분산 실시간 시스템에서 신뢰성 향상을 위한 통신 부하 분석

구현우° 홍영식
동국대학교 컴퓨터공학과
{hwgoo°, hongys}@dgu.ac.kr

Communication Overhead Analysis for Improving Reliability in Distributed Real-Time Systems

Hyun-Woo Goo° Young-Sik Hong
Dept. of Computer Engineering, Dongguk University

요 약

실시간 시스템은 논리적 정확성뿐만 아니라 시간적 정확성을 요구한다. 시간적 정확성을 만족시키기 위해 실시간 시스템의 설계자는 작업들의 스케줄 가능성에 대한 연구를 선행해야만 한다. 그리고 스케줄 가능성 분석을 위해 프로그램들에 대한 실행 시간 예측이 필요하다. 작업들의 실행 시간 예측을 위한 방법으로 측정과 정적 분석이 연구되었다. 측정 및 정적 분석은 비용 및 확장성에 문제점을 지니고 있고 실시간 시스템의 발전을 따라가지 못하여 분석 결과의 정확성 및 신뢰성이 만족스럽지 못하다. 본 논문에서는 정적 분석을 단일 실시간 시스템이 아닌 분산 실시간 시스템에 적용할 수 있는 확장된 정적 분석 도구의 개발에 초점을 둔다. 먼저 확장된 정적 분석 도구의 개발을 위해 통신 영향 요소의 분석 과정을 설계한다. 특히, 통신 부하의 영향 요소 중 통신 준비에 필요한 과정을 선행 예측 테이블로 작성하여 원시 프로그램 분석에 이용하고자 한다. 실행 시간에 영향을 미치는 요소들의 분석을 통해 원시 프로그램에서 자동적으로 예측된 실행 시간의 정확도와 신뢰도를 높인다.

1. 서 론

실시간 시스템의 가장 큰 특징은 시스템에서 요구되는 논리적 정확성, 뿐만 아니라 시간적 정확성을 필요로 한다는 것이다. 시간적 정확성을 만족시키기 위해 스케줄 가능성(Schedulability)에 대한 연구가 필요하였고 지금까지 많은 연구가 진행되어 왔다. 스케줄 가능성 분석은 실시간 시스템 설계에 있어 가장 중요하고 어려운 작업이다. 스케줄 가능성 분석을 위해 시스템에서 작동할 원시 프로그램의 실행 시간 분석 및 예측이 선행되어야 한다.

점차 사용자들의 컴퓨팅 파워에 대한 요구가 증대되고 하드웨어의 발달 및 네트워크의 발달로 인해 실시간 시스템에서 작동될 많은 프로그램들이 복잡해지고 프로그램의 크기가 증가되었다. 단일 실시간 시스템을 이용하던 사용자들이 분산 실시간 시스템을 이용하여 보다 효율적인 시스템 구축을 원하게 됨으로써 전통적인 정적인 실행 시간 분석만으로는 정확하고 신뢰할 수 있는 최악 실행 시간의 예측이 불가능하게 되었다.[1]

분산 실시간 시스템에서의 정확하고 신뢰할 수 있는 실행 시간의 예측을 위해서는 운영체제 및 통신 부하를 예측하여 확장된 정적 분석 기법 및 동적 분석 기법이 필요하다. 이에 본 논문에서는 분산 실시간 시스템에서 통신에 의해 발생되는 부하의 정적이고 자동적인 분석을 위한 실시간성 통신 요소 분석 방법 및 분석기의 설계를 제시한다.

본 논문의 2장에서는 정적 실행 시간 분석을 위한 관련 연구를 살펴보고 3장에서는 실시간성 분석도구의 전체 구조를 살펴본다. 4장에서는 RPC(Remote Procedure Call)와 관련된 리눅스 커널 소스의 분석을 통해 실시간성 분석에 영향을 미치는 요소인 통신 영향 요소를 살펴보고 분석된 정보를 이용하여 실시간성 분석 도구와 통합한다. 5장에서 실행 시간 분석의 신뢰도를 높이기 위한 향후 연구 과제를 언급한기로 한다.

2. 관련 연구

실시간 시스템의 중요한 요소인 시간적 정확성을 만족하기 위해 실시간 작업 스케줄링 기법에 관한 많은 연구가 진행되어 왔다. 그와 병행하여 작업들의 스케줄 가능성의 분석을 위한 실시간 시스템에서 실행될 프로그램의 실행 시간 분석에 관한 연구 또한 활발히 진행되어 왔다.

전통적인 실행 시간 분석 기법은 실시간성 분석을 위해 원시 프로그램 단위 요소의 흐름을 분석하는 흐름 분석과 하드웨어 영향 요소의 분석, 그리고 분석된 요소들의 실행 시간을 계산하는 단계로 나뉜다. 하드웨어 영향 요소에 대

본 연구는 정보통신부 대학 IT연구센터 육성•지원사업의 연구결과로 수행되었습니다.

한 연구는 캐시와 파이프라인에 대해 많은 연구가 진행되었으며 Tree-Based Calculation, Path-Based Calculation 그리고 IPET-Based Calculation 기법을 이용하여 요소들의 실행 시간을 예측하였다. [2, 3, 4]

대부분의 연구는 원시 프로그램을 분석의 주요 대상으로 하고 있다. 프로그램의 컴파일 시간에 생성되는 중간 코드 또는 어셈블리 코드를 분석 대상으로 한다.

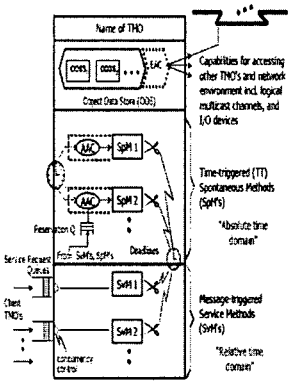
이러한 정적 분석 기법들은 몇 가지 문제점을 지니고 있다. 원시 프로그램 상에 반복문이 포함 되어 있는 경우에 설계자 또는 사용자가 직접적으로 반복 파라미터를 입력하여야 한다. 그리고 반복 파라미터의 예측이 힘들다. 따라서 분석에 추가적인 시간 및 비용이 필요하고 분석 결과의 정확도를 보장하지 못하는 문제점을 가지고 있다. 그리고 분산 실시간 시스템에 적용은 통신에 따르는 부하를 고려하지 않음으로 인해 정확하고 신뢰할 수 있는 실행 시간 분석을 보장하지 못한다.

반복 파라미터의 직접 입력 문제를 해결하기 위해 분석 과정에서 표현식을 이용하여 실행 시간을 계산하는 방법이 연구되었다. 그리고 분산 실시간 시스템에서 통신에 따른 부하를 고려하기 위해 통신 지연 시간 및 작업의 응답 시간 분석에 관한 연구도 진행되었다. [5, 6, 7]

하지만 통신에 따른 부하를 고려한 많은 연구들은 통신을 위한 준비 작업을 고려하지 않았으며 작업의 실행 시간 및 통신 시간에 대한 예측이 결정되어 있음을 가정하고 있다. 본 논문에서는 통신을 위한 준비 작업을 고려하고 반복 파라미터의 표현식 기법을 적용하여 신뢰성 있고 자동적인 예측을 가능케 한다.

3. 실시간성 분석 도구

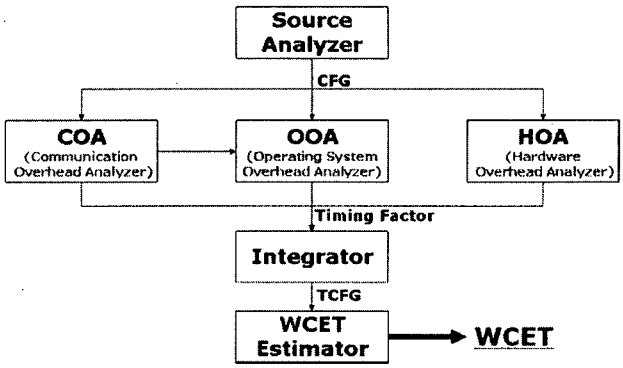
실시간성 분석 도구의 전체 구조를 살펴보기 이전에 신뢰성 있는 분산 실시간 환경 구축 도구이면서 이 논문의 분석 대상인 TMO(Time-triggered Message-triggered Object)살펴 본다. TMO는 기존 실시간 시스템 환경에 객체 지향 개념을 적용하고 원격 객체(Remote Object) 호출에 대한 시간적 처리 과정, 시스템 및 데이터의 동기화를 부여하여 통신에 의해 발생할 수 있는 응답 시간의 미보장성을 줄여 개발자가 고려해야 하는 통신에 대한 문제를 최소화한다. 응답 시간의 미보장성을 줄일 수 있는 특성을 TMO가 가지고 있음으로써 쉽고 정확한 통신 영향 요소의 예측이 가능하다.



[그림 1] TMO의 기본구조

[그림 1]은 TMO의 기본 구조를 나타낸다. SpM과 SvM의 이용을 통해 작업의 동기화를 부여하여 시간적 정확성을 높이고 SvM을 통해 객체간의 통신을 손쉽게 설계 및 구현이 가능하다. TMO를 이용한 분산 실시간 시스템의 설계는 객체 지향 개념의 적용을 통해 체계적인 설계가 가능하다. 그러나 분산 실시간 시스템의 설계 과정 중 선행 되어야 하는 스케줄 가능성 검사를 위한 많은 실행 시간 분석 도구들은 구조적 프로그램 분석 방법을 택하고 있기 때문에 객체 지향 프로그램 분석 방법의 개발이 필요하다.[8]

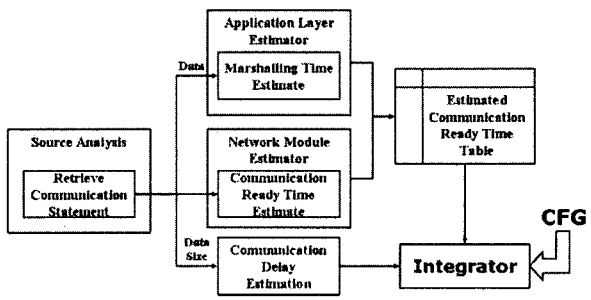
[그림 2]는 분산 실시간 시스템을 위한 실시간성 분석



[그림 2] 실시간성 분석 도구

도구의 전체적인 구조를 나타낸다. 소스 분석기는 컴파일러의 전단부에 해당하는 작업을 진행하여 CFG(Control Flow Graph)를 생성한다. CFG는 원시 프로그램의 동작 의미를 가지는 문장별 흐름의 정보를 표현한다. 소스 분석기의 추가적인 동작은 통신 부하 분석기 및 운영체제 부하 분석기에 사용될 통신, I/O 그리고 메모리 할당에 해당하는 문장 및 시스템 콜을 분석하여 저장한다. 파악된 문장별 흐름을 그래프 형태로 나타내어 영향 요소와 관련되어 저장된 정보와 함께 다음 단계 분석기의 입력으로 사용한다. 통신, 운영체제 그리고 하드웨어 영향 요소 분석기는 원시 프로그램에서 추출된 영향 요소들의 시간 영향 요소(Timing Factor)를 분석하여 TCFG (Timing Control Flow Graph)를 생성한다. 본 연구에서 실행 시간 계산의 단계는 관련 연구에서 언급한 Tree-Based Calculation을 기반으로 하여 실행 시간을 예측한다.

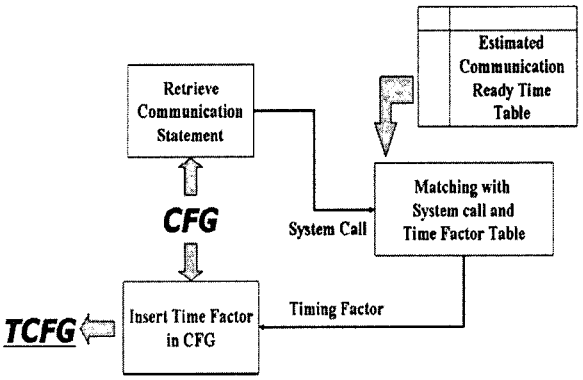
4. 통신 부하 요소 분석 및 통합



[그림 3] 통신 부하 요소 분석

분산 실시간 시스템에서 통신에 의해 발생하는 부하 요소의 분석은 [그림 3]과 같은 과정을 따른다.

어플리케이션 레벨에서 통신에 필요한 준비 작업인 어플리케이션과 커널에서 통신을 위한 준비시간의 분석을 선행하여 통신 부하 분석에 필요한 선행 예측 테이블을 작성한다. 소스 분석기에서 생성된 CFG와 선행 예측 테이블 및 통신 지연 시간을 통합하여 TCFG를 생성한다. 그리고 통신 지연 시간을 예측하기 위해 통신에 사용되는 데이터의 크기는 소스 분석기에서 추출하고 상수 값으로 예측하지 못한 실행시간 표현식의 반복 파라미터로 사용한다.



[그림 4] 통합 단계

통합 단계의 과정은 [그림 4]와 같다. 원격 함수 호출의 발생 여부를 확인하기 위해 원시 프로그램에서 함수 호출 부분에 대한 문장을 추출하고 추출된 함수의 이름이 선행 예측 테이블내의 네트워크 모듈 함수의 이름과 비교하여 해당 네트워크 모듈 함수의 최악 실행 시간을 CFG에 반영하여 CFG를 구성하는 노드의 필드에 추가함으로써 TCFG를 생성한다.

선행 예측 테이블 작성은 통신에 참여하는 리눅스 커널의 네트워크 모듈을 정적 실행 시간 분석 기법을 기초로 하여 예측을 통해 이루어진다. 리눅스에서 제공하는 GCC 컴파일러를 이용하여 해당 네트워크 모듈 함수의 어셈블리 코드를 생성한다. 생성된 어셈블리 코드에서 문장 흐름에 따른 최장의 인스트럭션 경로를 계산한다. 이때 만약 모듈 함수 내에 반복문이 존재하면 앞서 관련연구에서 살펴본 표현식 형태로 실행 시간을 나타낸다. [표 1]은 원격 함수 호출에 사용되는 대표적인 네트워크 모듈 함수의 실행 시간을 나타낸다. 원격 함수 호출의 시작을 위한 바인딩 및 소켓 생성 네트워크 모듈 함수의 최장의 인스트럭션 경로와 각각의 실행 시간을 표시하고 있다.

5. 결론 및 향후 연구

본 논문에서는 분산 실시간 시스템을 위한 실시간성 분석 도구의 설계에 있어 통신에 의해 발생하는 부하를 분석하기 위한 모델 설계를 논하였다. 통신 부하의 분석을 위해 필요한 과정 및 네트워크 모듈 함수의 정적인 예측 값을 선행 예측 테이블로 저장하는 과정을 설명하고 있다. 그리고 원시 프로그램의 실행시간 분석에 선행 예측 테이블의 사용함으로써 정확하고 신뢰성 있는 예측을 보장한다. 그리고 네트워크 모듈 함수 중 반복문의

[표 1] 네트워크 모듈 함수의 실행 시간

Function Name	Longest Path Line Number	WCET	Function Name	Longest Path Line Number	WCET
sys_accept	34	55	sys_recv	15	21
sys_bind	52	72	sys_setsockopt	51	72
sys_connect	51	69	sys_shutdown	41	55
sys_getpeername	51	74	sys_socket	34	55
sys_getsocketname	50	72	sys_socketpair	92	134
sys_getsocketopt	48	68	sys_sendmsg	171	223
sys_listen	45	67	sys_recvmsg	186	256
sys_recvfrom	69	87	sys_sendto	71	92

경우 기존 분석과정에서 필요한 반복 파라미터의 직접 입력을 없애기 위해 상수 값으로 표시되던 실행 시간을 식으로 표현하여 분석 과정에 발생하는 설계 및 분석자의 노력을 최소하였다.

실시간 시스템에서 실행 시간 분석에 고려되어야 할 영향 요소 중 하나인 운영체제 영향 요소의 분석을 향후 연구과제로 남긴다. 스케줄링과 관련된 모듈 분석 및 분하 예측을 통해 보다 정확하고 신뢰성 있는 분석 도구의 개발을 진행한다.

6. 참고문헌

[1] Scott A. Brandt, "The Case for Dynamic Real-Time Task Timing in Modern Real-Time Systems", IEEE IPDPS'04, June, 2004

[2] Ermedahl. Andreas, "A Modular Tool Architecture for Worst-Case Execution Time Analysis", Doctoral Theis, Uppsala University, 2003

[3] J. Engblom, A. Ermedahl, M. Sjoedin, J. Gubstafsson, and H. Hansson. "Worst-case execution-time analysis for embedded real-time systems", Journal of STTT, vol. 4, p. 437-455, no. 4. 2003

[4] G. Ottosson and M. Sjodin, "Worst-Case execution time analysis for modern hardware architectures". In Proceeding of ACM SINGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems. 1997

[5] G. Bernat and A. Burns. "An approach to symbolic worst-case execution time analysis". In Proc. 25th Workshop on Real-Time Programming, Palma, Spain, May 2000.

[6] B. Lisper. Fully automatic, parametric worst-case execution time analysis. In Proc. of 3rd Int. Workshop on WCET Analysis, pages 99-102, July 2003.

[7] K. Tindell, and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", Microprocessing & Microprogramming, Vol. 40, Issue 2-3, April 1994, pp.117-134.

[8] K. H. (Kane) Kim, "APIs for Real-Time Distributed Object Programming, Computer", v.33 n.6, p.72-80, June 2000.