확장성을 고려한 계층적 시스템 성능 모델 및 시뮬레이션*

Hierarchical Performance Modeling and Simulation of Scalable Computer System

김 흥 준**, 안 효 범**, 조 경 산**
HeungJun Kim, HyoBeom Ahn, and Kyungsan Cho

Abstract

The performance of a computer system depends on the system architecture and workload, and the high performance required in many applications can be achieved by the scalability of the system architecture and workload.

This paper presents scalable workload, a performance metric of scalable speedup and hierarchical modeling for the scalable computer system as well as the development of the object-oriented simulator smplC^{++} which is an advanced C^{++} version of the discrete event-driven simulation environment smplE.

In addition, this paper presents two examples of applying scalable speedup, hierarchical modeling and simulator smplC^{++} to analyze the performance effect of the scalability in a multiprocessor system and a network-based client/server system.

1. 서 론

컴퓨터 시스템의 성능은 시스템의 구조 및 실행 프로그램인 작업부하에 의해 영향을 받는데, 많은 실 응용 분야에서 요구되는 고성능을 위해 시스템 구조를 선형적으로 확장시킬 필요성이 증가되고 있다[1]. Gorden Bell[2]은 확장적 컴퓨터의 3 가지 개발 이념을 다음과 같이 제시하였다.

1) 시스템 크기의 확장성: 잘 정의된 작업부하에 대해 시스템의 구조적 크기(프로세서의 수와 같은)를 증가시켜 이에 상응(비례)하는 성능을 얻을 수있는 시스템의 확장성을 의미한다. 이 개념은, 시스템 크기의 확장성에 의해 대단위 병렬 처리 시스템을 구현할 수 있음을 나타낸다.

2) 세대적 확장성: 프로세서의 처리 능력, 메모리 용량 등과 같이 일정한 주기로 확장되는 추세를 의미한다.

3) 문제의 확장성: 처리해야 할 자료의 양을 확장시키는 것을 의미하며, 주어진 문제의 입도(granularity)에서 효율적으로 동작하도록 작업부하를 크게 증가시키는 것이다.

최근의 시스템 구조의 개발 경향은 확장성에 있으며 이에 대한 평가는 성능, 신뢰성, 유효성, 현실성, 가격 등의

* 이 논문은 1994년도 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음.

** 단국대학교 전산통계학과

여러 측면에서 고려되어야 한다. 그 중에서 특히 성능적 측면이 강조되어, 구조의 확장성은 확장된 구조에 따라 비례하는 성능을 추구한다. 하지만 시스템 구조의 확장성에 비례하는 성능을 얻는 것은 불가능한 실정이다. 한 예로 프로세서의 수가 n배로 증가되었을 경우 n배의 성능 개선, 즉, 동일한 작업부하 하에서 n배의 처리시간 감소를 얻거나 동일시간 내에 n배의 작업량을 처리하는 것은 불가능하다. 따라서, 확장성에 대한 현실적인 성능 측도 및 분석이 고려되어야 한다. 즉, 시스템의 확장성은 Gorden Bell이 제시한 구조 및 작업부하의 확장성과 이에 대한 성능 분석 방법의 개선까지도 제시되어야 한다.

성능 향상과 연계하여 다음과 같은 구조 및 작업부하의 확장 환경들이 고려되었다[3].

첫째는 일정한 작업부하 하에서의 구조적 확장성으로 이는 구조의 확장에 비례하는 처리시간의 개선을 성능 목적으로 하는 경우이다. 이 경우에, 시스템의 시스템의 과부하가 무시되면 Amdahl의 법칙[1]에서와 같이 일정한 작업부하에서는 순차적(sequential) 병목(알고리듬의 직렬성에 의한 과부하)이 성능 향상에 저해가 되어, 확장적 구조의 성능 개선도는 비관적 예측을 제시한다.

둘째는 구조적 확장성에 비례하여 작업 부하의 크기를 증가시키는 것으로 처리 시간의 개선보다는 처리율의 개선을 목적으로 하는 환경으로, 시스템의 과부하를 제외하면 Gustafson의 모델[4]과 같은 의미를 갖는다. 이 경우의 성능 목적은 동일한 시간 내에 더 많은 작업을 처리하고자 함이고, 앞의 일정한 작업부하에서 제시된 확장적 구조의 제한점을 극복하게 된다.

결론적으로, 확장성을 고려한 컴퓨터 시스템에서는 구조적 확장성과 다양한 작업 환경에 따른 실제 응용 작업부하의 확장성을 고려하여, 확장성에 따른 성능의 개선도와 성능 개선을 제한하는 요소(작업부하의 불균형에 의하거나, 시스템의 구성 요소 사이의 불균형 등에 의한)들을 분석할 수 있는 성능 분석의 방법을 종합적으로 연구해야 한다.

이러한 목적을 위해 본 논문에서는 다양한 구조적 특성에 따른 확장성과 작업부하의 확장성을 소개하고, 확장성에 의한 성능 개선을 효율적으로 나타낼 수 있는 확장적 개선도(scalable speedup)라는 새로운 성능 측도를 제시하고, 이들을 합리적으로 분석하기에 적합한 계층적 성능 모델을 소개한다. 또한 이러한 개념을 실제 성능 분석에 적용할 수 있도록 기존의 이산 사건 기반의 시뮬레이터 smplE를 객체 지향적으로 변환한 새로운 시뮬레이터인 smplC++의 개발을 설명하고, 이의 실제 시스템 분석에의 활용 예를 보인다.

본 논문의 구조는 2장에서는 확장적 구조 및 작업 부하, 성능 측도와 계층적 성능 모델을 제시하고, 3장에서는 객체 지향적 시뮬레이터인 smplC++의 개발을 설명한다. 4장에서는 다중 프로세서 시스템과 클라이언트/서버 시스템에서 구조 및 작업의 확장성에 따른 성능 분석의 예를 제시하고, 5장의 결론으로 본 논문을 마무리한다.

2. 확장적 구조와 계층적 성능 모델

2.1 구조적 확장성

컴퓨터 시스템에서의 구조적 확장성은 시스템을 구성하는 내적 또는 외적인 구조적 요소의 종류나 수의 증가를 나타내며, 이 개념은 다양한 기능과 처리능력을 가진 대단위 시스템의 구현을 가능케 한다. 하지만 구조적 확장성이 단지 프로세서의 수, 메모리 용량 또는 입출력 및 통신 처리 능력 등의 선형적 증가만을 의미하는 것은 아니며, 본질적으로는 구조적 확장에 따른 선형적 성능의 개선을 포함하고 있다. 하지만, 구조적 확장에 비례하는 성능을 얻는 것은 불가능하므로 현실적인 성능 분석이 요구된다.

시스템의 구조적 확장은 시스템의 전체적 균형을 고려해야만 시스템 전체의 성능적 확장이 수반된다. 예를 들어 프로세서의 수를 증가시키는 경우, 시스템 내의 전체적인 메모리/입출력/통신의 요구량도 따라서 증가한다. 이로 인한 메모리/입출력 등의 상대적 처리 능력의 저하에 의한 처리 시간의 지연을 발생할 수 있다. 따라서 이러한 능력 저하 요소의 기능을 시스템의 균형에 맞게 향상시켜야 한다. 이를 위하여 구조의 확장에서 발생되는 각 개별적 병목 지역을 분리시키고 이들을 개별적으로 처리하기 위하여 각 구조적 구성 요소의 특성을 상/하위 수준으로 표현하는 계층적 구조와 이에 따른 계층적 성능 모델의 구축 및 분석이 필요하다. 구조적 확장의 예는 프로세서의 수, 클럭주기, 메모리양, 통신 과부하, 입출력 요구 등이 있다[1].

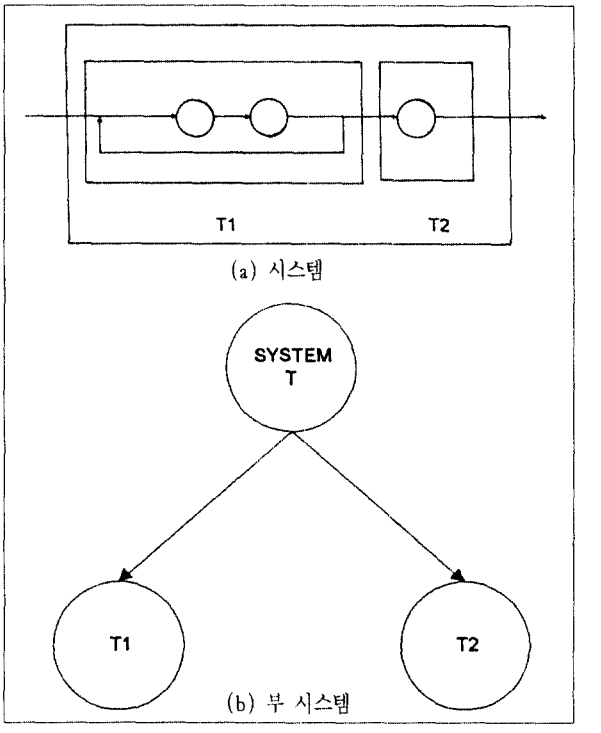
2.2 성능 분석 방법 및 계층적 성능 모델

시스템 성능 분석에는 여러 방법이 있으나, 측정에 의한 성능 분석이 가장 정확하며 필요한 성능 관련 정보를 자세히 제공해 줄 수 있다는 장점이 있어 실제로 많은 경우에 사용된다[5][6]. 하지만, 실제 시스템에 측정을 위한 소프트웨어 또는 하드웨어적 모니터를 설치하기 위해서는 운영체제 또는 기존 하드웨어의 수정이 필요하며, 특히 여러 기종에 대한 성능 분석이나 본 연구에서와 같은 구조의 확장성에 따른 성능 분석에는 적합치 못하다. 또한 구조의 확장에 따른 영향의 분석은 비교적 복잡하고 다양한 환경이 고려되어야 하므로 분석적 모델은 많은 가정이 필요하고, 정확한 분석이 어렵다는 단점이 있다. 확장적 시스템에서의 성능 분석은 확장성에 따른 성능 개선 또는 확장성으로 인한 성능 개선의 저해 요소들을 분석하기에 적합해야 하는데, 이를 위하여 본 연구에서는 계층적 성능 모델링과 시뮬레이션을 결합하는 방법을 제시하도록 한다.

컴퓨터 시스템은 구조 및 운영적 특성에 따라 이를 구성하는 여러 부시스템들로 분해(partition)가 가능하고, 또 각 부시스템은 이를 구성하는 모듈로 순환적으로 분해할 수 있다. 이러한 분해 과정은 독립적 단위 기능을 가진 구조적 요소까지 순환적으로 가능하다. 이러한 분해 과정으로 형성된 결과는 계층적 구조 및 각 구조적 요소에 따른 계층적 모델로 표현이 가능하다[7]. 모델은 컴퓨터 시스템의 설계 taxonomy 상의 특수성에 의해 구축하며, 각 요소의 구조 및 운영의 특성에 따라 구성되도록 한다. 이 때 가장 낮은 계층은 독립적 기능을 가진 요소의 구현 예(instance)가 되며, 각 하위 계층의 요소가 모여 상위 계층의 요소를 구성하게 되어 상속성이 각 계층을 통해 유지된다. 각 요소가 하나 이상의 형태로 구현이 가능하다면, 가능한 구현들은 그 요소의 instance들이 된다. 또한 새로운 구조적 요소의 옵션이 추가되면, 이를 계층적 구조의 적합한 위치에 삽입하기만 하면 된다.

계층적 모델링은 구조적 계층도와 같이 전체 시스템 모델을 작은 부모델로 순환적으로 분해하는 것이다. 이 부모델들은 각각 개별적으로 분석되고 그 결과가 상위 모델의 분석에 적용되는 과정을 순환적으로 반복하는 것이다. 이러한 계층적 모델의 예로는 FESC(flow equivalence service center)[8], 또는 상위 작업 그래프(또는 스레드 그래프)와 하위의 작업 모델의 계층 모델[9], 또는 본 논문에서와 같은 계층적 구조에 따른 계층적 모델[7]등이 있다.

〈그림 2-1〉은 계층적 구조와 모델에 대한 간단한 예를 보인 것이다. 대상 시스템은 구조 및 운영 특성에 따라 그림(a)와 같이 분해 될 수 있으며, 이는 그림(b)와 같이 계층적으로 표현되어진다. 그림(a)에서 시스템의 처리시간 T는 시스템을 구성하는 부 시스템의 처리시간인 T1과 T2로 구해질 수 있다.



〈그림 2-1〉 계층적 구조와 모델의 예

확장적 구조의 계층적 성능 모델은 성능적 영향을 고려하여 각 계층의 독립성과 상속성을 유지할 수 있도록 객체 지향적 모델링으로 구현하는 것이 효율적이므로, 4장에서 설명될 객체 지향적 시뮬레이터를 활용하도록 한다.

2.3 성능의 측도

실행 성능의 측도는 시스템의 응용 목적에 의해 다양하게 정해질 수 있다. 고정적 구조에서의 성능은 흔히 처리

시간과 처리율의 2가지 측면으로 분석된다. 확장적 구조를 갖는 시스템에서는 고정적 구조의 시스템 성능에 덧붙여 구조의 확장에 의한 성능 영향을 분석할 수 있도록 처리 시간의 개선도(speedup) 또는 효율성 등이 분석되었는데[1][10] 이에 덧붙여 본 연구에서는 새로운 측도인 확장적 개선도를 제안한다.

1) 처리시간의 개선도(speedup)

절대적인 성능의 측도인 처리시간은 시스템의 구조 또는 작업환경에 따라 여러 경우로 정의될 수 있다. 예를 들어, 범용 시스템에서는 작업 또는 명령어의 처리시간을, 클라이언트/서버 시스템에서는 클라이언트의 요청에 대한 처리 시간을, 입출력 시스템에서는 입출력 요청의 처리 시간을 나타낸다. 처리 시간의 개선도는 동일한 작업부하라는 전제 조건에서 정의된 처리 시간의 개선율을 표시한다.

예를 들어 다중프로세서 시스템에서 프로세서 수의 증가에 따른 처리시간의 개선도(Sn)는 다음과 같이 표현된다[1].

$Sn = T(1)/T(n)$

T(i): 동일 작업부하에서 프로세서가 i개인 경우의 처리시간

하지만, 이의 대표적 모델인 Amdahl의 법칙[1]은 시스템의 과부하를 무시하고, 고정된 작업부하라는 전제로 인해 작업부하의 순차적 요소에 의해 성능 개선이 제한되는 등의 문제점을 가진다. 이러한 문제점을 개선하기 위해 확장적 시스템에 비례하는 확장적 작업부하를 제시한 Gustafson의 모델[4]은 실제 작업의 처리시간은 일정하므로, 확장적 구조(프로세서 수의 증가)의 성능 개선을 표현하기 위한 처리시간의 개선도를 확장된 작업부하의 환경 하에서 다음과 같이 제시하였다.

$Sn = T'(1)/T'(n)$

T'(i): n배로 확장된 작업부하에서 프로세서가 i개인 경우의 처리시간

Gustafson의 법칙에서도 시스템의 과부하가 무시되고, T'(1)는 기본 구조에 확장된 작업부하를 적용한 값으로 기본 구조에 대한 과도한 작업부하로 인해 실제로 T(1)은 구할 수 없거나 의미가 없을 수도 있다.

따라서 구조 및 작업부하의 확장에 따른 성능의 개선과 과부하로 인한 성능의 저하를 함께 표현할 수 있도록 다음에 제시되는 새로운 측도인 확장적 개선도가 바람직하다.

2) 확장적 개선도(Scalable Speedup)

시스템의 확장성에 비례하여 작업부하가 증가하는 경우에는 작업의 처리량은 증가하나, 각 작업의 처리시간은 시스템의 과부하로 인해 감소하게 된다. 따라서, 이 두 성능 요소를 결합하여, 이전의 모델에서 무시되었던 병렬적 작업처리에 의한 과부하를 고려하며 동시에 확장성에 의한 성능 개선을 포함할 수 있다. 이를 위해서는 각 처리 시간을 처리하는 작업의 양에 대해 정규화(normalize)하여 이들을 비교하는 것이 바람직하다.
제안되는 확장적 개선도(Se)는 다음과 같다.

$Se = (T(1)/W(1)) \,/\, (T'(n)/W(n))$

T(1): 프로세서가 1개인 경우의 실제 처리시간

T'(n): n배로 확장된 작업부하에 프로세서가 n개인 경우의 실제 처리시간

W(i): i배로 확장된 작업부하

이 측도의 의미는 단위 작업당 처리 시간의 개선도를 나타내는 것으로, 이의 장점은 구조 및 작업부하의 확장성을 모두 고려하고 시스템의 과부하에 의한 영향이 고려되며, Gustafson 법칙에서처럼 T'(1)과 같은 별도의 값을 계산할 필요가 없다는 것이다. 이 측도는 앞에서 설명된 처리시간의 개선을 구조의 확장성(프로세서의 수)에 대한 비율로 나타내는 시스템의 효율성(efficiency, En = Sn/n)과도 구별되어야 한다. 확장적 개선도의 값이 증가되면 구조 및 작업부하의 확장성에 대한 긍정적 성능 확장을 제시하며, 확장적 개선도의 값이 감소되면 구조 및 작업부하의 불균형으로 확장성에 대해 실제 성능 효율은 감소함을 나타낸다.

즉, 확장적 개선도는 앞에서 설명된 처리 시간의 개선도와 효율성이라는 2가지 개념이 함께 포함되며, 이의 실 활용 예는 4장에서 소개된다.

2.4 작업부하

작업부하는 단순히 실 프로그램의 특성을 제공하는 이외에 평가하려는 시스템의 확장적 구조의 영향을 분석하기에 적합하게 자동적인 확장성과 성능 예측 분석에 필요한 기능을 제공해야 한다. 아울러, 성능모델 또는 시뮬레이터의 실제 수행할 내용인 작업부하는 실행 환경의 실제성과 모델에의 적용성이 모두 만족되어야 한다. 본 연구에서는 기존의 커널, 벤치마크[11][12], 인위적 작업부하[13]등을 분석/활용하여 확장적 구조의 특성을 반영하고 계층적 모델에 적용하기에 적절한 확장적 작업부하를 고려하였다.

작업부하의 확장성은 추구하는 성능의 목적과 실제 응용 작업환경에 따라 다음과 같이 정해진다.

확장적 구조에서 처리시간을 감소시켜 처리율을 증가시키기 위해서는 동일한 작업부하를 사용한다. 하지만, 확장된 구조로 동일한 작업부하를 실행하는 경우는 매우 제한적이다. 따라서, 처리 시간보다는 확장된 구조적 요소의 활용도를 높여 처리율을 증진시키기 위해서는 작업부하(즉, 프로세서의 확장에 따른 계산 처리양, 메모리의 확장에 따른 메모리 요구양, 입출력의 확장에 따른 입출력 요구양, 외부 연결의 증가에 따른 통신 요구양등)를 구조의 확장에 비례하여 증가시켜 적용하게 된다.

실제 작업부하의 확장은 다음의 2 방법으로 구현된다.

첫째로는 동일한 작업을 반복 수행하는 것으로, 이는 수행할 작업의 양은 증가시키나 확장된 작업부하로 인한 시스템에의 성능 영향 분석은 의미가 적게 된다. 예로서 AIM suite III에서는 사용자 수의 증가를 동일한 작업의 반복 수행(사용자의 수에 비례한)으로 구현한다.

둘째로는 작업부하 내용을 실제로 확장하는 것으로, 이의 예로는 4장에서 제시하는 바와 같이 다중 프로세서에서 모든 프로세서가 동일한 작업부하(프로세서의 수와 무관한)를 갖도록 하는 경우처럼 확장되는 처리 능력에 비례하여 작업부하를 증가하는 경우와, TPC-C에서 확장된 구조에 비례하여 처리 능력의 확장이 없이 단위 시간당 발생되는 작업부하가 증가하도록 구현하는 경우 등이 있다.

결국, 구조의 확장에 따라 인수(parameter)화된 작업량의 확장이 스스로 가능하게 되어 동적으로 현실적으로 작업부하를 조정할 수 있으며, 기존 자료를 활용하여 계층적 모델/시뮬레이터에 적용이 용이하고, 목적에 부합하는 성능 측도를 예측할 수 있는 작업부하가 바람직하다. 인수 화하여 확장 가능토록 할 작업부하의 내용에는 작업의 종류/양/수, 프로세스의 수, 순차적/병렬적 작업 비, 통신요구양, 입출력 요구양, 메모리 요구양등이 있다.

3. 객체 지향적 시뮬레이터 smplC++

앞장에서 제시된 여러 요건을 만족할 수 있으며, 또한 확장적 구조의 분석에 적절한 계층적 모델을 가장 효율적으로 구현하는 방법은 다음과 같은 이유로 객체 지향적 시뮬레이션이다.

- 상속성등을 사용하여 계층적 모델의 구현이 용이하다.
- 구조 및 작업부하의 확장성을 쉽게 추가할 수 있다.
- 구현이 비현실적 구조나 네트워크 등의 분석이 가능하다.

3.1 객체 지향적 시뮬레이터의 개발 의의

시뮬레이션 모델의 구축에는 시뮬레이션 전용 언어를 사용하는 방법과 일반 범용 언어를 이용하는 방법이 있다. C 언어를 시뮬레이션 툴 구현에 활용함으로써 범용 언어의 특성을 활용하여 컴퓨터 시스템의 시뮬레이션에 필요한 기본 구조를 제공하는 smpl이 개발되었다[14]. MacDougal이 제안한 smpl은 범용 목적의 프로그래밍 언어인 C를 이용하여 구현된 함수들의 집합 형태의 라이브러리로서 구성된 이산 사건 중심의 시뮬레이션 언어이다. 본 연구팀은 다양한 시스템의 시뮬레이션이 가능하고 효율적 처리가 가능하도록 smpl의 자료구조를 변경하고, 기타의 함수적 기능을 추가한 smplE을 에서 제안한 바 있다[15].

smplC++의 기존 이산 사건 기반 시뮬레이터인 smpl, smplE와 비교한 장점만을 정리하면 다음과 같다. smpl에 비해 smplE에서는 기능적인 확장과 자료구조의 변경을 통해 여러 대상 모델의 시뮬레이션과 효율적 처리가 가능하도록 확장하였으나, 본 연구에서 고려되는 확장적 구조 및 작업부하와 이에 따른 계층적 모델에 쉽게 적용하고, 대상 시스템의 성능 특성과 영향 분석을 효율적으로 수행하도록 기존 smplE에 객체 지향적 패러다임을 도입하여 새

로운 객체 지향적 시뮬레이터 smplC++를 객체 지향 언어인 C++로 구현하였다. 시뮬레이터에서 입출력의 자동화 부분은 구현 중으로, 일부 수작업을 통하여 시뮬레이션의 실행은 현상태로 가능하다.

3.2 smplC++의 구조 및 구성

smplC++은 이산 사건 중심 시뮬레이션 특성을 갖는 smplE의 기본적 방법과 기능을 그대로 유지하면서, 개선 목적 따라 여러 가지 추가적인 기능을 부가하였다.

smplC++에는 smplE와 같이 Facility, Token, 그리도 Event라는 3 가지 실체(entity)가 있고, 이 들의 기능은 smplE에서와 같다. Facility는 컴퓨터 시스템에서의 시스템 내의 독립적인 기능을 수행하는 물리적, 논리적 구성 요소 또는 자원을 나타내며, 계층적 모델 구조를 유지하기 위한 attribute들이 포함된다. smplC++에서 하나의 Facility는 하나의 큐 시스템으로 표현한다.

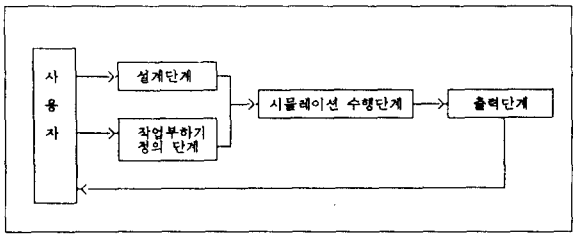
Facility 간의 연결 관계는 Token이라는 정보 전달 매개체에 의해 결정된다. Token은 시스템의 동적인 실체를 나타내는 것으로서, Facility간 Token의 움직임에 의해 시스템의 동적 행위가 결정된다. Token은 컴퓨터 시스템 내에서의 하나의 작업일 수 있고, 네트워크 모델에서는 하나의 전달 패킷일 수 있다. 또한 각 객체간 정보 전달의 메시지 기능을 수행한다.

Event는 시스템의 상태 변화를 야기하는 사건을 의미한다.

smplC++의 전체적인 구성은 〈그림 3-1〉과 같이 기능적으로 4 단계로 구분된다. 설계 단계에서는 시뮬레이션 대상 모델의 구성에 대한 정보와 운영에 대한 정보를 입력받고, 그 정보를 이용하여 Facility들로 구성되는 대상 모델의 구성 작업과 확장 작업을 수행한다. 작업부하기 정의 단계에서는 작업부하의 특성에 대한 정보를 입력받아 그 특성에 맞는 작업부하기들을 구축한다. 출력 단계에서는 시뮬레이션 수행 후 성능 분석에 필요한 활용도, 대기시간, 처리시간 등을 출력하여 준다.

3.3 smplC++의 기본 객체

smplC++에는 기본적인 실체인 Facility, Event, Token를 기본 구성 객체로 정의하고, 개별 객체인 Facility들의 계층적 구성을 유지하면서 각 Facility들에 대한 연결

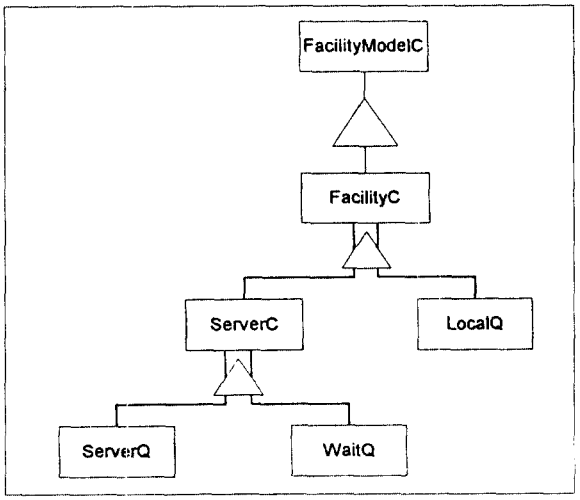


〈그림 3-1〉 smplC++의 기능적 구성단계

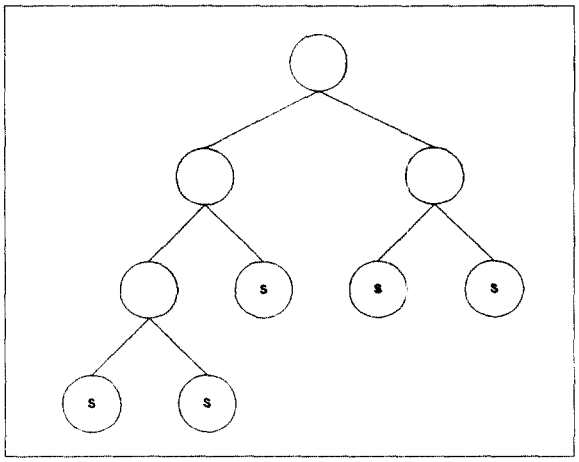
(interface) 기능을 담당하는 객체 실체를 포함하였다. 또한 다양한 작업 부하를 고려하여, 작업부하 모델의 구성을 별도로 처리하기 위한 작업부하기를 객체로 정의하였고, 이러한 작업부하기 객체들을 유지하면서 외부와 연결 기능을 담당하는 객체 실체를 포함하고 있다. 주요 객체들 간 조립형 연결 관계는 〈그림 3-2〉와 같고, smplC++의 주요 객체들은 다음 표와 같다.

〈표 3-1〉 smplC++의 주요 객체

객체 클래스 이름	특 성
DlistC	각 객체에 필요한 큐기능 클래스의 기본 클래스
StatisticC	확률 처리에 필요한 연산을 포함
ServerC	각 Facility에 부속되며 실제적인 시뮬레이션 운영 연산을 포함
FacilityC	Facility 계층간 연결 특성과 시뮬레이션 대상 구조의 독립적인 기능 요소를 포함
FacilityModelC	Facility들의 계층적 구성 유지 그리고 Facility와 외부와의 연결 기능을 수행
DesignC	사용자의 입력으로부터 대상 구조를 구축하며, 시뮬레이션 운영에 필요한 정보를 정형화
WorkloadC	작업부하의 특성을 포함하고 있는 작업부하기 객체
WorkloadModelC	작업부하기 들의 구성 유지와 각 작업부하기와 외부와의 연결기능
DesignWorkloadC	사용자의 입력으로부터 특성에 맞는 작업부하기들을 구축
EventListC	사건의 발생을 시간 순으로 등록 및 제거 기능
TokenC	FacilityC 들간의 동적 실체



〈그림 3-2〉 주요 객체간 연결 구조



〈그림 3-3〉 연결노드와 시뮬레이션 노드

3.4 사용자 정의

smplC++에서 사용자의 정의 항목은 구조 요소들의 계층적 구성과 확장에 관한 정보, 시뮬레이션 수행을 위한 정보, 및 작업부하 특성에 관한 정보의 3가지로 구분된다.

사용자의 구조 정보를 통해 생성되는 시뮬레이션 대상 구조의 요소들은 〈그림 3-3〉과 같이 계층적으로 구성되며, 이에는 시뮬레이션 노드(그림에서 s노드)와 연결 노드의 두 종류의 노드가 존재한다. smplC++에서 각 노드들의 하나의 Facility 클래스의 객체로서 정의되며, 시뮬레이션 노드에는 연결 노드의 attribute로 서버의 수, 서버의 수행 시간, 수행 시간의 분포 특성 등이 추가된다.

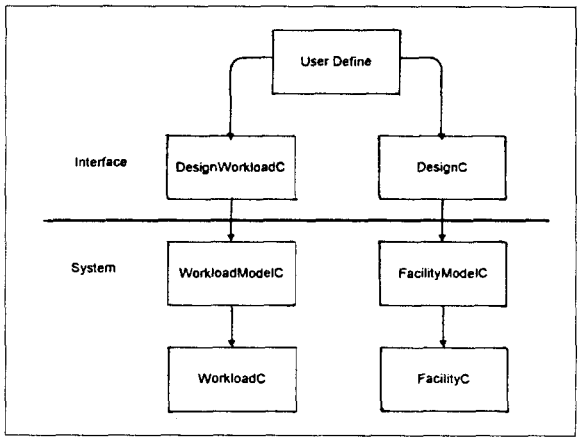
smplC++에서는 smplE에서와는 달리 시뮬레이션 수행 프로그램을 작성하지 않고 사용자의 정의에 의해 직접 시뮬레이션이 수행되도록 하기 위해 Token의 이동 경로를 결정할 수 있는 4 가지 유형을 분류하였다. 첫째 유형은 작업부하의 특성에 따라 Token의 이동 경로가 정해지며, 둘째는 확률적 특성(캐쉬 히트율과 같은)에 의해 Token의 이동 경로가 정해진다. 셋째는 랜덤하게 Token의 이동 경로가 달라지며, 마지막으로 Token이 경유한 노드에 의해 다음번 이동 경로가 정해진다. 또한 각 유형에서 Token이 원하는 목적지로 이동하기 위해 경유하는 경유형을 고려하였다. smplC++의 수행은 한 노드로부터 연결될 수 있는 모든 노드의 유형을 정의한 테이블을 통하여 각 유형이 정의된 테이블을 참조하도록 구성되어 있다. 시뮬레이션 수행에 관련된 입력 정보 형식은 앞의 유형에 따라 달리 입력되어질 수 있다.

여러 특성의 작업 부하가 복합적으로 발생할 수 있도록 smplC++에서는 작업부하의 특성을 확률적인 특성으로 고려하였다.

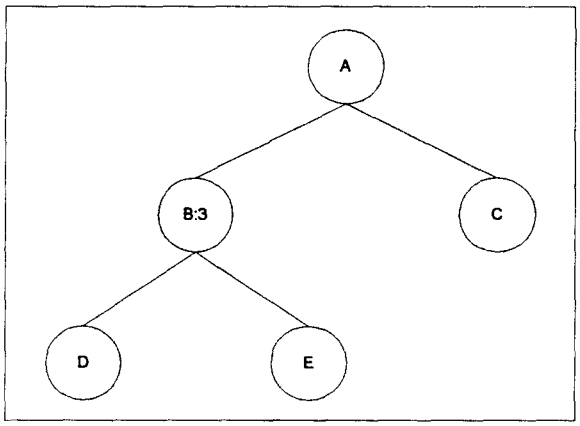
3.5 설계단계

사용자 정의로부터 입력된 정보는 DesignC와 DesignWorkIodC의 객체에 의해 소프트웨어적으로 구축된다. 〈그림 3-4〉은 사용자 입력과 설계 단계에서 필요한 객체간의 연결 관계를 보인다.

이 단계에서 〈그림 3-5〉와 같이 동일한 기능을 갖는 구성 성분의 확장에 대한 사용자 정의가 있는 경우 〈그림 3-6〉과 같이 프로그램 내에서 스스로 확장하는 기능과 기존의 구성 정보와 수행 정보를 변경하는 기능을 포함하였다. 확장시 수행 과정을 보면, 〈그림 3-5〉에서와 같은 기본적인 계층 구성에서 부시스템 B를 3개로 확장하는 경우 〈그림 3-6〉과 같이 B와 동일한 부 시스템 B1, B2, B3가 만들어지며 B1, B2, B3와 원래 B 노드와 연결된 A 노드 사이에 B′이 새롭게 생성된다. B′의 기능은 확장된 여러 개의 노드들과의 연결을 원활하게 운영하기 위한 목적으로 만들어지며, 현재 smplC++ 에서는 만들어진 여러 개



〈그림 3-4〉 객체간 연결 구조

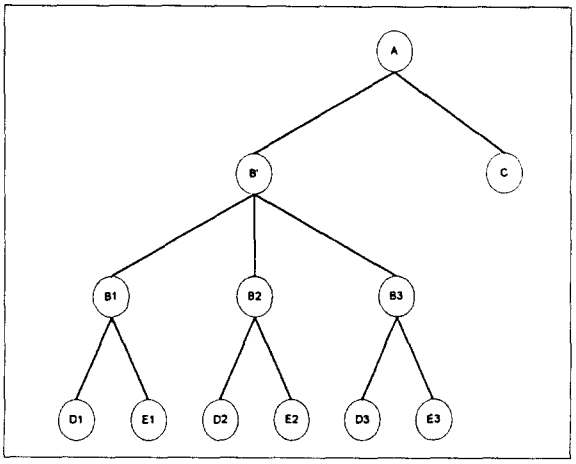


〈그림 3-5〉 사용자 정의에 의한 구조

의 노드들과 random 하게 연결하는 기능만을 포함하고 있다. 〈그림 3-5〉에서의 B 노드내에 정의 되어 있는 정보는 확장되어진 부 시스템의 헤드 노드들인 B1, B2, B3에 그대로 복사되어지고, 연결정보는 첨가된 B′ 노드로 인하여 새롭게 갱신되어진다. 이러한 확장과 새로운 노드의 첨가, 그리고 정보의 갱신 작업은 기본적으로 주어진 사용자 정보를 참고하여 설계 단계에서 자동으로 처리되어진다.

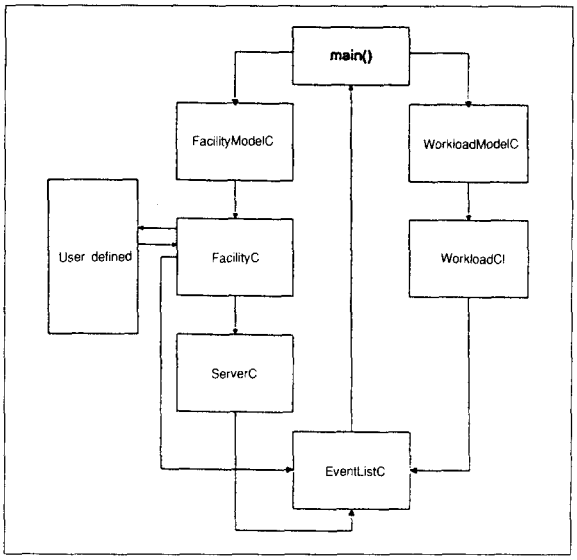
3.6 수행 단계

시뮬레이션 수행 단계에서는 일반적인 이산 사건 시뮬



〈그림 3-6〉 확장 적용된 구성

레이션 프로그램의 동작과 같은 방법으로 동작된다. smplC++에서 시뮬레이션이 진행되는 과정을 살펴보면 〈그림



〈그림 3-7〉 객체간 연결 관계

3-7〉에서와 같이 첫사건은 main() 함수로부터 WorkloadModelC 객체를 호출하고, 여러 개의 WorkloadC를 갖고 있는 경우 WorkloadModelC는 적절한 WorkloadC 객체를 호출한다. WorkloadC는 미리 정의된 특성에 따라 사건을 TokenC 객체로 생성하여 EventListC 객체에 전달한다. 다

음 시뮬레이션 진행은 이 최초의 사건에 의해 연속적으로 발생하게 된다. 다음 동작은 EventListC 객체로부터 최초의 사건 정보가 기록된 TokenC 객체를 가져와 FacilityModelC 객체로 전달한다. FacilityModelC 객체는 TokenC에 명시된 다음 사건을 처리할 FacilityC 객체로 TockenC 객체를 전달한다. FacilityC 객체는 사용자 정보를 검색하여 다음 처리할 사건을 발생하여 EventlistC 객체로 전달한다. 이 FacilityC 객체가 실제 시뮬레이션 수행 노드인 경우 ServerC 객체를 호출하여 변화에 따른 각종 측정치 값을 갱신하고 다음 사건을 발생하여 EventlistC 객체로 전달한다. 이와 같은 동작이 어떤 종료 조건을 만족할 때까지 계속 반복되며, 종료시 측정된 값들을 종합적으로 계산하여 최종 보고서를 작성하여 출력하고 시뮬레이션을 종료한다.

다음은 주요 클래스 객체의 간략한 코드이다.

```
class TokenC
{
        unsigned long TokenNum;                 //Token의 유일한 번호
        int Target;                             //Token의 첫번째 목적지
        int             WorkloadType;           //발생된 작업부하기 번호
        String TypeSignal;                      //발생 사건 유형
        int Priority;                           //우선순위
        double          EnterInTime;            //Token 발생 또는 변화 시간
    public:
        TokenC();                               //TokenC 생성자
};

class FacilityC
{
        int UniqueNum;                          //FacilityC 객체의 식별 번호
        String Name;                            //FacilityC 객체 이름
        int ChildNum;                           //연결된 자식 노드의 수
        int UniqueParentNum;                    //연결된 부모 노드의 식별 번호
        FacilityC * * Child;                    //자식 노드와의 연결
        int         NodeType;                   //노드 구분
        ServerC     * Server;                   //시뮬레이션 수행 노드인 경우 필요한 구성 객체
        // 사용자 정의 정보
    public:
        FacilityC();                            //생성자
        int Set();                              //설계 단계에서의 모델 구성 연산
        int Operation();                        //수행 단계에서의 운영 연산
};


class EventListC:public DlistC
{
    public:
        void EventIn();
        NodeC * EventOut();                     //TokenC 객체의 입력
        void EventShow();                       //TokenC 객체의 출력
};
```

4. 시스템 성능 분석의 실 예

본 연구에서는 다중 프로세서 시스템과 네트워크 기반의 클라이언트/서버 시스템을 앞에서 제시한 확장적 구조/작업부하, 성능 측도 및 계층적 성능 모델에 시뮬레이터 smplC++를 이용하여 분석하도록 한다.

4.1 다중 프로세서 시스템

구조적 확장은 공유 버스, 공유 메모리의 다중 프로세서 구조에서 프로세서의 수를 증가시키며, 이에 비례하여 작업부하가 증가하는 환경을 분석하였다. 이를 위하여 전체 시스템을 다음과 같이 프로세서, 연결 구조 및 메모리, 입출력의 부시스템으로 분해하여 모델토록 한다.

1) 프로세서의 모델

프로세서는 내부 파이프라인 구조만을 모델에서 고려한다. 각 프로세서는 동일한 작업부하를 갖도록 하며, 프로세서의 수에 비례하여 작업부하를 확장하여 이에 따른 성능 분석을 가능케 한다. 또 각 프로세서는 충분한 수의 콘텍스트가 대기한다고 가정하여 콘텍스트 전환에 의해 여러 콘텍스트들이 처리 가능하도록 한다. 따라서, 입출력 요구시 등에는 콘텍스트 전환을 실시하고, 콘텍스트의 수를 증가함에 따른 성능 영향을 분석케 한다.

2) 연결 구조(공유 버스) 모델

프로세서, 메모리, 입출력 부시스템은 공유 버스를 통해 연결된다. 공유 버스는 동기적 운영을 가정하고 프로세서간의 공정한 중재를 고려한다. 실패한 공유 버스의 요청은 큐에서 대기하여 처리하도록 하며, 공유 버스를 통한 메모리 접근은 packet switching 방식에 의한 분할(split) 전송 처리를 모델 한다.

3) 입출력 부시스템 모델

시스템 성능 저하 영향이 최대 44%로 중요한 입출력 부시스템은 디스크 요구 처리 시간을 모델하며, 다양한 입출력 환경에서의 입출력 충돌과 과부하를 분석하도록 다음 사항들을 모델에 고려한다. 다양한 입출력 환경 및 입출력 요구율의 변경에 따른 모델을 구축하며, 디스크 입출력은 페이지 폴트, 스왑핑, 순수 입출력(read, write)을 고려한다.

4) 작업 처리 시간의 모델

프로세서에 할당되어 처리되는 작업의 처리 시간은 다음과 같이 표시된다[1].

작업 처리 시간(Tj)
= 명령어 수(Ni) * CPI(명령어 당 처리 사이클 수) * 사이클 시간(Tcyc)

위의 모델에 [5]의 모델을 확장하여, CPI를 프로세서, 연결구조/메모리, 입출력 사이클에 의한 항목으로 다음과 같이 분해할 수 있다.

$CPI = p + m^*k + IO^*k'$

- p: 지역 메모리를 참조하는 프로세서에 의한 사이클 수
- m: 공유 메모리의 참조에 의한 메모리 사이클 수
- IO: 입출력 처리에 의한 입력 사이클 수
- k(k'): 메모리(입출력) 사이클의 프로세서 사이클 수

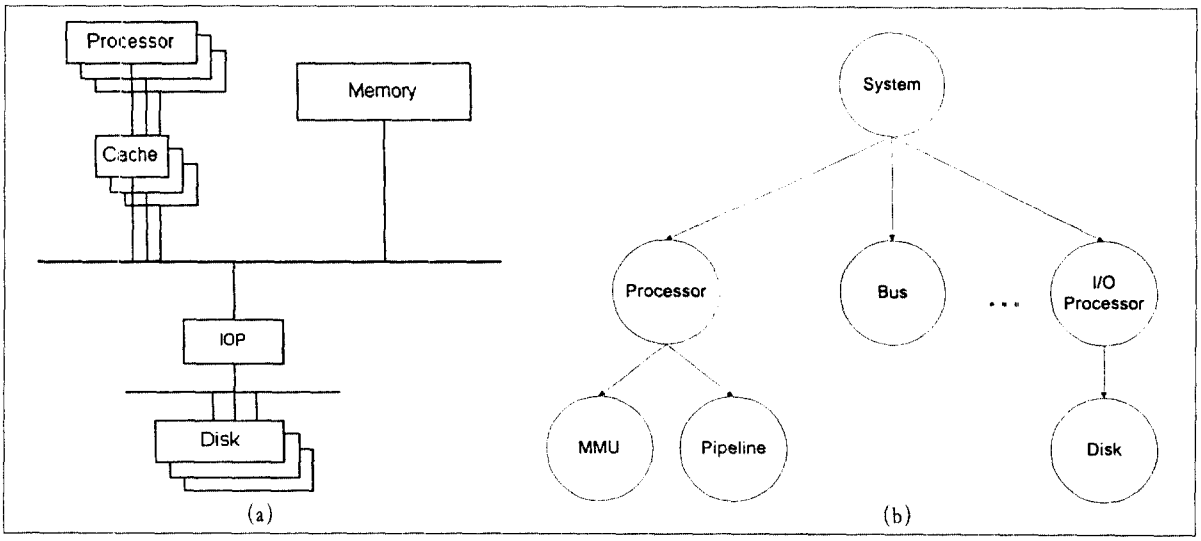
각 분해된 항목으로 Tj를 표시하면 다음과 같이, 다음과 같이 Tlm(지역 메모리를 이용한 처리 시간), Tsm(공유 메모리를 이용한 처리 시간), Ti/o(입출력 처리시간)으로 분해할 수 있다.

$Tj = Ni^*p^*Tcyc + Ni^*m^*k^*Tcyc + Ni^*IO^*k'^*Tcyc$

또 각 시간 성분은 관련 자원의 특성에 따라 충돌 처리 시간, 대기 시간, 실 수행 시간 등으로 분해하여 작업부하 또는 구조의 불균형 및 병목현상을 분석할 수 있다.

5) 대상 시스템의 구조, 작업부하 및 계층적 모델

본 연구에서 고려하는 대상 모델은 하나의 버스를 여러 개의 프로세서, 메모리, 입출력 프로세서가 공유하는 다중 프로세서 시스템으로서, 입출력 부시스템을 포함하고 있

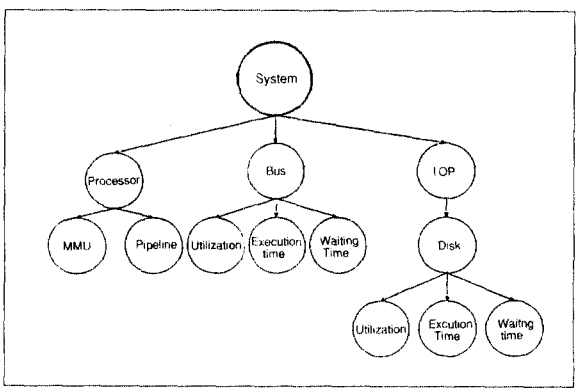


〈그림 4-1〉 다중 프로세서 구조

다. 기본적인 구성은 〈그림 4-1a〉와 같으며, 기본적인 계층적 구성 모델은 〈그림 4-1b〉와 같다. 본 연구에서 고려하는 작업부하로서의 프로세서의 확장에 따른 시스템의 계층적 구성은 〈그림 4-2〉와 같다.

대상 시스템의 구조에서, 구조적 확장 따른 작업부하의 특성은 모든 프로세서가 동일한 작업 부하를 갖는다. 즉, 프로세서의 수가 n배 확장되어도 각 프로세서의 작업부하는 동일하나, 전체 작업부하는 n배로 증가된다. 운영적인 특성으로 각 프로세서는 지역 메모리에서 미스되면 공유메모리를 참조하고, 공유 메모리 폴트 또는 명시적 입출력 요청시에는 컨텍스트 전환을 수행하며, 충분한 수의 컨텍스트가 대기한다고 가정한다. 지역 메모리의 미스율과 메모리 폴트율은 작업부하의 인수가 되면, 실 작업부하는 smplE을 활용한 기존의 연구[15]에서 분석된 작업부하와 유사한 환경을 기준으로 하여 프로세서를 작업부하로 하여 프로세서의 수를 2 개씩 증가하면서 26개 까지 확장하였다.

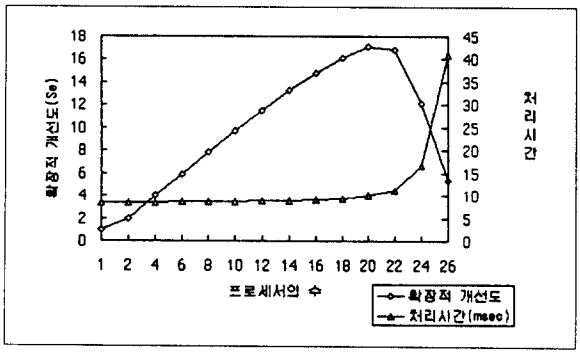
프로세서의 수 : 1 에서부터 확장
입출력 프로세서의 수 : 1
디스크의 수 : 4
명령당 메모리 참조 비율 : 1.2
지역 메모리 미스율 : 10%
공유메모리 폴트율 : 1%
페이지의 크기 : 1 Kbyte



〈그림 4-2〉 시스템의 계층적 모델

6) 분석 결과

smplE을 활용한 [15]와 유사한 환경에서 smplC^{++}를 통해 대상 시스템을 분석한 결과를 프로세서 수의 증가에 따른 확장적 개선도와 처리 시간에 대해 정리한 결과는 다음과 같다.



〈그림 4-3〉 분석 결과

위의 결과는 확장적 작업부하에서 분석되었으므로 Amdahl의 모델과는 직접 비교할 수는 없다. 또한 Gustafson의 모델과는 달리 시스템의 과부하에 의한 성능 영향이 고려되었으므로 프로세서 수에 따라 처리 시간이 일정하지 않다. 즉, 구조 및 작업부하의 확장에 따라 처리 시간은 과부하에 의해 증가하게 되는데, 이를 다시 작업부하의 양으로 정규화 하였음으로 처리 시간과 확장적 개선도를 비교하여 시스템의 작업 처리 능력의 분석이 가능해진다. 그림에서 프로세서의 수가 20까지는 확장적 개선도가 증가하다가 그 후에는 감소한다. 확장적 개선도가 감소하는 것은 시스템의 처리량은 증가하더라도 구조/작업부하의 불균형등이 성능에 대한 장애가 됨을 나타낸다. 따라서, 제시된 구조와 작업부하의 확장성에 따른 최적 성능 효율은 프로세서의 수가 20개인 경우임을 알 수 있다.

이 상황에서 앞에서 제시한 바와 같이 분해된 성능 요소의 분석을 통해 성능 저해 원인을 분석하여 이의 개선이 가능해진다. 본 논문의 목적상 상세한 하부 구조의 성능 분석 및 개선안은 여기서 다루지 않는다.

4.2 클라이언트/서버 시스템

클라이언트/서버 시스템에서 네트워크 화일 서버(NFS)의 처리에 대한 성능을 분석한다. 이를 위하여 작업부하로서 클라이언트의 다양한 NFS요청 (작업부하)의 종류 및 특성을 조사하고, 이를 활용하여 클라이언트 수 및 요청율의 증가에 따른 시스템 확장에 대한 서버 및 통신로에서의 영향을 분석한다.

1) 클라이언트/서버 시스템에서의 처리 시간

클라이언트/서버 시스템에서 대표적으로 사용되는 성능 측도인 처리시간은 클라이언트에서 관측한 요청에 대한 처리시간을 나타내는 서버/네트워크에 대한 성능 측도이다[16].

클라이언트의 요청에 대한 처리는 서버의 구조/처리 능력과 네트워크의 특성/처리능력에 따라 정해지므로 처리시간 Tc/s는 다음과 같이 나타낼 수 있다.

$$Tc/s = 1/Xs + Dn(= Ms/Tn) + Qn$$

Xs : 서버의 처리율
Dn : 네트워크에서 발생되는 지연
Ms : 네트워크 시스템을 통해 전송되는 평균 전송량의 크기
Tn : 네트워크의 전송률
Qn : 시스템 과부하

2) 네트워크 화일 서버(NFS)시스템에서의 작업부하의 분석

각 클라이언트가 동일한 작업부하를 갖도록 하고, 클라이언트의 수에 비례하여 작업부하를 확장하도록 하기 위해, 클라이언트/서버 구조에서의 네트워크 화일 서버(NFS)의 작업부하 분석 자료를 활용하였다. NFS의 작업부하 분석은 실제 시스템에서의 측정 과 NFSSTONE 벤치마크 프로그램의 측정을 비교 제시되었으며[12], 또한 Sun 3/50 클라이언트 상에서 SUE라는 모니터를 사용하여 NFS 연산에 대해 6주동안 측정된 결과가 제시되었다[13].

[13]에서는 클라이언트의 작업부하를 생성하기 위해 클라이언트의 요청을 제어 요구와 최종(End)요구로 구별하고, 실제 제어 요구는 최종 요구를 지원하기 위해 발생하므로 최종 요구만으로 다음 표와 같이 클라이언트의 요청을 제시하였다. 실제 발생 빈도수가 매우 적은 것(1% 미만)들은 생성 작업부하에서 제외시키도록 한다.

표를 분석하면, 클라이언트의 요청은 순수 자료의 요청과 화일/디렉토리의 속성 자료의 요청으로 분류된다. 이 자료 요청들의 신속한 처리를 위해 서버에 각각 버퍼캐쉬 및 DNLC(directory name lookup cache) 라는 캐쉬를 두며, 특히 화일/디렉토리 속성 자료의 요청은 네트워크에의 작업부하를 감소시킨다.

NFS 요청처리에 필요한 읽기/쓰기의 크기 속성은 다음 표와 같이 분석되었다[13].

〈표 4-2〉에서의 자료를 보면 대부분 읽기/쓰기를 위한

〈표 4-1〉 End 요구들의 빈도 분산

요 청	측정치	조정치
read	62.44	63.32
write	20.26	20.54
create	4.16	4.21
remove	1.71	1.7
rename	0.70	-
symlink	0.14	-
link	0.42	-
mkdir	0.02	-
readdir	10.13	10.25
rmdir	0.01	-

〈표 4-2〉 NFS에서의 읽기/쓰기 비율

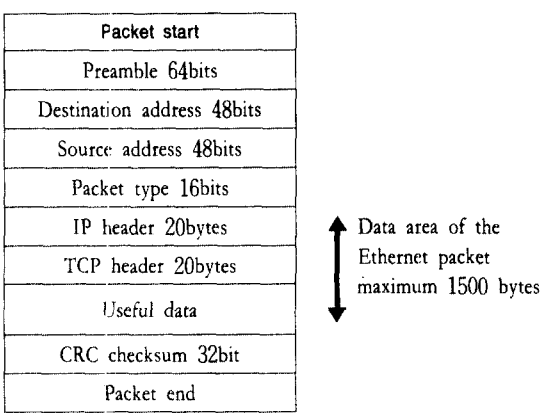
크기 (Bytes)	% Read	% Write
〈 500	29.62	6.74
501-8000	6.65	19.21
8001-8192	63.73	74.05
〉 8192	0	0

전송 크기는 8001-8192 Byte가 대부분 임을 알 수 있는데 NFS의 논리적인 패킷의 크기가 8192 Byte이기 때문이다.

3) 네트워크 상에서 NFS 처리

본 연구에서 고려하는 네트워크의 전송 자료의 구성은 다음 그림과 같다.

〈그림 4-4〉 TCP/IP를 사용한 Ethernet 패킷의 구조



4) 대상 시스템의 구성 및 NFS 처리

본 분석에서 고려되는 시스템의 구조는 다음과 같다.

클라이언트의 NFS 요청들은 poisson 분포를 갖으며, 이 때 클라이언트에서의 요청 발생 주기는 실험적으로 0.507초라고 제시되어[13] 이를 활용할 수 있다. 하지만 서버와 네트워크의 처리 시간의 분포는 환경에 의해 정해지는데 일반적으로 클라이언트/서버 시스템은 M/M/1으로 모델하나[16] 이는 여러 문제점을 초래한다. 즉, 서버와 네트워크를 하나의 서버로 처리하므로 실제 각 부분에서의 성능 병목 및 과부하를 구하기 어렵고, 앞에서 제시한 표와 자료로부터 서버와 네트워크의 처리 시간이 negative exponential분포를 갖는다고 입증할 수없다. 또, 네트워크와 서버를 M/M/1 큐 또는 M/D/1 및 M/M/1 큐의 네트워크로 처리하는 것도 문제가 된다. 따라서, 앞장에서 제시한 객체지향 시뮬레이터가 이 경우에 좋은 해결책이 될 수 있다.

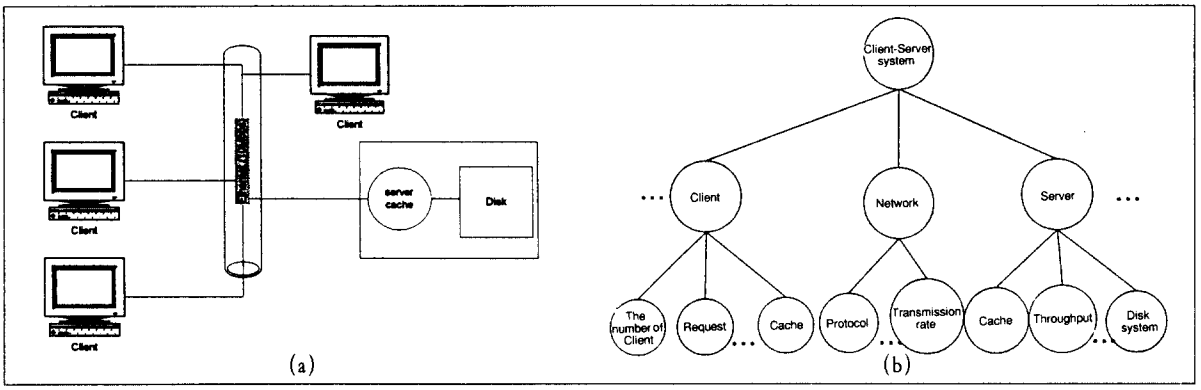
앞에서 제시한 구조에 다음과 같이 동작하는 동기 쓰기(synchronous write)방식을 고려한다.

① 클라이언트의 쓰기 요청시 서버는 안정된 기억 장치에 쓰기 전에는 쓰기 요청의 응답을 주지 않는다.
② 클라이언트가 자료를 수정 한후 쓰기 요청을 보낼 때, 클라이언트 캐쉬는 수정되며, 클라이언트는 수정된 내용을 서버에게 바로 쓰기 요청을 보낸다.
③ 서버의 처리 시간을 계산할 때 네트워크 프로토콜로 발생하는 지연은 무시한다. 단 요청이 발생할 때 생기는 네트워크의 전송 지연은 추가한다.
④ 쓰기 요청 처리 시간은 클라이언트가 서버에 요청을 보낸 후 요청을 처리했다는 응답이 올 때까지를 서버의 쓰기 요청 처리 시간으로 간주한다.

5) 분석 결과

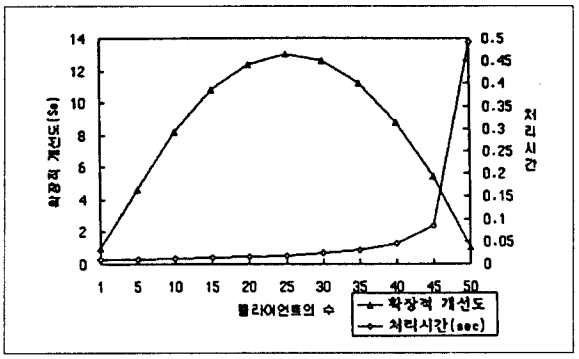
NFS에서의 클라이언트의 수에 비례하여 서버에 대한 NFS 요청의 수가 확장할 때 발생하는 처리시간 및 확장적 개선도의 변화는 〈그림 4-6〉과 같고, 이의 분석 환경은 다음과 같다.

- 서버 디스크 캐쉬 접근 시간 : 40 ns
- 서버의 디스크 접근 시간 : 20ms
- 서버의 디스크 캐쉬 히트 율 : 85%
- 캐쉬 라인의 크기 : 32bytes
- 디스크 블럭의 크기 : 8kbyte



〈그림 4-5〉 네트워크 기반의 서버/클라이어트 모델

- 네트워크의 전송률 : 10 Mbits / sec
- 클라이언트의 요청율(λ) : 1/0.507 요청/초



〈그림 4-6〉 분석결과

이 경우는 클라이언트의 수에 비례하는 작업부하의 증가에는 처리 능력의 확장이 동반되지 않음이 앞의 경우와는 다른 경우이다. 단순하게는 작업부하의 증가에 따라 처리 시간은 증가하지만 처리되는 작업의 양은 증가한다. 이를 확장적 개선도가 종합적으로 분석하고 있다. 즉, 확장적 개선도가 상승하는 경우는(클라이언트의 수가 25이하인 경우) 서버 및 통신로의 처리 능력이 작업 부하를 초과하는 경우로 이 경우에는 작업부하의 증가에 의해 시스템의 성능적 효율성은 지속적으로 증가한다. 하지만 확장적 개선도가 감소되는 경우(클라이언트의 수가 25 이상인 경우)는 작업부하의 증가에 의해 시스템의 과부하가 증가하여 시스템의 최적 확장 상태를 초과하였음을 나타낸다. 이 경우에는 시스템의 각 요소에 대한 병목현상을 분석하면 그 원인을 알 수 있다.

물론, 확장적 개선도가 최고인 경우(클라이언트의 수가 25인 경우)가 주어진 구조에서 작업을 가장 효율적으로 처리할 수 있는 상황이다.

5. 결 론

본 연구는 많은 응용 분야에서 요구되는 시스템의 고성능을 충족시킬 수 있는 구조 및 작업부하의 확장성의 의의와 이의 성능 개선을 표현할 새로운 성능 측도로 단위 작업당 처리 시간의 비인 확장적 개선도를 제시하고, 확장적 시스템을 표현하고 분석하기에 적합한 계층적 모델링을 소개하고, 이러한 개념으로 실제 성능 분석에 적용할 객체 지향적 시뮬레이터인 smplC++의 개발을 제시하였다. 성능 측도는 구조 및 작업부하의 확장성에 의한 성능 개선과 시스템의 과부하에 의한 성능 저하가 모두 포함되도록 선정되어 최적의 확장 상태를 구할 수 있다. 시뮬레이터는 확장적 시스템의 계층적 성능 모델의 시뮬레이션의 구축 작업을 용이하도록 객체 지향적으로 구현하였으며 시뮬레이터의 일부 입출력의 자동화 부분은 현재 구현 중에 있다.

실 작용 예로 분석적 모델이 복잡하거나 측정이 어려운 다중 프로세서 시스템의 구조 및 작업부하의 확장에 따른 분석과 네트워크 기반의 서버/클라이언트 시스템의 클라이언트 수의 증가에 따른 분석을 논문에서 제시한 성능 측도, 확장적 구조/작업부하 및 계층적 성능 모델과 시뮬레이터 smplC++로 실시하였다. 구조 또는 작업부하의 증

가에 따른 확장적 개선도(scalable speedup)를 구하여, 확장성에 의한 성능 영향을 분석하고 작업부하/구조의 최적의 균형 상태와 성능 효율 변화를 분석할 수 있다.

본 논문의 주목적은 구조나 운영의 새로운 제안이 아닌 확장성에 대한 성능 측도, 분석 방법 및 시뮬레이터의 개발이었으며, 여러 분야의 확장적 시스템에 대한 실제 성능 개선에 본 연구의 활용이 실시될 것이다.

참고문헌

[1] Kai Hwang, *Advanced Computer Architecture Parallelism Scalability Programmability*, McGraw-Hill, 1993.

[2] Gorden Bell, "Ultracomputer: A Teraflop Before Its Time," *Communications of ACM*, Vol 35, No 8, 1992, pp. 27-47.

[3] A. Sivasubramaniam et al., "An Approach to Scalability Study of Shared Memory Parallel Systems," *Proceedings of SIGMETRICS 1994*, May 1994,pp. 171-180.

[4] John L. Gustafson, "Reevaluating Amdahl's Law," *Communications of ACM*, Vol 31, No 5, 1988, pp. 532-533.

[5] Janaki Akella and Daniel P. Siewiorek, "Modeling and Measurement of the Impact of Input and Output on System Performance," *Proceedings of International Symposium on Computer Architecture*, 1991, pp. 390-399.

[6] C. Natarajann et al., "Measurement-based Characterization of Global memory and network Contention, Operating system and parallelization Overheads: A Case Study on a Shares-Memory Multiprocessor," *Proceedings of International Symposium on Computer Architecture*, 1994, pp. 71-80.

[7] Kyungsan Cho et al., "An Intelligent Simulation Environment For Computer System Design," *Simulation Series*, Vol. 22, No. 3, April 1990, pp. 59-64.

[8] E. D. Lazowska, *Quantitative System Performance Computer System Analysis Using Queuing Network Models*, Prentice-Hall, Inc., 1984.

[9] Xiaodong Zhang et al, "Performance Predictions in Implicit Communication Systems," *Proceedings of Parallel and Distributed Processing*, 1994, pp. 560-568.

[10] D. L. Eager et al., "Speedup Versus Efficiency in Parallel System," *IEEE Trans. Computers*, vol. 38, No. 3, 1989, pp. 408-423.

[11] Jim Gray, *The Benchmark Handbook*, Second Edition, 1993.

[12] Barry Shein and Mile Callahan, "NFSSTONE: A Neteork File Sever Performance Benchmark," *Summer USENIX Conference Proceedings*, 1989, pp. 269-275.

[13] Roberta R. Bodnarchuk and Richard B. Bunt, "A Synthetic Workload Model for a Distributed System File Server," *Proceedings of SIGMETRICS 1991*, Vol. 19, No 1, May 1991. pp. 50~59.

[14] M.H. MacDougall, *Simulating Computer Systems - Techniques and Tools*, The MIT Press, 1987.

[15] 조 성만, 조 경산, "시뮬레이션 도구 smplE의 개발 및 멀티프로세서 시스템 성능 분석에의 활용," 「한국 시뮬레이션학회 논문지」, Vol. 1, No. 1, 1992, pp. 87-102.

[16] Paul E. Renaud, *INTRODUCTION TO CLIENT/SERVER SYSTEMS*, Wiley, 1993, pp. 281-283.

● 저자소개 ●

조경산
1979년 서울대학교 전자공학과 졸업(학사)
1981년 한국과학원 전기 및 전자공학과 졸업(공학석사)
1988년 텍사스대학교(오스틴 소재) 전산·전기공학과 졸업(Ph.D.)
1988-1990년 삼성전자 컴퓨터부문 책임연구원
1990-현재 단국대학교 전산통계학과 부교수
관심분야 : 컴퓨터 구조, 성능분석, 병렬구조, 시뮬레이션

김홍준
1989년 단국대학교 전자계산학과 졸업(학사)
1993년 단국대학교 전산통계학과 졸업(석사)
1993년-현재 단국대학교 전산통계학과 박사과정
관심분야 : 컴퓨터 구조, 성능분석, 병렬구조, 객체지향 시뮬레이션

안효범
1992년 단국대학교 전자계산학과 졸업(학사)
1994년 단국대학교 전산통계학과 졸업(석사)
1994년-현재 단국대학교 전산통계학과 박사과정
관심분야 : 컴퓨터 구조, 성능분석, 병렬구조, 분산시스템