□ 기술해설 □

다중프로세서 운영 체제의 성능 모니터링

국방정보체계연구소 박상서*
중앙대학교 김성조**

● 목 차 ●

1. 서 론

다중프로세서 시스템을 구성하는 논리적 요소 중 시스템의 성능에 영향을 주는 것[1]에는 하드웨어, 운영 체제, 그리고 응용 프로그램 등이 있지만, 가변적이지 않고 성능에 미치는 영향을 정형화하기 용이한 요소는 하드웨어와 운영 체제이다. 하드웨어는 일부 부품을 보다 빠른 것으로 교체하거나 시스템의 구조를 변경함으로써 시스템의 성능을 향상시킬 수도 있지만, 일반적으로 이것은 개발이 일단 완료되면 그 성능의 수준이 이미 고정(fixed)된 경우가 대부분이다. 반면, 운영 체제는 주요 성능 변수를 조정(tuning)하거나 성능을 저하시키는 일부 커널 서비스 루틴만을 재코딩함으로써 시스템의 전체적인 성능 향상을 쉽게 꾀할 수 있으며, 그 비용도 하드웨어에 비해 저렴하다. 또한, 운영 체제는 동작중인 응용 프로그램에 대해 계속적인 서비스를 제공하여야 하므로 시스템의 전체적인 성능에 직접적으로 영향을 주는 중요한 요소이다. 따라서, 운영 체제의 성능은 시스템의 성능에 가장 큰 영향을 미치는 요소 중 하나로 간주될 수 있으므로 다중프로세서 시스템의 성능은 주로 운영 체제를 대상으로 측정되어야 한다.

다중프로세서 시스템용 운영 체제의 성능을 측정하기 위해서는 시스템이 현재 발휘하고 있는 성능 통계에 관한 데이타가 수집되어야 하며, 운영 체제가 동작하거나 응용 프로그램을 지원하는데 소요한 시간 역시 측정되어야 한다[2]. 성능 통계 데이타를 계속적으로 수집하면 시스템의 운용에 관련된 주요 변수들을 저속적

*정 회 원
**종신회원

으로 관찰할 수 있어 동작중인(active) 시스템이 만족할 만한(acceptable) 상태를 유지하는지 파악할 수 있다[3, 4, 5, 6]. 또한, 운영 체제의 처리 시간을 계속적으로 측정하게 되면 핫 스폿(hot spot)이나 병목현상(bottleneck)이 발생하는 지점 또는 최적화가 필요한 코드를 찾아낼 수 있다[2]. 따라서, 병목현상의 해결과 코드 최적화를 통하여 프로세스들이 유휴 상태로 남아있게 되는 시간을 감소시킬 수 있어 다중프로세서 시스템 성능의 전체적 향상이 가능해진다.

운영 체제의 성능을 측정하기 위해서는 명령어 혼합, 벤치마크, 합성(synthetic) 프로그램 또는 시뮬레이션 기법 등이 활용될 수 있으나, 이들 보다는 모니터링이 보다 적합한 것으로 알려져 있다[7, 8]. 따라서, 본 고에서는 다중프로세서 시스템용 운영 체제의 성능 모니터링에 관련된 모니터링 기법, 성능 모니터링 도구의 모델, 도구 구현 방법, 그리고 다중프로세서 시스템의 성능 모니터링에 이용되는 도구들을 소개한다.

2. 성능 모니터링 도구 요건

다중프로세서 시스템용 운영 체제의 성능을 측정하기 위한 도구는 다음과 같은 요건[1, 9, 10, 11, 12]을 만족하여야 한다. 첫째, 성능 모니터링 도구의 실행으로 인하여 대상 시스템에 부과되는 부하를 가능한 한 최소화하여야 한다. 이는 성능 모니터링 도구가 대상 시스템의 행위에 얼마나 영향을 주는가에 따라 측정된 성능 데이타의 정확성이 결정되기 때문이다. 둘째, 각종 오버헤드가 최소화되어야 한다. 시스템의 성능을 측정하기 위하여 필수적일 수밖에 없는 프로브(probe)[13] 설치 또는 커널 자료 구조 접근 등으로 인한 모니터링 오버헤드뿐만 아니라 데이타의 저장이나 네트워킹 등에 관련된 오버헤드 등도 최소화되어야 한다. 셋째, 측정된 성능 데이타는 새로운 정보로 가공되거나 추후 필요한 정보를 추출할 수 있도록 이력(historical) 형태로 저장되어야 한다. 일정 기간 동안 이력 형태로 저장된 데이타를 분석하여 성능 변화 추세를 파악하면 시스템의 성능 발휘 정도를 보다 넓은 관점에서 파악할 수 있을 뿐만 아니라, 추후 시스템의 성능 예측을 통한 시스템 확장 및 시스템 대체 계획에도 반영할 수 있다. 마지막으로, 데이타가 다양한 형태로 표현될 수 있어야 한다. 대부분 숫자와 문자만으로 구성되는 성능 데이타를 단순히 나열하는 것은 사용자가 시스템의 현재 성능 상태를 정확하게 파악하는데 도움을 주지 못한다. 따라서, 측정된 성능 데이타를 텍스트뿐만 아니라 다양한 형태의 그레픽과 다중 윈도우 등으로 시각화함으로써 시스템 성능에 대한 사용자의 이해를 보다 증진시킬 수 있어야 한다.

3. 성능 모니터링 기법

다중프로세서 시스템용 운영 체제의 성능을 측정하기 위한 모니터링 기법에는 프로그램의 실행 과정을 한 명령어 단위로 추적하는 추적 구동형(trace-driven) 기법, 이벤트의 발생시 구동되는 이벤트 구동형(event-driven) 기법, 그리고 지정된 시간 간격마다 정기적으로 성능 데이타의 샘플을 채취하는 시간 구동형(time-driven) 기법, 그리고 운영 체제의 함수가 실행될 때마다 함수의 처리 시간을 측정하는 함수 구동형(function-driven) 기법이 있다.

3.1 추적 구동형 기법

추적 구동형 모니터링 기법[2, 5, 9, 14, 15]은 다중프로세서 운영 체제의 실행 과정을 어셈블리 수준에서 한 명령어 단위로 추적하여 제어의 흐름(thread of control)으로 표현한다. 이 기법을 이용하면 운영 체제의 실행 궤적과 경쟁이 빈번히 발생하는 코드 부분을 파악할 수는 있으나 여러 프로세서가 운영 체제의 코드를 수행하는데 소요된 시간은 측정할 수 없다. 또한, 이 기법에서는 제어의 흐름을 추적하기 때문에 운영 체제에서 사용하는 자료 구조와 같은 다양한 성능 인자에 대한 측정이 불가능해 성능 통계 데이타를 수집할 수도 없다. 뿐만 아니라, 이 기법은 운영 체제를 어셈블리 수준의 명령어 단위로 추적하기 때문에 측정 단위(granularity)가 너무 작고, 측정 대상에

대하여 추상화된 뷰(view)를 제공하지도 않는다.

3.2 이벤트 구동형 기법

이벤트 구동형 모니터링 기법[1, 2, 5, 9, 13, 16, 17, 18]은 추적 구동형 기법을 추상화한 것으로서 프로그램 또는 시스템의 동적 행위를 이벤트의 순서로 표현한다. 이 기법을 이용하면 다중프로세서 시스템에서 발생하는 다양한 이벤트의 발생 시각과 빈도를 알 수 있기 때문에 핫 스폿 또는 병목현상이 발생하는 지점을 쉽게 파악할 수 있다. 그러나, 이벤트가 발생한 상황과 이벤트 간의 상관 관계 등에 대한 데이타가 주로 수집될 뿐 시스템 성능에 관한 통계 데이타를 측정할 수 없다. 또한, 이벤트를 처리하는데 소요된 경과(elapsed) 시간도 측정할 수 없어 운영 체제의 처리 시간 측정에도 직접 이용할 수 없다. 뿐만 아니라, 이벤트가 하드웨어 인터럽트의 발생에서부터 실행 제어의 특정 주소 도달, 특정 공유 메모리의 접근, 특정 변수 값의 변경, 시스템 콜 발생 여부, 프로세스의 생성과 종료에 이르기까지 광범위하게 정의[9,11,17,19,20,21]되어 있어 측정 단위가 너무 다양하며 이벤트에 대하여 일관된 뷰를 제공하지도 않는다. 더구나, 이벤트는 추상화 수준이 너무 높아 구체적인 성능 측정이 어렵다[22]. 예를 들어, 디스크 입출력에는 캐쉬, 메모리, 화일 시스템, 버퍼 캐쉬, 그리고 입출력 드라이버 등이 함께 고려되어야 하는데, 디스크 입출력 이벤트라 함은 화일 시스템에서의 작업을 의미하는지, 입출력 드라이버에서의 작업을 의미하는지 아니면 이들 모두를 의미하는지 정확하지 않다.

3.3 시간 구동형 기법

샘플링이라고도 불리는 시간 구동형 모니터링 기법[2, 5, 11, 16, 18]은 주기적으로 운영 체제의 자료 구조로부터 시스템의 성능에 대한 통계 자료를 수집하는데 주로 이용된다. 대부분의 의미있는 성능 데이타는 메모리에서의 위치가 고정되어 있으므로 주기적으로 그 주소가 가리키는 곳의 값을 읽으면 시스템 성능에 대한 통계 자료를 수집할 수 있다[2, 5, 22]. 이 기법을 이용하면 시간에 따른 성능 변화를 매우 자세히 측정할 수 있어 시스템의 현재 성능 발휘 정도와 일정 기간 동안의 성능 통계 데이타 수집에 적합하다. 그러나, 이 기법 역시 운영 체제의 처리 시간에 관련된 메트릭을 측정할 수는 없다.

3.4 함수 구동형 기법

함수 구동형 모니터링 기법[2, 22]은 운영 체제의 이진코드 이미지 중 함수의 시작점과 종료점에 프로브를 삽입하여 트랩을 발생시킴으로써 해당 함수의 처리 시간을 측정하는 기법이다. 이 기법은 함수 이미지 확장, 함수 구성 정보 작성, 함수 선택, 프로브의 삽입 및 삭제, 함수 실행 정보 로깅, 그리고 프로그램 카운터 전진 기능으로 구성된다. 함수 이미지 확장 기능은 운영 체제의 컴파일시 함수의 시작점과 종료점에 프로브를 삽입할 별도 공간을 생성하는 것이고, 함수 구성 정보 작성 기능은 모든 운영 체제 함수의 이름과 각 함수의 시작점과 종료점의 절대 주소를 파악함으로써 운영 체제가 메모리에 적재되었을 때 프로브가 삽입될 위치를 파악하는 것이다. 함수 선택 기능은 사용자에게는 함수의 이름을 보여준 뒤, 사용자가 프로브를 삽입하거나 삽입되어 있는 프로브를 삭제할 함수를 선택할 수 있도록 지원하는 기능이다. 프로브의 삽입시에는 프로브가 삽입되어 있지 않는 함수들의 목록만이 보여지며 프로브의 삭제시에는 프로브가 삽입되어 있는 함수의 목록만이 보여진다. 프로브의 삽입 및 삭제 기능은 사용자가 삽입한 운영 체제 함수의 이미지에 프로브를 삽입하거나 삽입되어 있는 프로브를 삭제하는 기능이고, 함수 실행 정보 로깅 기능은 삽입된 프로브에 의하여 트랩이 발생하였을 때 해당 함수의 실행에 관한 정보 즉, 트랩의 종류, 트랩이 발생한 주소, 해당 함수를 실행시킨 프로세스의 번호, 이 프로세스가 실행되고 있는 프로세서의 번호, 트랩이 발생한 순간의 타임스탬프 등을 로그에 기록하는 기능이다. 마지막으로, 프로그램 카운터 전진 기능은 운영 체제 함수의 실행에 관한 정보를 기록한 뒤, 운영 체제 함수의 원래(original) 코드를 정확하게 실행할 수 있도록 지원

하는 기능이다.

이 기법을 이용하면 운영 체제 중 핫 스폿 또는 병목현상이 발생하는 지점을 파악할 수 있으며, 운영 체제 함수가 실행되는데 소요된 경과 시간을 측정할 수 있어 병목현상의 해결 및 코드 최적화에 필요한 정보를 제공할 수 있다. 그러나, 이 기법을 이용할 경우 운영 체제의 자료 구조에는 접근할 수 없어 성능 통계 데이타의 측정에 직접 이용할 수 없다.

3.5 성능 모니터링 기법 비교

본 절에서 기술한 기존 성능 측정 기법의 특성을 간략히 요약 비교하면 표 1과 같다.

표 1 기존 성능 모니터링 기법의 특성

특성 / 기법	측정 단위	추상화 수준	측정 대상	
			성능 통계자료	운영체제 처리시간
추적 구동형	일정	너무낮다	불가능	불가능
이벤트 구동형	다양	너무높다	불가능	불가능
시간 구동형	일정	적 정	가 능	불가능
함수 구동형	일정	적 정	불가능	가 능

먼저, 추적 구동형 기법, 시간 구동형 기법 및 함수 구동형 기법에서의 측정 단위는 각각 어셈블리 명령어, 시간 및 운영 체제 함수로 일정한 반면, 이벤트 구동형 기법에서의 측정 단위는 무엇을 이벤트로 정의하는가에 따라 달라질 수 있어 일관적이지 않다. 두 번째로, 추적 구동형 기법을 적용하여 수집된 성능 데이타는 어셈블리 명령어 수준의 프로그램 실행 궤적이므로 추상화 수준이 너무 낮고, 이벤트 구동형 기법의 측정 대상이 되는 이벤트는 추상화 수준이 높다. 반면, 시간 구동형 기법과 함수 구동형 기법에서는 성능 데이타 수집 대상이 각각 운영 체제 자료 구조 및 운영 체제 함수이므로 추상화 수준이 적절하다. 마지막으로, 성능 측정 대상인 성능 통계 데이타와 운영 체제의 처리에 소요되는 시간 측면에서 각 기법을 비교하면, 시간 구동형 기법과 함수 구동형 기법을 적용하는 경우는 각각 성능 통계 자료와 운영 체제의 처리 시간을 측정할 수 있으며, 그 외의 기법들은 모두 성능 통계 데이타와 운영 체제의 처리 시간을 측정할 수 없다.

4. 성능 모니터링 도구 모델

사용자와 대상 시스템의 중간에 위치하면서 성능 데이타의 수집, 분석 및 브라우즈 기능을 지원하는 성능 모니터링 도구는 다음과 같은 다섯 가지의 기본적인 요소들[2, 13]로 구성된다.

- 모니터 : 하나 이상의 모니터링 기법을 이용하여 대상 시스템을 모니터링하면서 성능 통계 데이타 또는 운영 체제의 처리 시간에 관련된 성능 데이타를 수집한다.
- 분석기 : 모니터에서 수집된 성능 데이타를 사용자가 쉽게 이해할 수 있는 고수준 정보로 변환함으로써 사용자의 성능 분석 시간을 감소시키며, 사용자가 다중프로세서 시스템의 성능 상태를 보다 정확히 파악할 수 있도록 지원한다.
- 데이타베이스 : 추후 성능 비교, 성능 변화 추세 파악 또는 예측 등에 사용하기 위해 측정된 성능 데이타를 이력 형태로 저장하기 위한 안정된(stable) 저장 장소이다.
- 브라우저 : 분석된 성능 정보들은 그래픽 또는 다중 윈도우 등을 이용하여 디스플레이함으로써 사용자가 측정된 성능을 보다 쉽게 이해할 수 있도록 지원한다.
- 사용자 인터페이스 : 사용자 인터페이스는 X-윈도우나 Motif 등과 같은 그래픽을 기반으로 한 인터페이스로서 브라우저가 성능 데이타 또는 정보를 텍스트, 도표, 다중 윈도우, 각종 그래프 및 그래픽으로 표현할 수 있도록 지원한다. 또한, 윈도우와 메뉴를 기반으로 사용자의 명령을 입력받거나 성능 모니터링 도구에서 사용자에게 메시지나 안내 등을 출력할 수 있는 방법을 제공한다.

시스템 성능을 측정하기 위한 성능 모니터링 도구의 모델[22]로는 호스트 모델, 클라이언트/서버 모델, 생산자/소비자 모델, 그리고 에이

전트/매스터 모델이 있다.

4.1 호스트 모델

호스트 모델은 그림 1에서 볼 수 있는 바와 같이 모니터링 도구의 각 구성 요소가 모두 대상 시스템 상에서 실행되는 것이다. 이 모델은 간단하며 구현이 비교적 쉽다는 장점이 있는 반면, 대상 시스템에 불필요한 부하를 부과한다는 단점이 있다.

모니터는 반드시 대상 시스템에 존재하여야 한다. 그러나, 사용자 인터페이스를 위한 그래픽스 시스템의 작동, 성능 데이타의 분석 또는 브라우징 등의 부하는 모니터링과 직접적인 관련이 없는 것들이므로 이와 관련된 요소들은 대상 시스템에서 실행되지 않아도 된다. 뿐만 아니라 이 요소들은 대상 시스템의 실행에 불필요한 영향을 준다.

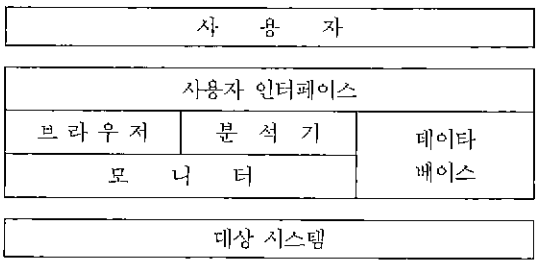

그림 1 호스트 모델

4.2 클라이언트/서버 모델

클라이언트/서버 모델은 호스트 모델의 단점을 해결할 수 있는 모델로서 그림 2와 같이 모니터와 다른 요소(분석기, 브라우저, 데이타베이스 및 사용자 인터페이스)들을 물리적으로 다른 시스템 상에 분리시킨 뒤, 네트워크를 통하여 각각 성능 데이타를 주고 받는다. 클라이언트는 성능 데이타가 필요할 때마다 서버에 성능 데이타를 요구하고, 서버는 요구를 받으면 성능 데이타를 측정하여 클라이언트에 전달한다.

이 모델은 대상 시스템에서 실행되지 않아도 될 구성 요소를 다른 시스템으로 이동시켰기 때문에 대상 시스템에 부과되는 불필요한 부하를 최소화할 수 있다. 그러나, 수집된 성능 데이타가 네트워크를 통해 클라이언트에 전달되어야 하므로 네트워킹 오버헤드가 추가된다. 뿐만 아니라, 클라이언트는 성능 데이타가 필요할 때마다 서버에 요구를 전송해야 하므로 [23] 비효율적이다. 특히, 성능 데이타는 짧은 간격(예 : 60초)마다 주기적으로 요구되는 경우가 대부분이므로 클라이언트가 매번 요구를 보내는 것은 매우 비효율적이다.

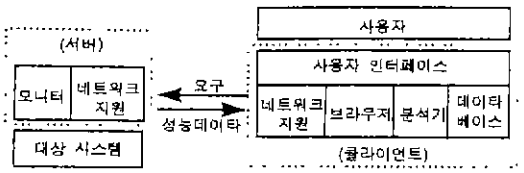

그림 2 클라이언트/서버 모델

4.3 생산자/소비자 모델

클라이언트/서버 모델에서 발생하는 요구의 반복 전달 문제점을 개선하기 위한 방법은 그림 3과 같이 생산자/소비자 모델을 이용하는 즉, 생산자는 소비자가 요구를 보내지 않아도 항상 시스템의 성능을 측정하여 소비자에게 전달하고, 소비자는 요구를 보내지 않고 생산자가 전달하는 데이타를 분석하여 브라우즈한다. 이 때, 생산자와 소비자는 모니터링 도구의 실행 초기에 맺은 규약에 따라 지속적으로 성능 데이타를 주고 받는다.

이 모델은 클라이언트/서버 모델에서 발생할 수 있는 반복적인 요구를 없앨 수 있으며, 호스트 모델에 비하여 대상 시스템에 부과되는 부하를 최소화시킬 수도 있다. 그러나, 수집된 성능 데이타가 생산자에서 소비자로 일방적으로 전달될 뿐 소비자로부터 생산자에게 제어가 전달될 수 없다. 즉, 성능 데이타 전달 주기의 변경이나 모니터링의 시작과 종료 등과 같은 제어가 전달되지 못하는 문제가 있다.

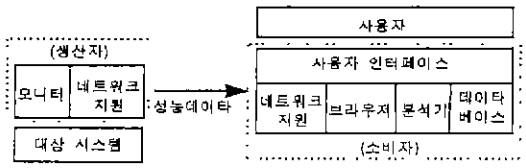

그림 3 생산자/소비자 모델

4.4 에이전트/매스터 모델

성능 모니터링 도구가 대상 시스템에 부과하는 부하를 최소화하기 위해서는 클라이언트/서버 모델 또는 생산자/소비자 모델의 접근 방법처럼 모니터를 브라우저, 분석기, 데이타베이스 및 사용자 인터페이스와 분리시켜야 한다. 이와 동시에, 클라이언트/서버 모델에서 발생하는 문제점인 요구의 반복적인 전달을 피할 수 있어야 하며, 생산자/소비자 모델에서는 불가능한 모니터 제어를 가능하게 하여야 한다. 이것은 그림 4와 같은 구조를 갖는 에이전트(agent)/매스터(master) 모델에서 가능하다. 이 모델에서 매스터는 모니터링의 시작과 종료, 또는 성능 데이타 전달 주기의 변경 등과 같이 에이전트를 제어할 필요가 있을 때마다 제어 명령을 에이전트에 전달하여 처리시키고 그 결과를 되돌려 받는다. 이 때 매스터와 에이전트는 종속 관계(master/slave)를 갖게 된다. 반면, 일단 모니터링이 시작되면 에이전트는 별도의 모니터링 종료 제어를 전달받기 전까지 매스터가 성능 데이타를 요구하지 않아도 계속적으로 대상 시스템으로부터 성능 데이타를 수집하여 매스터에게 전달한다. 이 때는 매스터와 에이전트가 동료 관계(peer-to-peer)를 갖게 된다.

이 모델은 모니터와 다른 구성 요소가 물리적으로 다른 시스템 상에 분리되어 있어 대상 시스템에 부과되는 부하를 최소화시킬 수 있다. 또한, 클라이언트/서버 모델에서 발생할 수 있는 요구의 계속적인 반복 전달을 제거할 수 있으며, 생산자/소비자 모델에서는 불가능했던 제어의 전달도 가능하다. 이 모델의 단점은 성능 데이타, 제어 및 재어 명령의 처리 결과를 전송하는데 소요되는 네트워킹 오버헤드이다. 여기서, 성능 데이타 전달 오버헤드는 대상 시스템에 대한 불필요한 부하를 감소시키기 위한 타협(trade-off)이고, 제어 와 처리 결과 전송 오버헤드는 반복적인 요구를 제거하면서 모니터를 제어하기 위한 타협이다. 이 모델에서는 모니터를 제어할 필요가 있는 경우에만 제어를 전달하나, 그러한 경우가 매우 드물기 때문에 전체적인 네트워킹 오버헤드는 클라이언트/서버 모델에 비하여 매우 적고 생산자/소비자 모델에 비해서는 조금 크다.

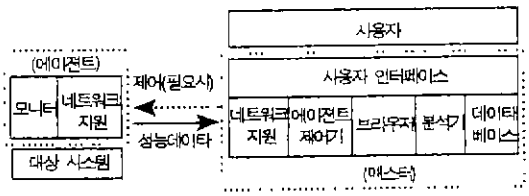

그림 4 에이전트/매스터 모델

4.5 성능 모니터링 도구 모델 비교

본 절에서 기술한 기존 성능 모니터링 도구 모델의 주요 특성을 요약 비교하면 표 2와 같다. 먼저, 모니터와 모니터링 도구의 다른 구성 요소들을 분리시킨 클라이언트/서버 모델, 생산자/소비자 모델, 그리고 에이전트/매스터 모델이 호스트 모델에 비해 대상 시스템에 대한 부하가 적은 것을 알 수 있다. 두 번째로, 호스트 모델을 제외한 모든 모델은 모니터가 다른 구성 요소들로부터 물리적으로 분리되어 있으므로 수집된 성능 데이타를 전송하는 네트워킹 오버헤드가 존재한다. 클라이언트/서버 모델은 성능 데이타뿐만 아니라 성능 데이타에 대한 요구도 매번 전송되어야 하므로 네트워킹 오버헤드가 많다. 반면, 생산자/소비자 모델에서는 성능 데이타만을 전송하고, 에이전트/매스터 모델에서는 성능 데이타와 필요한 경우에 한해서 제어 명령을 전달하기 때문에 클라이언트/서버 모델에 비해 상대적으로 네트워킹 오버헤드가 적다. 마지막으로, 생산자/소비자 모델에서는 모니터를 제어하는 능력이 제공되지 않는다.

표 2 기존 성능 모니터링 도구 모델의 특성 비교

특성 / 모델	대상 시스템에 대한 부하	네트워킹 오버헤드	도구 제어 능력
호스트	많다	없다	있다
클라이언트/서버	적다	많다	있다
생산자/소비자	적다	적다	없다
에이전트/매스터	적다	적다	있다

5. 성능 모니터링 도구 구현 방법

성능 모니터링 도구를 구현하는 방법[5,12,

13,16]은 대상 시스템에 성능 모니터링을 위한 별도의 하드웨어 장비를 부착하는 방법, 시스템 내에 펌웨어 형태로 구현하는 방법. 소프트웨어만으로 구현하는 방법, 그리고 하드웨어 장비, 펌웨어 또는 소프트웨어를 함께 이용하여 구현하는 방법이 있다.

5.1 하드웨어를 이용한 구현

성능 모니터링 도구의 구현시 하드웨어 장비를 이용한다는 것은 논리 분석기(logic analyzer), 회로(in-circuit) 에뮬레이터 또는 특수 목적용 VLSI 등과 같은 별도의 하드웨어 장비를 대상 시스템에 접속시키는 것을 의미한다[5, 11, 12, 23]. 이 방법으로 구현된 모니터링 도구는 대상 시스템에 미치는 영향과 모니터링 오버헤드를 최소화할 수 있다. 그러나 이들은 시스템 버스, 메모리 포트 또는 입출력 채널 등과 같은 특정 하드웨어에서 발생하는 전기적 신호를 감지해야 하기 때문에 저수준의 성능 데이타만을 수집할 수 있으며, 수집된 신호를 평균 실행 큐의 길이나 페이징 작업 통계와 같은 일반적 수준의 성능 데이타로 변환하기도 힘들다. 뿐만 아니라, 모니터의 구현 및 설치시 하드웨어에 대한 높은 전문성과 전반적 지식을 필요로 하여 운영 체제 성능 모니터링 도구를 구현하기에는 부적합한 방법이다.

5.2 펌웨어를 이용한 구현

펌웨어를 이용하는 성능 모니터링 도구는 프로세서의 마이크로코드를 수정함으로써 구현된다[5]. 이 방법으로 구현된 모니터링 도구는 측정 단위와 기능이 극히 제한적이며 저수준의 데이타만을 수집할 수 있다. 따라서, 네트워크의 모니터링이나 마이크로코드의 최적화를 위한 주소 목록을 작성하는데 주로 이용될 뿐 운영 체제의 성능 모니터링에는 거의 이용되지 않는다.

5.3 소프트웨어를 이용한 구현

소프트웨어만으로 모니터를 구현하기 위해서는 관련 원시 코드를 수정하여 소프트웨어 프로브를 삽입하여야 한다[4, 5, 14, 23]. 소프트웨어적으로 성능 모니터링 도구를 구현하면 대상 시스템의 실행에 시간과 공간적으로 영향(perturbation)을 주며, 시스템이 정지하면 모니터가 작동될 수 없는 단점이 있다. 그러나, 다양한 성능 메트릭에 대하여 고수준의 성능 데이타를 측정할 수 있고 수집된 성능을 기록하기에도 적합할 뿐만 아니라, 개발과 수정도 용이하기 때문에 데이타베이스와 같은 고수준 소프트웨어나 운영 체제의 성능 측정을 위한 모니터링 도구의 구현[5]에 널리 이용되고 있다.

5.4 합성 구현

하드웨어 장비, 펌웨어 또는 소프트웨어를 함께 이용하여 모니터링 도구를 구현하면 각각의 장점을 활용할 수 있다[5]. 이 방법으로 구현된 대부분의 모니터링 도구는 하드웨어 장비를 이용하여 성능 데이타를 수집하고 수집된 성능 데이타를 소프트웨어적으로 분석한다. 그러나, 하드웨어 장비에서 수집된 성능 데이타는 저수준이기 때문에 이를 소프트웨어로 구현된 도구에서 수집한 것과 유사한 수준의 성능 데이타로 변환하기가 쉽지 않다.

6. 성능 모니터링 도구 소개

시스템 성능 모니터링 도구는 개인용 컴퓨터, 워크스테이션 또는 분산 시스템용으로는 많은 도구들이 발표되고 있으나 다중프로세서 전용 성능 모니터링 도구는 이에 비해 상대적으로 적은 편이다. 본 장에서는 다중프로세서 시스템 성능 모니터링 도구의 예로서 Mtop, TMP, Rowe의 모니터 및 Perfscope를 간략히 소개한다.

6.1 Mtop

Mtop[24]은 다중프로세서 시스템에서 병렬 프로세스의 처리 상태를 파악하기 위한 모니터링 도구로서 시간 구동형 기법을 사용하며 호스트 모델을 따라 소프트웨어로 구현되었다. 이 도구는 시스템을 사용하고 있는 사용자의 수, 평균 작업부하(workload), 이용가능한 실제 메모리의 크기, CPU 이용률, CPU를 사용하는 프로세스에 대한 정보 등을 curses[25]

를 기반으로 한 윈도우 상에 출력한다.

6.2 TMP

TMP[12]는 다중프로세서 시스템을 포함하는 분산 시스템을 위한 성능 모니터링 도구로서 이벤트 구동형 모니터링 기법을 사용하며 생산자/소비자 모델을 따르고 있다. 이 도구의 구현에는 하드웨어와 소프트웨어가 함께 이용되었는데 하드웨어 부분은 시스템 버스를 감시하면서 프로세서에 의하여 생성되는 주소를 감시하고, 각 주소에 관련된 이벤트를 그루핑하며, 각 이벤트에 타임스탬프 정보를 기록하는 등 성능 데이타 수집에 관련된 작업을 수행한다. 수집된 이벤트는 FIFO 메모리에 저장되는데 이때마다 인터럽트가 발생되어 소프트웨어가 성능 데이타를 사용자에게 디스플레이한다.

TMP 하드웨어에서 감시하는 주요 이벤트에는 프로세스의 시작, 종료, 또는 대기 등과 같은 프로세스 상태에 관련된 디스팻처(dispatcher)의 오퍼레이션 이벤트, 메시지의 송신 및 수신에 관련된 통신 활동 이벤트, 그리고 동기화 메카니즘, 입출력 큐의 초기화 또는 통신 서브시스템 등에 관련된 커널 이벤트 등이 있다. TMP 소프트웨어에서 디스플레이하는 주요 성능 메트릭은 프로세스에 관련된 것과 시스템에 관련된 것이 있다. 프로세스에 관련된 메트릭에는 경과 시간, 준비(ready) 큐에서 대기한 시간, 각종 대기(wait) 조건에 의하여 블럭되었던 시간 등이 있고, 시스템에 대한 메트릭에는 CPU 시간, 스케쥴러 큐의 길이, 송수신된 메시지의 수, 그리고 프로세스가 시스템 모드 및 사용자 모드에서 동작한 시간 등이 있다.

6.3 Rowe의 모니터

Rowe의 모니터[23]는 추적 구동형 기법을 사용하고, 생산자/소비자 모델을 기반으로 하고 있으며, 하드웨어와 소프트웨어를 함께 이용하여 구현되었다. 이 모니터는 호스트에 의하여 원격으로 제어되는 논리 분석기를 하나의 대상 프로세서에 연결시킨 뒤 성능을 측정하여 지속적으로 수집하여 호스트에 전달하고, 정보 수집이 완료되면 다음 프로세서에 연결시켜 성능을 측정하는 과정을 순차적으로 반복한다. 호스트에서 동작하는 소프트웨어는 전달받은 성능 데이타를 분석하여 각 프로세서의 이용률을 전체 시간의 퍼센트로 변환하여 히스토그램으로 표현한다.

6.4 Perfscope

Perfscope[2, 22]는 에이전트/매스터 모델을 기반으로 하며 소프트웨어적으로 구현되었다. 이 도구는 그림 5에서 보는 바와 같이 함수 구동형 기법과 시간 구동형 기법을 함께 사용한다. 여기서, 프로버는 함수의 시작점과 종료점에 트랩을 발생시키기 위한 프로브를 삽입하거나 삭제하고, 로거는 트랩이 발생하였을 때, 함수의 실행에 관련된 데이타를 로그에 저장하며, 수집기는 사용자 레벨에서 동작하면서 특정 주기마다 타이머에 의해 호출(invoke)되어 성능에 관련된 커널 자료 구조와 로그를 읽어 대상 시스템의 하드 디스크 또는 네트워크 소켓으로 플러쉬시킨다.

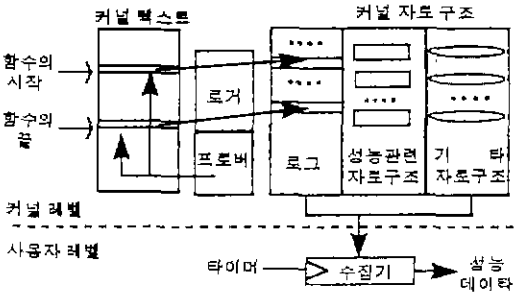

그림 5 Perfscope의 모니터링 모델

이 도구는 시간 구동형 기법과 함수 구동형 기법을 혼합하여 대상 시스템의 커널 자료 구조로부터 프로세스, CPU 이용률, 시스템 통계, 디스크 이용률 및 메모리 이용률 등과 같은 운영 체제의 성능 통계 데이타뿐만 아니라, 메모리, 프로세스, 시스템 콜, 화일 시스템, 락과 세마포어 및 디바이스 드라이버 등과 같은 상위 레벨 언어로 작성된 모든 운영 체제 함수의 실행에 소요된 시간을 측정할 수 있다. 또한, 측정된 성능 데이타는 분석을 거쳐 Motif를 기반으로 하는 윈도우 시스템 상에 다중 윈도우와 다양한 형태의 그래픽으로 표현된다.

7. 결 론

본 고에서는 다중프로세서 운영 체제의 성능을 모니터링 하기 위한 도구가 갖추어야 할 기본적인 요건들을 정의하였으며, 성능 측정에 이용될 수 있는 성능 모니터링 기법, 성능 모니터링 도구의 모델, 그리고 도구 구현 방법을 기술하고 각각의 장단점을 비교 분석하였다. 또한, 성능 모니터링 도구의 예로서 Mtop, TMP, Rowe의 모니터 및 Perfscope를 간략히 소개하였다.

다중프로세서 시스템의 하드웨어를 개발할 때에는 병렬성을 최대한으로 보장할 수 있도록 메모리, 캐쉬 및 상호연결(interconnection) 네트워크 등의 성능을 측정함으로써 시스템의 성능을 개선할 수 있다. 이 경우에는 추적 구동형 기법과 이벤트 구동형 기법이 유용하며, 도구의 구현시에는 하드웨어, 펌웨어 또는 이들과 소프트웨어의 합성 구현이 적합할 것으로 사료된다. 그러나, 일단 개발이 완료된 시스템의 성능을 개선하기 위해서는 다중프로세서 시스템의 성능 발휘 정도에 대한 통계 데이타와 운영 체제의 각 함수가 실행되는데 소요되는 시간을 측정하여야 한다. 이 경우에는 시간 구동형 기법, 함수 구동형 기법 또는 이들을 혼합한 혼합 기법이 유용하고, 도구는 소프트웨어적으로 구현되는 것이 적합할 것이다.

다중프로세서 시스템 운영 체제의 성능을 측정에 있어서 계속적으로 연구되어야 할 분야는 우선, 대상 시스템에 대한 영향을 최소화하는 방안일 것이다. 하드웨어나 펌웨어 또는 합성 형태로 구현된 성능 모니터들은 대부분이 저급의 데이타를 수집할 수 밖에 없지만 대상 시스템에 미치는 영향이 매우 적다. 반면, 운영 체제의 성능 모니터링 도구의 구현에 많이 이용되는 소프트웨어적 구현 방법은 대상 시스템의 실행에 많은 영향을 줄 수 밖에 없으므로 이것을 최소화 하기 위한 방안들이 연구되어야 한다. 또한, 측정된 성능 데이타를 왜곡(distortion)없이 정확하게 분석할 수 있도록 여러 다중프로세서 시스템에 공통적으로 적용될 수 있는 정형화된 다양한 기법들이 제시되어야 할 것이다. 특히, 제안되는 분석 기법들은 성능 데이타들 간의 상관 관계를 다중프로세서 측면에서 적절히 조명할 수도 있어야 할 것이다. 마지막으로, 분석된 성능 정보들을 사용자가 보다 잘 이해할 수 있는 형태로 브라우즈하기 위한 공학적 방법론 등이 제안되어야 할 것이다.

참고문헌

[1] M. Loukides, System Performance Tuning, O'Reilly & Associates, Inc., 1991.

[2] 김성조, 다중프로세서 시스템 성능의 동적 측정 및 분석 도구에 관한 연구, 중간보고서, 중앙대학교, 한국학술진흥재단, 1995.

[3] T. Bemmerl, "Distributed Monitoring Systems-a Basis for General Purpose Distributed Multiprocessors," Panel Session, Proc. of the Int'l Conf. on Distributed Computing Systems, May 1991, pp. 380-380.

[4] P. C. Bates, "Effective Instrumentation is the Key to Effective Monitoring," Panel Session, Proc. of the Int'l Conf. on Distributed Computing Systems, May 1991, pp. 379-379.

[5] R. Jain, The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling, John Wiley & Sons, Inc., 1991.

[6] A. Sheehan, Performance Analysis : An Introduction, Technical Report 89.3, Part No. 19150, Mar 1989.

[7] A. Mink, et al., "Multiprocessor Performance - Measurement Instrumentation," IEEE Computer, Vol. 23, No. 9, Sep. 1990, pp. 63-75.

[8] H. C. Lucas, Jr., "Performance Evaluation and Monitoring," ACM Computing Surveys, Vol. No. 2, June, 1971, pp. 79-91.

[9] H. Davis, et al., "Multiprocessor Simulation and Tracing Using Tango," Proc. of the Int'l Conf. on Parallel Processing, Vol. II, 1991, pp. 99-107.

[10] G. Lomow, et al., "Monitoring Distributed Systems," ACM Trans. on Computer

Systems, Vol. 5, No. 2, May 1987, pp. 121-150.

[11] J. E. Lumpp, Jr., et al., "Specification and Identification of Events for Debugging and Performance Monitoring of Distributed Multiprocessor Systems," Proc. of the 10th Int'l Conf. on Distributed Computing Systems, May 1990, pp. 476-483.

[12] D. Haban and D. Wybranietz, "A Hybrid Monitor for Behavior and Performance Analysis of Distributed Systems," IEEE Trans. on Software Engineering, Vol. 16, No. 2, Feb. 1990, p. 197-211.

[13] D. M. Ogle, et al., "Application-Dependent Dynamic Monitoring of Distributed Systems," IEEE Trans. on Parallel and Distributed Systems, Vol. 4, No. 7, Jul. 1993, pp. 762-778.

[14] A. Poursepanj, "The PowerPC Performance Modeling Methodology," Communications of the ACM, Vol. 37, No. 6, June 1994, pp. 47-55.

[15] K. Beck, et al., "Integrating Profiling into Debugging," Proc. of the Int'l Conf. on Parallel Processing, Vol. II, 1991, pp. 284-285.

[16] R. Hoffmann, et al., "Distributed Performance Monitoring : Methods, Tools, and Applications," IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 6, June 1994, pp. 585-598.

[17] D. Haban and D. Wybranietz, "A Hybrid Monitor for Behavior and Performance Analysis of Distributed Systems," IEEE Trans. on Software Engineering, Vol. 16, No. 2, Feb. 1990, pp. 197-211.

[18] D. A. Reed, "Performance Monitoring : Fact and Fancy," Panel Session, Proc. of the Int'l Conf. on Distributed Computing Systems, May 1991, pp. 382-382.

[19] J. J. P. Tsai, et al., "A Noninterference Monitoring and Replay Mechanism for Real-Time Software Testing and Debugging," IEEE Trans. on Software Engineering, Vol. 15, No. 12, Dec. 1989, pp. 1615-1629.

[20] J. J. P. Tsai, et al., "A Noninvasive Architecture to Monitor Real-Time Distributed Systems," IEEE Computer, Mar. 1990, pp. 11-23.

[21] C. Q. Yang and B. P. Miller, "Performance Measurement for Parallel and Distributed Programs : A Structured and Automated Approach," IEEE Trans. on Software Engineering, Vol. 15, No. 12, Dec. 1989, pp. 1615-1629.

[22] 박상서, 김성조, "혼합 기법을 이용한 다중프로세서 Unix 성능 모니터링 도구," 한국정보과학회논문지(A), 제22권 9호, 1995, pp. 1337-1351.

[23] P. K. Rowe, et al., "A Multiprocessor Performance Measurement Tool," Proc. of the USENIX Summer Conf., June 1985, pp. 421-432.

[24] 김정녀 외, "UNIX SVR4 MP 환경 하에서 다중처리 기능 검사 도구," '93 가을 정보과학회' 학술발표논문집, 제20권 2호, 1993, pp. 539-542.

[25] Goodheart, B. 1991. UNIX Curses Explained. Prentice-Hall, Englewood Cliffs, N.J.

박 상 서

1991 중앙대학교 전자계산학과 공학사.
1993 중앙대학교 전자계산학과 공학석사.
1995.12 중앙대학교 컴퓨터공학과 박사과정수료.
1996.1~ 국방정보체계연구소.
관심분야 : 병렬 및 다중처리, 운영체제, 디버깅, 성능평가, 시스템 관리, CALS임.

김 성 조

1975 서울대학교 응용수학과 공학사.
1977 한국과학기술원 전산과 이학석사.
1977~1980 ADD(연구원).
1980~현재 중앙대학교 컴퓨터공학과 부교수.
1987 Univ. of Texas at Austin 이학박사.
1987~1988 Univ. of Texas at Austin(Research Fellow).
관심분야 : 병렬 및 다중처리, 디버깅, 시스템 망관리, 멀티미디어임.

KOREA-JAPAN Joint Workshop on Algoritm and Computation

- 일 자 : 1996년 8월 23~24일
- 장 소 : 한국과학기술원 전산학과
- 관련분야 : Automata, Languages, Computability
  Combinatorial/Graph/Geometric/Randomized Algorithms
  VLSI/Parallel Algorithms, Networks/Distributed Algorithms
  Theory of Learning/Robotics, Number Theory/Cryptography
  Graph Drawing, Computational Logic
- 주 최 : 한국정보과학회 컴퓨터이론 연구회
  위원장 장직현 교수(서강대 전산학과)
  jchang@alglab.sogang.ac.kr
- 문 의 : 신찬수(KAIST 전산학과)
  cssin@jupiter.kaist.ac.kr
  Tel : 042-869-3553 Fax : 042-869-3510