TMO 모델 기반의 동적 분석 프레임워크 설계: 구성요소 및 측정지수

(Design of TMO Model based Dynamic Analysis Framework: Components and Metrics)

정 윤 석 † 김 태 완 †† 장 천 현 †††
(Yoon Seok Jeong) (Tae Wan Kim) (Chun Hyon Chang)

요 약 컴퓨터 시스템이 등장한 이후 시스템 성능을 측정하고 분석하기 위한 많은 연구가 시스템 모델링, 성능 측정, 감시, 그리고 성능 예측 등 여러 분야에서 진행되었다. 그럼에도 불구하고, 각 성능 관련 분야를 하나로 묶는 통합 프레임워크에 관한 연구는 거의 이루어지지 않았다. 특히 TMO(Time-Triggered Message-Triggered Object) 실시간 프로그래밍 모델의 경우, 간단한 감시 도구를 제외하고 성능 측정 도구나 분석 프레임워크가 없어, TMO 모델 기반 시스템 및 태스크를 분석하는데 어려움이 있다. 이에 따라, 본 논문에서는 TMO 모델 기반의 동적 분석 프레임워크인 TDAF(TMO based Dynamic Analysis Framework)를 제안한다. TDAF는 성능 측정 및 분석 단계를 전체적으로 다루며, 구성 요소인 부하 모델, 성능 모델, 그리고 보고 모델을 유기적으로 결합하여 보다 신뢰할 수 있는 정보를 개발자에게 제공한다. 이를 지원하기 위해 기존 부하 모델에 TMO 모델을 결합하여 확장한 부하 모델을 제안하고, TMO 객체 부하를 파악할 수 있는 부하 계산 알고리즘을 제안한다. 또한 TMO 객체 부하를 고려하여 성능 측정지수를 구현한 성능 알고리즘과, 부하 및 성능을 기초로 실시간 태스크의 주기 및 데드라인을 도출할 수 있는 보고 모델과 알고리즘을 제안한다. 마지막으로 부하 계산 알고리즘의 타당성을 입증하기 위한 실험을 수행하고 그 결과를 제시한다.

키워드 : TMO 모델 기반의 동적 분석 프레임워크, 부하 모델, TA-MVA, 성능 모델, 보고 모델

Abstract A lot of studies to measure and analyze the system performance have been done in areas such as system modeling, performance measurement, monitoring, and performance prediction since the advent of a computer system. Studies on a framework to unify the performance related areas have rarely been performed although many studies in the various areas have been done, however. In the case of TMO(Time-Triggered Message-Triggered Object), a real-time programming model, it hardly provides tools and frameworks on the performance except a simple run-time monitor. So it is difficult to analyze the performance of the real-time system and the process based on TMO. Thus, in this paper, we propose a framework for the dynamic analysis of the real-time system based on TMO, TDAF(TMO based Dynamic Analysis Framework). TDAF treats all the processes for the performance measurement and analysis, and provides developers with more reliable information *systematically combining a load model, a performance model, and a reporting model.* To support this framework, we propose a load model which is extended by applying TMO model to the conventional one, and we provide the load calculation algorithm to compute the load of TMO objects. Additionally, based on TMO model, we propose performance algorithms which implement the conceptual performance metrics, and we present the reporting model and algorithms which can derive the period and deadline for the real-time processes based on the load and performance value. In last, we perform some experiments to validate the reliability of the load calculation algorithm, and provide the experimental result.

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음
† 비 회 원 : 건국대학교 컴퓨터공학과
ysjeong@konkuk.ac.kr
†† 학생회원 : 건국대학교 컴퓨터공학과
twkim@konkuk.ac.kr
††† 종신회원 : 건국대학교 컴퓨터공학과 교수
chchang@konkuk.ac.kr
논문접수 : 2004년 12월 30일
심사완료 : 2005년 5월 20일

Key words : TMO Model based Dynamic Analysis Framework, Load Model, TA-MVA, Performance Model, Reporting Model

1. 서 론

컴퓨터 시스템이 등장한 이후 시스템 성능을 측정하고 분석하기 위한 많은 연구가 진행되었다[1-11]. 모델링 분야에서는 정확한 성능 분석의 기초가 되는 시스템 모델링 기법을 연구하였다. 시스템은 사용자 요청을 처리하기 위해 여러 자원을 이용하므로, 각 작업 별 자원 사용량 분포를 파악하면, 작업 별 부하 특성을 모델링할 수 있다[1]. 반대로, 메모리나 I/O 등 시스템 자원 관점에서 작업들이 해당 자원을 참조하는 패턴을 파악하여 시스템을 모델링할 수 있다[2,3]. 큐 네트워크의 경우, 시스템에서 발생하는 부하의 크기를 정량적으로 파악하기 위해서 큐 길이를 사용하였다[4].

성능 측정 분야에서는 성능 측정을 위한 기초 데이타를 정의하고, 측정지수를 만드는데 초점을 맞추었다. 성능 분석을 위한 기초 데이타는 기존 연구 결과를 정리하여 RUM(Resource Usage Model)이라는 데이타 모델로 제안되었다[5]. 더불어 RUM 기반의 부하 프로파일이나 데드라인 미스 프로파일과 같은 정교한 모델들이 제안되었고, 과도한 데드라인 미스 비율, 안정화 시간 등의 측정지수들도 제안되었다[6].

시스템 환경이 분산 및 실시간 환경으로 바뀌면서, 실시간 태스크의 성능과 관련된 실시간 모델링, 실시간 감시, 성능 예측과 관련된 연구들이 수행되었다[7-10]. 실시간 모델링 연구에서는 마코브 모델을 이용하여 실시간 태스크를 모델링하고자 시도하였고, DMR(Deadline-Missed-Reliability), DMBF(Deadline-Missed-Before-Failure)와 같은 측정지수들을 제안하였다[7,8]. 성능 예측 분야에서는 통계학적인 방법에 근거하여 동적 환경에서도 정확한 성능 분석이 가능하도록 성능 예측 모델을 제시하였다[11].

분석을 통해 기존 연구들은 다음의 한계를 가지고 있는 것으로 나타났다.

- 한계 1: 성능 측정 및 분석을 위한 여러 연구들이 수행되었지만, 통합적인 프레임워크에 기초하지 않았다.

시스템 성능과 관련해 이루어진 다수의 연구에도 불구하고, 통합된 프레임워크에 기반하여 수행된 연구나 통합 프레임워크에 관한 연구는 거의 이루어지지 않았다. 통합 프레임워크는 성능 분석이 수행되는 분석 주기 동안 구성요소들을 결합하여, 체계적으로 성능 분석을 수행하고, 일관성 있는 결과를 도출하기 위해 필요하다. 특히 본 논문에서 대상으로 하는 TMO(Time-Triggered Message-Triggered Object) 실시간 모델의 경우, 간단한 감시 도구를 제외하고 성능 측정 및 분석 도구가 거의 제공되지 않아, TMO 모델 기반 실시간 시스템의 성능을 분석하는데 어려움이 있다.

- 한계 2: 기존 성능 분석 연구들은 시스템과 태스크를 동시에 고려하지 않았다.

기존 실시간 시스템 성능 분석을 위한 측정지수들은 데드라인과 같은 실시간 특성만을 고려하였다. 그러나 실시간 태스크는 시스템 작업 부하에 의해 수행 능력이 달라지므로, 실시간 시스템 부하 및 실시간 태스크의 시간 특성 모두를 고려한 모델과 측정지수들이 제시되어야 한다.

본 논문은 이러한 한계점을 극복하기 위해 실시간 시스템 성능을 분석하기 위한 통합 체계인 TDAF(TMO Model based Dynamic Analysis Framework)를 제안한다. TDAF는 실시간 프로그래밍 모델인 TMO 모델에 기반한 실시간 시스템을 대상으로, 실시간 태스크 모델링, 성능 측정, 성능 보고에 이르는 성능 측정 단계를 체계적으로 지원한다. 본 논문에서는 TDAF 구조 및 TDAF를 구성하는 각 요소를 위한 알고리즘과 측정지수를 제안한다.

2. 배경 연구

2.1 TMO모델의 구조

본 논문에서는 실시간 프로그래밍 모델인 TMO 모델에 기반한 실시간 시스템을 성능 측정 대상으로 인식한다. 보다 정확하게 말하면, TMO모델을 지원하는 LTMOS(Linux TMO System) 미들웨어 상에서 동작하는, TMO 실시간 태스크를 대상으로 한다[12-14].

TMO 모델은 기존 실시간 모델을 확장한 형태이며, 실시간 시스템을 구현할 때, 신뢰성 있는 디자인이 가능하도록 명확한 문법 구조와 실행 시맨틱을 지원한다[10, 15]. 그림 1은 TMO 모델의 구조를 보여준다.

TMO 모델은 다음의 4가지 요소로 구성된다.

- EAC(Environment Access Capability): 외부 객체간 통신 및 입출력 기능을 담당하며, 논리적으로 객체간의 연결을 위한 채널 형태로 다룸
- ODS(Object Data Store): TMO 객체의 상태 및 객체간 공유 데이타를 저장하는 공간. SpM이나 SvM에 의해 접근되며, ODSS(ODS Segment) 단위로 다루어짐. ODSS에 저장된 데이타는 EAC의 채널을 통해 전달됨

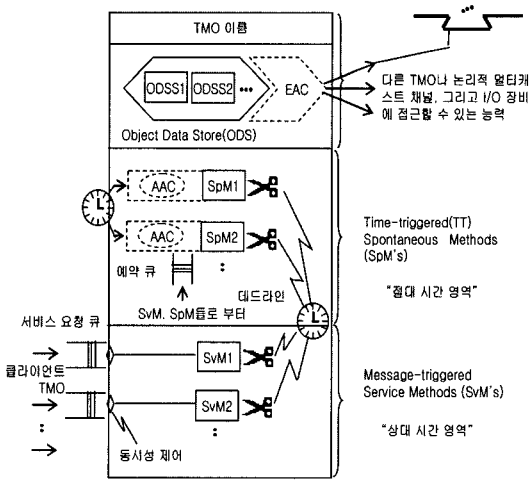

그림 1 TMO 모델의 구조

- SpM(Spontaneous Method): 시간에 의해 호출되는 메소드로서 일정 주기마다 동작. SpM의 구동시간을 정의한 AAC(Autonomous Activation Condition)의 시간 중 최적 시간 조건에 구동됨
- SvM(Service Method): 메시지에 의해 호출되는 메소드로서 외부 서비스 요청에 의해 동작

본 논문에서는 TMO 실시간 태스크인 SpM과 SvM의 수행을 분석하고, 분석 결과를 개발자에게 제공하여, 견고한 실시간 시스템 개발이 가능하도록 지원한다. 모든 TDAF 구성요소는 TMO 모델을 고려하며, 실험 및 구현도 TMO모델을 기반으로 수행한다.

2.2 TMO의 시간 개념

그림 2는 시간 개념에 근거한 TMO 실시간 태스크의 동작 구조를 보여준다[14]. TMO 실시간 태스크는 앞서 언급한 것과 같이 SpM과 SvM이라는 실시간 태스크로 구성된다. SpM은 시간 주기에 따라 동작하는 태스크이다. 예를 들어, 그림 2에는 각각 250ms, 200ms, 그리고 500ms 의 주기를 갖는 3개의 SpM이 존재한다. 이들 SpM은 250ms, 200ms, 그리고 500ms마다 주기적으로 호출되어 수행된다. SpM은 데드라인을 갖는데, 이는 SpM이 데드라인 내에 수행되어야 함을 의미한다. 반면, SvM은 SpM에 의해 호출되며, SvM의 수행 시간은 SpM의 수행 시간에 포함된다. 편의 상, 본 논문에서는 시간을 일정 간격 t로 나눈 시간 프레임 단위로 TMO 객체의 수행과 분석을 다룬다.

2.3 동적 분석과 컨텍스트

동적 개념은 [8]에서와 같이 '조건이나 상태가 변동되는' 것을 나타내거나, [16]에서처럼 '상황 변화에 자동적으로 반응하는' 것을 의미하기도 한다. 경우에 따라 '특정 프로그램이 동작하고 있는 상태'를 의미한다. 본 논문에서는 세 번째의 동적 개념을 사용한다. 즉, 동적 분석은 시스템 혹은 실시간 태스크가 실행 중인 동적 컨텍스트에서 수행되는 분석을 의미한다. 참고로, 정적 분석은 시스템 혹은 태스크가 실행되지 않는 정적 컨텍스트에서 수행되는 분석을 말한다. 그러나 두 가지 분석은 서로 엄격하게 분리되지 않는다. 본 논문에서도 동적 분석은 정적 분석의 결과를 이용한다.

3. 관련 연구

3.1 자원 사용량 모델

[5]는 시스템에서 동작하는 어플리케이션의 자원 요구량이 변화할 때, 이에 반응할 수 있는 적응형 자원 할당

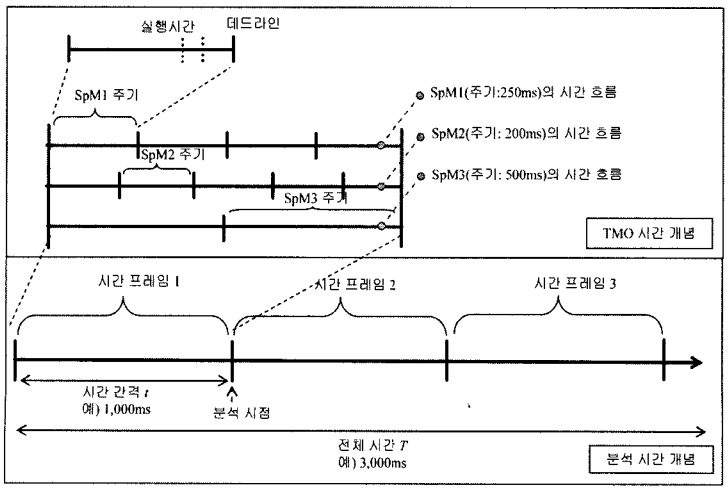

그림 2 TMO의 시간 개념

(Adaptive Resource Allocation) 메커니즘을 위한 모델과 측정지수를 제안하였다. 모델의 일부로서 제안된 어플리케이션의 자원 사용량 모델인 RUM은 어플리케이션이 수행될 때 사용할 것으로 예상되는 자원 사용량 값과 실제로 사용한 자원 사용량을 나타기 위한 모델이다. 전자를 정적 RUM, 후자를 동적 RUM이라 한다.

정적 RUM은 알고리즘을 통한 분석이나 코드 프로파일링 등의 전통적인 방법으로 추정하는 값으로 구성된다. 다음은 정적 RUM의 측정지수들을 나타낸다. 부하를 발생시키는 기본 단위는 어플리케이션 내의 컴포넌트이며, 각 컴포넌트간 혹은 컴포넌트 내부에서 데이타 전달이나 호출 등을 통해 커뮤니케이션이 발생하는 것을 가정하였다.

- Parallelism Level
- Execution Time
- Intra-Communication Protocol
- Maximum Outgoing Intra-Communication Message Size
- Total Number of Outgoing Intra-Communication Messages
- Total Amount of Outgoing Intra-Communication Data
- Inter-Communication Protocol
- Total Number of Outgoing Inter-Communication Messages
- Total amount of Outgoing Inter-Communication Data
- Process Speed Factor

동적 RUM은 실제 실행시간에 측정된 값을 포함하는 측정지수들로 정적 RUM과 달리 축약적으로 표현되었으나, 동일한 구조를 취하고 있다. 다음은 동적 RUM의 측정 요소이다.

- Execution Factor
- Total Amount of Intra-Component Data Factor
- Maximum Intra-Component Message Size Factor
- Total Amount of Inter-Component Data Factor

[5]는 RUM의 추가 확장이 가능하며, 실행시간, 커뮤니케이션, 어플리케이션 내부 커뮤니케이션(Intra-Communication)은 측정지수들은 각각 CPU, I/O, 메모리 자원과 관련이 있음을 언급하였다. 이를 근거로 상기 측정요소들을 시스템 자원 요소로 변환하여 사용할 수 있다. 그러나, RUM은 전체 어플리케이션 관점에서 부하를 고려하기 때문에, 본 논문에서 고려하는 실시간 태스크들의 부하를 분리해 낼 수 없는 문제점을 내포하고 있다.

3.2 부하 모델과 성능 측정지수

[6]은 부하 또는 자원 변화에 반응하는 적응형 실시간 시스템을 위한 제어 프레임워크를 제안하였다. 프레임워크의 일부로서 시스템 부하 모델링을 위한 부하 프로파일(Load Profile)과 특정 부하 상태에서의 시스템 성능을 나타내기 위한 데드라인 미스 비율 프로파일(Miss-ratio Profile)을 제안하였다.

부하 프로파일 $L(t)$는 시간 함수로 표현되는 시스템 부하를 설명하기 위한 모델로서 다음의 두 가지 모델로 구분된다.

- 계단형 증가 모델 $SL(t)$: 평균적인 부하 L_{nom}에서 L_{max}로 증가한 후, 안정적 상태를 유지하는 모델. $SL(L_{nom}, L_{max})$으로 표현.
- 경사형 증가 모델 $RL(t)$: 일정 시간 내에 점진적으로 부하가 증가하며, 보다 완곡하고 현실적인 부하 변화 모델임. $RL(L_{nom}, L_{max}\ T)$로 표현.

부하 프로파일을 통해 표현된 특정 부하 하에서 시스템 성능을 파악하기 위해 데드라인 미스 비율에 근거한 세부 성능 측정지수를 다음과 같이 정의하였다.

- 안정성: 계단형 또는 경사형 부하에서 데드라인 미스 비율이 '0'에 수렴하는 상태를 안정적이라고 할 때, 현재 어느 정도 안정되었는지를 나타냄
- 상태 변화 응답성: 시스템 상태가 변화하였을 때 적응이 얼마나 빠르고 효율적으로 이루어지는지를 설명. 세부 측정지수로 과도한 데드라인 미스 비율(Overshoot), 시스템의 데드라인 미스 비율이 안정 상태로 들어가는데 걸리는 시간을 의미하는 안정화 시간을 포함
- 안정 상태의 데드라인 미스 비율: 안정 상태에서의 평균 데드라인 미스 비율. 시스템이 얼마나 잘 회복했는지를 나타냄
- 민감성: 특정 변수의 변화에 대해 안정 상태 데드라인 미스 비율이 어떻게 변화하는지를 나타냄

[6]은 부하 표현을 위한 부하 프로파일과 성능 측정지수들의 개념을 제안하였으나, 구체적인 추출알고리즘을 제시하지 못하였다. 따라서 부하 및 각 성능 측정지수별로 실제 성능 값을 추출하기 위한 알고리즘이 제안되어야 하며, 특히, 본 논문에서 목표로 하는 TMO 모델을 반영하여 확장되어야 한다.

4. TDAF(TMO Model based Dynamic Analysis Framework)

4.1 성능 측정 및 분석을 위한 요소

[4]는 성능 측정 및 분석을 위한 일반화된 단계를 그림 3과 같이 제시하였다.

각 단계마다 성능 측정 및 분석을 위해 몇 가지 요소들이 필요하다. 이를 정리하면 표 1과 같다.

단계 (1)의 성능 측정 대상은 시스템 자체이거나 시

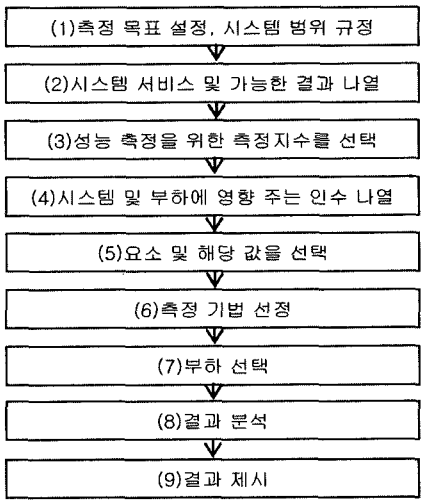

그림 3 성능 측정 및 분석 단계

표 1 단계 별 성능 측정 및 분석 요소

단계	필요한 요소
1	성능 측정 대상
2	N/A
3	성능 측정지수
4	부하요소(시스템, 실시간 태스크)
5	부하요소(시스템, 실시간 태스크)
6	측정 기법, 성능 측정지수
7	부하 모델
8	분석 기법, 성능 모델, 분석 관련 측정지수
9	보고 모델

스템 자원, 혹은 해당 시스템에서 동작하는 태스크다[4]. 단계 (3)의 성능 측정지수는 에러율이나 실행시간과 같은 성능 측정기준을 의미하며, 일반적으로 속도, 정확성, 가용성과 관련이 있다. 단계 (4)에서 필요로 하는 요소에는 시스템 인수와 서비스 요청 인수가 있다. 시스템 인수는 하드웨어나 소프트웨어와 관련된 인수이고, 서비스 요청 인수는 시스템에 부하를 발생시키는 서비스 요청과 관련된 인수이다. 실시간 시스템의 경우, 시스템 인수는 시스템 부하요소로, 서비스 요청 인수는 실시간 태스크 부하요소로 바꾸어 생각할 수 있다. 부하요소는 하위 단계에서 부하 계산을 위해 필요한 데이타들로 구성되며, 본 논문에서는 이를 부하 프로파일로 정의한다. 단계 (6)의 성능 측정지수는 본 논문에서는 성능 프로파일로 정의한다. 단계 (7)의 부하 모델은 시스템 부하를 모델링하기 위해 사용되며[5,6], 단계 (8)의 성능 모델은 분석한 성능을 설명하기 위한 명세이다[6]. 마지막으로, 단계 (9)의 보고 모델은 분석 결과를 개발자가 이해할 수 있는 형태로 보여주기 위한 모델이며, 보고 프로파일과 포맷 정보를 결합하여 개발자에게 제공한다. 상기 내용을 기초로, 성능 측정 및 분석과 관련된 요소를 정리하면 그림 4와 같다.

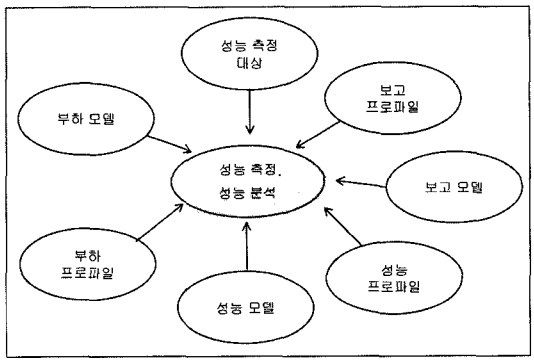

그림 4 성능 측정 및 분석을 위한 요소

4.2 TDAF의 구조

TDAF는 TMO 모델 기반의 실시간 시스템의 성능을 분석하기 위해 성능 측정 및 분석 요소들의 관계를 규정하고, 각 요소들을 연동하기 위해 제안하는 프레임워크이다. TDAF는 분산 환경에서 TMO 어플리케이션이 동작하는 실시간 시스템을 대상으로 한다. 또한 각 TMO 객체들은 데드라인 미스 발생이 허용되는 소프트 실시간 시스템에서 동작하는 것으로 가정한다.

그림 5는 4.1절에서 파악한 각 성능 측정 및 분석 요소들을 결합하여, 요소간의 관계, 그리고 동적 분석 실행 단계를 나타낸 프레임워크를 보여준다.

TDAF는 다음의 5가지 요소를 포함하도록 설계하였다.

- 부하 프로파일
- 부하 모델
- 성능 프로파일
- 성능 모델
- 보고 모델

[6]에서 제안한 부하 프로파일은 부하를 측정하기 위해 필요한 데이타, 데이타를 계산하기 위한 알고리즘, 계산 결과인 부하를 표현하는 부하 모델을 구체적으로 구별하지 않고 사용하였다. 본 논문에서는 명시적으로 각각을 구별하여 정의한다. 부하 프로파일은 시스템과 실시간 태스크의 부하를 계산하기 위해 필요한 데이타 집합이다. 부하 프로파일은 추정 단계에서 사용하는 추정 부하 프로파일(Estimated Load Profile)과 실행 단계에서 사용하는 수행 부하 프로파일(Running Load Profile)로 구분된다. 또다시 각 부하 프로파일을 시스템 부하와 실시간 태스크 부하로 구분하여, 각 부하를 상세히 파악할 수 있도록 설계하였다. 반면, 부하 모델은 시

스템 및 실시간 태스크의 부하를 계산하여, 시스템과 실시간 태스크가 발생시키는 부하를 정형화하기 위한 모델이다. 성능 프로파일은 시스템 및 실시간 태스크 성능을 파악하기 위한 측정지수들의 집합이다. 성능 모델은 분석한 성능을 설명하기 위한 명세를 의미한다. 마지막으로 보고 모델은 개발자가 필요로 하는 데이타의 집합이며, TDAF 구성 요소들의 결과를 통합하여 제시한다.

TDAF 요소들은 분석 단계에 따라 각자의 역할을 수행한다. 분석 단계는 그림 5와 같이 3단계로 구분할 수 있다.

- 추정 단계(Estimating Phase)
- 수행 단계(Running Phase)
- 보고 단계(Reporting Phase)

추정 단계는 정적 컨텍스트에, 수행 및 보고 단계는 동적 컨텍스트에 속한다. 수행 단계와 보고 단계는 기능 구분을 위해 독립적인 단계로 분리하였다.

추정 단계에서는 시스템 및 실시간 태스크 데이타를 수집하여 추정 부하 프로파일을 구성하고, 부하 계산 알고리즘을 이용하여 시스템 및 실시간 태스크 부하를 추정한다. 수행 단계는 시스템과 실시간 태스크의 실제 부하 측정과 관련된 활동들을 수행하는 단계이다. 시스템 모니터와 RP(Run-time Process) 모니터는 데이타를 수집하여 실행 부하 프로파일을 구성하고, 부하 계산 알고리즘은 시스템 및 실시간 태스크의 실제 부하를 측정하는데 이용된다[9]. 추정 부하와 실제 부하를 이용해 부하 모델을 구성하고, 성능 알고리즘을 이용해 성능 프로파일을 구성하는 것도 이 단계에서 이루어진다. 마지막으로 보고 단계에서는 보고 알고리즘을 이용해 수행 단계에서 생성된 성능 프로파일을 가공하고, 포맷 정보와 결합하여 보고 모델을 만들어 개발자에게 제공한다.

다음은 TDAF의 세부 요소를 모델링하기 앞서, 관련 기호들을 정리한 내용이다.

Ba_{ij} : 허용가능 범위, SpM_{ij}의 데드라인
Bl_i : TMO 객체 i의 처리율 하한 범위
Bu_i : TMO 객체 i의 처리율 상한 범위
Cp_{ij} : i번째 TMO 객체의 j번째 SpM의 데드라인
D_ip : TMO 객체 i의 SpM의 평균 요구 시간
D_ip_{avg} : T 시간 동안 일어난 TMO 객체 i의 SpM의 평균 요청 수
D_ip_{max} : T 동안 수행된 TMO 객체 i의 SpM 중 최다 요청 수를 가진 SpM의 요청 수
E_i : TMO 객체 i의 안정화 시간
Ep_{ijk} : 시간 프레임 t에서 SpM_{ij}의 k 번째 수행 시 종료 시간
$\overline{FPp_{ij}}$: SpM_{ij} 주기의 평균 여유분
$\overline{FCp_{ij}}$: SpM_{ij} 데드라인의 평균 여유분
Ip_{ijk}, Ep_{ijk} : 시간 프레임 t에서 k번째 SpM_{ij}의 시작시간과 종료시간
Iv_{ijk}, Ev_{ijk} : 시간 프레임 t에서 k번째 SvM_{ij}의 시작시간과 종료시간
Lp_{ij} : SpM_{ij}의 수행에 의해 발생하는 부하
Lp_{ijk} : 시간 프레임 k에서 SpM_{ij}에 의해 발생된 부하
Lv_{ij} : SvM_{ij} 수행에 의해 발생하는 부하
M_i : TMO 객체 i 내의 SpM 수
Ma_i : TMO 객체 i의 과도 비율에 대한 접근 정도
N : 시간 프레임 t 동안 존재하는 TMO 객체의 수
O_{ij} : SvM_{ij}가 SpM_{ij}에서 차지하는 로드 비율
Pl_{ij} : 성능여유분, Ba_{ij} - SpM_{ij}의 실행시간

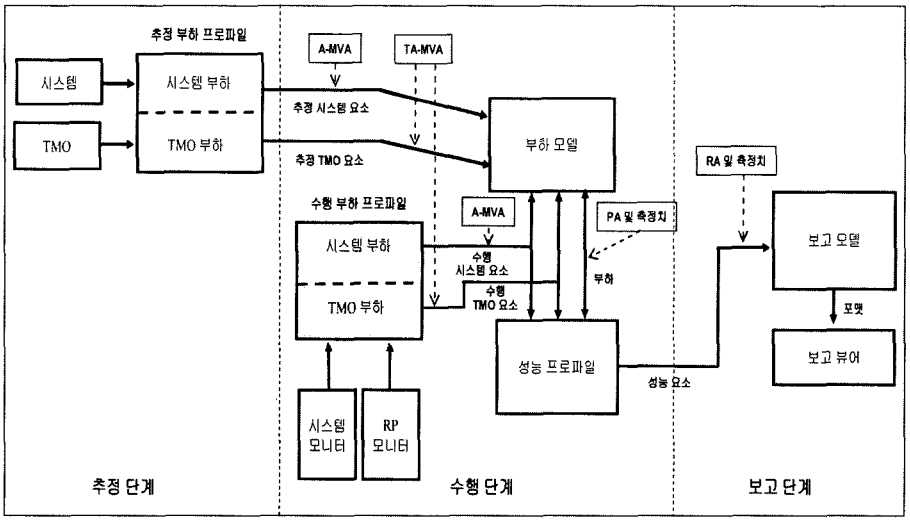

그림 5 TDAF의 구조

Pp_{ij} : SpM_{ij}의 주기
R : TMO 객체의 응답시간
R_{ij} : 시스템에 변화(TMO 객체 상의 변화)가 발생했을 경우, SpM_{ij} 응답시간
Rp_{ij} : SpM_{ij}의 응답시간, 추정단계에서는 응답시간=데드라인임
RPp_{ij} : SpM_{ij} 추천 주기
RCp_{ij} : SpM_{ij} 추천 데드라인
S : 안정성, T 동안 m_k가 SMB 이하일 비율
SMB : 데드라인 미스 비율 허용 기준(사용자 정의)
Sp_{ij} : SpM_{ij}의 서비스 처리 시간
Sp_{ijk} : 시간 프레임 t에서 k번째 수행된 SpM_{ij}의 서비스 처리 시간
T : 전체 시간
Vp_{ij} : 시간 프레임 내에 일어난 SpM_{ij} 의 요청 수
Vp_i : 시간 프레임 t 동안 발생한 SpM_i의 요청 수
X : TMO 객체의 처리율
X_i : TMO 객체 i의 처리율
Z : TMO 실시간 태스크의 완충시간 SpM_{ij} 주기 - SpM_{ij} 데드라인의 총합
e : 부하에서 허용 가능한 에러율
f : 전체 시간 내의 시간 간격 개수
m_k : 시간 프레임 k 동안 발생하는 실시간 태스크 데드라인 미스 비율
t : 시간 간격
te_i : TMO 객체 i가 안정 상태가 되는 시점
ts_i : TMO 객체 i가 불안정 상태가 되는 시점

4.3 부하 모델링

4.3.1 부하 프로파일(Load Profile)

부하란 '일정 기간 동안 시스템에 입력되는 모든 사용자의 요구'를 의미한다[1]. 사용자 요구는 어플리케이션 형태로 나타나거나, 실시간 태스크 형태로 나타난다. 각 어플리케이션의 수행을 위해서는 일정량의 자원이 요구된다.

[5]는 어플리케이션의 자원 요구량을 파악하기 위해, RUM을 제안하였다. 그러나 실시간 태스크 분석에 RUM을 직접 적용하기에는 어려움이 있다. 이는 RUM을 통해 어플리케이션이 요구하는 자원량을 파악할 수 있지만, 실시간 태스크 수행에 필요한 자원량을 구별할 수 없기 때문이다. 물론, 코드 프로파일링을 통해 실시간 태스크의 자원 사용량을 파악할 수 있다. 그러나 이러한 방법은 본 논문에서 목표하는 어플리케이션 계층에서의 분석과는 거리가 있다.

본 논문에서는 RUM의 구조를 취하되, 실시간 태스크 부하를 구별할 수 있도록, TMO 부하 관련 요소도 추가하였다. 또한 시스템 관련 측정지수들은 각 자원 별 요소로 분할하여 사용하는 방안을 도입하였다. 표 2는 RUM을 확장한 부하 프로파일을 나타낸다. RUM은 어플리케이션이 요구하는 자원 사용량, 즉, 부하 계산을 목적으로 사용되는 데이타 모델이므로 본 논문에는 이를 부하 프로파일이라 정의한다.

표 2 부하 프로파일

	추정/수행 단계
시스템 부하	· CPU 요소 · I/O 요소 · 메모리 요소
TMO 부하	· SpM 요소 · SvM 요소

표 2와 같이, 각 데이타들은 시스템 부하 관련 요소와 TMO 부하 관련 요소로 구분될 수 있다. 전자는 시스템과 관련된 데이타들의 집합이며, CPU 요소, I/O 요소, 그리고 메모리 요소로 나누어진다. 사용 단계에서 각 요소들을 보다 구체적인 데이타로 바꾸어 사용해야 한다. 예를 들어, CPU 요소는 사용자 모드에서의 CPU 시간, 커널 모드에서의 CPU 시간, 그리고 대기 큐의 실행 가능한 태스크 수로 구성할 수 있다. 또한 각 요소에는 데이타를 추가하여 사용할 수 있다.

TMO 부하 관련 요소는 전체 시스템 부하에서 실시간 태스크의 부하를 분리하기 위해 만들어졌다. 이는 실시간 태스크 부하를 구별하지 못했던 RUM의 문제를 해결하기 위해 추가한 것이다. TMO 부하 관련 요소는 SpM 요소와 SvM 요소로 구성된다. TMO 요소 역시 구체적인 데이타로 바꾸어 사용해야 한다. 예를 들어, 주기, 데드라인, 그리고 실행시간으로 SpM 요소를 구성할 수 있다.

4.3.2 부하 계산 알고리즘: TA-MVA

본 논문에서는 시스템 부하와 실시간 태스크 부하를 계산하기 위해 두 개의 알고리즘을 이용한다. 시스템 부하 계산 알고리즘은 기존의 시스템 모델링 알고리즘 중 MVA를 개선한 A-MVA(Approximate Mean Value Analysis)를 이용한다[17,18]. A-MVA는 시스템 상의 서비스 요청이 증가하면, 각 자원의 큐 길이도 비례적으로 증가한다는 Schweitzer 가정에 기초한다[17]. Q_i가 i 번째 장치에 도달하는 평균 서비스 요청 수, N이 사용자 수라 할 때,

$$\frac{Q_i(N)}{N} = a_i(\text{상수}) \quad \forall N$$

이며, 이는 곧

$$\frac{Q_i(N-1)}{N-1} = \frac{Q_i(N)}{N}$$

혹은

$$Q_i(N-1) = \frac{N-1}{N} Q_i(N)$$

이다. R_i가 i 번째 장치의 응답시간, S_i가 i 번째 장치의

서비스 시간, X가 시스템 처리율, V_i가 i번째 장치에 도달한 서비스 횟수, Z가 서비스 요청과 요청 사이의 충돌을 막기 위한 여유시간을 의미하는 완충시간(think time)이라 할 때, A-MVA의 방정식은 다음과 같다.

$$R_i(N) = S_i(1 + \frac{N-1}{N} Q_i(N))$$

$$X(N) = \frac{N}{Z + \sum V_i R_i(N)}$$

$$Q_i(N) = X(N) V_i R_i(N)$$

실시간 태스크 부하를 계산하기 위해서는, 본 논문에서 설계한 TA-MVA(TMO Model based Approximate MVA)를 사용한다. 기본 골격은 A-MVA를 따르지만, TMO의 실시간 태스크 부하를 계산할 수 있도록 실시간 정보들을 고려하여 설계하였다. A-MVA에서는 시스템 부하를 나타내기 위해, 큐 길이라는 개념을 이용하였다. 큐 길이는 곧 부하이며, 부하의 증감은 시스템 상에서 수행되는 어플리케이션 실행시간과 처리율의 증감을 의미한다. 실시간 태스크 관점에서 보면, 실시간 태스크 부하가 실시간 태스크 응답시간 및 처리율의 변화를 가져온다고 생각할 수 있다. 반대로 생각하면, 응답시간 및 처리율을 통해 실시간 태스크 부하를 파악할 수 있다. TA-MVA는 이러한 가정을 근거로 한다.

TMO 시간 개념을 기초로, i 번째 TMO 객체의 j번째 SpM의 서비스 시간 Sp_{ij}는 다음을 통해 구할 수 있다. 시간 프레임 t 에서 k번째 수행된 SpM_{ij}의 서비스 처리 시간이 Sp_{ijk}이고, 서비스 요청 횟수가 Vp_{ij}라 할 때, 시간 프레임 t에서 SpM_{ij}의 평균 서비스 처리 시간은

$$Sp_{ij} = \frac{1}{Vp_{ij}} \sum_{k=1}^{Vp_{ij}} Sp_{ijk}$$

이다. SpM_{ij}의 수행에 의해 발생하는 부하 Lp_{ij}를 고려하면, N이 TMO 객체 수라고 할 때, SpM_{ij}의 응답시간은

$$Rp_{ij} = Sp_{ij}\left(1 + \frac{N-1}{N} Lp_{ij}\right)$$

이다. 그리고 TMO 전체 응답시간은 전체 SpM_{ij}의 응답시간과 서비스 요청 회수를 이용해 나타낼 수 있다.

$$R = \sum_{i=1}^{N} \sum_{j=1}^{M_i} Rp_{ij} Vp_{ij}$$

TMO의 처리율 X와, SpM_{ij}의 수행으로 인해 발생하는 부하는 다음의 방정식을 통해 구할 수 있다. Z가 실시간 태스크의 수행과 다음 수행간의 충돌을 막기 위한 여유시간을 나타내는 완충시간, N이 시간 프레임 t 동안 존재하는 TMO 객체의 수를 의미할 때,

$$X = \frac{N}{Z + R}$$

$$Lp_{ij} = X Vp_{ij} Rp_{ij}$$

이다. 참고로 Cp_{ij}, Pp_{ij}가 SpM_{ij}의 데드라인과 주기를 의미할 때, 완충시간은

$$Z = \sum_{i=1}^{N} \sum_{j=1}^{M_i} (Pp_{ij} - Cp_{ij})$$

이다. 이제 SvM_{ij}의 부하 길이를 파악해 보자. SvM의 수행은 SpM에 포함되기 때문에, SvM_{ij}의 부하 길이는 SpM_{ij}의 부하 길이를 통해 파악할 수 있다. SvM_{ij}가 SpM_{ij}의 수행 시간 중 어느 정도의 비율을 차지하는지 다음의 공식을 통해 파악할 수 있다. Ip_{ijk}, Ep_{ijk}가 시간 프레임 t에서 k번째 SpM_{ij}의 시작시간과 종료시간을 의미하고, Iv_{ijk}, Ev_{ijk}가 시간 프레임 t에서 k번째 SvM_{ij}의 시작시간과 종료시간일 때, SvM_{ij}의 로드 비율은

$$O_{ij} = \frac{1}{Vp_{ij}} \sum_{k=1}^{Vp_{ij}} \frac{Ev_{ijk} - Iv_{ijk}}{Ep_{ijk} - Ip_{ijk}}$$

이다. 따라서, SvM_{ij}의 로드 비율과 SpM_{ij}의 부하 길이를 곱하여 SvM_{ij}의 부하 길이를 구할 수 있다.

$$Lv_{ij} = Lp_{ij} O_{ij}$$

상기의 공식을 결합하여 표 3의 TA-MVA 알고리즘을 제안한다.

4.3.3 부하 모델

부하 모델은 일정기간의 부하 상태 변화를 정형화하기 위한 모델이다. 본 논문에서는 [6]에서 제시한 두 가지 부하 모델 중 현실적인 경사형 증가 모델을 차용하였다. 다만, 실시간 태스크 부하를 고려할 수 있도록 부하 모델을 다음과 같이 확장하였다.

$$L(S_{nom}, S_{max}, P_{nom}, P_{max}, T)$$

부하 모델은 5개의 인수를 취하는데, 각각이 의미하는 바는 다음과 같다.

- S_{nom}: 평균 시스템 부하 혹은 시스템 부하 추정치
- S_{max}: T시간 동안에 변화한 시스템 부하
- P_{nom}: 실시간 태스크 수행에 의해 발생하는 평균적인 시스템 부하 혹은 시스템 부하 추정치
- P_{max}: T 시간 동안에 변화한 실시간 태스크 부하
- T: 주어진 시간

S_{nom}와 S_{max}는 태스크 부하를 포함한 시스템 부하를 나타내며, A-MVA를 통해 도출한다. 반면, P_{nom}와 P_{max}는 실시간 태스크 부하를 나타내기 위하여 추가하였다. 이는 TMO 객체가 TMO 미들웨어에 의해 스케줄링되므로, TMO 객체의 시간 조건, TMO 객체의 수 등에 의해 수행 여부 및 부하가 결정되기 때문이다. 따라서 실시간 태스크 부하 정보를 구별하여, 정확한 성능

표 3 TA-MVA 알고리즘

Initialization:
X=0
FOR i=1 TO N DO
BEGIN
  FOR j=1 TO M_i DO
  BEGIN
    $Lp_{ij}=N/M_i$
    $Lv_{ij}=0$
    $Sp_{ij}=\frac{1}{Vp_{ij}}\sum_{k=1}^{Vp_{ij}}Sp_{ijk}$
  END
END
i=0

Iteration:
WHILE max_{ij} {$|Lp_{ij}-XRp_{ij}Vp_{ij}|$} > e DO
BEGIN
  FOR i = 1 TO N DO
  BEGIN
    FOR j=1 TO M_i DO
    BEGIN
      $Rp_{ij}=Sp_{ij}\left(1+\frac{N-1}{N}Lp_{ij}\right)$
      $O_{ij}=\frac{1}{Vp_{ij}}\sum_{k=1}^{Vp_{ij}}\frac{Ev_{ijk}-Iv_{ijk}}{Ep_{ijk}-Ip_{ijk}}$
    END
  END
  $R=\sum_{i=1}^{N}\sum_{j=1}^{M_i}Rp_{ij}Vp_{ij}$
  $Z=\sum_{i=1}^{N}\sum_{j=1}^{M_i}(Pp_{ij}-Cp_{ij})$
  $X=\frac{N}{Z+R}$
  FOR i=1 TO N DO
  BEGIN
    FOR j=1 TO M_i DO
    BEGIN
      $Lp_{ij}=XVp_{ij}Rp_{ij}$
      $Lv_{ij}=Lp_{ij}O_{ij}$
    END
  END
END

측정 및 보고가 가능해진다. P_{nom}와 P_{max}는 앞서 제안한 TA-MVA를 통해 도출한다.

구현된 부하 모델은 [6]에서 제안한 바와 같이 작업 부하를 추상화한 형태이다. 즉, 부하 모델의 실현 예는 시스템 용량, 수행 중인 어플리케이션 수와 종류, 수행 특성 등에 따라 달라지게 된다. 따라서 부하 모델은 특정 부하에 대응하는 시스템 및 실시간 태스크 성능을 상대적으로 파악할 수 있는 근거와, 정확한 성능 측정을 위한 기초 데이타를 성능 모델에 제공한다.

4.4 성능 모델링

4.4.1 성능 프로파일

성능 프로파일이란 주어진 시스템 부하 조건에서 시스템의 성능을 파악하기 위한 측정지수 집합이다. 성능 모델을 구성하는 주요 측정지수에는 그림 6과 같이 안정성, 과도한 데드라인 미스 비율, 안정화 시간, 성능여유분이 있다[5,6]. 이 측정지수들이 모여 하나의 성능 프로파일을 구성한다.

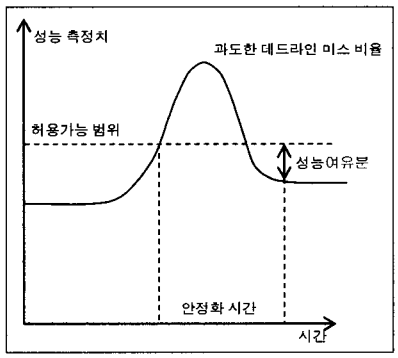

그림 6 성능 모델

- 안정성(Stability): 특정 시점에서 실시간 태스크가 정상적으로 서비스를 할 수 있는 확률을 나타냄
- 과도한 데드라인 미스 비율(Overshoot): 가장 큰 데드라인 미스 비율을 나타냄. 과도한 데드라인 미스 비율(이하, 과도 비율)은 시스템 실패의 원인이 되기 때문에 이를 파악해야 하나, 과도 비율에 도달하는 것은 시스템 실패를 의미하기 때문에, 과도 비율 근접성을 통해 간접적으로 파악함
- 안정화 시간(Settlement Time): 안정적인 상태로 회복하기 위해 필요한 시간을 나타냄. 시스템 혹은 실시간 태스크에 미치는 부하의 심각성을 파악하기 위해 사용
- 성능여유분(Performance Laxity): 데드라인 미스 비율이 임계치를 넘은 이후, 다시 안정적인 상태에 접어들었을 때, 실시간 태스크 허용 값과 실시간 태스크 성능간의 거리를 나타냄. 시스템 변화에 적응한 정도를 파악하기 위해 사용

본 논문에서는 [5,6]에서 개념적으로 정의된 성능 측정지수들을 TMO 모델 기반 시스템에서 사용할 수 있도록 추출 알고리즘으로 구성하여 제안하였다.

4.4.2 성능 측정지수

1) 안정성

일정 시간 동안 데드라인 미스가 '0'에 수렴한다면, '시스템은 안정적'이라고 할 수 있다[6]. 이는 사용자가 요구하는 서비스 요청에 대해 데드라인 미스 없이 서비스가 가능한 것을 의미한다. 가용성 관점에서 본다면, 일정 기간 중에 실시간 태스크가 기능을 발휘하고 있을

시간 비율로 안정성을 파악할 수 있다[19]. 본 논문에서는 가용성 개념에서 차용한 MTTR(Mean Time To Repair), MTBF(Mean Time Between Failure)를 이용해 안정성을 구한다.

먼저, 일정 기간 동안 허용 가능한 실시간 태스크의 데드라인 미스 비율을 SMB(Steady-State Miss-Ratio Bound)라 하자[6]. 만일 실시간 태스크의 데드라인 미스 비율이 SMB 보다 큰 경우, 이를 실패로 정의한다. TMO 실시간 태스크들이 실패 상태에 있는지를 파악하기 위해서는 일정 기간 동안의 평균 데드라인 미스 비율을 파악해야 한다. Lp_{ijk}가 시간 프레임 내에서 k번째 수행되는 SpM_{ij}에 의한 부하라면, SpM_{ij}의 종료시간은 Lp_{ijk}에 따라 변화하게 된다. 따라서 부하를 고려한 SpM_{ij}의 종료시간 Ep_{ijk}와 데드라인 Cp_{ij}를 비교해 보면, 데드라인 미스가 발생했는지 알 수 있다. 또한 전체 실행 횟수와 데드라인 미스 횟수를 비교하면 실시간 태스크의 데드라인 미스 비율을 구할 수 있다.

이를 위해 다음의 방정식을 제안한다. SpM_{ij}의 k 번째 수행 시 종료시간이 Ep_{ijk}이라 할 때, 시간 프레임 내에서 발생하는 실시간 태스크의 데드라인 미스 비율 m_k 는 다음과 같다.

$$m_k = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{M_i}\sum_{j=1}^{M_i}\left(\frac{1}{Vp_i}\sum_{k=1}^{Vp_{ij}}\mathrm{int}\left(\frac{Ep_{ijk}(1+Lp_{ijk})}{Cp_{ij}}\right)\right)$$

만일, m_k가 SMB 보다 크다면, 실패를 의미하며, 시간 프레임의 시간 t를 실패 시간으로 본다. 이러한 정의를 기초로 MTTR와 MTBF의 개념을 차용하면, 시스템 안정성 S는 다음과 같이 나타낼 수 있다.

$$S = 1 - \frac{n}{f}$$

상기의 주요 공식들을 이용해, 안정성을 추출하기 위한 알고리즘을 표 4와 같이 설계하였다.

표 4 안정성 추출 알고리즘

Initialization:

$f = \frac{T}{t}$

Iteration:

FOR k = 1 to f DO

BEGIN

$$m_k = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{M_i}\sum_{j=1}^{M_i}\left(\frac{1}{Vp_i}\sum_{k=1}^{Vp_{ij}}\mathrm{int}\left(\frac{Ep_{ijk}(1+Lp_{ijk})}{Cp_{ij}}\right)\right)$$

if (m_k > SMB)

$n = n + 1$

END

$$S = 1 - \frac{n}{f}$$

2) 과도 비율

과도 비율은 가장 큰 데드라인 미스 비율을 의미한다. 과도 비율이 중요한 이유는 과도 비율에 도달하는 시점에서 시스템 실패가 일어나기 때문이다. 따라서 본 논문에서는 과도 비율 자체를 파악하지 않고, 데드라인 미스 비율이 과도 비율에 어느 정도 접근하는지 파악하여, 간접적으로 과도 비율을 찾아내는 방법을 고안하였다.

다음은 [4]의 운영 법칙(Operation Law)을 이용해 유도한 결과이다. C_o가 완료된 서비스 수, T가 전체 시간이라 할 때, 처리율 X는

$$X = \frac{C_o}{T} \tag{1}$$

이다. 다시 전체 서비스 요청 수 관점에서 미스 비율을 파악해 보자. V가 T시간 동안 일어난 서비스 요청 수라고 할 때, 미스 비율 m은

$$m = \frac{V - C_o}{V} \tag{2}$$

이다. (1)과 (2)를 연동하기 위해, (1)을 다음과 같이 변형한다.

$$C_o = XT \tag{3}$$

(2)와 (3)을 이용하면, 아래와 같이 유도할 수 있다.

$$m = \frac{V - XT}{V}$$

$$Vm = V - XT$$

$$X = \frac{V(1-m)}{T} \tag{4}$$

(4)를 통해 데드라인 미스 비율과 처리율은 반비례 관계에 놓인 것을 알 수 있다. Vp_i가 시간 프레임 t 동안 발생한 SpM_i의 요청 수라 할 때 (4)는 다음과 같이 표현할 수 있다.

$$X_i = \frac{Vp_i(1-m_k)}{k}$$

이제 m_k가 과도 비율에 어느 정도 접근하는지 파악해 보자. 이를 위해 병목 분석 기법의 BJB(Balanced Job Bound)를 이용하였다[20]. Zahorjan 등은 병목현상이 발생하지 않는 균형잡힌 시스템(a balanced system)을 관찰하여 특정 자원의 성능 변화가 다른 자원의 병목현상을 야기하지 않는 상한과 하한 기준을 도출하였다. 만일 특정 자원의 처리율이 처리율 값의 상한과 하한 사이에 위치한다면 이는 해당 자원이 다른 자원에 부하를 주지 않고, 서비스를 처리하고 있음을 나타낸다. 본 논문의 관점에서 보면, TMO 객체의 처리율을 통해서 TMO 객체가 데드라인 미스를 발생시킬 수 있는 가능성이 어느 정도 커지는지를 파악할 수 있다. 다시 말해

TMO 객체 i의 처리율이 하한 범위 Bl_i과 상한 범위 Bu_i에 근접할수록 데드라인 미스의 가능성이 커진다는 것을 의미한다.

본 논문의 목적에 따라 BJB 방정식을 변형하면 다음과 같다. D_ip가 TMO 객체 i의 SpM의 평균 요구 시간, D_ip_{max}가 가장 많은 요청을 받은 SpM의 요청 수라 할 때, 처리율 X_i의 하위 범위 Bl_i는 다음과 같다.

$$Bl_i=\frac{N}{Z+D_ip+(N-1)D_ip_{\max}\frac{(N-1)D_ip}{(N-1)D_ip+Z}}$$

D_ip_{avg}가 T 시간 동안 일어난 TMO 객체 i의 SpM들의 평균 요청 수라 할 때, 처리율 X_i의 상위 범위 Bu_i는 다음과 같다.

$$Bu_i=\min\left\{\frac{1}{D_ip_{\max}},\frac{N}{Z+D_ip+(N-1)D_ip_{avg}\frac{D_ip}{D_ip+Z}}\right\}$$

m_k가 과도 비율에 어느 정도 접근하는지 파악하기 위해서는, 처리율 X_i가 상위 범위와 하위 범위 사이의 어느 지점에 위치하는지를 파악하여, 이를 위치 비율로 나타내면 된다. Ma_i이 과도 비율에 대한 접근 정도라 할 때,

$$Ma_i=\frac{X_i-Bl_i}{Bu_i-Bl_i}$$

이다. Ma_i는 0.5를 기준으로 0.5에서 멀어질수록 과도 비율에 근접하는 것으로 이해할 수 있다. 상기 제시한 공식들을 이용해 표 5와 같이 과도 비율 근접성 추출 알고리즘을 설계하였다.

3) 안정화 시간

안정화 시간은 시스템에 변화가 생긴 후, 다시 안정될 때까지 걸리는 시간을 나타내는 측정지수이다. 이는 시스템 혹은 실시간 태스크에 미치는 부하 심각성을 이해하기 위해 사용한다. 안정화 시간을 파악하기 위해, 병목 분석 기법의 BJB를 이용한다. 안정화와 관련하여 안정과 불안정의 두 가지 상태가 존재한다. 불안정 상태는 처리율 X가 BJB를 벗어나는 경우에 발생하는 것으로 가정한다.

$$X(N)\le \text{하위범위}$$

혹은

$$X(N)\ge \text{상위범위}$$

반면, 안정 상태는 처리율 X가 BJB안에 위치하는 경우이다. 즉, 다음의 경우, 안정 상태로 인식한다.

$$\text{하위범위}\le X(N)\le \text{상위범위}$$

TMO 객체 i가 불안정 상태가 되는 시점을 ts_i, 안정 상태가 되는 시점을 te_i라 하면, 안정화 시간은 다음과 같이 구할 수 있다.

$$E_i = te_i - ts_i$$

상기의 공식들은 이용하여 안정화 시간 추출 알고리즘을 구성하면 표 6과 같다.

표 5 과도 비율 추출 알고리즘

```
FOR k = 1 to f DO
BEGIN
   FOR i=1 to N DO
   BEGIN
      FOR j=1 to M_i DO
      BEGIN
         Dp_ij=Sp_ij Vp_ij
         D_ip_max = max{Dp_ij}
         X_i = Vp_i(1 - m_k) / k
         D_ip_avg = Σ_{i=1}^{N} (1/M_i) Σ_{j=1}^{M_i} Dp_ij
         Bl_i = N / (Z + D_ip + (N-1)D_ip_max (N-1)D_ip / ((N-1)D_ip + Z))
         Bu_i = min{ 1/D_ip_max , N / (Z + D_ip + (N-1)D_ip_avg D_ip/(D_ip+Z)) }
         If (Bl_i ≤ X_i AND Bu_i ≥ X_i)
         BEGIN
               Ma_i = (X_i - Bl_i) / (Bu_i - Bl_i)
         Else If (Bl_i ≥ X_i)
               Ma_i = 0
         Else
               Ma_i = 1
         END
      END
   END
END
```

4) 성능여유분

성능여유분은 데드라인 미스 비율이 임계치를 넘은 이후, 다시 안정 상태에 접어들었을 때, 허용값과 성능간의 거리로 나타낸다. 여기서 임계치는 SMB로, 허용값은 해당 성능치의 최대 허용범위로 이해할 수 있다. 예를 들어, 실시간 태스크 응답시간의 허용값은 데드라인이 된다. 성능여유분은 성능 모델에 기초해 구할 수 있다. 이때, 기준 성능치는 응답시간으로 한다.

$$\text{성능여유분} = \text{허용가능범위} - \text{응답시간} \quad (5)$$

Ba_{ij}가 허용값, Pl_{ij}가 성능여유분이라면, (5)를 다음과 같이 바꾸어 표현할 수 있다.

$$Pl_{ij} = Ba_{ij} - R_{ij}$$

장비 i의 서비스 시간과 부하를 S_i, Q_i라 하면, 응답시간 R_i은 [4]의 운영 법칙에서 차용하여 다음과 같이 나타낼 수 있다.

표 6 안정화 시간 추출 알고리즘

```
Initialization:
        state=1
Iteration:
FOR k = 1 to f DO
BEGIN
    FOR i=1 to N DO
    BEGIN
        FOR j=1 to Mi DO
        BEGIN
            Xi = Vpi(1-mijk) / k
            If (Xi< Bli OR Xi > Bui)
            BEGIN
                state = 0
                ts=time()
            END
        ...
            If (state = 0 AND ( X≥ Bli AND X≤Bui)
            BEGIN
                te=time()
                Ei = tei - tsi
            END
        ...
        END
    END
END
```

표 7 성능여유분 추출 알고리즘

```
Initialization:
        state=1
        f = T / t
Iteration:
FOR k = 1 to f DO
BEGIN
    FOR i=1 to N DO
    BEGIN
        FOR j=1 to Mi DO
        BEGIN
            Xi = Vpi(1-mijk) / k
            If (Xi< Bli OR Xi > Bui)
            BEGIN
              state = 0
            END
        ...
            If (state = 0 AND ( X≥ Bli AND X≤Bui)
            BEGIN
                Rij = Spij(1-Lpij)
                Plij = Baij - Rij
            END
        END
    END
END
```

$$R_i = S_i(1-Q_i) \quad (6)$$

마지막으로, (6)을 실시간 태스크를 고려하여 변형할 수 있다. Sp_{ij}가 SpM_{ij}의 서비스 시간, Vp_{ij}가 SpM_{ij}의 요청이 발생한 수, 그리고 Lp_{ij}는 SpM_{ij}의 부하라고 할 때, (6)은 다음과 같이 나타낼 수 있다.

$$R_{ij} = Sp_{ij}(1-Lp_{ij})$$

상기 공식을 토대로, 표 7과 같이 성능여유분 추출 알고리즘을 설계하였다.

4.5 보고 모델링

4.5.1 보고 모델

TDAF를 통해 도출되는 분석 결과는 실시간 프로그램을 작성하는 개발자가 보다 정확한 데이타에 근거하여 실시간 프로그램을 개발할 수 있도록 지원한다. 그런 점에서 보고 모델이 TDAF의 마지막 구성요소가 되는 것은 의미가 있다.

보고 모델이란 개발자에게 제공할 보고서를 구성하는 데이타 및 데이타 형태를 의미하며, 보고 프로파일과 보고 포맷으로 구성된다.

4.5.2 보고 프로파일

보고 프로파일은 개발자에게 제공될 최종 데이타 집합을 의미한다. 이는 상황에 따라 다르지만, 본 논문에서는 TMO 모델 기반의 시스템을 개발하는 개발자 관점에서 프로파일을 구성한다.

TMO 모델 기반의 시스템 개발자들은 다음 사항을 고려한다. SpM_{ij}의 주기와 데드라인을 1,000ms와 800ms로 설정했다면, 이는 적절한 값인가? 개발자는 해당 SpM의 주기 값을 설정할 때, 자신의 경험이나 시행 착오 방식을 이용한다. 주기 및 데드라인 값이 너무 작다면, SpM은 시간 제한조건을 준수할 수 없다. 반대로, 너무 큰 값이라면, SpM은 유휴 상태에서 많은 시간이 소모된다. 또한 시스템 상황이 변경되거나, 다른 시스템에서 운영되어야 한다면, 처음에 부여한 주기와 데드라인 값은 여전히 적절한가?

상기 고려사항을 근거로 보고 프로파일에 포함할 기본 데이타를 다음 두 가지로 결정하였다.

- SpM 추천 주기: 각 SpM에 유효한 추천 주기 값
- SpM 추천 데드라인: 각 SpM에 유효한 추천 데드라인 값

상기 데이타를 우선적으로 선택한 이유는 개발자가 실시간 프로그래밍할 때, 가장 먼저 실시간 태스크의 주기 및 데드라인를 결정을 해야 하기 때문이다.

1) SpM의 추천 주기

SpM 추천 주기는 실제 실행을 근거로 작성되며, 주기의 평균 여유분(이하, 주기 여유분)이라는 개념에 근거하여 계산된다. SpM 주기 여유분은 설정 주기와 실시간 태스크 응답시간을 비교하여 파악한다. Pp_{ij}가 특

정 시간 프레임에서 SpM_{ij}의 주기, Vp_{ij}가 SpM_{ij}의 서비스 요청 횟수, 그리고 Sp_{ijk}가 시간 프레임에서 k번째 SpM_{ij}의 요청을 처리하는 서비스 시간이라 할 때, SpM_{ij} 주기 여유분은 다음을 통해 구할 수 있다.

$$\overline{FPp_{ij}}=\frac{1}{Vp_{ij}}\sum_{k=1}^{Vp_{ij}}\frac{Pp_{ij}-Sp_{ijk}}{Pp_{ij}}$$

SpM 주기 여유분을 기초로 추천 주기를 제안할 수 있다. 이때, SpM 주기 여유분만이 아니라, 시스템의 안정성도 고려한다. 만일 안정성이 낮다면, 데드라인 미스 비율이 높기 때문에, 보다 긴 주기를 부여해야 한다. 반대의 경우 보다 짧은 주기를 부여해도 상관이 없다.

시스템이 가장 안정적인 상태라면, 4.4.2절에서 제시한 안정성 측정지수의 값이 1이 되고, 추천 주기는 $-\overline{FPp_{ij}} * Pp_{ij} + Pp_{ij}$가 되므로, SpM_{ij}의 주기인 Pp_{ij}에서 여유분이 제거된다. 만일 시스템이 불안정 상태라면, 안정성 측정지수는 1보다 작게 되고, $(1-S)*Pp_{ij}$만큼 주기가 증가하게 된다.

그러나 시스템 상태와 상관없이 개발자가 초기에 부여한 주기 값이 작았다면, 주기 여유분 값은 음의 값을 취하게 되고, 실체 추천주기를 증가하게 된다. 반면, 초기에 부여한 주기 값이 컸다면, 주기 여유분은 양의 값을 취하게 되므로, 실제 추천주기는 감소하게 된다. 상기 논리에 근거하여 SpM_{ij}의 추천주기는 다음과 같이 구할 수 있다.

$$RPp_{ij}=(1-S-\overline{FPp_{ij}})Pp_{ij}+Pp_{ij}$$

2) SpM의 추천 데드라인

SpM의 추천 주기와 마찬가지로, SpM의 데드라인도 추천할 수 있다. SpM 추천 데드라인은 평균 데드라인 여유분(이하, 데드라인 여유분) 개념에 기초한다. 데드라인 여유분은 각 SpM의 데드라인과 실행시간을 비교하여 구할 수 있다. 다음 공식을 통해 계산한다.

$$\overline{FCp_{ij}}=\frac{1}{Vp_{ij}}\sum_{k=1}^{Vp_{ij}}\frac{Cp_{ij}-Sp_{ijk}}{Cp_{ij}}$$

데드라인을 추천할 경우에도, 시스템 안정성을 고려해야 한다. 추천 주기를 도출한 논리와 동일하게 추천 데드라인은 다음 공식을 이용하여 구할 수 있다.

$$RCp_{ij}=(1-S-\overline{FCp_{ij}})Cp_{ij}+Cp_{ij}$$

보고 파일을 구성하는 추천 값들을 구하는 알고리즘은 표 8과 같다.

4.5.3 보고 포맷

보고 포맷은 보고 프로파일을 어떤 형식으로 제시할지 결정할 수 있도록 포맷 정보를 모아놓은 것이다. 표 9는 개발자가 선택할 수 있는 포맷 유형과 각 유형에 해당하는 측정지수를 보여준다. 필요에 따라 포맷 유형을 추가할 수 있다.

표 8 보고 프로파일을 구성하기 위한 알고리즘

```
Iteration:
FOR i = 1 to N DO
BEGIN
    FOR j=1 to Mi DO
    BEGIN
        For k=1 to Vpij DO
        BEGIN
            FPp_ij = (1/Vp_ij) Σ_{k=1}^{Vp_ij} (Pp_ij - Sp_ijk)/Pp_ij
            FCp_ij = (1/Vp_ij) Σ_{k=1}^{Vp_ij} (Cp_ij - Sp_ijk)/Cp_ij
        END
        RPp_ij = (1 - S - FPp_ij) Pp_ij + Pp_ij
        RCp_ij = (1 - S - FCp_ij) Cp_ij + Cp_ij
        ...
        print(fo_type_period, RPpi)
        print(fo_type_deadline, RCpi)
        ...
    END
END
```

표 9 보고 포맷

포맷 유형	포맷	표현가능 측정지수
fo_ver	막대 그래프	N/A
fo_pie	파이 그래프	N/A
fo_line	선 그래프	N/A
fo_text	텍스트	· SpM 추천주기 · SpM추천 데드라인
fo_slide	슬라이드	· SpM 추천주기 · SpM추천 데드라인

5. 실 험

TDAF는 여러 요소로 구성되며, 각 요소를 지원하기 위한 알고리즘 및 측정지수들을 포함한다. 따라서 TDAF를 검증하려면 각 알고리즘을 대한 적합성 및 신뢰성 검증 실험을 수행해야 한다. 그러나 본 논문에서는 우선적으로, TDAF 요소 중 부하 모델을 구성하기 위한 TA-MVA알고리즘과 관련된 실험을 수행하는 것으로 한정한다.

5.1 실험 환경 및 설계

그림 7은 실험 환경을 보여준다. 먼저, 실험을 위해 리눅스에 LTMOS미들웨어를 탑재하였다. 어플리케이션 계층에는 TMO 프로그램, RP 모니터, 그리고 TSA (TMO based Static Analyzer)가 동작하도록 하였다. TMO 프로그램은 3개의 SvM과 이를 호출하는 3개의

SpM으로 구성된 하나의 TMO 객체를 포함하도록 설계하였다. 각 SpM은 각각 일정 시간의 CPU 바운드 작업을 수행하도록 하였다. RP 모니터는 TMO 프로그램을 감시하며, 관련 데이타를 수집한다. 마지막으로 TSA는 수집한 TMO 관련 데이타를 이용해 정적 컨텍스트에서 정적 분석을 수행한다.

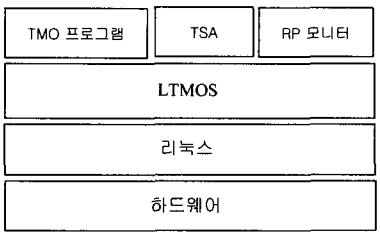

그림 7 실험 환경

그림 8은 실험 요소의 구성 및 데이타 흐름을 보여준다. TMO 프로그램의 설정 데이타는 수행 전에 DB에 저장되고, TMO 프로그램의 수행 데이타는 RP 모니터가 수집하여 DB에 저장한다. TSA는 DB로부터 수행 데이타와 설정 데이타를 입력 받아 정적 분석을 수행한다.

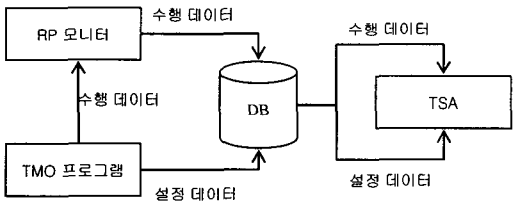

그림 8 실험 및 분석 구조

5.2 실험: TA-MAV를 이용한 SpM의 부하 계산

앞서, 큐 길이는 부하이며, 부하의 증감은 어플리케이션의 실행시간과 처리율의 증감을 의미한다고 언급하였다. 또한 실시간 태스크 관점에서 보면, 실시간 태스크 부하가 응답시간 및 처리율 변화를 가져오는 것으로 가정하였다. 본 실험은 이러한 가정에 근거하여 시스템 부하와 실시간 태스크 부하 유형이 어느 정도 유사한지 파악하고자 실시하였다. 또한, 실시간 태스크가 부하에 따라 어떠한 응답시간, 처리율, 부하 길이의 유형을 보이는지를 확인하였다.

표 10, 11은 SpM과 SvM의 설정 정보를 보여준다. 표 10에서 각 SpM은 호출 즉시 메소드를 수행하도록 시작시간을 '0'으로 설정했으며, 강제적으로 종료하기 전에는 종료되지 않도록, 운영유형을 'FOREVER'로 설정하였다. 표 11에서 SvM은 SpM에 의해 호출되므로, 데드라인만 갖도록 설정하였다.

표 10 SpM 설정 정보

이름	주기	데드라인	시작시간	유형
SpM1	1,000	500	0	FOREVER
SpM2	2,000	500	0	FOREVER
SpM3	3,000	500	0	FOREVER

표 11 SvM 설정 정보

이름	주기	데드라인	시작시간	유형
SvM1	N/A	500	N/A	N/A
SvM2	N/A	500	N/A	N/A
SvM3	N/A	500	N/A	N/A

실험 결과, 하나의 TMO를 동작시켰을 때, SpM1, SpM2, 그리고 SpM3는 평균적으로 36.1ms, 111.0ms, 429.3ms의 응답시간을 보였다. 이 데이타를 기초로, TA-MVA를 통해 응답시간, 처리율, 각 SpM의 부하길이를 계산하여, 표 12와 같은 결과를 얻었다. 단, 계산시 허용 에러율 e는 0.01이며, 첫 번째 열이 의미하는 바는 TA-MVA의 연산 반복 횟수를 의미한다.

TMO 개수를 증가시키면서 동일한 계산을 수행하면, 그림 10, 그림 11, 그리고 그림 12의 결과를 얻을 수 있다.

그림 9는 TMO 개수에 따른 TMO의 응답시간을 의미한다. 응답시간은 부하 증가에 비례하여 길어지는 것을 알 수 있다. 즉, 부하가 증가할수록, 요청한 서비스 처리를 위해 더 많은 시간이 요구되는 것을 알 수 있다.

그림 10은 TMO 개수에 따른 처리율 변화를 보여준다. TMO 개수가 증가함에 따라, 처리율의 증가 속도는 감소하는 것을 알 수 있다. 결국, 부하가 일정 수준 이

표 12 실험 결과(TMO 개수가 5일 때)

No	응답시간				TMO 처리율	부하 길이		
	SpM1	SpM2	SpM3	TMO		SpM1	SpM2	SpM3
1	0.2338096	0.703296	2.718144	8.914752	0.466298204	0.638164531	0.983837	2.534931331
2	0.054379138	0.198364724	1.298988433	3.519345869	0.938553667	0.306226439	0.558528	2.438340713
3	0.044819321	0.16059727	1.265838533	3.282384805	0.982244013	0.264141061	0.473237	2.486724641
4	0.043607263	0.153023456	1.282443897	3.285601738	0.981623664	0.256835525	0.450634	2.517754553
5	0.043396863	0.151016329	1.293093363	3.299616892	0.978930117	0.254894978	0.443503	2.531696073
6	0.043340975	0.150383093	1.297878092	3.306951315	0.977526411	0.254201688	0.441010	2.537420226

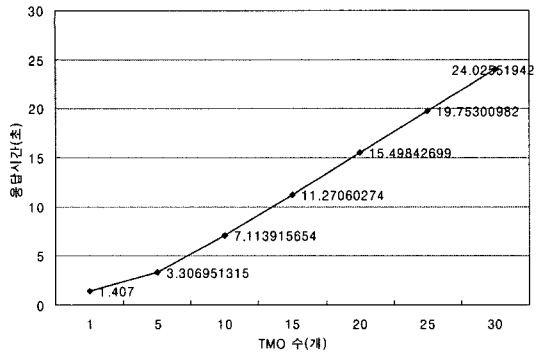

그림 9 응답시간(TMO 기준)

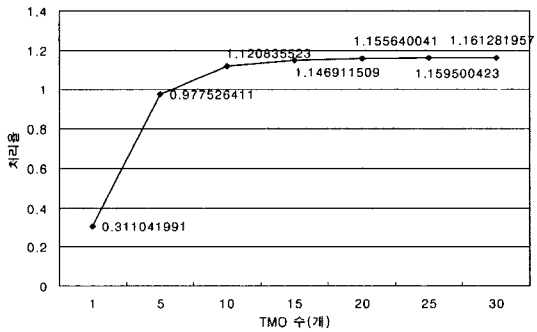

그림 10 처리율(TMO 기준)

상인 경우, 일정 처리율 이상을 기대하기 어렵다. 그림 10의 경우, TMO 개수가 15를 넘는 시점부터는 처리율 증가가 둔화되는 것을 확인할 수 있다.

그림 11은 TMO 개수 증가에 따른, 각 SpM 부하 길이의 변화를 보여준다. 부하 길이는 각 SpM 수행에 의해 발생하는 부하 정도를 의미한다. 부하 길이는 응답시간이 클수록, 부하 길이의 증가 속도가 커지는 것을 확인할 수 있다.

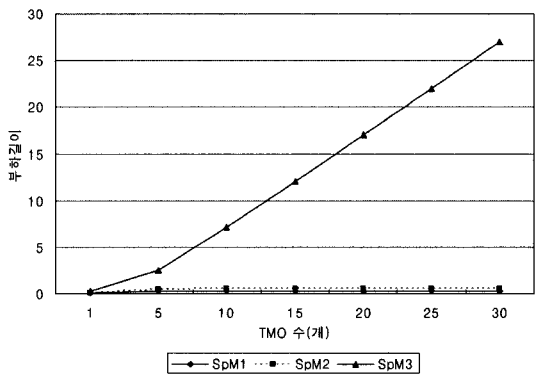

그림 11 부하 길이(SpM 기준)

5.3 실험의 결론

실험을 통해, TA-MVA를 통해 도출한 결과 유형과 [19]에서 제시한 A-MVA의 결과 유형이 동일함을 확인할 수 있었다. 이는 앞서 제시한 가정을 입증해 주는 결과이며, TA-MVA는 실시간 태스크 부하를 확인할 수 있는 도구임을 의미한다.

부하 증가에 따른 실시간 태스크의 응답시간, 처리율, 부하 길이의 변화 유형은 그림 9-11을 통해 제시한 바와 같다. 즉, 개발자가 실시간 태스크의 수행 작업을 어떻게 설계하느냐에 따라, 응답시간과 처리율 등의 차이가 발생할 수 있지만, 응답시간, 처리율, 부하 길이의 기본적인 유형은 그림 9-11에서 확인한 유형을 따른다는 것을 알 수 있다.

6. 결론 및 향후 방향

6.1 결론

본 논문에서는 TMO 모델의 성능 측정 및 분석을 통합적으로 지원할 수 있도록, 실시간 태스크 성능 분석 체계인 TDAF를 제안하였다. 또한 TDAF의 구성요소를 실현하기 위한 모델링 알고리즘과 측정지수들을 새롭게 제안하였다. 구체적으로 기존 부하 프로파일에 TMO 실시간 요소를 추가하여 부하 프로파일 영역을 시스템 요소에서 실시간 요소까지로 확장하고, 실시간 태스크의 부하를 계산할 수 있는 TA-MVA 부하 계산 알고리즘을 제안하였다. 둘째로, 부하를 이용하여 실시간 시스템과 실시간 태스크를 모델링할 수 있는 부하 모델을 제안하였다. 이는 기존 모델과 달리, 부하 변동 및 시스템과 실시간 태스크 부하간의 관계로 표현할 수 있도록 설계한 것이다. 셋째로, 특정 부하 모델 하에서 실시간 태스크의 성능을 측정할 수 있는 성능 모델을 제안하였다. 성능 모델에서 요구하는 성능 측정지수들은 TMO 실시간 태스크의 특성을 고려하여 설계하였다. 마지막으로 사용자에게 분석 데이타를 제공하기 위한 보고 모델을 제안하였다. 본 논문에서는 사용자들의 요구를 반영하여 SpM 추천 주기, SpM 추천 데드라인으로 보고 프로파일을 구성하였고, 유연한 보고를 위한 보고 포맷을 제안하였다.

또한 본 논문에서는 TA-MVA의 유효성을 입증하기 위해, TMO 실시간 태스크가 보여주는 응답시간, 처리율, 그리고 부하 길이의 기본적인 유형을 도출하기 위한 실험을 수행하였다. 실험 결과, TA-MVA와 A-MVA의 결과 유형이 동일함을 확인하였다. 이는 앞서 제시한 가정을 입증함과 동시에 TA-MVA가 실시간 태스크 부하를 확인할 수 있는 도구임을 나타낸 것이다.

6.2 향후 방향

본 논문에서는 단일 노드 환경의 TMO모델을 고려한

분석 체계를 제안하였다. 그러나, TMO모델은 분산 실시간을 지원하므로, TMO 성능을 보다 정확하게 측정하고 분석하기 위해서는 단일 노드 뿐 아니라, 네트워크에 기반한 다중 노드 환경도 고려해야 한다. 향후에는 TDAF 및 TDAF 구성 요소들이 다중 노드 환경의 TMO 태스크를 분석할 수 있는 동적 분석 체계와 관련된 연구가 요구된다.

참 고 문 헌

[1] Agrawala, A. K., J. M. Mohr, "A model for Workload Characterization," *Annual Simulation Symposium Proceedings of the Symposium on Simulation of Computer Systems*, 1975.

[2] Ferrari, D., "Characterization and Reproduction of the Referencing Dynamics of Programs," *Proceedings of the Performance*, 1981.

[3] Mincer-Daszkiewicz, J. and Weiss, Z., "an Approach to Program I/O Reference Behavior Modeling," *Performance Evaluation*, 1984.

[4] Raj Jain, *"the Art of Computer Systems Performance Analysis,"* John Wiley & Sons, Inc., 1991.

[5] Rosu, D., Schwan, K., Yalamanchili, S., Jha, R., "On Adaptive Resource Allocation for Complex Real-time Applications," *Proceedings of the 18th IEEE Real-Time Systems Symposium*, 1997.

[6] Chenyang Lu, J. A. Stankovic, T. F. Abdelzaher, Gang Tao, S. H. Son, M. Marley, "Performance Specifications and Metrics for Adaptive Real-time Systems," *Proceedings of the 21st IEEE Real-Time Systems Symposium*, 2000.

[7] Brandt, S. A., "Performance Analysis of Soft Real-Time Systems," *1997 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, 1997.

[8] Brandt, S.A., "Performance Analysis of Dynamic Soft Real-time Systems," *Proceedings of the IEEE Performance, Computing, and Communications*, 2001.

[9] Yoon-Seok Jeong, Tae Wan Kim, Chun Hyon Chang, "Design and Implementation of a Run-time TMO Monitor on LTMOS," *Proceedings of the Embedded Systems and Applications*, 2003.

[10] Kane Kim, et al., "A Timeliness-Guaranteed Kernel Model: DREAM Kernel and Implementation Techniques," *RTCSA*, 1995.

[11] Murthy V. Devarakonda and Ravishankar K. Iyer, "Predictability of Process Resource Usage: A Measurement-Based Study on UNIX," *IEEE Transactions on Software Engineering*, 1989.

[12] Hyun-Jun Kim, et al., "TMO-Linux: a Linux-based Real-time Operating System Supporting Execution of TMOs," *Proceedings of the 15th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2002.

[13] Kim, J. G. and Cho, S. Y., "LTMOS: An Execution engine for TMO-Based Real-Time Distributed Objects," *Proceedings of the PDPTA*, 2000.

[14] S.H.Park, *"LTMOS(LinuxTMO System)'s Manua,l,"* HUFS, 2000.

[15] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials," *Proceedings of the 18th IEEE Computer Software & Applications Conference*, 1994.

[16] Chenyang Lu, "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm," *Proceedings of the 20th IEEE Real-Time Systems Symposium*, 1999.

[17] Schweitzer, P., "Approximate Analysis of Multiclass Closed Networks of Queues," *International Conf. on Stochastic Control and Optimization*, 1979.

[18] Bard, Y., "Some Extensions to Multiclass Queues Queueing Network Analysis," *Performance of Computer Systems*, 1979.

[19] 권오운, 이홍철, "설비 Loss 구조에 의거한 가동효율 관련 지표의 향상방안 연구", 대한설비관리학회, 2003.

[20] Zahorjan, et al., "Balanced Job Bound Analysis of Queueing Netwokrs," *Communications of the ACM*, 1982.

정 윤 석

2000년 건국대학교 컴퓨터공학과(공학석사). 2000년~현재 건국대학교 컴퓨터공학과 박사과정. 관심분야는 실시간 시스템 모니터링, 객체지향 프로그래밍, XML

김 태 완

1996년 건국대학교 전자계산학과(공학석사). 1996년~2001년 현대중공업 기전연구소 연구원. 1996년~현재 건국대학교 컴퓨터공학과 박사과정. 2004년~현재 건국대 컴퓨터공학부 강의교수. 관심분야는 프로그래밍 언어, 실시간 프로그래밍, 자동화 소프트웨어, 산업기기 감시 진단 제어 시스템

장 천 현

1977년 서울대학교 계산통계(학사). 1979년 KAIST 전산학(석사). 1985년 KAIST 전산학(박사). ~현재 건국대학교 컴퓨터 공학과 정교수. 관심분야는 프로그래밍 언어, 컴파일러, 실시간 시스템