 체크포인팅 과정을 나타낸다. 전체 태스크의 실행시간을 T, 롤백 시간을 R, 체크포인팅 비용을 C라 하고, 오류발생율을 λ 라 하고 포아송(poisson) 프로세스를 따른다고 할 때, 최적의 체크포인트 간격 T_{opt} 는 아래와 같다.

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

위 식에서 알 수 있듯이 체크포인팅 비용이 일정할 경우, λ 가 증가함에 따라 체크포인팅 간격을 좁게 해주어야 함을 알 수 있다. λ 와 T_{opt}는 제곱근에 반비례적인 관계이고 C와는 제곱근에 비례함을 알 수 있다. 그러나, 이러한 최적의 체크포인트 간격(optimal checkpoint interval)은 평균 수행시간(average execution time)을 줄이는 것 보다 데드라인 준수가 중요한 실시간 컴퓨팅 환경에 비효과적인 경우가 있다. 만약 수행 시간 분포가 표준 정규분포 $N(\mu, \sigma^2)$ -μ : 평균, σ : 표준편차-를 따른다면, 평균 수행시간을 줄이는 것이 유리한 비실시간 컴퓨팅에서는 $N(10,2^2)$는 $N(11,1)$보다 좋은 성능을 보인다. 반면에 만약 데드라인이 14인 실시간 컴퓨팅 환경에서는 $N(11,1)$가 $N(10,2^2)$보다 좋은 성능을 보인다. 데드라인이 14일 때, $N(11,1)$은 데드라인을 99.9%를 준수하는 반면 $N(10.2^2)$은 97.7%를 준수한다. 그러나, 만약 데드라인이 11이면, $N(10,2^2)$은 $N(11,1)$보다 실시간 컴퓨팅 환경이나 비실시간 컴퓨팅 환경 모두에서 좋은 성능을 보인다. 그러므로 데드라인과 실행 시간의 표준 편차는 평균 수행시간 이외에도 실시간 컴퓨팅 환경에서 성능에 영향을 미치는 중요한 요소가 된다. 다시 말하면, 만약 실행 시간의 확률 밀도 함수가 f(t)이고, 데드라인이 D라고 하면, 데드라인 준수율은 아래와 같이 계산할 수 있다.

$$\int_o^D f(t)dt$$

체크포인트 간격에 따라 실행 시간의 표준 편차를 조정할 수 있으며 평균 수행 시간을 최소로 하였을 때 최대의 성능을 보이는 비실시간 컴퓨팅 환경과 다른 실시간 컴퓨팅 환경에서 최적의 체크포인트 간격을 구할 수 있다. 체크포인트 간격을 C, 이 때, 실행 시간의 확률 밀도 함수를 f(t,C), 데드라인을 D라고 하면, 실시간 컴퓨팅 환경을 위한 최적의 체크포인트 간격은 $C_{rt\text{-}opt}$ 은 아래의 식을 만족한다.

$$\int_0^D f(t, C_{rt-opt})dt \geq \int_0^D f(t, C)$$

for all $0 < C \leq e$

체크포인팅은 결함이 발생하였을 때, 컴퓨팅의 손실을 줄일 수 있으며 태스크의 완료 시간을 최소화 할 수 있다. 체크포인트 간격은 태스크의 수행시간에 영향을 미친다. 만약 체크포인트 간격이 좁으면, 결함발생 후에 다시 프로세싱하는 시간을 줄일 수 있는 반면에 체크포인팅 비용이 증가하여 전체 태스크 수행 시간이 증가된다. 결함허용 실시간 시스템을 위하여 중복 기법에 관한 연구가 이루어졌다[6]. 많은 실시간 시스템에서는 성능과 신뢰성을 요

구하게 된다. 이를 만족시키기 위하여 시간 제약을 만족시키면서 동시에 실행할 수 있는 적정한 태스크의 수를 결정해야 한다. 여러 상황에서 태스크에 대한 성능이나 신뢰성이 상충되는 경우가 많다. 실시간 결함허용 시스템의 검증을 위해서 성능-신뢰성의 상충에 대한 연구가 이루어졌다. 만약 신뢰성 향상을 위해서 태스크를 수행시킬 때 똑같은 카피(copy)를 두는 방법을 사용한다면, 그 카피의 정도에 대한 결정이 필요하다.

Ⅲ. 성능평가

실시간 시스템에서는 비실시간 시스템에서와 다른 최적의 체크포인트 간격이 존재한다. 이는 실시간 시스템에서의 시간 제약성 때문에 체크포인트 간격을 설정함에 있어서 실시간 시스템에서는 오류발생율이 높아질수록 체크포인트 간격을 좁게 해야 한다. 실시간 시스템에서는 주어진 시간 안에 작업을 끝내는 것이 중요하므로 평균 태스크 수행 시간을 크게 증가시키지 않는 범위 내에서 비실시간 시스템에서의 체크포인트 간격보다 더 자주 체크포인팅을 실시해야 하는 경우가 많다. 데드라인이 길어질수록 여유 시간이 많으므로 체크포인트 간격을 좁게 하여 성능을 향상시킬 수 있다. 실시간 시스템에서의 최적의 체크포인팅 간격을 결정하기 위해 아래와 같이 시뮬레이션 변수를 가정하였다. 애플리케이션에서 하나의 태스크가 작업을 한다고 가정한다. 실시간 시스템에서 성능 평가 기준은 여러 가지가 있을 수 있지만(데드라인 준수율, 데드라인을 지나친 정도 등) 본 논문에서는 데드라인 준수율을 성능평가의 기준으로 삼았다. 성능평가에 사용된 환경변수는 다음과 같다.

- R = 1 : 롤백 오버헤드(rollback overhead)
- C = 1 : 체크포인팅 오버헤드 (checkpointing overhead)
- 수행시간 : 100
- 오류발생율 : {0.001, 0.005, 0.01, 0.02, 0.05} Poisson process로 가정
- Tc : 체크포인트 간격 (체크포인트 간격이 수행 시간과 동일 할 때는 체크포인트를 수행하지 않음을 나타냄)
- laxity 비율 : {10%, 20%, 30%, 40%, 50%} ((수행시간 + laxity)/수행시간 x 100)%)

오류발생율과 체크포인팅 간격, laxity 비율을 변화하면서 시뮬레이션을 통하여 성능을 측정하였다 (laxity = 데드라인 - 수행시간). 전체 실험 횟수는 100,000번으로 데드라인 미스(miss)율을 구하였다.

그림 2에서 "optimal"은 비실시간 시스템에서 최적의 체크포인팅 간격을 나타내고, "Tc=x"는 체크포인트 간격이 x임을 나타낸다. 그림 2에서와 같이 비실시간 시스템에서 최적인 체크포인트 간격이 실시간 시스템에서는 적용되지 않는다. 예를 들면, laxity가 20%일 때의 체크포인트 간격을 보면 비실시간일 때, 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 태스크가 종료되므로 성능향상에 도움이 된다는 것을 알 수 있다. 반면에 체크포인팅을 적게 할 경우, 시스템 장애가 발생했을 때, 이전 체크포인트한 지점으로 다시 되돌아가는데 손실이 많아져서 종료시간을 예측하기 힘들기 때문에 실시간 시스템에서 성능 향상을 기대할 수 없다. 그러나 laxity가 적어(10%인 경우) 체크포인트 오버헤드가 부담이 되거나 오류발생율이 매우 낮을 때는 체크포인팅을 자주하는 것이 성능향상에 도움이 되지 못한다.

그림3에서는 체크포인트 간격에 대한 데드라인 미스율을 나타내고 있다. 간격이 아주 작거나 클 때에는 전체적인 성능의 향상을 기대할 수 없다. 그림 3에서도 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.

표1에서 나타난 값들은 오류발생율에 따른 비실시간과 실시간 시스템에서의 최적 또는 최적에 가까운(sub-optimal) 체크포인팅 간격을 나타낸다.

Ⅳ. 결론 및 향후계획

체크포인팅은 결함이 발생하였을 때, 결함에 의한 컴퓨팅 손실 시간을 줄일 수 있으며 태스크 수행시간을 최소화할 수 있다. 체크포인트 간격은 태스크의 수행시간에 영향을 미친다. 만약 체크포인트

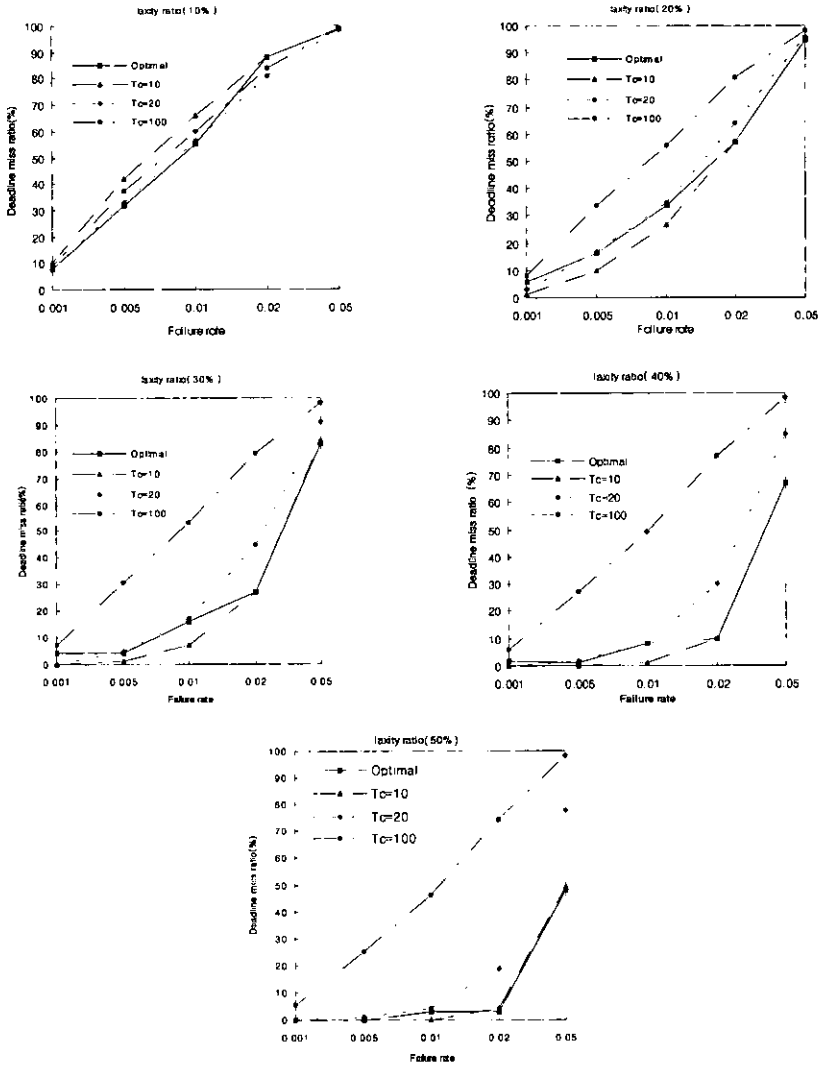
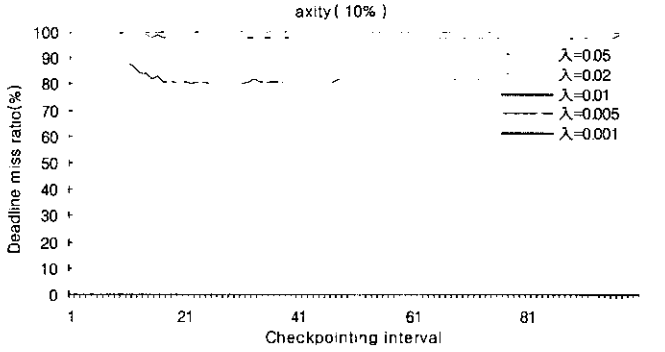
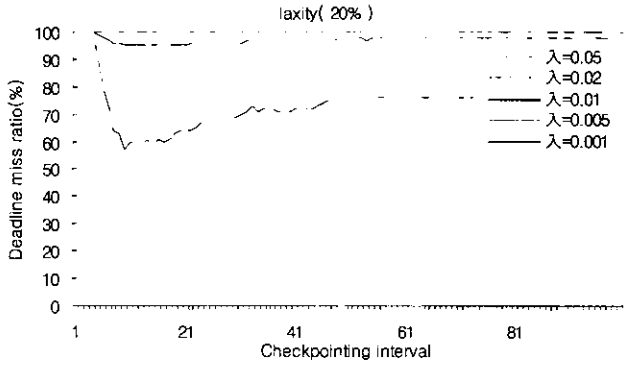


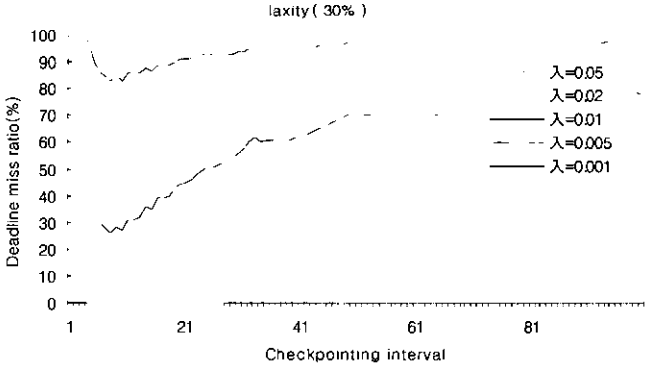
그림 2. 오류발생율에 따른 데드라인 미스율(laxity 비율=10%,20%,30%,40%,50%)

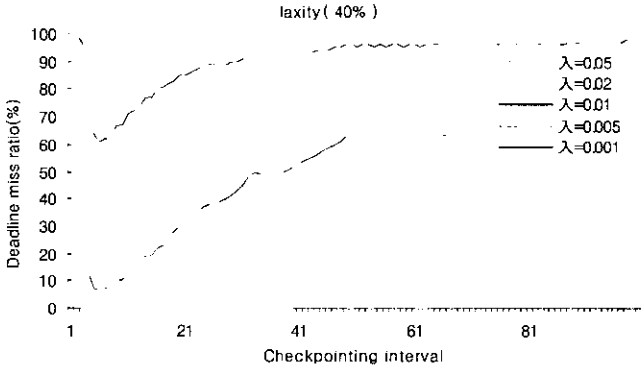
표 1. 비실시간대 실시간 시스템의 최적 체크포인트 간격

Failure rate(λ)	0.001	0.005	0.01	0.02	0.05
Non real-time	44	20	14	10	6
Real-time(laxity=10%)	17~41	20, 30	20~32	25~31	17~96
Real-time(laxity=20%)	10, 12	13	13	13	15
Real-time(laxity=30%)	4~27	6~10	6, 7	8	8,10
Real-time(laxity=40%)	3-39	4-17	5-8	5-7	6
Real-time(laxity=50%)	3-53	3-23	3-13	5	5


axity(10%)
Deadline miss ratio(%)
Checkpointing interval
入=0.05
入=0.02
入=0.01
入=0.005
入=0.001


laxity(20%)
Deadline miss ratio(%)
Checkpointing interval
入=0.05
入=0.02
入=0.01
入=0.005
入=0.001


laxity(30%)
Deadline miss ratio(%)
Checkpointing interval
入=0.05
入=0.02
入=0.01
入=0.005
入=0.001


laxity(40%)
Deadline miss ratio(%)
Checkpointing interval
入=0.05
入=0.02
入=0.01
入=0.005
入=0.001

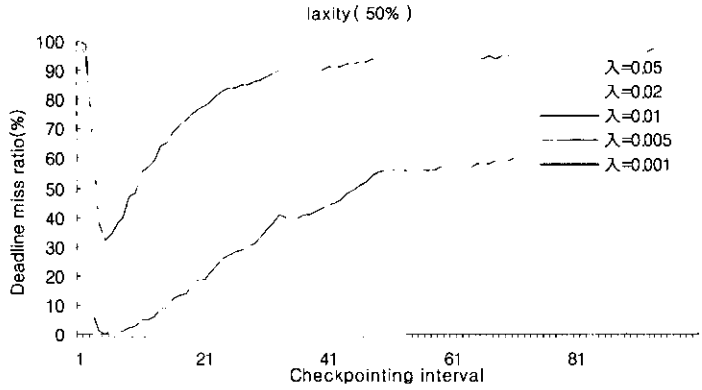


그림 3. 체크포인팅 간격에 따른 데드라인 미스율(%) (laxity=10%,20%,30%,40%,50%)

간격이 잦으면, 결함발생 후에 다시 프로세싱하는 시간을 줄일 수 있는 반면에 체크포인팅 비용이 증가된다. 데드라인 준수가 중요한 실시간 시스템에서는 평균 수행시간의 단축이 중요한 비실시간 시스템과는 다른 최적의 체크포인팅 간격이 존재하며 이를 시뮬레이션을 통하여 검증하였다. 일반적으로 laxity가 늘어나게 되면 실시간 컴퓨팅을 위한 최적의 체크포인팅 간격이 좁아짐을 알 수 있다. 일반적으로 laxity에 여유가 있다면 좀 더 자주 체크포인팅을 할 경우 상대적으로 예측 가능한 태스크 종료 시간을 데드라인 이내에 보장 받을 수 있다.

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. Elnozahy, 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, pp. 125-130, May 1999.

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

[5] N. H. Vaidya, "On Staggered Checkpointing," Symposium on Parallel and Distributed Processing(SPDP '96), 1996.

[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, pp. 292-301, No. 2, February 1995.

[7] J. Young, "A first order approximation to the optimal checkpoint interval," *Communication of the ACM*, Vol.17, pp. 530-531, Sept. 1974.

변 계 섭(Kyue-Sup Byun) 학생회원

1999년 : 아주대학교 정보 및 컴퓨터 공학부 졸업(학사)

2000년 : 현재 아주대학교 정보통신전문대학원 석사과정

<주관심 분야> 실시간 시스템, 이동컴퓨팅, 결함허용 시스템

김 재 훈(Jai-Hoon Kim) 정회원

1984년 : 서울대학교 제어계측공학과 (학사)
1993년 : Indiana University, Computer Science (석사)
1997년 : Texas A&M University, Computer Science (공학박사)
1984년~1991년 : 대우통신㈜ 컴퓨터연구실, 팀장
1995년~1997년 : Texas A&M University, Graduate Research Assistant
1997년~1998년 : 삼성전자㈜ 컴퓨터시스템팀, 수석연구원
1998년~현재 : 아주대학교 정보통신전문대학원, 조교수
<주관심 분야> 분산시스템, 이동컴퓨팅, 실시간시스템