프로그램 성능조율 대안을 평가하는 방법론
(A Methodolgy to Evaluate Program Tuning Alternatives)

엄 현 상 †
(Hyeonsang Eom)

요 약 본 논문에서는 프로그램의 성능을 향상시키기 위하여 프로그래머들이 다른 성능조율 대안들을 평가하는 것을 돕는 새로운 평가 방법론을 소개한다. 이 방법론은 조율 대안이 채택된 경우의 성능을 평가할 수 있게 한다. 구체적으로 말하자면, 성능 병목 지점의 확인을 위하여 프로그램 구성 요소들에서 소요되는 시간을 수량화하는 전통적인 성능 평가 방법론과는 대조적으로 분산 또는 병렬 프로그램이 처리하는 일의 이동 이후의 성능을 예측한다. 따라서, 이 방법론은 일의 처리 장소를 변경함으로써 성능을 향상시키는 것에 대한 가이드라인을 제공한다. 이 방법론을 사용하면 기반 네트워크 변경에 따른 성능에 대한 파급효과도 예측할 수 있다. 이 방법론은 조율할 프로그램이 실행되는 동안 점진적으로 그리고 온라인으로 성능을 평가할 수 있다. 본 논문에서는 이 방법론을 구현한 후 사용했을 때 여섯 프로그램들의 검사 집합에 대하여 다른 조율 대안들의 성능을 정확히 예측할 수 있다는 실험 검증 결과를 보인다.

키워드 : 성능조율 대안 평가, 측정, 예측, 도구, 온라인 분석, 분산 및 병렬 계산

Abstract We introduce a new performance evaluation methodology that helps programmers evaluate different tuning alternatives in order to improve program performance. This methodology permits measuring performance implications of using tuning alternatives. Specifically, the methodology predicts performance after workload migration for a distributed or parallel program in contrast to traditional performance methodlogies that quantify time spent in program components for bottleneck identification. The methodology thus provides guidance on workload migration. The methodology also permits predicting the performance impact of changing the underlying network. The methodology may evaluate performance incrementally and online during the execution of the program to be tuned. We show that our methodology, when it is implemented and used, permits accurately predicting the performance of different tuning alternatives for a test suite of six programs.

Key words : Evaulation of Performance Tuning Alternatives, Measurements, Prediction, Tools, Online Analysis, Distributed and Parallel Computing

1. 서 론

프로그램을 성공적으로 (성능) 조율하기 위해서는 성능 병목 현상의 원인을 파악하여야 하고 그 해결책을 제안하고 구현하여야 한다. 마지막으로, 조율한 프로그램에서 문제가 해결되었다는 것을 증명하기 위하여 성능을 다시 측정하여야 한다. 이 과정 각각의 단계는 어렵고 시간이 많이 걸리는 작업이다. 따라서, 성능 결함 제거 도구는 조율 작업을 하는 프로그래머를 돕기 위하여 사용하게 된다. 그러나, 성능 도구에 관한 대부분의 연구는 병목 확인/파악 단계에 초점을 맞추어 왔다. 이것은 중요한 단계이나 단지 첫번째 단계이다. 본 논문에서는 그 다음 단계에서 조율 대안들 중 최선의 것을 선택할 수 있도록 가이드라인을 제공하는 기술을 제시하고자 한다.

성능 저하의 원인을 발견한 후에는 조율 대안들을 파악하여야 한다. 빈번이, 데이타의 분해 방법의 변경, 프로세서에 대한 프로세스 지정의 변경, 또는 계산 혹은 통신 자원의 변경 등을 시도하는 서로 다른 여러가지 방법들을 대안들으로서 고려한다. 그러나, 각각의 이 대안들은 그 구현을 위하여 프로그램을 변경하고 해당 결함을 제거한 다음, 재실행하는 현저한 노력이 필요할 수 있다. 프로그래머가 이러한 노력을 기울이면서 코드 한 줄한줄을 변경하기 전에 여러가지 조율 대안들의 잠재적인 파급효과들을 평가하는 것을 지원하는 성능 도구들이 필요하다.

성능 도구가 조율 대안들의 잠재적인 혜택에 대한 정

† 정 회 원 : 서울대학교 컴퓨터공학부 교수
hseom@cse.snu.ac.kr
논문접수 : 2006년 7월 10일
심사완료 : 2006년 10월 19일

보를 제공하는 방법은 다양하다. 먼저 도구는 소스코드 분석을 바탕으로 변경된 프로그램의 성능을 정적 예측하는 방법을 사용할 수 있다. 그러나, 이 접근방법은, 해당 예측 결과가 프로그램의 실제 동작에 대한 중요한 정보를 제공하는 동적 (실행) 정보를 반영하지 않는 문제가 있다. 두번째 접근방법은, 그 동적 동작을 측정하기 위하여 프로그램에 측정을 위한 코드를 추가하여 변경한다(Instrumentation). 이 코드의 실행을 통하여 실행 중 얻은 프로그램 동적 동작에 대한 데이타를 사용함으로써 조율 대안의 성능을 오프라인으로 예측한다. 이 접근방법은 상당한 양의 데이타 수집이 필요할 수 있다. 이 대신에 본 논문에서 제시하는 평가 방법론은 프로그램 현 버전을 실행하고, 이 실행을 온라인으로 측정하면서 동시에 다른 조율 대안의 영향을 예측하는 제삼의 접근방법을 사용한다. 이 아이디어는 프로그램에 대하여 조율 대안으로서 제안된 변화를 모의실험(시뮬레이션)하면서 동시에 원 프로그램을 실행하는 것이다. 이 기술은 컴퓨터 구조 분야에서 여러가지 변경 부분들을 모의실험하는데 성공적으로 사용되고 있다[1]. 시스템 대부분을 직접적인 실행하면서 수정된 부분들에 대해서만 모의실험하는 것은 전체 프로그램을 모의실험하는 것보다 빠르게 수행할 수 있다. 그리고, 이러한 직접적인 실행은 프로그램의 동적 행동을 고려할 수 있게 해주며 제한적인 모의실험은 성능을 예측하기 위하여 수집하고 처리되어야 하는 데이타의 양을 감소시키는 장점이 있다.

순차적 프로그램과는 다르게 분산 또는 병렬 프로그램에서는, 계산이 어떻게 수행되는가 뿐만 아니라 그것이 어디에서 수행되는가에 대한 조율이 가능하다. 예를 들면, 생산/소비자의 파이프라인 프로세스는 데이타의 친화성(Affinity)을 가진다. 데이타는 정적(디스크 파일인 경우)이거나 동적(생산 프로세스인 경우)일 수 있다. 일을 균형적으로 분담하거나(Load Balancing), 데이타의 친화성을 높이기 위하여 계산의 일부분을 한 프로세서에서 다른 프로세서로 이동하는 것이 더 생산적일 수 있다. 이러한 이동은 적절히 정의된 분산 또는 병렬 계산 입자크기(Granularity)에 대하여 효과적일 수 있다. 예를 들어, 일의 이동은 프로세스들, 큐어리들, 또는 프로시저들 수준에서 고려될 수 있다. 본 논문에서는 어떻게 결과가 계산되어지는가를 변경하는 것보다 어디서 계산이 수행되는가를 변경하는 것에 관련된 "이렇게 변경하면 어떨까"라는 질문의 답을 제공하는 방법론을 제시하고자 한다. 구체적으로 말하자면, 프로세서들 간에 프로세스 상의 계산을 이동하는 파급효과에 대한 피드백을 프로그래머들에게 제공하는 새로운 방법론 Process-level Performance Evaluation Methodology(Ps-PEM)를 소개한다. 덧붙여, 기반 네트워크 변경에 따른 성능 차이를 예측하게 하는, Networking Performance Evaluation Methodology(N-PEM)라고 불리는 Ps-PEM의 변형된 버전도 소개한다.

이 논문에서는 Ps-PEM을 제시하고 여러 병렬 프로그램들에 대하여 이 방법론을 실험적으로 검증한 결과를 보여준다. 2장에서 Ps-PRM을 소개하고 이 방법론의 구현에 대하여 설명하며 대안 성능 예측의 정확도를 수량적으로 보여준다. 3장에서는 네트워크 인프라의 성능 변화에 따른 응용 실행 시간의 변화를 예측하기 위하여 Ps-PEM을 네트워크에 대하여 변형한 형태인 N-PEM을 소개하고 그 사용법을 설명한다. 4장에서는 관련된 연구를 소개한다. 마지막으로, 5장에서는 본 연구를 요약하고 그 미래 추진 방향을 간략하게 제시한다.

2. Process-level Performance Evaluation Methodology(Ps-PEM)

Process-level Program Performance Evaluation Methodology(Ps-PEM)는 분산 또는 병렬 실행 환경에서 프로세스들을 프로세서들에 지정하는 것을 변화시키는데 대한 성능 파급효과를 예측함으로써 프로세스 이동의 성능에 대한 영향을 평가한다. 이 방법론의 목적은 프로세스 배치가 변경되었다는 가정 하에 실행 시간의 잠재적인 향상을 계산하는 것이다. 이 기술은 분산 또는 병렬 프로그램을 더 큰 수의 프로세서들 상에서 실행했을때 그 성능을 예측하기 위하여 사용할 수 있다.

해당 잠재적인 향상의 정도를 평가하기 위하여 현 프로세스 배치에서 실행하는 동안 가상적인 배치에서의 실행 시간을 예측하게 된다. 이 접근방법에서는, 목표 구성(여기서는 배치) 하에서 이벤트들의 실행을 모의실험하는 중앙 감시소(Cental Monitor)에 각 메세지 전달에 필요한 동작들에 해당하는 이벤트들에 대한 데이타를 전송하도록 응용 프로세스에 코드를 추가하여 변경한다(Instrumentation).

목표 배치 상에, 한 노드에서 한 CPU에 대하여 경쟁하는 한개 이상의 프로세스가 있을 수 있기 때문에 정확한 예측을 위하여 프로세스들을 스케줄하는 실제적인 정책을 채택하여 사용해야 한다. Ps-PEM은 운영체제가 정해진 시간 할당량(Quantum) 동안 한 프로세서에 하나의 비대기(Non-Waiting) 프로세스를 스케줄하고 그 다음 할당량 시간에 다음 비대기 프로세스로 전환하는 공정한 순환(Round-Robin) 스케줄 정책을 사용하는 것을 가정한다. Ps-PEM 에서 계산을 빠르게 수행하기 위하여 각각의 시간 할당량에 대하여 모의실험을 하지 않는다. 각 시간 간격에 대하여 비차단(Non-Blocking) 프로세스는, 시간 할당량을 효과적으로 무한히 작게 만

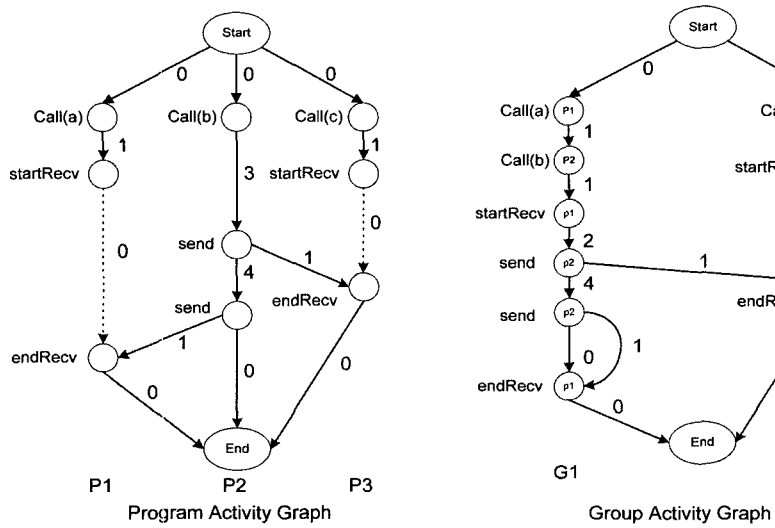

그림 1 PAG에서 GAG로의 변환

들면서 프로세서를 동등하게 공유하게 한다.

Ps-PEM 기반 성능 예측 알고리즘을 설명하기 전에, Ps-PEM 을 설명하기 위한 몇가지 용어들을 정의하면 다음과 같다.

이벤트(Event): 프로세스에 의하여 수행되는 관찰 가능한 동작. 프로세스는 다른 프로세스들과 메세지를 교환하며 통신한다. 메세지 전달은 send, startRecv, 및 endRecv 이벤트들을 발생시킨다. 메세지 이벤트들은 프로세스들 간에 "짝"을 이룬다. 예를 들어, 한 프로세스 상의 send 이벤트는 다른 프로세스 상의 해당 endRecv 이벤트와 정확히 짝을 이룬다.[1)]

프로세스 시간(Process Time): 프로세서 상에서 프로세스가 실행되면서 메세지를 기다리지 않을 때 작동하는 매 프로세스 시계의 시간.

프로그램 활동 그래프(Program Activity Graph, PAG): 하나의 프로그램 실행 중 발생하는 이벤트들의 그래프. 이 그래프의 노드들은 프로그램의 실행 상에서의 이벤트들을 나타낸다. 화살표는 프로세스 내에서의 이벤트 순서나 프로세스들 간의 통신 이벤트 의존성을 나타낸다. 각 화살표는 이벤트들 사이의 프로세스 시간이나 프로세스 간 통신시간 라벨이 붙는다. 그림 1의 왼쪽 반은 세 프로세스들 상에서 수행되는 병렬 프로그램의 간단한 PAG를 보여준다.

이전-발생(Happen-Before): 각 프로세스 상에서의 연속적인 근거리(Local) 이벤트들과 통신 동작들을 의미하는 이벤트들의 이행성(Transitive) 부분 순서. 근거리 이벤트들에 대하여 한 이벤트가 다른 이벤트에 비하여 그 프로세스 상에서의 프로그램 실행 중 먼저 일어났다면 그 이벤트는 다른 것보다 이전에 발생한 것이다. 원거리(Remote) 이벤트들에 대하여send 이벤트는 해당 endRecv에 비하여 이전에 발생하였다. 형식화하자면, 이전-발생은 Lamport의 이전-발생 관계가 의미하는 이벤트들 간의 선행 관계들의 집합이다[2].

결정적 경로(Critical Path, CP): PAG 상에서 최장 시간의 프로세스 /통신 시간이 부하된 경로. 전체 프로그램의 실행에 대하여 해당 CP의 길이는 각 프로세서에서 하나의 프로세스가 동작할 때의 그 프로그램의 실행 시간을 나타낸다.

프로세스 그룹(Process Group): 예측(목표) 구성에서 하나의 프로세서 상에서 동작하는 프로세스들의 집합.

그룹 시간(Group Time): 각 그룹의 모든 프로세스들이 한 프로세서 상에서 실행될 때 작동하는 그룹별 시계의 시간.

그룹 활동 그래프(Group Activity Graph, GAG): 한 프로그램의 실행 중 발생하는 이벤트들의 그래프. 그래프 노드들은 프로그램 실행 상의 이벤트들을 나타낸다. 화살표들은 하나의 그룹 내에서 발생하는 이벤트들의 순서나 그룹들 간의 통신 의존성을 나타낸다. 근거리 화살표에는 이벤트들 간의 그룹 시간 라벨이 붙는다. 그림 1의 오른쪽 반은, 왼쪽 반이 보여주는 세개의 프로세스들을 갖는 병렬 프로그램이 두개의 목표 그룹들을 갖는 경우에 해당하는 간단한 GAG를 보여준다. GAG는 사실상 하나의 "가상" 프로세스 안

1) 이 정의는 Lock 및 Barrier 같은 다른 동기성 또는 통신 이벤트들을 포괄하도록 쉽게 확장할 수 있다.

으로 하나의 해당 프로세스 그룹으로부터의 이벤트들이 수집되는, 모든 이벤트들로 이루어진 PAG이다.

최초 가능 시간(Earliest Possible Time, EPT): 목표 그룹 내에서 이벤트가 발생할 수 있는, 그룹 시간으로 측정된 가장 이른 시간. EPT는 각 그룹에 하나만의 프로세스가 있을 때 프로세스 시간과 동일하다.

목표 구성 상에서의 실행 시간을 계산하기 위하여, Ps-PEM에서는 그룹 활동 그래프 GAG를 만들고 그 다음에 GAG 상에서 가장 긴 경로의 길이를 계산한다. 본 논문에서는 이 과정을 명료하게 소개하기 위하여 사후(Postmortem) 처리 방법으로 PAG를 GAG로 전환하는 과정을 먼저 설명한다. 그 다음에 응용이 실행되는 동안 온라인으로 GAG를 만드는 실제 알고리즘의 세부사항들을 설명한다.

프로그램의 실행에 대하여 목표하는 프로세스 그룹화 방법이 주어졌을 때 해당 GAG는 한 그룹 내에 포함될 PAG 프로세스들의 구성요소들을 GAG의 한 "프로세스"로 합쳐 넣음으로써 해당 PAG로부터 만들 수 있다. PAG 로부터의 각 이벤트는 그룹 시간 순서로 GAG내에 배치된다. 동일 그룹 내의 인접 이벤트들 간의 화살표는 그들 사이에 경과된 시간 라벨이 붙는다.

그림 1은 PAG와 해당 GAG를 예시한다. GAG내의 화살표의 라벨 크기는 목표 그룹화 효과를 나타낸다. 그룹 1(G1) 내 startRecv의 최초 가능 시간(EPT)은 프로세스 1(P1)과 2가 한 프로세서를 그 startRecv까지 공유하기 때문에 2이다. 그룹 1 내 endRecv의 EPT는 프로세스 2가 그룹 프로세서에서 동작하고 해당 send 이벤트가 그 endRecv에 선행해야 하기 때문에 8이다. 그룹 2 내 startRecv와 endRecv의 EPT들은 그 그룹에 오직 하나의 프로세스가 있기 때문에 모두 1이다. GAG 내 endRecv들은 EPT 계산의 두 극단적인 경우들을 보여준다: 그룹 1 안의 endRecv의 EPT는 해당 send 이벤트의 것과 동일하다. 그룹 2 안의 endRecv의 EPT는 그 startRecv의 것과 같다. 각 메세지 수신의 예측 실행 시간은 endRecv 이벤트의 EPT와, send 이벤트의 EPT 및 메세지의 "비행" 시간 합의 최대값이다. Ps-PEM에서는 목표 그룹 내 프로세스들 간의 메모리에 대한 경합을 고려하지 않기 때문에 프로세스 이동 후 실행 시간의 근사치를 나타낸다. 알고리즘의 자세한 설명은 2장 2절에서 찾을 수 있다.

Ps-PEM에서의 성능 예측을 위한 계산 수행에 해당하는 오프라인 알고리즘은 먼저 PAG를 만들고, 그것을 해당 GAG로 전환한 다음, 그 GAG 상에서, EPT 및 통신시간 경로를 따라 가장 긴 CP를 계산할 것이다. PAG 상에서의 노드 갯수는 프로그램이 실행되는 동안의 이벤트 갯수와 같기 때문에 전체 그래프 만들기, 전환, 및 계산은 긴 시간 동안 실행되는 프로그램들에 대하여 과도한 오버헤드를 야기할 수 있다. 그 대신, 점진적으로 PAG를 만들고 GAG로 전환하며 계산하는 온라인 알고리즘을 채용하는 Ps-PEM을 개발하였다. 이 온라인 알고리즘은 현재 처리되는 GAG의 일부분만을 유지할 수 있게 한다. 점진적으로 필요한 GAG 일부만을 유지하기 위하여 Kimelman과 Zernak에 의하여 개발된 즉석 위상 정렬(On-the-Fly Topological Sort) 알고리즘을 적용시켰다[3]. 이와 같이 개발한 알고리즘은 프로세스들의 목표 그룹 상의 실제 실행을 모의실험한다. 프로그램이 실행되는 동안 목표 구성의 상의 실행 시간을 예측하기 위해서는 온라인 결정 경로 알고리즘을 사용한다[4].

목표 그룹화 방법이 주어질 때 점진적으로 GAG를 만들기 위하여 그룹화 이후 해당 이벤트들의 순서를 결정해야만 한다. 위상 정렬(Topological Sort)에서와 같이 이전-발생 관계에서 정해지는 순서에 의하여 모든 이벤트들을 처리할 수 있도록 이벤트들을 선택함으로써 다음에 처리할 이벤트를 결정하여야 한다. 이전-발생 관계에 의하여 정렬할 수 없는 이벤트들은 각 그룹의 프로세스들을 한 프로세서에 순환적으로 스케줄하는 방법을 바탕으로 정렬한다.

Ps-PEM에서는 GAG 에 추가할 다음 이벤트를 선택하는 것에 덧붙여 선택한 이벤트로의 화살표에 올바른 라벨 크기도 지정하여야 한다. 그룹 간 화살표에 대하여 PAG에서 얻어지는 통신시간이 사용된다. 근거리(같은 그룹) 이벤트들 간의 화살표 부하 중량 계산은 좀더 복잡하다; 그 중량은 해당 그룹 내에서 추가한 마지막 이벤트와 현재 선택하여 처리 중인 이벤트 사이에, 모든 비차단 응용 프로세스에 의하여 행해지는 일의 양을 합친 크기 또는 대기 시간의 크기로 정한다.

2.1 Ps-PEM 기반 성능 평가 도구 동작 구조

Ps-PEM를 기반으로하는 성능 평가 도구는 그림 2가 보여 주는 구조로 동작한다. 응용 프로세스들로부터 프

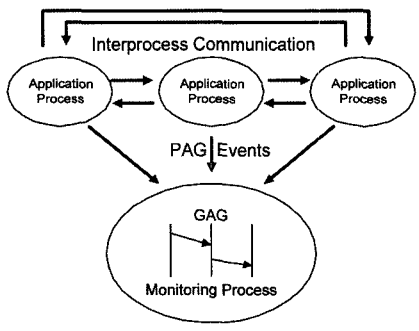

그림 2 Ps-PEM 기반 성능 평가 도구 동작 구조

로세스 간 통신 동작에 해당하는 이벤트들이 PAG 이벤트들로서 중앙 감시(Central Monitoring) 프로세스에 전송된다. 이곳에서 주어진 목표 구성에 대한 해당 GAG가 점진적으로 만들어지면서 해당 성능을 효율적으로 예측할 수 있게 된다.

2.2 알고리즘

본 절에서는 Ps-PEM에서 성능 예측을 위하여 사용하는 알고리즘을 자세하게 제시하고자 한다. 응용 프로세스들로부터 전송되는 일련의 프로그램 이벤트들(즉, PAG 이벤트들)을 어떻게 GAG로 변환하는가를 설명한다. 이 변환 과정을 직관적으로 설명하면, PAG 상에서 시간 흐름의 방향과 직각으로 줄자를 위치시켜 시간 흐름 방향으로 줄자를 이동하면서 만나게 되는 이벤트들을 GAG 상의 해당 그룹에 위치시키는 것이다. 변환 결과로서 얻어지는 GAG를 통한 가장 긴 경로의 길이를 계산함으로써 제안된 그룹화 후의 실행 시간을 계산한다. 이것을 위하여 이벤트들은 응용 프로세스들로부터 보내지고 GAG에 추가될 때까지 해당 정보가 유지된다. 이벤트들이 더이상 필요하지 않을 때 해당 정보가 제거된다. 한 이벤트가 처리되는 동안 다음 네 상태 중 하나를 갖는다:

대기(Queued): 이벤트가 중앙 감시 프로세스에 도착했지만 그 바로 전 이벤트가 아직 보고되지 않았으면 그 이벤트는 대기중이다.

현재(Current): 현재 이벤트는 처리 후보이다. 프로세스 당 많아야 하나의 현재 이벤트가 있을 수 있다.

미결정(Pending): 미결정 이벤트는 해당 send 이벤트가 처리되는 것을 기다리는 endRecv 이벤트를 의미한다.

보고완료(Reported): 이벤트 처리가 완료되고 그룹 활동 그래프(GAG)에 추가될 때 이벤트는 보고된다. 해당 DAG 자료 구조는 그것의 근거리 및 장거리 후임자(Successor)들이 모두 보고되면 자유화(Free)된다.

각 이벤트는 중앙 감시 프로세스에 도착한 후 EventArrival (그림 3의 19-44 줄) 함수에 의하여 처리된다. EventArrival 과정에서, PAG에 새로운 이벤트가 추가될 때 그 이벤트의 초기 상태는 그 전임자(Predecessor) 이벤트의 상태를 바탕으로 결정된다. 이벤트의 상태는 다음 두가지 경우에 변경된다: 도착했을 때와 전임자 이벤트가 보고될 때이다. 이벤트의 모든 전임자 이벤트들이 보고되면 이벤트는 현재 상태가 된다(그림 3의 4줄). endRecv 이벤트들만이 두개의 전임자 이벤트들을 갖으며 각 프로세스들로부터 이벤트들은 FIFO 순서로 도착하기 때문에 endRecv 이벤트들만이 미결정으로 표시될 수 있다(그림 3의 2줄).

처리를 위하여 최초의 현재 이벤트가 선택된다. 다수의 현재 이벤트들 중에서 선택하기 위하여 *최초 이벤트 시간*(Earliest Event Time)을 사용한다(그림 3의 14-18 줄). 한 이벤트의 최초 이벤트 시간은 그것을 현재 이벤트로서 선택할 때의 이벤트 시간이다. 선택한 이벤트를 발생시킨 프로세스가 속한 그룹 내의 다른 모든 현재 및 미결정 이벤트들의 procTime과 waitTime이 수정되고, 그 이벤트 그룹의 groupTime이 소비된 프로세스 시간 또는 해당 waitTime만큼 증가된다(그림 3의 28-32 및 39-43줄). 해당 그룹의 다른 각각의 비차단(Non-Blocking) 프로세스의 procTime이, 목표 구성에서 그 현재 이벤트와 그 그룹에서 바로 전에 보고된 이벤트들 간에 실행되었을 시간의 크기를 모의실험하기 위하여 변경된다(그림 3의 31 및 42줄). 선택한 이벤트가 비차단(Non-Blocking) 이벤트인 경우에 해당 그룹 내 각 차단(Blocking) 프로세스에 대하여는 그 waitTime이 그 그룹에서 동작 가능한 이벤트들에 의하여 사용되는 총 프로세스 시간만큼 감소된다(그림 3의 30줄). 이와 같이, waitTime 필드는 GAG에 마지막 이벤트가 추가되었을 때부터 그룹에 의하여 소비되는 프로세스 시간을 반영한다.

정확한 예측을 위하여 예측 시간을 계산할 때 통신비용을 고려하는 것이 필요하다. 통신비용은 프로세스 끝단에서의 프로토콜 처리 시간과 메세지의 비행시간에서 비롯된다. 한 프로세스에 프로토콜 처리가 국한된다면 직접 측정이 쉽다. 그러나, 시계 동기화 문제 때문에 메세지 비행 시간을 정확히 측정하는 것이 일반적으로 불가능하다. 이 문제를 해결하기 위하여 메세지 바이트 수와 메세지가 근거리 (같은 프로세서) 또는 원거리인가를 바탕으로한 통신시간 참조표를 사용한다. 이 표의 값들은 다른 크기의 메세지들에 대한 왕복시간의 절반을 측정함으로써 오프라인으로(응용 실행 전에) 결정된다. 한 프로세스가 오랫동안 이벤트를 생성하지 않으면 (다른 프로세스들과 통신하지 않으면) 제시한 알고리즘에서는 다른 프로세스들에서 현재 이벤트들을 처리할 수 없다. 이것을 방지하기 위하여 추가 생명유지(Keep-Alive) 이벤트들을 만들기 위한 목적으로 각 응용 프로세스에서 주기적 알람을 사용한다. 이와 같은 추가 생명유지 이벤트들은 정상적인 이벤트들로 간주되어 목표 그룹의 그룹시간이 증가될 수 있도록 만든다. 하지만 이러한 이벤트(이와같이 증가된 차이)는 GAG에 추가되지 않고 폐기된다.

2.3 Ps-PEM의 실험적 검증

Paradyn 성능 측정 도구들을 기반으로 Ps-PEM을 구현했다[5]. Paradyn은 실행되는 프로그램에 코드를 추가하여 변경하고 주기적으로 표본 응답요청함수들(Callbacks)의 사용할 수 있게 해 주기 때문에 그 알고

```
1.  UpdateState(Event):
2.    IF Event's type is endRecv AND its send event has not been reported
3.      Event.state <- pending
4.    ELSE Event.state <- current
5.    IF Event's type is endRecv AND its send event has been reported AND
6.       Event.remotePred.Cs > Event.localPred.Cr
7.      Event.waitTime += (Event.remotePred.Cs - Event.localPred.Cr)

8.  Report(Event):
9.    add Event into GAG
10.   Event.state <- reported
11.   IF (Event.remoteSuc && Event.remoteSuc.state == pending)
12.     UpdateState(Event.remoteSuc)
13.   IF (Event.localSuc) UpdateState(Event.localSuc)

14. EarliestEventTime(Event): // Time when Event is chosen as the current one
15.   IF Event's type is endRecv
16.     return Event.waitTime + group(Event).time
17.   ELSE // CNER (Current Non-End-Receive) Events are all current events except endRecv ones
18.     return Event.procTime * |CNER events| + group(Event).time

19. EventArrival(Event):
20.   insert Event into PAG
21.   IF (there is no un-reported event for Event's Process) UpdateState(Event)
22.   ELSE Event.state <- queued
23.   WHILE (Each Process has a current or pending Event)
24.     neEvent <- CNER Event with the smallest EarliestEventTime(Event)
25.     eEvent <- current endRecv Event with smallest EarliestEventTime(Event)
26.     IF (neEvent AND
27.         (no eEvent OR EarliestEventTime(neEvent) < EarliestEventTime(eEvent)))
28.       FOR EACH (current or pending Event in neEvent's Group)
29.         IF (Event.state == pending)
30.           Event.waitTime -= |CNER Events in Event's Group| * neEvent.procTime
31.         ELSE Event.procTime -= neEvent.procTime
32.       group(neEvent).time += |CNER Events in neEvent's Group| * neEvent.procTime
33.       IF (neEvent is a send event)
34.         neEvent.Cs <- group(neEvent).time
35.       ELSE IF (neEvent is a startRecv event)
36.         neEvent.Cr <- group(neEvent).time
37.       Report(neEvent)
38.     ELSE
39.       FOR EACH (current or pending Event in eEvent's Group)
40.         IF (Event.state == pending)
41.           Event.waitTime -= eEvent.waitTime
42.         ELSE Event.procTime -= eEvent.waitTime/ |CNER Events in eEvent's group|
43.       group(eEvent).time += eEvent.waitTime
44.       Report(eEvent
```

그림 3 Ps-PEM 성능 예측 의사코드(Pseudo Code)

리즘을 쉽게 구현할 수 있게 해 준다. 응용 프로그램들의 집합을 실행함으로써 Ps-PEM을 검증했다. 대상 집합은 합성 병렬 응용(Synthetic Parallel Application, SPA), 여행하는 세일즈맨 문제(Traveling-Salesman Problem, TSP), 그리고 NAS 벤치마크 프로그램들의 한 부분집합을 포함한다. 이 NAS 응용들은 고도의 병렬 프로그램(Embarrassingly Parallel program, EP), 병렬 FFT 계산(parallel FFT computation, FT), 정수 정렬 프로그램(Integer Sort program, IS), 및 다중그리드 해결기(Multi-Grid solver, MG)이다. NAS 응용에 대한 자료 크기는 워크스테이션들의 네트워크 상에서 실행할 수 있는 크기에 해당하는 "A 등급"이었다. 모든 프로그램들은 IBM SP-2상에서 실행하였고 통신을 위하여PVM을 사용했다[6]. 실험적 검증을 위하여 프로그램들의 실행 시간들을 측정했고 해당 Ps-PEM예측 시간들을 그 시간들과 비교했다. 또한, 예측 시간을 계산하는 오버헤드도 파악했다.

모든 측정이 전용 SP-2 노드들 상에서 행하여졌고, 그래서 다른 응용과의 간섭은 없었다. 응용 프로세스들의 프로세스 시간들과 참조표 형태의 통신시간만을 기반으로 예측이 이루어지기 때문에 성능 예측을 위한 계산은 목표 응용과 같은 프로세서 상에서 동작하는 다른 응용 관련 오버헤드에 의하여 영향받지 않는다. 그러나, 시스템 상의 부하는 실제 구성에서의 실행 타이밍에 영향을 미친다.

그림 4는 성능 측정 및 예측 시간들의 요약하여 보여

응용 목표 구성	측정 시간	실제 구성 예측. 에러 시간 %	실제 구성 예측 에러 시간 %	실제 구성 예측 에러 시간 %
SPA		4/4	4/2	4/1
4/4	158.7	159.0 -0.2%	158.2 0.3%	158.5 0.1%
4/1	240.2	235.5 2.0%	235.1 2.1%	236.2 1.7%
TSP		4/4	4/2	4/1
4/4	85.6	85.5 0.1%	85.8 -0.2%	85.9 -0.4%
4/1	199.2	197.1 1.1%	197.7 0.8%	198.9 0.2%
EP (class A)		16/16	16/8	16/4
16/16	258.2	255.6 1.0%	260.7 -1.0%	267.4 -3.6%
FT(class A)		16/16	16/8	16/4
16/16	140.9	139.2 1.2%	140.0 0.6%	144.0 -2.2%
IS(class A)		16/16	16/8	16/4
16/16	271.2	253.3 6.6%	254.7 6.0%	255.0 6.0%
MG(class A)[3]		16/16	16/8	16/4
16/16	172.8	166.0 4.0%	168.5 2.5%	186.7 -8.0%

그림 4 측정 및 Ps-PEM 기반 성능 예측 시간

응용 구성	메 시지 수	메시지 바이트	시간 W/o Inst	 With Inst	오버헤드 초.	 %
SPA						
4/4	56	248	158.7	164.2	5.5	3.5%
4/1	56	248	240.2	247.0	6.8	2.8%
TSP						
4/4	6	2.3K	85.6	88.6	3.0	3.5%
4/1	6	2.3K	199.2	203.6	4.4	2.2%
EP (class A)						
16/16	45	1.8K	258.2	268.8	10.6	4.1%
FT (class A)						
16/16	3,480	1.8G	140.9	146.7	5.8	4.1%
IS (class A)						
16/16	7,725	670.5M	271.2	291.2	20.0	7.4%
MG (class A)						
16/16	3,396	400.2M	172.8	178.7	5.9	3.4%

그림 5 Ps-PEM 기반 계산 오버헤드

준다. N이 프로세스 수이고 M이 노드 수일 때 목표 또는 실제 구성을 나타내기 위하여 N/M이라고 표시한다. 각 목표 구성에 대하여 세가지 실제 구성들 상에서 프로그램을 수행한다: 목표 구성과 동일한 하나, 목표 구성의 노드 수의 절반인 노드들을 가진 다른 하나, 그리고 그 절반 수의 다시 절반인 노드들을 가진 나머지 하나를 사용한다. 실행하는 구성과 동일한 목표 구성의 성능을 예측함으로써 사용한 통신시간 예측 방법이 얼마나 정확했었는지를 평가할 수 있었다. 실험한 결과, 모든 경우에서 예측 값들이 실제 시간들의 8% 범위 내에 있다. 모든 그림에서 시간의 단위는 초이다.

Ps-PEM 기반으로 계산하는 오버헤드도 측정했다. 이를 위하여 Ps-PEM 기반 계산을 하게 하면서 그리고 그 계산을 하게 하지 않으면서 여섯 개의 응용들을 실행했다. 그림 5가 나타내는 오버헤드는 해당 응용을 실행하면서 Ps-PEM 기반으로 계산하게 하는데 추가로 필요한 시간을 나타낸다. 대부분의 응용과 구성에 대하여 Ps-PEM 기반으로 계산하는 오버헤드는 5% 이내이다. 그러나, IS 응용에 대하여 해당 오버헤드는 7.4%이다. 비교적 높은 이 오버헤드의 원인을 조사한 결과, Paradyn 성능 도구와 그 응용 프로그램을 동시에 실행하는 "상호작용"에 의하여 오버헤드가 주로 야기되었다고 결론내렸다.[2)]

3. Networking Performance Evaluation Methodology(N-PEM)

각 응용에 대하여 본 그림은 목표 구성을 보이고 두 번째 열은 이 목표 구성에서의 실행 측정 시간을 보여준다. 테이블의 나머지 부분은 세개의 다른 실제 구성에서 실행될 때 Ps-PEM을 기반으로 예측된 실행 시간들을 보인다.

Networking Performance Evaluation Methodology (N-PEM)는 분산 또는 병렬 실행 환경에서 통신 네트워크를 변경시키는 효과를 예측함으로써 네트워크 업그레이드의 영향을 사전 평가할 수 있게 해 준다. N-PEM의 목적은 네트워크를 변화시키는 조건에서 실행 시간의 잠재적 향상을 계산하는 것이다. N-PEM 기반 성능 예측 알고리즘은, 응용이 근거리 네트워크 상에서 실행되면서 장거리 네트워크 상에서 성능에 영향을 미치는 성질들의 동작을 모의실험하는데 역시 쓰여질 수 있다. Ps-PEM과 유사하게, N-PEM은 현재 사용 가능한 네트워크 대신 가상의 네트워크를 사용할 때의 잠재적인 성능 향상을 평가하기 위하여 그 네트워크 상에서의 프로그램 실행 시간을 예측하게 해 준다.

N-PEM 기반으로 계산하기 위하여, 현재 (실제) 네트워크의 통신시간 참조표를 예측 (목표) 네트워크의 참조표로 교체한 후 Ps-PEM 기반으로 계산하기 위하여 사용하였던 같은 알고리즘을 사용했다. 본 연구에서는 실제 및 목표 네트워크들를 모두 사용할 수 있었기 때문에 각 네트워크에 대하여 메시지 전송시간을 측정함으로써 해당 참조표를 만들었다. 그러나, 제안된 (가상의) 네트워크를 평가하고자 했다면 단지 그것의 기대되는 성능을 바탕으로 적당한 값들로 구성된 해당 참조표를 만들었을 것이다. N-PEM 기반으로 계산하는 오버헤드는 Ps-PEM 기반으로 계산하는 것과 동일하다.

N-PEM에서는 현재 네트워크의 통신시간 참조표 대신에 목표 네트워크의 참조표를 사용함으로써 Ps-PEM의 변형된 형태로서 N-PEM을 구현했다. Ps-PEM을 평가하기 위하여 사용한 NAS 벤치마크 응용들의 동일한 부분집합에 대하여 실험함으로써 N-PEM의 유효성을 검증했다. 이 NAS 응용들에 대하여 10Mbps 이더넷 상에서 320Mbps 고성능 스위치 상의 실행 시간을 실제

2) 이것이 Paradyn과 많은 차단 시스템 제어 호출들(Blocking System Calls)을 하는 프로그램들 안의 ptrace 실행의 상호작용에 의한 것임을 의심하지만 아직 이 점을 조사하고 있다.

실행 시간의 8% 이내로 예측할 수 있었다. 예를 들어, FT에 대하여 예측 에러는 4% 이내였고 이 경우의 실제 실행 시간은 목표 구성 상의 시간보다 30배 느렸다. 또한, 목표 구성과 목표 네트워크 통신비용을 동시에 적용함으로써 Ps-PEM과 N-PEM의 조합을 구현하여 실험했다. NAS 응용들 중 네가지에 대하여 실험한 결과, 모든 경우에 16/8 및 이더넷 현재 구성으로부터 얻은 16/16 및 고성능 스위치 목표 구성 상의 성능 예측 값들이 실제 목표 구성에서의 실행 시간들의 7% 이내에 있다는 것을 확인했다.

4. 관련 연구

본 논문에서 제시하는 온라인 "이렇게 변경하면 어떨까" 계산과 밀접히 관련된 두 분야들이 있다: 성능 측정 도구와 성능 예측 도구 분야들이다. 성능 측정 도구들은 실제 프로그램 실행의 동작을 수량화하고 특정 동작들 또는 프로그램 구성 요소들에 시간을 할당한다. 성능 예측은 알고리즘 또는 프로그램 실행 시간을 예측하기 위하여 모델 또는 모의실험을 사용한다.

성능 측정 도구들의 세가지 주요 유형들이 있다: 프로파일러, 시각화 및 탐색 도구들이다. 프로파일러 메트릭들[7,8]은 분산 또는 병렬 응용의 각 구성 요소(주로 프로시저)에 해당 값을 부여하고 정렬된 테이블의 형태로 소개된다. 시각화 도구[9,10]는 그림을 사용하여 응용의 성능을 보여준다. 탐색 도구들[11,12]은 성능 병목점을 찾는 것을 탐색 문제로 간주함으로써 사용자들이 성능 데이타 정보를 얻는 오버헤드를 조절할 수 있게 한다. 그러나, 이 모든 도구들은 하나의 실행에 대하여 특정 프로그램을 측정하거나 분석하는데 초점을 맞춘다. 대안들을 프로그래머가 선택하게 하는 한가지 종류의 도구는 응용 조정(Application Steering)[13,14] 도구이다. 응용 조정은 프로그램이 실행되는 동안 프로그래머들이 그것의 선택된 면들을 변화시킬 수 있게 한다. 이 기술은 프로그램 파라미터들을 조율할 때 아주 효과적일 수 있다. 그러나, 현재 실행되는 실행 이미지 안에서 가능한, 데이타를 분해하고 변화시킬 수 있는 알고리즘 종류에는 필연적으로 제약이 따른다. 복잡한 알고리즘 변경을 위해서는 프로그램 수정이 필요하다.

성능 예측은 제어된 환경에서 프로그램 실행을 연장하는 기법 또는 정적 프로그램 분석에서 도출된 추계(Stochastic) 모델들을 기반으로 할 수 있다. 실종된 싸이클 분석(Lost Cycles Analysis)[15]은 상호 간섭 효과가 없는 파라미터들을 변화시키고 그 실행 시간을 기록하는 제어된 일련의 실험들을 실행함으로써 다른 동작 점들에서 성능을 예측할 수 있게 한다. 그러나, 이 기술의 사용을 위해서는 실행 가능한 다른 조율 대안들의 구현이 필요하다. 정적 예측[16,17]은 프로그램 실행 시간을 예측하는 모델링 언어들 또는 소스코드 분석 방법을 사용한다. 필연적으로, 이 기술은 응용, 시스템 소프트웨어, 그리고 하드웨어 간의 상호 작용의 많은 자세한 면들을 고려하지 않는다.

5. 결론 및 미래 연구 방향

본 논문에서는 후보 조율 대안들이 어떻게 응용의 실행 시간을 향상시킬 것인가에 대하여 평가할 수 있도록 연구개발한 새로운 성능 예측 방법론을 제안하였다. 이 방법론의 유효성을 실험적으로 검증하기 위하여 여섯개의 프로그램들에 대하여 제안한 방법론이 가상으로 변경된 구성의 실행 시간을 정확하게 예측할 수 있다는 것을 보였다.

이 방법론이 현재의 형태로 유용한 경우를 정리하면 다음과 같다.

Ps-PEM(Process-level Performance Evaluation Methodology): 분산 또는 병렬 프로그램 수행 시 더 크거나 더 작은 수의 프로세서를 사용할 때의 실행 시간을 예측함으로써 프로세서 수의 변경이 유익한지를 판단하여 사용 프로세서의 수를 변경하고자 하는 경우

N-PEM(Networking PEM): 분산 또는 병렬 프로그램 수행 시 기반 네트워크를 변경한 후의 실행 시간을 예측함으로써 기반 네트워크의 변경이 유익한지를 판단하여 기반 네트워크를 변경하고자 하는 경우

PEM은 다른 성능분석 도구들과 병행하여 사용하면 더욱 효과적이다. 예를 들어, 병목 지점 측정 도구가 기반 네트워크를 병목 지점으로서 파악하는 경우 N-PEM을 사용하여 기반 네트워크를 변경한 후의 성능을 예측할 수 있다.

PEM은 현재의 형태로도 프로그래머들에게 유용하지만, 이 연구를 확장할 여러가지 연구 방향들이 있다. 첫째, PEM은 어떤 프로그램 조율 대안을 평가할 것인가에 대한 정보를 제공하지 않는다. 대부분의 경우에, 고려할 다수의 조율 대안들이 있다. 하나의 미래 연구 방향은 후보 조율 대안들의 선택 자동화를 연구하는 것이다. 둘째, 대안들의 자동 선택에 대한 연구 결과를, 성능 향상을 위하여 관찰된 동작을 기반으로, 실행되는 동안 자동적으로 프로그램을 수정하는 동적 프로그램 적응 연구의 바탕으로서 사용할 수 있다. 셋째, 이와 같은 자동 적응을 가능하게 하기 위하여 동적 이동을 고려하고 해당 이동 비용을 PEM에 포함하는 이슈에 대한 연구가 필요하다.

참 고 문 헌

[1] S. K. Reinhart, J. R. Larus, and D. A. Wood, "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," SIGMETIRCS, pp. 46-60, May 1993.

[2] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," Comm. ACM, Vol.21, No.7, pp. 558-564, 1978.

[3] D. Kimelman and D. Zernik, "On-the-Fly Topological Sort - A Basis for Interactive Debugging and Live Visualization of Parallel Programs," ACM/ONR Workshop on Parallel and Distributed Debugging, San Diego, CA, pp. 12-20, May 17-18, 1993.

[4] J. K. Hollingsworth, "An Online Computation of Critical Path Profiling," SPDT'96: SIGMETRICS Symposium on Parallel and Distributed Tools, Philadelphia, PA, pp. 11-20, May 22-23, 1996.

[5] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, "The Paradyn Parallel Performance Measurement Tools," IEEE Computer, Vol.28, No.11, pp. 37-46, 1995.

[6] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, PVM: Parallel Virtual Machine, The MIT Press, Cambridge, Mass, 1994.

[7] A. J. Goldberg and J. L. Hennessy, "Performance Debugging Shared Memory Multiprocessor Programs with MTOOL," Supercomputing'91, Albuquerque, NM, pp. 481-490, Nov. 18-22, 1991.

[8] M. Martonosi, A. Gupta, and T. Anderson, "MemSpy: Analyzing Memory System Bottlenecks in Programs," SIGMETRICS, Newport, RI, pp. 1-12, June 1-5, 1992.

[9] F. Lange, R. Kroger, and M. Gergeleit, "JEWEL: Design and Implementation of a Distributed Measurement System," IEEE Transactions on Parallel and Distributed Systems, Vol.3, No.6, pp. 657-671, 1992.

[10] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera, Scalable Performance Analysis: The Pablo Performance Analysis Environment, Scalable Parallel Libraries Conference, A. Skjellum, Editor, IEEE Computer Society, pp. 104-113, 1993.

[11] J. K. Hollingsworth and B. P. Miller, "Dynamic Control of Performance Monitoring on Large Scale Parallel Systems," 7th ACM International Conference on Supercomputing (ICS), Tokyo, Japan, pp. 185-194, July 1993.

[12] W. Williams, T. Hoel, and D. Pase, The MPP Apprentice Performance Tool: Delivering the Performance of the Cray T3D, Programming Environments for Massively Parallel Distributed Systems, North-Holland, 1994.

[13] W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, J. Vetter, and N. Mallavurupu, "Falcon: On-line Monitoring and Steering of Large-Scale Parallel Programs," Frontiers '95, McLean, VA, pp. 422-429, Feb. 6-9, 1995.

[14] D. A. Reed, K. A. Shields, W. H. Scullin, L. F. Tavera, and C. L. Ellford, "Virtual Reality and Parallel Systems Performance Analysis," IEEE Computer, Vol.28, No.11, pp. 57-68, 1995.

[15] M. E. Crovella and T. J. LeBlanc, "Parallel Performance Prediction Using Lost Cycles," Supercomputing'94, Washington DC, pp. 600-609, Nov. 14-18, 1994.

[16] V. Balasundaram, G. Fox, K. Kennedy, and U. Kremer, "A Static Performance Estimator to Guide Data Partitioning Decisions," 1991 ACM SIGPLAN Symposium on Principals and Practice of Parallel Programming, Williamsburg, VA, pp. 213-223, April 21-24, 1991.

[17] A. J. C. v. Gemund, "Performance Prediction of Parallel Processing Systems: The PAMELA Methodology," 7th ACM International Conference on Supercomputing (ICS), Tokyo, Japan, pp. 318-327, July 1993.

엄 현 상

1992년 서울대학교 계산통계학과 (학사) 1996년 미국 University of Maryland 컴퓨터과학과(석사). 1997년 미국 Sun Microsystems, Inc. Data Engineering Group, 인턴. 2003년 미국 University of Maryland 컴퓨터과학과(박사). 2003년~2005년 삼성전자 정보통신총괄 책임연구원. 2005년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 컴퓨터 시스템/네트워크/응용 소프트웨어 성능공학, 모바일 응용/미들웨어(보안 포함), 임베디드 소프트웨어