임계집합을 이용한 컴퓨터 시스템 성능향상 투자배분*

Budget Distribution for Computer System Performance Enhancement Using Critical Sets

박기현**, 장명숙***, 최준구****
Park, Kee Hyun · Jang, Meung Sook · Choi, Jun Gu

Abstract

Performance measurement and analysis for computer systems has been studied for a long time. However, little attention has been focused on how to distribute a given budget to each part of a computer system for enhancing system performance maximally. In traditional approaches, performance enhancement is achieved by identifying a dominant system bottleneck and enhancing the bottleneck's performance. These approaches, however, often bring problems since the removal of a bottleneck may result in other bottlenecks.

This paper proposes an approximated method for such budget distribution problem. For budget distribution, a critical set is defined. The set contains the servers of which performance changes affect the overall system performance significantly. Then, the given budget is distributed properly to every server in the critical set. To verify the proposed method, two benchmark experiments are carried out in SLAM environments. The experimental results show that the proposed method provides better results than the traditional method does in many cases.

1. 서 론

컴퓨터를 비롯한 시스템의 성능을 분석하고 예측하는 연구는 오래 전부터 계속되어 왔다. 큐잉분석과 같은 해석적인 연구 [1, 4, 9, 11]에서 부터 Petri net을 이용한 연구 [2, 3], 시뮬레이션을 이용한 연구 [6, 12, 13, 14, 15]에 이르기까지 다양한 접근방법들이 사용되어 왔다. 또한 성능분석 도구들도 개발되어 예전보다 훨씬 편리하게 시

* 이 연구는 1994년도 한국과학재단 핵심전문 연구비 지원에 의한 결과임. (과제번호: 941-0900-062-1)

** 계명대학교 전자계산학과 부교수.
*** 경북대학교 컴퓨터공학과 박사과정.
**** 계명대학교 전자계산학과 박사과정.

스템의 성능을 분석하거나 예측할 수 있게 되었다 [7, 8, 16].

그러나 이런 방법이나 도구들은 모두 처리율(throughput), 효율(utilization) 및 대기시간(waiting time) 등과 같은 일차원적인 성능수치를 구하는데 그친다. 이런 수치들은 성능향상 대책을 수립하는데 중요한 자료가 되지만, 아직까지 이들을 체계적으로 이용하여 효율적인 성능향상을 꾀하는 연구는 미흡하다. 성능향상을 위한 일반적인 방법은 성능분석을 통하여 얻은 성능수치로써 병목(bottleneck)을 찾은 후, 단순히 이 부분의 성능을 높혀주므로써 시스템 전체의 성능향상을 꾀하는 병목분석(bottleneck analysis)이 대부분을 차지한다 [6, 10].

문제는 국지적인 성능향상만으로 전체적인 성능향상을 꾀하는 것에는 한계가 있다는 것이다. 한 부분의 병목제거는 다른 부분의 새로운 (때로는 더 심각한) 병목을 유발시킬 수 있기 때문에 시스템 전체적으로는 성능향상의 효과가 그다지 크지 않을 수도 있기 때문이다. 예를 들어 그림 1과 같은 간단한 큐잉 네트워크를 생각해 보자. 이 시스템에서는 2개의 서버(server)들 (즉, CPU 및 DISK)이 큐잉 네트워크를 구성하고 있다. 시스템으로의 평균 도착간격은 2.0이다. CPU (DISK)의 평균 실행시간(execution time)은 평균 0.3 (0.8)으로서 지수분포를 이루고 있다. CPU (DISK)에서 DISK (CPU)로 가는 확률은 1.0 (0.2)이다. 시뮬레이션 도구인 SLAM [15]을 이용하여 계산하면 CPU의 효율은 0.17이고 DISK의 효율은 0.45이므로 DISK가 병목이 된다.

만약에 단순히 병목부분의 성능을 3배 더 향상 (즉, 실행비율(execution rate)을 4배로 증가)시켜서 DISK의 병목을 제거하고 전체적인 성능향상을 꾀한다면 전체 시스템 시간(system time)은 0.66이 된다. (시스템 시간은 시스템에 들어가서 나올때까지의 평균 지연시간이다.) 병목부분의 성능을 3배 만큼이나 더 향상시켰으나 시스템 전체적으로는 그다지 큰 성능향상을 보지 못했다. 왜냐하면 DISK의 성능향상이 CPU를 새로운 병목으로 만들어서 체증을 일으키기 때문이다. 따라서 CPU에 대해서 추가로 성능을 향상시켜야 되고, 시스템이 복잡할수록 이런 일이 많이 되풀이 된다. 만약에 한 부분에 대해서 집중적으로 성능향상을 위해 투자하는 대신에, CPU를 0.7배, DISK를 2.3배 더 성능을 높힌다면 전체 시스템 시간은 0.54가 되므로, 전체적으로 성능을 크게 향상시킬 수가 있다.

만약에 컴퓨터 시스템 설계자나 관리자가 병목부분만을 계속 찾아다니면서 제거하려 한다면, 체계적인 성능향상 대책 부재로 인해서 시간과 경비를 허비하는 셈이 된다. 그러므로 이러한 국지적인 성능향상 방법에서 야기되는 불합리한 점들을 개선하여 보다 효율적이고 전체적인 성능향상 대책을 수립할 수 있도록 하는 방법에 대한 연구가 필요하다.

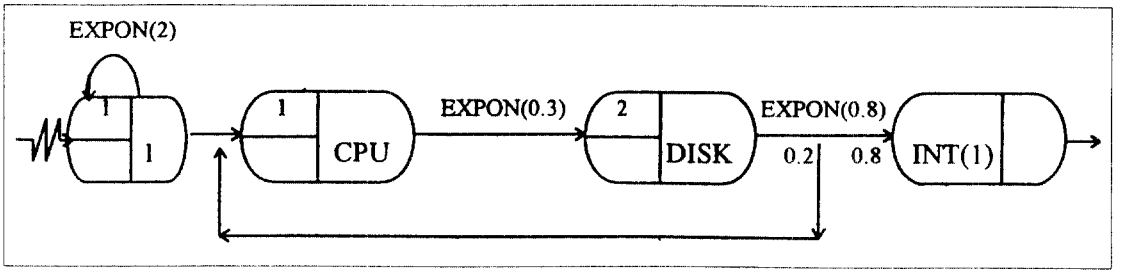
따라서 본 연구에서는 한 두 곳의 처리능력을 대폭적으로 향상시키는 대신에, 주요한 여러 곳들의 처리능력을 동시에 소폭 향상시키므로써 경제적인 성능향상을 꾀할 수 있는 방법을 제시한다. 즉, 본 연구는 "고정된 예산을 시스템의 어떤 부분들에게 어떻게 분배하여 이들의 성능을 향상시켜야 전체적으로 성능향상에 관한 최선의 효과를 얻을 수 있겠는가?" 하는 물음에 대한 근사적인 대답을 제시하고자 한다. 이를 검증하기 위해서 시뮬레이션 도구인 SLAM을 이용한다.

본 연구의 구성은 다음과 같다: 2절에서는 본 연구에서 사용될 시스템 환경에 대해서 설명한다. 성능향상 분배방법은 3절에서 제시되며, 그 결과 및 분석은 4절에서 논의된다. 마지막으로 5절은 결론 및 앞으로의 연구방향에 대해서 언급한다.

2. 시스템 환경

본 연구에서 사용되는 시스템은 큐잉 네트워크로 표현한다. 큐잉 네트워크는 성능을 분석하기 위하여 시스템을 표현하는 일반적인 표현방법이다. 큐잉 네트워크의 시작부분에서 모든 서버들에게 도달될 수 있고 각 서버에서 종단노드(terminate node)로 도달될 수 있다고 가정한다. 본 연구에서 사용하는 기본적인 자료는 큐잉 네트워크의 구성 및 각 서버에 대한 효율이다. 효율은 시뮬레이션 도구인 SLAM을 통하여 얻은 결과를 사용한다. 큐의 길이나 대기시간 등을 효율과 함께 사용하면 좀 더 나은 성능향상 방법을 얻을지도 모르지만, 이런 성능수치들은 직접 측정하기가 쉽지 않다. 따라서 본 연구에서는 SLAM으로 부터 얻을 수 있는 다양한 성능수치들 중에서, 현장에서도 측정하기 쉬운 효율만을 사용하므로써 실용적인 측면도 고려하고자 한다.

임의로 정한 대상 시스템들은 〈그림 2〉와 〈그림 3〉에 SLAM 방식으로 표현되어 있다. 그림 2에서는 CPU, DI



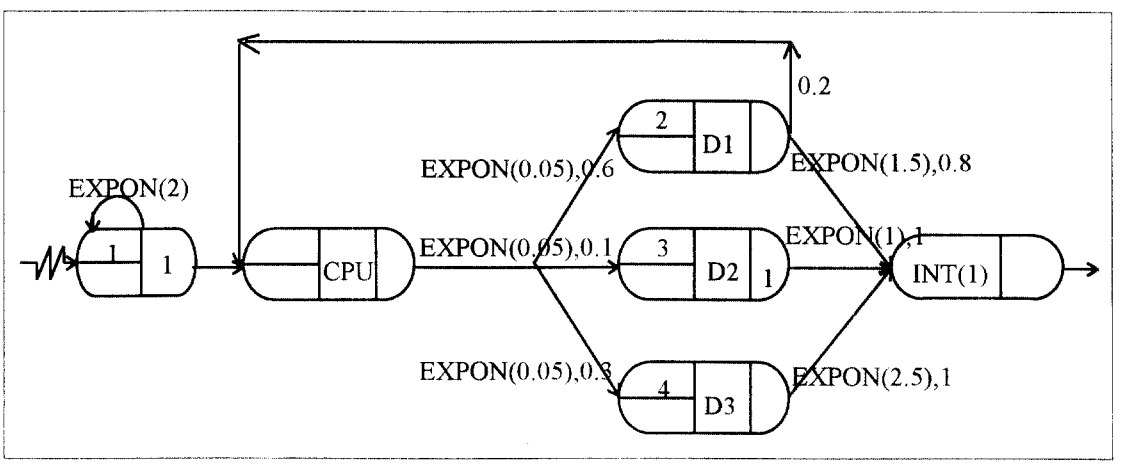
〈그림 1〉 시스템 1의 큐잉 네트워크

(DISK1), D2(DISK2), D3(DISK3)의 4개 서버들로 이루어진 시스템이다. 시스템으로의 평균 도착간격은 2.0이다. CPU, D1, D2, D3의 평균 실행시간은 각각 0.05, 1.5, 1.0, 2.5이고 이들은 모두 지수분포를 이룬다. CPU에서 D1, D2, D3에 이르는 확률은 각각 0.6, 0.1, 0.3이다. 이 시스템에서는 D1에서 CPU로의 확률 0.2의 피드백(feedback)이 있다. 그림 3은 C1(CPU1), C2(CPU2), C3(CPU3), D1(DISK1), D2(DISK2), D3(DISK3)의 6개 서버들로 이루어진 시스템이다. 시스템으로의 평균 도착간격은 2.0이다. C1, C2, C3, D1, D2, D3의 평균 실행시간은 각각 0.5, 1.5, 1.0, 1.5, 1.0, 1.5이고 이들은 모두 지수분포를 이룬다. 이 시스템에서는 D1(D3)에서 C1(C3)로의 확률 0.2(0.1)의 피드백이 있다.

서버에 x배 더 투자한다는 것은 그 서버의 평균 실행비율을 (1+x)*m으로 향상시키는 것을 의미한다.

예를 들면, 〈그림 2〉에서 평균 실행시간이 1.5인 D1에 3배 더 투자한다는 것은, DISK의 평균 실행비율이 0.67 (= 1/1.5)이므로, 평균 실행비율을 2.68 (= (1+3)*0.67)로 높히는 것을 의미하며 따라서 평균 실행시간은 0.37 (= 1/2.68)로 짧아짐 (혹은 향상됨)을 뜻한다.

1절에서 국지적인 성능향상보다 전체적으로 여러 곳의 성능을 조금씩 향상시키는 것이 바람직하다는 예를 보았다. 그러나 복잡한 큐잉 네트워크에서는 수많은 서버들이 있으므로 서버들을 모두 고려하여 조금씩 성능향상을 이룩하는 것은 바람직하지 않다. 그런데 어떤 서버의 성능을 변화시키면 관련되는 다른 서버들이 영향을 받을 수

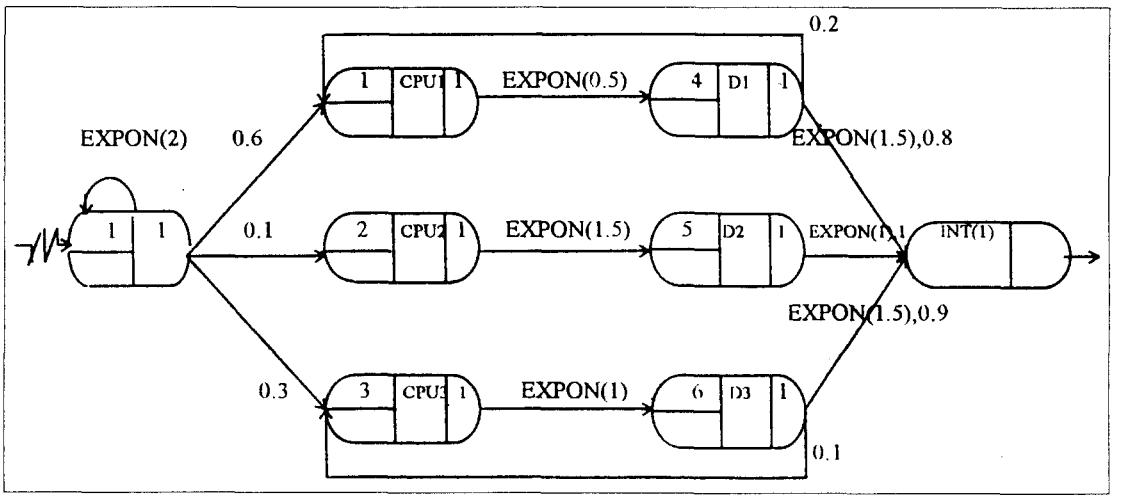


〈그림 2〉 시스템 2의 큐잉 네트워크

3. 성능향상 분배방법

[정리 1] 평균 실행비율(execution rate)이 m인 어떤 서

있으므로 각 서버가 다른 서버들에게 영향을 미치는 정도를 정의하는 것이 필요하다.



〈그림 3〉 시스템 3의 큐잉 네트워크

[정의 1] Flow 행렬 F의 요소 F (i,j)는 주어진 큐잉 네트워크에서 서버 i의 성능변화가 서버 j에게 미치는 영향의 정도를 나타낸다. 즉, $F = \sum_{i=1}^{\infty} A^i$이다. 여기서 A는 주어진 큐잉 네트워크에 관한 전이확률행렬(transition probablity matrix)이다.

예를 들면, 그림 2의 시스템에 관한 전이확률행렬 A와 flow 행렬 F는 〈그림 4〉와 같다. 즉, 그림 4의 flow 행렬 F를 보면, CPU에서의 흐름이 궁극적으로 CPU에는 0.14만큼, D1에는 0.68만큼 영향을 미친다. 다시 말하면, CPU의 성능을 향상시킬 경우, 영향을 제일 많이 받는 D1의 성능도 함께 향상시키지 않는다면 D1에 병목현상이 발생할 가능성이 많아서 전체적으로 성능향상의 효과가 크지 않을 수도 있다. 또한 D1의 성능을 향상시킬 때도 CPU의 성능을 어느 정도 같이 향상시켜야 한다는 것도 알 수 있다.

Flow 행렬 F는 알고리즘 1을 이용하여 구할 수 있다.

```
/* A는 주어진 큐잉 네트워크의 전이확률행렬                    */
/* B, C, 그리고 D는 A와 같은 크기의 행렬                      */
  B = D = A
  repeat
    begin
      C = A X B
      D = D + C
      B = C
      end
  until (no non-zero elements in C)
```

알고리즘 1. 큐잉 네트워크를 위한 flow 행렬 F 계산 알고리즘

알고리즘 1은 주어진 큐잉 네트워크의 전이확률행렬을 이용하여 계산하는데, n번째 행렬 곱셈으로 얻은 C(i,j)는 서버 i에서 (n+1)번 거쳐서 서버 j로 갈 수 있는 확률을 나타낸다. 따라서 2절에서 모든 서버에서 종단노드까지 도달할 수 있는 큐잉 네트워크를 가정했으므로, 알고리즘 1

$$A = \begin{array}{c} \\ \text{CPU} \\ \text{D1} \\ \text{D2} \\ \text{D4} \end{array} \begin{array}{c} \begin{array}{cccc} \text{CPU} & \text{D1} & \text{D2} & \text{D3} \end{array} \\ \begin{bmatrix} 0.00 & 0.60 & 0.10 & 0.30 \\ 0.20 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix} \end{array} \qquad F = \begin{array}{c} \\ \text{CPU} \\ \text{D1} \\ \text{D2} \\ \text{D4} \end{array} \begin{array}{c} \begin{array}{cccc} \text{CPU} & \text{D1} & \text{D2} & \text{D3} \end{array} \\ \begin{bmatrix} 0.14 & 0.68 & 0.11 & 0.34 \\ 0.23 & 0.14 & 0.02 & 0.07 \\ 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix} \end{array}$$

〈그림 4〉 시스템 2에 대한 행렬 A와 행렬 F

의 곱셈이 수렴한다는 것은 당연하다.

일반적으로 큐잉 네트워크에는 많은 서버들이 있으므로 이들 모두의 성능을 변화시키려는 것은 바람직하지 않다. 따라서 시스템 전체의 성능에 많은 영향을 미치는 서버들을 가려내는 일이 필요하다. 이런 중요한 서버들의 집합을 임계집합(critical set)이라고 정의한다.

[정의 2] 주어진 큐잉 네트워크에서 서버 n의 k-임계집합, k-CSET(n)은, 자기자신을 포함하여, 그 큐잉 네트워크에서 서버 n에 영향을 미치는 서버들과 서버 n에 의해서 영향을 받는 서버들 중에서 가장 영향이 큰 k개의 서버들의 집합이다. 자기자신은 스스로에게 가장 큰 영향을 주고받는다고 규정한다. 즉, k-CSET(n)은 $F(n,j)$와 $F(i,n)$에서 가장 큰 값을 가진 (k-1)개의 서버 i 혹은 서버 j들의 집합에 서버 n이 포함된 집합이다.

k-임계집합은 flow 행렬 F를 이용하여 구할 수 있다. 예를 들면 그림 2의 시스템에서는 그림 4의 flow 행렬이 보여주는 것 처럼, CPU의 1-CSET(CPU)는 {CPU}이고 2-CSET(CPU)는 {CPU, D1}이다. 일단 flow 행렬과 임계집합이 결정되면 알고리즘 2는 주어진 예산범위에서 성능향상을 위하여 각 서버에 대한 투자분배를 결정한다.

시뮬레이션 도구인 SLAM을 이용하여 평균 도착간격이 2.0인 경우에 그림 2의 시스템을 시뮬레이션하면 CPU, D1, D2, D3의 효율은 각각 0.03, 0.54, 0.07, 0.44이고 시스템 시간은 3.66이다. 이 시스템에서 효율이 평균을 상회하는 서버는 D1과 D3이므로 1-CSET(D1) = {D1}, 1-CSET(D3) = {D3}, 2-CSET(D1) = {CPU, D1}, 2-CSET(D3) = {CPU, D3}가 된다. 만약에 FUND = 3.00이고 k = 1이면, 알고리즘 2에 의해서, BUDGET(D1) = 1.65 (= 3*0.54/(0.54+0.44)), BUDGET(D3) = 1.35가 되어 D1과 D3에만 효율에 비례하여 투자된다. 이럴 경우에는 D1의 평균 실행비율은 1.77 (= (1+1.65)*1/1.5)이 되어서 DISK의 평균 실행시간은 0.56으로 짧아진다. D3의 평균 실행시간은 1.06이 된다.

이 값들을 이용하여 SLAM을 실행시켜 보면 투자후의 시스템 시간으로 1.03을 얻는다. 즉, 3배의 투자를 D1과 D3에 각각 1.65, 1.35배 투자하면 상대적으로 71.9% 향상된 시스템 시간을 얻을 수 있다. 참고로 이 시스템의 최고 병목인 D1에만 3배 집중투자하면 1.58의 시스템 시간 밖에 얻지 못한다. 만약에 FUND = 3.00이고 k = 2이면, CPU, D1, D3에 투자되며 BUDGET(CPU) = 0.09, BUDGET(D1) = 1.60, BUDGET(D3) = 1.60이 된다. SLAM을 실행시켜 투자후의 향상된 시스템 시간으로 1.08을 얻는다.

```
/* NSERV는 주어진 큐잉 네트워크에서의 서버 수 */
/* F는 주어진 큐잉 네트워크의 flow 행렬 */
/* UTIL(i)는 서버 i의 효율 */
/* FUND는 투자될 총 액수 */
/* BUDGET(i)는 서버 i를 위한 투자분배량 */
  BUDGET(i) = 0 for i = 1,...,NSERV
  AVGUTIL = average of UITL(i)s for all i = 1,...,NSERV
  COUNT = UNDER = 0
  THRESHOLD = AVGUTIL
  repeat
    begin
      COUNT = COUNT + 1
      find the next largest server BOTTLENECK in terms of utilization
      if (COUNT > NSERV or UTIL(BOTTLENECK) < THRES-
        HOLD)
        then exit the loop
      obtain k-CSET(BOTTLENECK)
    end
  until false
  for every BOTTLENECK
    for every i in k-CSET(BOTTLENECK)
      UNDER = UNDER + UTIL(i)
  for every BOTTLENECK
    for every i in k-CSET(BOTTLENECK)
      BUDGET(i) = FUND * UTIL(i) / UNDER
```

알고리즘 2. 주어진 예산범위내에서의 각 서버를 위한 투자분배 결정 알고리즘

4. 결과 및 분석

시스템 2와 시스템 3에 대한 알고리즘 2와 SLAM의 실행결과는 각각 〈표 1〉과 〈표 2〉에서 볼 수 있다. 〈표 1〉을 살펴보면, 평균 도착간격이 2.0일때 시스템 2의 기본적인 시스템 시간은 3.66이다. 만약에 총 투자가 3배이고 이를 최고 병목인 D1에만 집중투자한다면 시스템 시간으로 1.58을 얻는다. 이는 기본 시스템 시간 3.66에 관해서 상대적으로 56.8%의 향상효과가 있다. 반면에 알고리즘 2를 이용하면 k = 1일 경우에는 최고 병목인 D1과 그 다음

〈표 1〉 시스템 2의 투자분배 및 그 효과

평균 도착간격	총 투자 (배)	k	투자분배 (배)				시스템 시간 (상대적 향상율)
			CPU	D1	D2	D3	
2.0	0.00		0.00	0.00	0.00	0.00	3.66 (0.0%)
	3.00		0.00	3.00	0.00	0.00	1.58 (56.8%)
		1	0.00	1.65	0.00	1.35	1.03 (71.9%)
		2	0.09	1.60	0.00	1.31	1.08 (70.5%)
	1.00		0.00	1.00	0.00	0.00	2.26 (27.3%)
		1	0.00	1.00	0.00	0.00	1.68 (54.1%)
		2	0.03	0.53	0.00	0.44	1.80 (50.8%)
1.0	0.00		0.00	0.00	0.00	0.00	20.2 (0.0%)
	3.00		0.00	3.00	0.00	0.00	3.61 (82.1%)
		1	0.00	1.68	0.00	1.32	1.42 (93.0%)
		2	0.10	1.63	0.00	1.27	1.47 (92.7%)
	1.00		0.00	1.00	0.00	0.00	4.69 (76.8%)
		1	0.00	0.56	0.00	0.44	3.92 (80.6%)
		2	0.04	0.54	0.00	0.42	3.21 (84.1%)
0.5	0.00		0.00	0.00	0.00	0.00	206 (0.0%)
	3.00		0.00	3.00	0.00	0.00	42.7 (79.3%)
		1	0.00	1.50	0.00	1.50	4.14 (98.0%)
		2	0.16	1.42	0.00	1.42	2.90 (98.6%)
	1.00		0.00	1.00	0.00	0.00	84.0 (59.2%)
		1	0.00	0.50	0.00	0.50	94.0 (54.4%)
		2	0.05	0.47	0.00	0.48	128 (37.9%)

〈표 2〉 시스템 3의 투자분배 및 그 효과

평균 도착간격	총 투자 (배)	k	투자분배 (배)						시스템 시간 (상대적 향상율)
			C1	C2	C3	D1	D2	D3	
2.0	0.00		0.00	0.00	0.00	0.00	0.00	0.00	4.66 (0.0%)
	3.00		0.00	0.00	0.00	3.00	0.00	0.00	2.04 (56.2%)
		1	0.00	0.00	0.00	2.10	0.00	0.09	1.81 (61.2%)
		2	0.50	0.00	0.45	1.43	0.00	0.62	1.58 (66.1%)
	1.00		0.00	0.00	0.00	1.00	0.00	0.00	2.80 (39.9%)
		1	0.00	0.00	0.00	0.70	0.00	0.30	2.50 (46.4%)
		2	0.17	0.00	0.15	0.48	0.00	0.20	2.44 (47.6%)
1.0	0.00		0.00	0.00	0.00	0.00	0.00	0.00	34.8 (0.0%)
	3.00		0.00	0.00	0.00	3.00	0.00	0.00	2.52 (92.8%)
		1	0.00	0.00	0.00	2.04	0.00	0.96	2.34 (93.3%)
		2	0.52	0.00	0.45	1.38	0.00	0.65	2.17 (93.8%)
	1.00		0.00	0.00	0.00	1.00	0.00	0.00	3.24 (90.7%)
		1	0.00	0.00	0.00	0.68	0.00	0.32	4.24 (87.8%)
		2	0.17	0.00	0.15	0.46	0.00	0.22	3.98 (88.6%)
0.5	0.00		0.00	0.00	0.00	0.00	0.00	0.00	137 (0.0%)
	3.00		0.00	0.00	0.00	3.00	0.00	3.00	8.22 (94.0%)
		1	0.00	0.00	0.00	1.50	0.00	1.50	8.02 (94.1%)
		2	0.57	0.00	0.57	0.93	0.00	0.93	55.1 (59.8%)
	1.00		0.00	0.00	0.00	1.00	0.00	0.00	29.1 (78.8%)
		1	0.00	0.00	0.00	0.50	0.00	0.50	69.3 (49.4%)
		2	0.19	0.00	0.19	0.31	0.00	0.31	98.0 (28.5%)

병목인 D3에 각각 1.65와 1.35로 분배투자되어서 시스템 시간이 1.03으로 줄게되고 상대적으로 71.9%의 성능향상 효과를 볼 수 있다. k = 2일 경우에는 D1과 D3에 가장 영향을 받는 CPU가 추가되어서 CPU에 0.09, D1에 1.60, D3에 1.31로 분배투자되므로 시스템 시간이 1.08로 되고 상대적으로 70.5%의 성능향상 효과를 볼 수 있다. 총 투자가 1배일 때도 같은 현상을 볼 수 있다.

평균 도착간격이 1.0이고 총 투자가 3배일 때도 기본 시스템 시간 20.2에 비해서 k = 1 및 k = 2 경우, 각각 93.0% 및 92.7%로서, 최고 병목에만 집중투자한 82.1%에 비해서 높은 성능향상 효과를 얻는다. 평균 도착간격이 0.5이고 총 투자가 3배일 때는 알고리즘 2의 결과가 아주 좋음을 알 수 있다. 그러나 총 투자가 1배일 경우에는 오히려 최고 병목에 집중투자하는 것이 바람직하다는 것을 보여준다.

〈표 1〉을 통해서 보면, 일반적으로 알고리즘 2를 사용하는 것이 최고 병목의 집중투자보다 높은 성능향상 효과를 얻음을 알 수 있다. k = 1의 경우가 k = 2인 경우보다 대체로 좋은 결과를 얻는다. 그러나 평균 도착간격이 0.5이고 총 투자가 1.0의 경우와 같이 서버의 효율이 대부분 1.00이면서 총 투자가 적다면 집중투자가 약간 나은 결과를 얻는다. (최고 병목의 집중투자는 알고리즘2의 k와 THRESHOLD 값을 조정하므로써 얻을 수 있다.)

〈표 2〉의 시스템 3에서도 이와 비슷한 결과를 얻는다. 즉, 알고리즘 2에 의한 분배투자가 최고 병목에 대한 집중투자보다 대부분 성능향상 효과가 높음을 시뮬레이션을

통해 알 수 있다. 또한 〈표 1〉과 마찬가지로 평균 도착간격이 0.5이고 총 투자가 1.0의 경우에는 집중투자가 나은 결과를 얻고 있다.

5. 결론 및 앞으로의 연구방향

이 연구는 서버들의 성능향상을 위하여 투자를 어떻게 하는 것이 컴퓨터 시스템 전체 성능을 효과적으로 향상시키는가에 대한 방법을 제시하였다. 이 방법은 시스템의 구성, 자료흐름 및 각 서버의 효율을 바탕으로 각 서버에 성능향상을 위하여 투자할 양을 결정한다. 시스템의 모든 서버들을 대상으로 성능향상을 고려하는 것은 비실용적이므로, 시스템의 성능에 결정적으로 영향을 미치는 서버들을 중심으로 임계집합을 구하고 이들의 성능을 향상시키도록 고려했다.

임계집합을 이용한 이 방법의 성능향상 효과를 검증하기 위하여 SLAM을 이용하였다. SLAM의 실행결과, 대부분의 경우 최고 병목에만 집중적으로 성능향상을 도모하는 것보다 나은 성능향상 효과를 얻었다. 단지 서버의 효율이 1.00일 정도로 체증이 몹시 심하나 소규모 성능향상 투자 밖에 할 수 없는 경우에는 한 곳에 집중투자하는 것이 바람직하다는 것을 알 수 있다.

큐의 길이나 대기시간 등을 효율과 함께 사용하면 좀 더 나은 성능향상 방법을 얻을지도 모르지만, 이런 성능수치들은 컴퓨터 시스템에서 직접 측정하기가 쉽지 않다. 따라서 본 연구는 SLAM으로 부터 얻은 다양한 성능수치들 중에서, 현장에서도 측정하기 쉬운 효율만을 사용하므로써 실용적인 측면도 고려했다.

본 연구를 좀 더 개선하기 위해서는 (1) 알고리즘 2의 적절한 k와 THRESHOLD 값에 대한 결정방법이 모색되어야 하며, (2) 몹시 심한 체증에 비해서 성능향상 투자가 미미할 경우에 대한 보다 효율적인 고려가 필요하다.

참고문헌

[1] Akyildiz, I. F., "Product form approximations for queueing networks with multiple servers and blocking," *IEEE trans. on Computers*, vol. 38, no. 1, Jan 1989, pp. 99-114.

[2] Ajmone-Marsan, M., *et.al.*, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *Computing Surveys*, vol. 2, May 1984, pp. 93-122.

[3] Balbo, G., *et.al.*, "Combining queueing networks and generalized stochastic Petri nets for the solutions of complex models of system behavior," *IEEE Trans. Computers*, vol. C-37, no. 10, Oct. 1988, pp. 1251-1268.

[4] Baskett, K. M., *et.al.*, "Open, closed, and mixed networks of queues with different classes of customers," *J. ACM*, vol. 22, 1975, pp 248-260.

[5] Bruell, S. C., *et.al.*, "A mean value analysis based package for the solution of product-form queueing networks models," *Proc. Int'l Conf. Modelling Techniques Tools Perform. Anal.*, May 1984.

[6] Clymer, J. R., *System Analysis Using Simulation and Markov Models*, Prentice-Hall, NJ, 1990.

[7] Couvillion, J. A. *et.al.*, "Performability Modeling with UltraSAN," *IEEE Software*, vol. 8, no. 5, Sept. 1991, pp. 69-80.

[8] Funka-Lea, C. A. *et.al.*, "Interactive visual modeling for performance," *IEEE Software*, vol. 8, no. 5, Sept. 1991, pp. 58-68.

[9] Kleinrock, L., *Queueing Systems vol. I, II*, New York: Wiley, 1975.

[10] Lazowska, E. D., *et.al.*, *Quantitative System Performance - Computer System Analysis Using Queueing Network Models*, Englewood Cliffs: Prentice-Hall, 1984.

[11] Onvural, R. O., "Survey of closed queueing networks with blocking," *ACM Computing Surveys*. vol. 22, no. 2, June 1990, pp. 83-121.

[12] Sauer, C. M. and MacNair, E. A., *Simulation of Computer Communication Systems*, Prentice Hall, 1983.

[13] Smith, A. J.,"Long term file migration: development and migration of algorithms," *Comm. of ACM*, vol. 24, 1981, pp. 521-532.

[14] Smith, A. J.,"Cache memories," *Computing Surveys*, vol. 14, 1982, pp. 473-530.

[15] Pritsker, A. A. B., *Introduction to Simulation and Slam*

II, Halsted Press, 1986.

[16] Veran, M. and Potier, D., "QNAP 2: a portable environment for queueing networks modelling," *Proc. Int'l Conf. Modelling Techniques Tools Perform. Anal.*, May 1984.

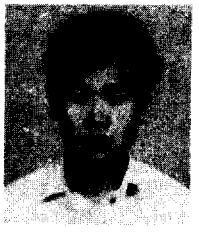
● 저자소개 ●

박기현

1979.2 경북대학교 전자공학과 졸업(공학사)
1981.2 한국과학기술원 전자계산학과 졸업(이학석사)
1990.8 미국 Vanderbilt 대학교 전자계산학과(공학박사)
현재 계명대학교 전자계산학과 부교수
관심분야 : 병렬 분산 운영체제, 성능분석/예측

장명숙

1975.2 경북대학교 수학교육학과 졸업(이학사)
1990.2 미국 Vanderbilt 대학교 전산학과 대학원 졸업(이학석사)
현재 경북대학교 컴퓨터 공학과 박사과정 수료
관심분야 : 병렬처리, 기계학습, 시스톨릭 어레이, 시뮬레이션



최준구

1990.2 계명대학교 공과대학 전자계산학과 졸업(공학사)
1992.2 계명대학교 대학원 전자계산학과 졸업(공학석사)
현재 : 계명대학교 대학원 전자계산학과 박사과정 재학중
경산대학교 정보처리학과 강의조교
관심분야 : 컴퓨터 시뮬레이션, 병렬 분산 운영체제