k-ary n-큐브 상에서의 동적 부하 균등 알고리즘

박 경 욱°, 임 형 석
전남대학교 전산통계학과

Dynamic Load Balancing Algorithm for k-ary n-cubes

Kyoung-Wook Park°, Hyeong-Seok Lim
Dept. of Computer Science and Statistics, Chonnam National Univ.

Abstract

In parallel system, the execution times of tasks cannot be accurately estimated and tasks may arrive at any time. In case, processors are overweighted with workload, the system utilization deteriorates. To solve this problem, dynamic load balancing that rearranges tasks in overloaded processors is required.

We propose the improved dynamic load balancing algorithm for k-ary n-cubes using the property of ring and MWA which is the dynamic load balancing one for mesh. The proposed algorithm which uses the global load information has less communication cost than GDE, DDE and load difference within 1.

1. 서 론

다중 컴퓨터 시스템은 균일한 계산량과 통신 패턴을 갖는 태스크들로 이루어진 문제들을 효율적으로 해결할 수 있다. 시스템에 제출되는 태스크의 수행시간을 예측할 수 있다면 각 프로세서의 부하 균등을 쉽게 제공할 수 있는 정적 스케줄링이 가능하다. 그러나, 많은 문제들에서는 태스크의 수행시간을 예측할 수 없고 태스크의 도착이 수시로 이루어지므로 효율적인 문제 해결을 위해 동적 스케줄링을 사용한다.

동적 스케줄링 환경에서는 특정 프로세서에 태스크 할당이 편중될 수 있으며 이로 인하여 전체 시스템의 성능이 저하된다. 이때, 동적 부하 균등을 이용하여 과부하상태에 놓인 프로세서들의 태스크들을 저부하 상태의 프로세서들로 이동시켜 처리한다면 시스템 활용도를 증가시키고 태스크에 대한 응답시간을 줄일 수 있으므로 전체적인 시스템의 성능을 향상시킬 수 있다.

동적 부하 균등 알고리즘은 크게 상호연결망의 형태에 무관하게 적용할 수 있는 부류[2,3,5]와 특정 상호연결망에만 적용되는 부류[9,10,11]들로 나눌 수 있다. 상호연결망의 형태에 무관하게 적용할 수 있는 부류의 알고리즘들은 여러 상호연결망에 적용할 수 있으나 상호연결망에 따라 다른 성능을 보여주며[7,8], 특정 상호연결망에만 사용되는 부류들에 비해 효율적이지 못하다[9,10]. 그러므로, 특정 상호연결망의 특성을 이용하여 보다 효율적인 동적 부하 균등 알고리즘이 요구된다.

본 논문에서는 Cosmic Cube, Connective Machine 등과 같은 많은 다중 컴퓨터 시스템에 사용된 k-ary n-큐브의 동적 부하 균등 알고리즘을 제시하고자 한다. k-ary n-큐브는 재귀적 구조와 확장성을 가지고 있고 체인, 링, 메쉬 그리고 토러스를 포함하는 상호연결망이다[1,6].

k-ary n-큐브의 동적 부하 균등 알고리즘으로는 DEM을 일반화한 GDE와 DDE가 제안되었다[10,11]. 본 논문에서는 메쉬를 위한 MWA[9]와 링의 특성을 이용하여 성능이 개선된 k-ary n-큐브의 동적 부하 균등 알고리즘을 제시하고, 시뮬레이션을 통해 제시한 알고리즘이 GDE, DDE보다 효율적임을 보이고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구를 기술하고 3장에서는 개선된 k-ary n-큐브의 동적 부하 균등 알고리즘을 제시한다. 4장에서는 시뮬레이션을 통해 다른 알고리즘과 비교 평가하고 5장에서는 결론을 기술한다.

2. 관련연구

부하 균등의 목적은 각 프로세서마다 같은 작업량을 갖도록 하여 전체 시스템의 활용도를 높이고 태스크의 응답시간을 줄이는데 있다. 이를 위해서는 태스크의 수행시간 예측이 요구되는데 태스크의 수행시간을 예측하는 것은 매우 어렵다. 그러므로, 일반적인 부하 균등 알고리즘에서는 각 프로세서마다 같은 수의 태스크를 갖도록 태스크를 할당한다.

동적 부하 균등 문제는 다중 컴퓨터 시스템에서 N개의 프로세서가 상호연결망으로 연결되어 있고, 각 프로세스 i에 w_i개의 태스크가 있을 때 각 프로세서마다 같은 수의 태스크 w_{avg}개를 갖도록 태스크들을 재배치하는 것으로 태스크의 평균수 w_{avg}는 다음과 같이 정의된다[9].

$$w_{avg} = \frac{\sum_{i=0}^{N-1} w_i}{N}$$

동적 부하 균등시 가장 많은 비용을 차지하는 부분

은 태스크를 이주시킬 때 발생하며 이를 줄이기 위해서는 이주할 태스크 수와 이동거리를 최소화해야 한다.

이는 최소비용-최대흐름 문제[4]로 바꿔 생각할 수 있으며 알고리즘의 복잡도는 노드수가 N일 때 $O(N^2)$으로 병렬 수행시 $O(N)$의 복잡도를 가지게 된다. 이러한 높은 복잡도는 부하 균등을 통해 얻는 이득을 감소시키므로 보다 낮은 복잡도를 갖는 휴리스틱 알고리즘이 요구된다.

k-ary n-큐브는 k^n개의 노드를 가지며, 각 노드는 k진수인 n개의 숫자 $x_{n-1}...x_i...x_0$로 나타내고, 노드 $x_{n-1}...x_i...x_0$와 노드 $x_{n-1}...(x_i \pm 1 \bmod k)...x_0$ 사이에 에지를 갖는다. 본 논문에서는 알고리즘 기술의 편의를 위해 k진수인 n개의 숫자를 10진수 형태로 바꾸어 표기한다.

기존의 k-ary n-큐브에서의 동적 부하 균등 알고리즘으로는 GDE와 DDE가 있다[10,11].

k-ary n-큐브를 위한 GDE는 $2n$개의 색상으로 에지-채색 후 같은 색상의 에지에 연결된 프로세서끼리 태스크 교환을 한다. 그리고, 나머지 색상 에지들에 대해서도 반복 수행한다. GDE는 한번 수행으로 부하 균등이 이루어지지 않으므로 각 프로세서의 이웃한 프로세서들과의 차이가 1이내일 때까지 반복 수행한다. 특히, k값이 클수록 많은 반복 수행이 요구되므로 k가 큰 k-ary n-큐브에 대해서는 효율적이지 못하다.

DDE는 k-ary n-큐브를 1차원 에지로만 구성된 k^{n-1}개의 링으로 분할하여 각각 부하 균등을 수행하고 이러한 과정을 각각의 차원 에지에 대해서 n번 반복 수행한다. DDE는 GDE와는 달리 알고리즘을 반복 수행할 필요는 없으나 부하차이가 n으로 완전한 부하 균등을 이루지 못한다.

MWA는 메쉬 상에서의 부하 균등 알고리즘으로 시스템 전체의 부하 정보를 수집하여 각 프로세서마다 할당될 태스크의 수를 결정한다[9]. 그 후 각 행에 있는 태스크의 수가 동일하도록 태스크를 교환한다. 이때 태스크를 전송할 프로세서들은 앞 열로 전송해야 할 태스크들을 제외한 나머지 태스크들을 다른 행으로 전송한다. 마지막으로 동일한 행에 있는 프로세서마다 같은 수의 태스크를 갖도록 태스크를 교환함으로써 부하 균등을 이룬다.

3. k-ary n-큐브 동적 부하 균등 알고리즘

제시된 알고리즘은 k-ary n-큐브에서 효율적인 부하 균등을 이루기 위해서 MWA와 DDE-링 알고리즘을 이용한다.

본 논문에서 제시하는 부하 균등 알고리즘은 그림 1과 같다.

---

k-ary n-큐브는 Q_n^k로 표기하며 전체 노드수 N은 k^n, 각 노드 i에 할당되어 있는 태스크 수는 w_i이다.

단계1. 부하 정보 수집

$w_i^0 = w_i \quad (0 < i < k^n - 1)$

단계2. 평균 부하 계산

$$T = \sum_{i=0}^{N-1} w_i^0, \quad R = T \bmod k^n$$

$$w_{avg} = \lfloor T / k^n \rfloor$$

단계3. 할당량 계산

$$q_i^0 = \begin{cases} w_{avg}+1 & \text{if } i < R \\ w_{avg} & \text{otherwise} \end{cases}$$

For d=1 to n-1

$$q_i^d = \sum \{ q_j^{d-1} \mid \lfloor j / k^d \rfloor = \lfloor i / k^d \rfloor \text{ and } j \bmod k^{d-1} = i \bmod k^{d-1} \}$$

For d=0 to n-1

$$Q_i^d = \sum \{ q_j^d \mid j \le i \text{ and } j \bmod k^d = i \bmod k^d \}$$

단계4. 태스크 교환

Task_Exchange(0, n)

Task_Exchange(i, n)

$$v_R = \begin{cases} v - k^{n-1} \times (k-1) & \text{if } \lfloor v / k^{n-1} \rfloor = k-1 \\ v + k^{n-1} & \text{otherwise} \end{cases}$$

$$v_L = \begin{cases} v + k^{n-1} \times (k-1) & \text{if } \lfloor v / k^{n-1} \rfloor = 0 \\ v - k^{n-1} & \text{otherwise} \end{cases}$$

$R^{-1} = L$

단계 4.1

For d=1 to n-1

For j= i to $i + k^n - 1$

$$w_j^d = \sum \{ w_p^{d-1} \mid \lfloor j / k^d \rfloor = \lfloor p / k^d \rfloor \text{ and } j \bmod k^{d-1} = p \bmod k^{d-1} \}$$

For d=0 to n-1

For j= i to $i + k^n - 1$

$$W_j^d = \sum \{ w_p^d \mid j \le p \text{ and } j \bmod k^d = p \bmod k^d \}$$

단계 4.2

For j= i to $i + k^n - 1$

$$\eta_j^L = \begin{cases} 0 & \text{if } j < k^{n-1} + i \\ Q_{j-k^{n-1}}^{n-1} - W_{j-k^{n-1}}^{n-1} & \text{otherwise} \end{cases}$$

$$\eta_j^R = W_j^{n-1} - Q_j^{n-1}$$

$$n_p = \{ \eta_{i+k^{n-1} \times p}^R \mid \eta_{i+k^{n-1} \times p}^R > 0 \text{ and } 0 \le p < k \}$$

$$n_z = \{ \eta_{i+k^{n-1} \times p}^R \mid \eta_{i+k^{n-1} \times p}^R = 0 \text{ and } 0 \le p < k \}$$

$$n_n = \{ \eta_{i+k^{n-1} \times p}^R \mid \eta_{i+k^{n-1} \times p}^R < 0 \text{ and } 0 \le p < k \}$$

$$x_m = \begin{cases} n_p\text{의 } \lceil k/2 \rceil \text{ 번째 큰 값} & \text{if } |n_n| + |n_z| - |n_p| < 0 \\ n_n\text{의 } \lceil k/2 \rceil \text{ 번째 작은 값} & \text{if } |n_p| + |n_z| - |n_n| < 0 \\ 0 & \text{otherwise} \end{cases}$$

For j= i to $i + k^{n-1} - 1$

$$\eta_j^R = \eta_j^R - x_m, \quad \eta_j^L = \eta_j^L + x_m$$

$v = i$

단계 4.3

For l = L to R

For j=0 to k-1

If $\eta_{v_l}^l \le 0$, break

$v = v_l$

For j=0 to k-1

$$cnt = | \{ w_{v+p}^0 \mid w_{v+p}^0 > q_{v+p}^0 \text{ and } 0 \le p < k^{n-1} \} |$$

$sum = \sum \{ w^0_{v+p} \mid w^0_{v+p} > q^0_{v+p}$
and $0 \le j < k^{n-1} \}$
$m = \lfloor (sum - \eta^l_u) / cnt \rfloor$
For $j = v$ to $v + k^{n-1} - 1$
If $\eta^l_j > 0$
If $n=1$, $\theta^l_j = \eta^l_j$
else

$$\theta^l_j = \begin{cases} w^0_j - m & \text{if } \eta^l_j \ge w^0_j - m > 0 \\ \eta^l_j & \text{if } w^0_j - m > \eta^l_j \\ 0 & otherwise \end{cases}$$

$\eta^l_u = \eta^l_u - \theta^l_j$
($u > j$ and $\lfloor u / k^{n-1} \rfloor = \lfloor j / k^{n-1} \rfloor$)

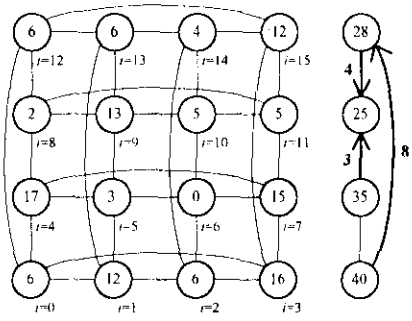
단계 4.4

For $j = v$ to $v + k^{n-1} - 1$
$w^0_j = w^0_j - \theta^l_j$ /* 태스크 송신 */
$w^0_{j_l} = w^0_{j_l} + \theta^l_j$ /* 태스크 수신 */
$v = v_l$
If $n=1$ return
For $j=0$ to $k-1$
Task_Exchange($i + j \times k^{n-1}$, $n-1$)

<그림 1> 새로운 k-ary n-큐브 동적 부하 균등 알고리즘

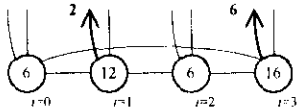
단계1에서는 kn번의 통신 단계를 거쳐 전체 프로세서들의 부하 정보를 수집한 다음 단계2에서 전체 태스크 수와 평균값을 계산하고, 단계3에서 각 프로세서마다 가지야할 태스크의 수를 계산한다. 단계4에서는 단계3에서 얻은 지역 부하 정보를 이용하여 각 프로세서끼리 태스크를 교환하면서 부하 균등을 이루게 된다.

k-ary n-큐브는 k개의 k-ary n-1-큐브들로 나눌 수 있다. 단계4에서는 이러한 특성을 이용하여 k-ary n-큐브를 하나의 노드가 k-ary n-1-큐브인 링으로 보고 DDE-링 알고리즘[10]을 이용하여, k개의 k-ary n-1-큐브들이 서로 교환할 태스크 수를 결정한다.



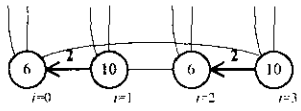
<그림 2> 4-ary 2-큐브에서 단계 4.2

그 다음, 태스크를 전송할 k-ary n-1-큐브 내에서 평균 이상의 태스크를 가진 프로세서의 개수 cnt와 태스크 수의 합 sum을 구한다. 그리고, sum에서 전송해야 할 태스크 수를 뺀 값을 cnt로 나누어 평균값 m을 얻는다. m보다 많은 태스크들을 갖는 프로세서들에서 m을 뺀 나머지 태스크들을 전송한다.



<그림 3> 4-ary 2-큐브에서 단계 4.3

다음 단계에서는 재귀적으로 k개의 k-ary n-1-큐브들을 각각 k개의 k-ary n-2-큐브들로 나누어서 위 과정을 반복해서 수행한다. n-1번의 재귀적 반복으로 k^{n-1}개의 k-링의 태스크 수의 합을 같게 한 후 k^{n-1}개의 k-링 각각에 대해서 DDE-링 알고리즘 수행한다.



<그림 4> 4-ary 2-큐브에서 단계 4.4

제시된 알고리즘에서 요구되는 통신 단계는 단계1의 부하 정보 수집에 kn단계, 단계4의 태스크 교환에서 최대 $\lfloor \frac{kn}{2} \rfloor$단계이다. 따라서, 알고리즘은 통신 단계 $\lfloor \frac{3}{2} kn \rfloor$이내에 수행된다. 또한 제시된 알고리즘 수행 후 k-ary n-큐브의 각 노드 부하차이는 1이내이다.

4. 성능평가

각 프로세서마다 0~1000개의 태스크를 임의로 할당하여 1000회 모의실험 하였다. GDE에서 λ는 16-ary에서 0.732, 32-ary에서는 0.837을 이용하였으며[11], DDE나 제시된 알고리즘에서 태스크 교환은 보내기-전-받기(Receive-before-send)방법을 이용하였다[10].

표준화 통신비용(Normalized communication cost)은 전체 태스크 수에 대한 이동된 태스크 수의 비율로 제시된 알고리즘은 GDE, DDE에 비해 낮은 통신비용을 갖는다.

<표1> 표준화 통신 비용

	GDE	DDE	제시된 알고리즘
16-ary 2-큐브	1.17	0.72	0.63
32-ary 2-큐브	2.05	0.95	0.83
16-ary 3-큐브	1.16	0.75	0.64
32-ary 3-큐브	2.02	0.98	0.84

부하차이(Load difference)는 최대 태스크 수를 갖는 프로세서와 최소 태스크 수를 갖는 프로세서의 태스크 수의 차이로 DDE에서는 n이내, GDE는 $\frac{nk}{2}$이내이며 제시된 알고리즘은 1이내이다.

<표2> 부하차이

	GDE	DDE	제시된 알고리즘
16-ary 2-큐브	10.19	1.87	1
32-ary 2-큐브	14.44	1.93	1
16-ary 3-큐브	11.73	2.82	1
32-ary 3-큐브	12.47	2.91	1

GDE는 $2n$번 반복을 하며 각 반복마다 부하 정보 교환에 두 번의 통신 단계와 부하 균등에 한번의 통신 단계가 필요하다. 그러므로, 알고리즘을 s번 수행하여 부하 균등 상태에 도달한다면 $6sn$번의 통신 단계가 필요하다. DDE는 각 차원의 부하 정보 수집을 위해 k단계와 부하균등시 최대 $\lfloor \frac{k}{2} \rfloor$단계가 필요하므로 최대 $\lfloor \frac{3}{2}kn \rfloor$의 통신 단계를 거친다. 제시된 알고리즘은 부하 정보 수집에 kn단계와 태스크 교환에 최대 $\lfloor \frac{kn}{2} \rfloor$단계가 필요하므로 $\lfloor \frac{3}{2}kn \rfloor$이내의 통신 단계로 수행된다.

<표3> 통신 단계 수

	GDE	DDE	제시된 알고리즘
16-ary 2-큐브	107.17	45.94	44.87
32-ary 2-큐브	160.56	91.39	88.23
16-ary 3-큐브	131.93	70.04	66.17
32-ary 3-큐브	170.56	140.33	128.94

지역성(Locality)은 전체 태스크 수에 대한 이동하지 않은 태스크 수의 비율로 제시된 알고리즘은 전역 부하 정보를 사용하여 과부하된 프로세서의 태스크들만을 전송하므로 GDE나 DDE에 비해 높은 지역성을 갖는다.

<표4> 지역성

	GDE	DDE	제시된 알고리즘
16-ary 2-큐브	53.67%	70.13%	72.54%
32-ary 2-큐브	44.73%	70.70%	72.56%
16-ary 3-큐브	53.41%	69.69%	72.50%
32-ary 3-큐브	44.63%	70.48%	72.52%

GDE는 부하 균등을 위해서 여러 번의 반복수행이 필요한 반면, DDE와 제시된 알고리즘은 한번의 수행으로 부하 균등을 이룰 수 있다.

<표5> 반복 수행 횟수

	GDE	DDE	제시된 알고리즘
16-ary 2-큐브	8.92	1	1
32-ary 2-큐브	13.45	1	1
16-ary 3-큐브	7.28	1	1
32-ary 3-큐브	9.39	1	1

4. 결 론

본 논문에서는 재귀적인 구조와 확장성을 가지고 있고 체인, 링, 메쉬 그리고 토러스를 포함하는 상호연결망인 k-ary n-큐브 상에서의 효율적인 동적 부하 균등 알고리즘을 제시하였다. 제시된 알고리즘은 $\lfloor \frac{3}{2}kn \rfloor$이내의 통신단계로 부하차이 1이내인 부하 균등이 가능하다. 또한, 모의 실험을 통하여 제시된 알고리즘이 GDE나 DDE보다 적은 통신비용과 통신 단계를 갖고 높은 지역성을 나타냄을 보였다.

참 고 문 헌

[1] W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Trans. Parallel and Distributed Systems*, vol.39, no.6, pp.775-785, 1990

[2] DL. Eager, E.D. Lazowska, and J.Zahorjan, "A Comparison of Receiver Initiated and Sender Initiated Adaptive Load Sharing," *Performance Evaluation*, vol.6, pp.53-68, 1986

[3] K.K. Goswami, M. Devarakonda, and R.K. Iyer, "Prediction-Based Dynamic Load-Sharing Heuristics," *IEEE Trans. Parallel and Distributed Systems*, vol.4, no.6, pp.638-648, 1993

[4] E.L. Lawler, *Combinatorial Optimization:Networks and Matroids*. Holt, Rinehar and Winston, 1976

[5] F.C.H. Lin and R.M. Keller, "The Gradient Model Load Balancing Method," *IEEE Trans. Software Eng.*, Vol.13, No.1, pp.32-38, 1987

[6] D.H. Linder and J.C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Trans. Computers*, vol. 40, no.1, pp.2-12, 1991

[7] P.K.K Loh, W.J. Hsu, C.Wentong and N.Sriskanthan, "How Network Topology Affects Dynamic Load Balancing," *IEEE Parallel & Distributed Technology*, pp.25-35 Fall. 1996

[8] M. Willebeek-LeMair and A.P. Reeves, "Strategies for Dynamic Load Balancing on Highly Parallel Computers," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 9, pp.979-993, 1993

[9] M.Y. Wu, "On Runtime Parallel Scheduling for Processor Load Balancing," *IEEE Trans. Parallel and Distributed System*, vol. 8, no. 2, pp.173-186, 1997

[10] M.Y. Wu and W. Shu, "DDE: A Modified Dimension Exchange Method for Load Balancing in k-Ary n-Cubes," *J. Parallel and Distributed Computing*, vol. 44, no. 1, pp.88-96, 1997

[11] C.Z. Xu and F.C.M. Lau, "The Generalized Dimension Exchange Method for Load-Balancing in k-Ary n-Cubes and Variants," *J. Parallel and Distributed Computing*, vol. 24, no. 1, pp.72-85, 1995