병렬 및 분산 시스템에서의 최적 고장 허용 자원 배치

(Optimal Fault-Tolerant Resource Placement in Parallel and Distributed Systems)

김 종 훈 † 이 철 훈 ††
(Jong-Hoon Kim) (Cheol-Hoon Lee)

요 약 본 논문에서는 병렬 및 분산 시스템에서 자원을 배치함에 있어서 최소한의 자원 복사(copy)만을 사용하면서 임의의 노드 및 링크 상에서 고장이 발생하더라도 주어진 성능 요건을 만족하게 하는 자원의 최적 배치 방법을 모색하고자 한다. 이러한 성능 요건의 만족과 시스템의 고가용성을 위하여, 모든 노드들에 대하여 최소한의 자원 복사를 사용하여 그 노드나 혹은 인접한 노드 중 적어도 두 개 이상에 자원 복사가 존재해야 하는데, 이것을 본 논문에서는 고장 허용 자원 배치 문제라고 부른다.

병렬 및 분산 시스템은 그래프로 표현할 수가 있다. 여기에서 고장 허용 자원 배치 문제는 그래프 상에서 가장 작은 고장 허용 dominating set을 찾는 문제로 변환이 된다. Dominating set 문제는 NP-complete로 증명이 되어 있으며, 본 논문에서는 A* 알고리즘을 사용하여 상태 공간 탐색 방법으로 최적 배치를 구한다. 또한, 최적 배치를 찾는 데에 걸리는 시간을 단축시키기 위하여, 고장 허용 dominating set의 특성들을 분석하여 유용한 휴리스틱 정보들을 도출한다. 또한 여러가지 정형 그래프와 임의 그래프 상에서의 실험을 통하여, 이들 휴리스틱 정보들을 사용하여 최적 고장 허용 자원 배치를 찾는 데에 걸리는 시간을 상당히 줄일 수 있음을 보인다.

Abstract We consider the problem of placing resources in a distributed computing system so that certain performance requirements may be met while minimizing the number of required resource copies, irrespective of node or link failures. To meet the requirements for high performance and high availability, minimum number of resource copies should be placed in such a way that each node has at least two copies on the node or its neighbor nodes. This is called the fault-tolerant resource placement problem in this paper.

The structure of a parallel or a distributed computing system is represented by a graph. The fault-tolerant placement problem is first transformed into the problem of finding the smallest fault-tolerant dominating set in a graph. The dominating set problem is known to be NP-complete. In this paper, searching for the smallest fault-tolerant dominating set is formulated as a state-space search problem, which is then solved optimally with the well-known A* algorithm. To speed up the search, we derive heuristic information by analyzing the properties of fault-tolerant dominating sets. Some experimental results on various regular and random graphs show that the search time can be reduced dramatically using the heuristic information.

· 본 연구는 1997년도 학술진흥재단 대학부설연구소 지원 과제에 의해 수행되었음
† 비 회 원 : 충남대학교 컴퓨터공학과
jhkim@ce.cnu.ac.kr
†† 정 회 원 : 충남대학교 컴퓨터공학과 교수
chlee@ce.cnu.ac.kr
논문접수 : 1999년 8월 26일
심사완료 : 2000년 4월 6일

1. 서 론

병렬 및 분산 시스템에는 입출력 프로세서나 프린터, 혹은 디스크와 같은 하드웨어 자원에서부터 에디터, 컴파일러, 혹은 데이타 화일 등과 같은 소프트웨어 자원에 이르기까지 여러 종류의 자원이 존재한다. 대규모 시스템인 경우에는 일반적으로 이들 각각의 자원에 대해서 여러 개의 복사(copy)가 시스템 내에 있게 된다. 그러므

로 이들 각 자원의 복사들을 시스템 내에 어떻게 배치하는 가에 따라 시스템의 성능 대비 가격 비 및 신뢰성에 큰 영향을 미치게 된다. 이들 자원의 복사들을 배치함에 있어서 다음과 같은 두 가지 극단적인 배치를 고려할 수가 있다. 첫째 모든 자원의 복사들을 각각의 프로세서에게 모두 배치시키는 것과, 둘째로 각 자원의 종류에 대해서 오직 하나의 복사만을 시스템 내의 특정 프로세서에 배치하는 것이다. 일반적으로, 첫번째 배치 방법은 시스템의 가격을 높임과 동시에 각 자원의 사용도(utilization)를 떨어뜨리게 된다. 반면에 두번째 배치 방법에서는 각 자원을 접근하는데 있어서 시간적인 지연(delay) 및 충돌(contention)로 인하여 시스템의 성능을 저하시키는 결과를 초래할 것이며 또한 자원이 배치된 프로세서 또는 그 프로세서를 연결하는 링크의 고장으로 인하여 그 자원을 사용할 수 없는 문제가 발생한다. 따라서 자원의 복사들을 배치함에 있어서 "가격 최소화", "성능 최대화", 및 자원의 가용성이라는 서로 상충하는 목적들 사이에 상호 보완 관계가 존재하게 된다. 가격을 줄이기 위해서는 여러 프로세서들이 각 자원의 복사를 공유하게 하여 각 자원의 전체 복사 수를 줄여야 하며, 시스템의 성능 저하를 막고 동시에 자원의 가용성을 높이기 위해서는 각 프로세서들이 각 자원의 적어도 두 개의 복사와는 일정한 거리 이내에 존재하게 하여 충돌 및 통신 지연을 막아야 막고 동시에 프로세서 및 링크의 고장 시에도 적어도 하나의 자원 복사에 접근할 수 있어야 한다.

본 논문에서 다루고자 하는 자원의 배치 문제는 다음과 같이 정의된다. 병렬 및 분산 시스템에서 어떤 자원을 배치함에 있어서 최소한의 자원 복사를 사용하여 배치를 하되, 임의의 프로세서 및 링크에 고장이 발생하더라도 각 프로세서들은 적어도 하나의 자원 복사와는 거리 1 이내에 존재하게 (즉, 자원을 가지고 있던지 혹은 자원을 가진 프로세서와 인접해 있던지) 해야 한다. 이 문제는 그래프 이론에서의 dominating set 문제보다 확장된 문제로서, 본 논문에서는 이것을 고장 허용 자원 배치 문제라고 부른다. 이 문제와 관련하여, 기존의 연구는 네트워크 상에서 r-dominating 집합과 p-center를 찾는 문제에 관련되어 있다. 먼저, Kariv와 Hakimi는 일반적인 구조에서의 exponential-time 알고리즘과 트리상에서의 polynomial-time 알고리즘을 제시했으며 [1], 다른 연구자들은 트리상에서 그 문제에 대한 효과적인 병렬 알고리즘을 개발하였다[2][3][4]. 그리고, *Kutten*과 *Peleg*는 *small k-dominating* 집합과 그것의 축소된 그래프 파티션을 계산하기 위한 분산 알고리즘을 제시했다[5]. 그 외에 다른 많은 연구자들은 k-ary n-cube 구조의 멀티프로세서 상에서의 자원 배치에 대하여 연구했다. 먼저, Reddy 등은 하이퍼 큐브 (즉, 2-ary n-cube) 상에서 입출력 프로세서들을 배치함에 있어서, 하이퍼 큐브의 모든 노드들이 적어도 하나 이상의 입출력 프로세서에 인접하도록 배치하는 문제를 다루었고[6], Livinston과 Stout는 특정 요건들을 만족하면서 하이퍼큐브에서의 자원 배치 문제를 고려 했다[7]. 그들은 또한 2차원 메쉬 (즉, k-ary 2-cube) 상에서의 해도 제시하였다[8]. Chiu와 Raghavendra는 하이퍼 큐브에서 자원 직경(diameter)을 최소화 한다는 목적 하에서 주어진 자원들을 배치하는 문제를 다루었다[9]. 그 외에 다른 연구자들은 k-ary n-cube 상에서 특정 제약하에서 j-인접 배치를 제안했다[10][11]. 그리고, 최근에, Lee는 일반적인 구조에서 각 노드들이 거리 k이내에서 자원을 접근할 수 있는 k-한도 자원 배치 문제를 다루었다[12].

이들 선행 연구와는 달리, 본 논문에서는 병렬 및 분산 시스템의 구조에 제한을 두지 않고, 일반적인 구조에서의 고장 허용 자원 배치 문제를 다루고자 한다. 병렬 및 분산 시스템은 그래프로 모델링되며, 이때 그래프의 각 노드는 각 프로세서를 의미하며, 두 노드 사이의 에지(edge)는 해당하는 두 프로세서 사이에 통신 링크가 있음을 의미한다. 본 논문에서는 먼저 고장 허용자원 배치 문제가 그래프 상에서 최소 고장 허용 dominating set을 찾는 문제와 동일 함을 보인다. Dominating set 문제는 NP-complete 문제임이 증명되어 있다[13]. 따라서 본 연구에서는 이 문제를 상태 공간 탐색(state-space search) 문제로 공식화하고 인공 지능 분야에서 사용되는 A^* 알고리즘을 이용하여 최적 배치를 찾는다. 또한 탐색 시간을 줄이기 위하여 고장 허용 dominating set의 특성을 분석하여 유용한 휴리스틱 정보를 도출하며, 여러 종류의 그래프 상에서의 실험 결과를 통하여 이들 휴리스틱 정보의 효율성을 보인다.

본 논문은 다음과 같이 구성되어 있다. 제 2 절에서는 자원 배치 문제를 시스템을 모델링한 그래프 상에서의 고장 허용 dominating set 문제로 공식화 한다. 제 3 절에서는 고장 허용 dominating set의 특성들을 분석하여 유용한 휴리스틱 정보들을 도출하며, 제 4 절에서는 A^* 알고리즘에 대한 설명과 함께 이들 휴리스틱 정보들을 사용하여 최적 배치를 찾는 알고리즘을 제시한다. 이 알고리즘을 적용하여 최적 배치를 찾는 예를 제 5 절에서 보이며, 제 6 절에서는 실험 결과를 통하여 본 알고리즘의 효율성을 보인다. 본 논문은 제 7 절에서 결

론을 맺는다.

2. 고장 허용 자원 배치 문제

병렬 및 분산 시스템은 여러 개의 프로세서들로 구성되어 있으며 이들 각 프로세서들은 통신 링크을 통해 서로 통신을 한다. 이러한 병렬 및 분산 시스템은 그래프로 모델링할 수 있는데, 이 때 그래프 상의 각 노드는 각 프로세서를 의미하며, 노드들 사이의 에지(edge)는 해당하는 두 프로세서 사이에 통신 링크가 존재함을 의미한다. 임의의 두 프로세서 사이의 거리(distance)는 그 두 프로세서를 연결하는 최소한의 링크 갯수로 정의한다. 이러한 병렬 및 분산 시스템에 값비싼 입출력 프로세서나 특정한 데이타 화일을 배치하는 문제를 생각해 보자. 시스템의 성능면에서 본다면, 이들 입출력 프로세서나 데이타 화일을 가능한 많은 프로세서에 배치를 하여, 각 프로세서들이 이들 자원을 접근하는 데에 걸리는 시간을 줄여야 하며, 또한 비용 면에서 본다면, 가능한 적은 프로세서들에게만 이들 자원들을 배치하여 전체 시스템 비용을 줄여야 한다. 또한 임의의 프로세서나 링크에 고장이 발생하더라도 자원을 접근할 수 있게 하기 위해서는 각 프로세서는 적어도 두 개 이상의 자원 복사와 인접해야 한다. 따라서 각 자원의 복사들을 배치하는 데에 있어서, 성능과 비용 및 가용성 면에서 상호 보완 관계가 존재한다. 그러므로 자원 배치에 있어서 다음과 같은 두 가지 목적을 동시에 고려하여야 한다: (1) 먼저 비용을 줄이기 위해서 배치되는 자원의 전체 복사 수를 줄여야 하며, (2) 또한 시스템의 성능 저하를 막고 동시에 자원의 가용성을 높이기 위해서 각 프로세서들이 적어도 두 개 이상의 복사와는 인접하게 존재하게 하여 임의의 고장에 대해서도 충돌 및 통신 지연 없이 자원을 접근할 수 있어야 한다. 본 절에서는 이와 같은 고장 허용자원 배치 문제를 그래프 이론에서의 고장 허용 dominating set 문제로 공식화한다.

병렬 및 분산 시스템을 모델링한 그래프를 $G(V, E)$라 하자. 여기에서 V는 노드들의 집합이고, E는 에지들의 집합이다. 임의의 두 노드 v_i와 v_j 사이의 거리 $d(v_i, v_j)$는 그들 사이의 최단 거리 상의 에지들의 수이다. 또한 임의의 노드 v_i와 임의의 노드 집합 S 사이의 거리 $d(v_i, S)$는 v_i와 S에 있는 가장 가까운 노드와의 거리로 정의한다. 즉, $d(v_i, S) = \min_{v_j \in S} d(v_i, v_j)$. 따라서, 임의의 노드 v_i에 대해서 $d(v_i, v_j) = 0$이고, 또한 $v_j \in S$ 이면, $d(v_i, S) = 0$이다. 임의의 노드 v_i에 대해서, v_i를 포함하여 그것과 인접한 모든 노드들의 집합을 $N(v_i)$로 정의한다. 즉, $N(v_i) = \{v_j | d(v_i, v_j) \le 1\}$. 이것을 본 논문에서는 노드 v_i의 *이웃* 집합이라고 부른다. 또한 임의의 노드 집합 $S(\subseteq V)$에 대해서 그것의 이웃 집합 $N(S)$은 $N(S) = \{v_j | d(v_j, S) \le 1\}$으로 정의한다. 노드 집합 S의 크기, 혹은 cardinality,는 $C(S)$로 표기한다.

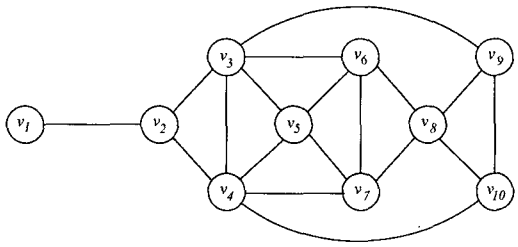

그림 1 예제 그래프

그래프 G 상에서의 *dominating set*은 그 그래프의 모든 노드 v를 다음과 같은 의미에서 "dominate"하는 노드들의 집합이다: 즉, 노드 v가 그 dominating set에 속해 있던지, 아니면 그 dominating set에 있는 하나 이상의 노드와 인접해 있어야 한다. 예를 들어, 그림 1에서 노드 집합 $\{v_2, v_5, v_8\}$와 $\{v_1, v_2, v_3, v_6, v_{10}\}$는 각각 dominating set이다. 본 논문에서는 dominating set의 정의를 변형한 고장 허용 dominating set을 (이것을 본 논문에서는 "FT-dominating set"이라고 부른다) 다음과 같이 정의한다.

정의 1: 어떤 그래프 $G(V, E)$ 상에서의 *FT-dominating set*은 노드의 집합으로서, 그래프 상의 모든 노드 v가 그 집합에 속해 있던지, 아니면 그 집합의 두 개 이상의 노드와 인접해 있는 그런 집합이다.

예를 들어, 그림 1에서 노드 집합 $\{v_1, v_3, v_7, v_{10}\}$은 FT-dominating set이다. 그래프 $G(V, E)$의 임의의 노드 집합 $S(\subseteq V)$에 대해서 그것의 FT-이웃 집합을 다음과 같이 정의한다.

정의 2: 그래프 $G(V, E)$의 임의의 노드 집합 $S(\subseteq V)$의 FT-이웃 집합 $N_{FT}(S)$은 집합 S에 있는 모든 노드들을 포함하여 S에 있는 적어도 두 개 이상의 노드와 인접한 노드들의 집합이다. 즉,

$$N_{FT}(S) = \{v_k | v_k \in S\} \cup \{v_l | v_l \in N(v_i) \text{ and } v_l \in N(v_j)\},$$
$$\text{for some } v_i, v_j \in S, i \neq j$$

예를 들어, 그림 1에서 노드 집합 $S = \{v_2, v_3, v_8\}$의 FT-이웃 집합 $N_{FT}(S)$는 $\{v_2, v_3, v_4, v_6, v_8, v_9\}$이다. 어떤 노드 v_i가 집합 S의 FT-이웃 집합에 속하면, 즉

$v_i \in N_{FT}(S)$이면, v_i는 집합 S에 *FT-dominated되었다*고 말한다. 임의의 그래프 $G(V,E)$의 모든 노드들이 노드 집합 $S(\subseteq V)$에 FT-dominated되었을 경우, 즉, $N_{FT}(S) \equiv V$ 일 경우, 집합 S는 그래프 G의 FT-dominating set이 된다. 어떤 FT-dominating set에 대해서 그 집합에 있는 어떠한 노드를 제외시키더라도 그 집합이 더 이상 FT-dominance 특성을 가질 수 없을 때, 그 집합을 *minimal FT-dominating set* 이라고 한다. 예를 들어, 그림 1에서 노드 집합 $\{v_1, v_2, v_5, v_8, v_9\}$은 minimal FT-dominating set이지만, $\{v_1, v_2, v_4, v_6, v_9\}$은 minimal FT-dominating set이 아니다. 왜냐하면, v_2를 제외시킨 집합 $\{v_1, v_4, v_6, v_9\}$이 또한 FT-dominating set이기 때문이다. 하나의 그래프에는 여러 개의 minimal FT-dominating set이 존재할 수 있으며, 그들의 크기는 서로 다를 수가 있다. 당연히, 크기가 가장 작은 FT-dominating set은 minimal FT-dominating set이다. 예를 들어, 그림 1에서 크기가 가장 작은 FT-dominating set은 $\{v_1, v_3, v_7, v_{10}\}$와 $\{v_1, v_4, v_6, v_9\}$이다. 만약에, 임의의 노드 집합 S를 그 집합에 있는 각각의 노드(즉, 프로세서)들에 복사를 하나씩 배치하는 자원 배치 방법으로 mapping시킬 경우, 노드 집합과 자원 배치에는 one-to-one mapping이 되며, 이때 그 배치에 필요한 전체 자원 복사의 수는 $C(S)$이 된다. 따라서, 자원의 복사수를 최소화 한다는 관점에서 보면 $C(S)$은 배치 S에 대한 성능 척도가 된다. 또한, S가 FT-dominating set이라면 그것에 해당하는 배치는 고장 허용 자원 배치가 된다. 그러므로, 최적의 고장 허용 자원 배치 S_o는 다음과 같이 모든 고장 허용 배치들(즉, 모든 FT-dominating set들) 중에서 자원 복사의 수가 최소인 배치이다:

$$C(S_o) = \min_S C(S).$$

3. FT-Dominating Set의 특성

$|V|$개의 노드 $\{v_1, v_2, \ldots, v_{|V|}\}$를 가지는 그래프 $G(V,E)$에 대해서 배치 영역을 $\Gamma = \{0, 1, X\}^{|V|}$로 정의한다. 이때 각 배치 $P \in \Gamma$는 그래프 G의 N개의 노드를 표현하는 $|V|$-tuple $P = (p_1, p_2, \ldots, p_{|V|})$로서, 여기에서 각 p_i는 배치 P 상에서 노드 v_i의 다음과 같은 상태를 표시한다:

$$p_i = \begin{cases} 0 & v_i \text{에 자원 복사를 배치하지 않기로 결정되었다.} \\ 1 & v_i \text{에 자원 복사를 배치하기로 결정되었다.} \\ X & v_i \text{에 대해서 어떠한 결정도 나지 않았으며,} \\ & v_i \text{를 } \textit{candidate} \text{ 노드 라고 부른다.} \end{cases}$$

P_0, P_1, 그리고 P_X를 각각 상태가 0, 1, 그리고 X인 노드들의 집합이라고 하자. 만약, $P_X = \emptyset$이면 (즉, $P_0 \cup P_1 = V$), 이때 P를 "완전 배치" ("complete placement")라고 부르며, 그렇지 않을 경우 "불완전 배치" ("partial placement")라고 부른다. 그러면, 모든 FT-dominating set S은 ($P_1 = S, P_0 = V - S$)인 완전 고장 허용 배치에 mapping된다. 최적 배치를 찾기 위한 상태 공간 탐색 트리 (제 4 절에서 설명 됨) 상에서 intermediate node들의 상태는 각각 불완전 배치이며, leaf node들의 상태는 완전 배치가 된다. (여기에서 프로세서를 의미하는 그래프 상의 "노드"와 혼돈을 피하기 위해 탐색 트리 상의 node는 영문자 "node"로 표기한다.)

임의의 불완전 배치 P 하에서, $S(P)$를 $P_1 \subseteq S(P)$와 $P_0 \cap S(P) = \emptyset$을 만족하는 하나의 FT-dominating set이라 하자. 그러면, 불완전 배치 P에 따라 이러한 FT-dominating set $S(P)$가 존재할

수도 있고 없을 수도 있다. 만약 이러한 FT-dominating set $S(P)$가 하나 이상 존재할 때, P는 *feasible*하다고 하며, 그렇지 않을 경우 P는 *infeasible*하다고 한다. 어떠한 불완전 배치 P에 대해서 $(P_1 \cup P_X)$이 FT-dominating set일 때, P는 feasible하다. 따라서, 불완전 배치 P의 feasibility에 대해 다음과 같은 특성이 존재함을 알 수 있다.

특성 1: 어떤 불완전 배치 P가 feasible할 필요 충분 조건은 $N_{FT}(P_1 \cup P_X) \equiv V$ 이다.

따라서 최적 배치를 찾기 위한 상태 공간 탐색에서 만들어진 각 불완전 배치에 대해서, 먼저 feasibility를 테스트한 후, feasible하지 않은 배치인 경우, 더 이상 고려하지 않아도 되므로 그 만큼 최적 배치를 찾는 시간을 단축시킬 수가 있다.

어떠한 불완전 배치 P 하에서, 임의의 candidate 노드 $v_i \in P_X$에 대한 *full-covered set*을, ($N_F(v_i|P)$로 표기함), P_1에는 FT-dominated되지 않고, ($P_1 \cup \{v_i\}$)에는 FT-dominated되는 노드들의 집합이라고 하자. 즉,

$$N_F(v_i|P) = \{v_j | v_j \notin N_{FT}(P_1) \text{ and } v_j \in (N_{FT}(P_1 \cup \{v_i\}))\}$$

또한, 임의의 candidate 노드 $v_i \in P_X$에 대한 *half-covered set*을, ($N_H(v_i|P)$로 표기함), P_1에는 이웃하지 않고, v_i에는 이웃하는 노드들의 집합이라고 하자. 즉,

$$N_H(v_i|P)=\{v_j|v_j\notin N(P_1) \text{ and } v_j\in N(v_i)\}$$

불완전 배치 P 하에서의 임의의 두 candidate 노드 v_i, v_j에 대해서, 만약 $N_F(v_j|P)\subset N_F(v_i|P)$ 이면서 $N_H(v_j|P)\subseteq(N_F(v_i|P)\cup N_H(v_i|P))$이면 v_j는 P 하에서 v_i에 의해 covered되었다고 한다. 만약 어떤 노드가 P 하에서 다른 노드에 의해 covered되었을 때, 그 노드는 P 하에서 trivial하다고 말한다. 예를 들어, 그림 1에서 불완전 배치 $P=(1,0,1,X,X,0,X,X,0,1)$를 생각해 보자. 그러면, $N_{FT}(P_1)=\{v_1,v_2,v_3,v_4,v_9,v_{10}\}$, $N(P_1)=\{v_1,v_2,v_3,v_4,v_5,v_6,v_8,v_9,v_{10}\}$.

여기에서, $N_F(v_4|P)=\{v_5\}$, $N_H(v_4|P)=\{v_7\}$, $N_F(v_7|P)=\{v_5,v_6,v_7,v_8\}$, $N_H(v_7|P)=\varnothing$. 그러므로, v_4는 P 하에서 v_7에 covered 되었다. 마찬가지로, v_5와 v_8도 P 하에서 v_7에 covered되었다.

이 경우, $\{v_1,v_3,v_7,v_{10}\}$이 P 하에서 가장 작은 FT-dominating set이다.

그래프 $G(V, E)$의 임의의 노드 집합 S의 *cover number*를 $\alpha(S)$로 표기하며 다음과 같이 정의한다; $\alpha(S)$는 집합 S에 FT-dominated된 노드의 수의 두배에 집합 S에 FT-dominated되지 않고 다만 이웃하는 노드의 수를 더한 값이다. 즉,

$$\alpha(S)=2\times C(N_{FT}(S))+C(N(S)-N_{FT}(S))$$

정의에 의해서, 임의의 FT-dominating set S의 cover number는 $2|V|$가 된다. 또한, 어떤 불완전 배치 P 하에서 P_X에 있는 candidate 노드 v_i의 cover number를 $\alpha(v_i|P)$로 표기하며 다음과 같이 정의한다:

$$\alpha(v_i|P)=\alpha(P\cup\{v_i\})-\alpha(P_1)$$

어떤 불완전 배치 P 하에서 P_X에 있는 모든 노드들이 (즉, 모든 candidate 노드들) 그들의 cover number의 크기에 따라 다음과 같이 non-increasing 순서로 배열되었다고 가정하자:

$$P_X=\{v_{x_i}|\alpha(v_{x_i}|P)\geq\alpha(v_{x_{i+1}}|P) \text{ for } 1\leq i<C(P_X)\}$$

이들 중 처음 t 개의 candidate 노드들의 cover number의 크기의 합을 $T(t|P)$라고 하자. 즉,

$$T(t|P)=\sum_{i=1}^{t}\alpha(v_{x_i}|P)$$

또한, $T(t|P)$이 다음의 부등식을 만족하는 가장 작은 t 값을 $t_{\min}$이라고 하자:

$$t_{\min}=\text{minimum } t \text{ such that } T(t|P)+\alpha(P_1)\geq 2|V|, \text{ for } 1\leq t\leq C(P_X) \quad (1)$$

임의의 불완전 배치 P가 feasible하면, 위의 (1) 식을 만족하는 $t_{\min}$이 존재하게 되며, 이러한 $t_{\min}$값은 현재의 불완전 배치 P 하에서 최적 고장 허용 배치를 위해 더 필요한 자원 복사 수에 대한 lower-bound가 된다.

특성 2: 어떤 feasible 불완전 배치 P 하에서의 가장 작은 FT-dominating set, 즉 최적 고장 허용 배치를 $S_o(P)$라 하자. 그러면, 최적 배치에서의 자원 복사 수는 현재 사용된 자원 복사 수에 $t_{\min}$을 더한 값보다 항상 같거나 크다. 즉,

$$C(S_o(P))\geq C(P_1)+t_{\min}$$

임의의 feasible 불완전 배치 P에 있어서, 어떤 하나의 candidate 노드 v_i가 다른 candidate 노드 v_j에 의해 covered되었다고 하자. 또한, 노드 v_i를 포함한 노드 집합 $S(P)$(즉, $v_i\in S(P)$)을 P 하에서의 하나의 가장 작은 FT-dominating set이라고 가정하자. 그러면, $N_F(v_i|P)\subset N_F(v_j|P)$이며 $N_H(v_i|P)\subseteq N_H(v_j|P)$이므로, $S'(P)=S(P)-\{v_i\}+\{v_j\}$인 또 다른 가장 작은 FT-dominating set $S'(P)$이 존재하게 된다. 다시 말해서, trivial 노드들에게 자원의 복사를 배치하지 않아도 항상 최적 배치를 구할 수가 있다. 즉, trivial 노드들은 자원 복사의 배치 대상으로 고려하지 않아도 되며, 이것은 다음의 특성 3에서 잘 설명이 되어 있다.

특성 3: 임의의 feasible 불완전 배치 P 하에서의 trivial 노드들의 집합을 $P_t(\subseteq P_X)$라고 하자. 또한, 다음과 같은 또다른 불완전 배치 P^*를 생각하자. 즉, $P_1^*=P_1, P_0^*=P_0\cup P_t$, 그리고 $P_X^*=P_X-P_t$. 그러면, $S_o(P)$와 $S_o(P^*)$를 각각 P와 P^* 하에서의 가장 작은 FT-dominating set이라고 하면, $C(S_o(P))=C(S_o(P^*))$이 성립한다.

임의의 feasible 불완전 배치 P에서의 하나의 trivial 노드 v_i에 대해, 만약 v_i가 candidate 노드 v_j에 의해 covered되었고, 또한 v_j는 v_i를 FT-dominate하는 (즉, $v_j\in N_{FT}(P_1\cup\{v_j\})$인) 유일한 candidate 노드일 경우, 노드 v_j는 P 하에서 *FT-essential* 하다고 말한다. 위의 특성 3에 의하면, 각 trivial 노드들을 자원 배치의 대상으로 고려하지 않아도 항상 최적 배치를 구할 수가 있다. 바꾸어 말하면, 각각의 trivial 노드들에게 자원을 배치하지 않은 상태에서도 항상 최적 배치가 존재한다. 그러나, 이 상태에서의 각 최적 배치는 각 FT-tirvial 노

드들을 유일하게 FT-dominate하는 FT-essential 노드들이 존재할 경우, 이들에게 반드시 자원을 배치하여야 한다. 또한, 그림 1의 v_1과 같은 각 말단 노드(dangling node)들은 어떠한 노드에 의해서도 FT-dominated되지 않으므로 당연히 FT-essential 노드가 된다. 따라서, 불완전 배치 P에서의 FT-essential 노드들의 집합을 $P_e(\subseteq P_X)$라고 했을 때, 다음과 같은 특성이 만족된다.

특성 4: 임의의 feasible 불완전 배치 P에 대해서 다음과 같은 또 다른 불완전 배치 P^*를 생각해 보자:

즉, $P_1^* = P_1 \cup P_e, P_0^* = P_0 \cup P_t$, 그리고 $P_X^* = P_X - P_t - P_e$. 여기에서 $S_o(P)$와 $S_o(P^*)$를 각각 P와 P^* 하에서의 가장 작은 FT-dominating set이라고 했을 때, $C(S_o(P)) = C(S_o(P^*))$이 성립한다.

위의 특성 3과 4에 의해서 임의의 feasible 불완전 배치 P 상에서 P_X에 있는 모든 trivial 노드들을 P_0로 옮길 수 있으며, 또한 P_X에 있는 모든 FT-essential 노드들을 P_1로 옮길 수 있으므로, 상태 공간 탐색에서 최적 배치를 찾는 데에 걸리는 시간을 상당히 줄일 수가 있다. 이들 특성들에 의한 시간 단축 효과는 제 6 절에서 보인다.

4. 상태 공간 탐색 방법

이 절에서는 최적 고장 허용 자원 배치를 찾는 문제를 상태 공간 탐색 문제로 공식화하고, 인공 지능

분야에서 잘 알려진 A^* 알고리즘[14]을 사용하여 그 해를 찾는다. A^* 알고리즘을 사용하여 최적 고장 허용 자원 배치를 찾을 수 있을 뿐만 아니라, 앞 절에서 도출한 휴리스틱 정보를 이용하여 최적 배치를 찾는 데 걸리는 시간을 단축할 수가 있다.

상태 공간 탐색 문제는 목적 상태에 도달할 때까지 상태 명세에 오퍼레이터를 적용시킴으로써 그 최적 해를 구할 수가 있다. 각각의 상태 명세는 *node*로써 표시된다. 이들 각 node에 적용되는 오퍼레이터는 그 node의 상속자들을 만들어 내는 것 (이것을 *node 확장*이라고 부름)으로 정의된다. 탐색 문제에서의 해 경로라는 것은 상태 공간 상의 경로로서, 출발 node에서 부터 하나의 목적 node에 이르게 하는 연속적인 오퍼레이터의 배열로 정의된다[14]. 여기에서 해 경로는 가장 작은 FT-dominating set에 해당하는 최적 고장 허용 자원 배치를 결정한다. 이와 같은 정의들로써, 상태 공간 탐색 방법을 다음과 같이 설명할 수가 있다.

상태 명세: 탐색 트리 (search tree) 상에서의 각 node n에 해당하는 불완전 배치를 N-tuple $P=(p_1, p_2, \ldots, p_N)$로써 표현한다. 여기에서 각 성분 $p_i \in \{0,1,X\}$는 각 노드 v_i에 대한 것으로서 제 3 절에서 설명한 의미를 가진다. 각각의 불완전 배치 P에 대해서 $P_0 = \{v_i | p_i = 0\}$, $P_1 = \{v_i | p_i = 1\}$, 그리고 $P_X = \{v_i | p_i = X\}$로 정의한다.

초기 상태 s: 초기 상태에서의 trivial 노드들의 집합과 FT-essential 노드들의 집합을 각각 $\varnothing_t$와 $\varnothing_e$라고 하자. 앞에서 설명했듯이, 모든 말단 노드들은 초기 상태에서 FT-essential 노드가 된다. 그러면, 특성 3과 4에 의해서, 초기 상태 **s**를 다음과 같이 설정할 수가 있다. 즉, $P_0 = \varnothing_t, P_1 = \varnothing_e$, 그리고 $P_X = V - \varnothing_t - \varnothing_e$.

오퍼레이터: 오퍼레이터는 P_X에서 하나의 candidate 노드 v_i를 추출하여 P_0과 P_1에 넣음으로써 각각 새로운 배치 P'와 P''를 만드는 것이다. 즉,

$$P' : P_0' = P_0 + \{v_i\}, P'_1 = P_1, \text{and } P'_X = P_X - \{v_i\},$$
$$P'' : P''_0 = P_0, P''_1 = P_1 + \{v_i\}, \text{and } P''_X = P_X - \{v_i\}.$$

그 다음, 그래프 G 상의 노드 인접(adjacency) 정보를 이용하여 P'의 feasibility를 검사한다. 앞의 특성 1에 의하여, P'가 feasible할 필요 충분조건은 $N_{FT}(P'_1 \cup P'_X) \equiv V$이다. 현재의 배치 P가 feasible하므로, P''는 당연히 feasible하다. 따라서 P''에 대한 feasibility 검사는 할 필요가 없다. 이런 식으로 만들어진 새로운 feasible 배치(들)에 대해서, 오퍼레이터는 최종적으로 각 node를 만든다.

목적 상태: 어떠한 상태 P에 있어서 $P_X = \varnothing$이라면 그것은 목적 상태가 된다. 다시 말해서, 모든 프로세서들에 대해서 자원 복사를 배치를 할 지 혹은 안 할 지가 결정이 되고 나면 탐색은 끝이 난다.

위와 같이 자원 배치 문제를 상태 공간 탐색 문제로 공식화하고 난 후, A^* 알고리즘을 사용하여 최적 고장 허용 자원 배치를 찾는다. A^* 알고리즘에서는 각 node의 확장 시에 순서를 정하기 위해 평가 함수를 사용하는데, 이러한 평가 함수가 적절히 잘 정의가 되어 있으면, 경로 비용이 최소인 최적 해 경로를 찾는 것이 보장된다. 보다 자세히 설명하자면, 만약 다음과 같은 평가 함수를 사용할 때:

$$\hat{f}(n) = g(n) + \hat{h}(n),$$

여기에서 $g(n)$은 출발 node에서 부터 현재 node n까지의 탐색 공간 상의 경로 비용이고, $\hat{h}(n)$은 node n에서 하나의 목적 node까지의 최소 경로 비용 $h(n)$에 대한

lower-bound에 대해서 가능한 휴리스틱 정보들을 사용하여 추정한 추정치이다. 여기에서, 모든 node n에 대해서 $\hat{h}(n) \le h(n)$이 만족된다면, A^*알고리즘은 상태 공간 상에서 어떠한 다른 "무정보" ("uninformed") 알고리즘에 비해 더 적은 node를 확장하고서 최적 해를 찾는 것이 보장되어 있다. 다시 말해서, 이러한 휴리스틱 탐색 알고리즘을 사용함으로써 최적 고장 허용 자원 배치를 찾는 시간을 단축시킬 수가 있으며, 병렬 및 분산 시스템이 대형일 경우 상당히 많은 시간 절약을 할 수가 있다.

앞에서 고장 허용 자원 배치 문제를 FT-dominating set 문제로 공식화하는 방법에 따르면, 상태 공간 상에서의 각 node n은 일부 프로세서들에 대해서 주어진 자원 복사를 가질 지 혹은 안 가질 지가 결정된 불완전 배치를 묘사한다. 따라서, 상태 공간 상의 각 node n에 대한 평가 함수를 $\hat{f}(n) = g(n) + \hat{h}(n)$로 설정했을 때, 여기에서 $g(n)$ 값은 node n에 해당하는 불완전 배치 P 하에서 자원 복사를 가지도록 결정된 프로세서의 수가 되고, 그리고 휴리스틱 함수 $\hat{h}(n)$ 값은 불완전 배치 P를 고장 허용 완전 배치로 만드는 데에 필요한 최소 자원 복사의 수인 $h(n)$ 값에 대한 lower-bound 추정치가 된다.

불완전 배치 P가 상태 공간 상의 node n에 해당한다고 했을 때, $g(n) = C(P_1)$이 된다. 그리고 휴리스틱 함수 $\hat{h}(n)$에 대해서는, 여러가지 다른 방법으로 그 값을 설정할 수가 있다. 가장 간단한 방법은 모든 n에 대해서 $\hat{h}(n) \equiv 0$로 설정하는 것으로서, 이것을 uniform-cost 탐색 방법이라고 하며[14], 이것 역시 최적 고장 허용 자원 배치를 찾는 것이 보장된다. 그러나 탐색 시간을 단축하기 위해서는, 0보다 큰 $\hat{h}(n)$값을 사용하여야 한다. 만약, $\hat{h}(n)$값을 앞 절에서 설명한 조건 (1) 식을 만족하는 t_{min} 값으로 설정을 하면, 특성 2에 의해서 $\hat{h}(n)$은 모든 n에 대해서 $h(n)$의 lower-bound임이 보장된다. $\hat{h}(n)$값은 다음의 알고리즘을 사용하여 계산할 수가 있다.

알고리즘 Comp_LB: Lower-bound $\hat{h}(n)$ 계산. 여기에서 P는 탐색 트리 상에서 node n에 해당하는 불완전 배치이다. 그래프 G의 노드 인접 정보를 이용하면, 각 노드 v_i의 이웃 집합 $N(v_i)$와 집합 P_1의 FT-이웃 집합 $N_{FT}(P_1)$을 쉽게 구할 수가 있다.

Step 1. 다음과 같이 $\alpha(P_1)$을 찾는다.

$$\alpha(P_1) = 2 \times C(N_{FT}(P_1)) + C(N(P_1) - N_{FT}(P_1)).$$

Step 2. 각 노드 $v_i \in P_X$ 에 대해서, 다음과 같이 P 하에서의 cover number $\alpha(v_i|P)$을 찾는다.

$$\alpha(v_i|P) = \alpha(P_1 \cup \{v_i\}) - \alpha(P_1).$$

Step 3. P_X에 있는 각 candidate 노드 v_i들을 $\alpha(v_i|P))$ 값에 대해서 non-increasing 순서로 정렬시킨다. 정렬된 순서가 $v_{x_1}, v_{x_2}, \ldots, v_{x_{C(P_X)}}$와 같다고 하자.

Step 3.1. 모든 노드 v_{x_i}에 대해서, 임의의 노드 v_{x_j}, $1 \le j \le i-1$,에 대해 $N_F(v_{x_i}|P) \subset N_F(v_{x_j}|P)$ 와 $N_H(v_{x_i}|P) \subseteq N_H(v_{x_j}|P)$이 만족되면, v_{x_i}는 trivial 노드로 선언한다. P_X로 부터 모든 trivial 노드 v_{x_i}를 추출하여 P_0로 넣는다. (즉, p_{x_i}를 0으로 만든다.)

Step 3.2. 모든 trivial 노드 v_{x_i}에 대해서, 만약 $v_{x_i} \in N_{FT}(P_1 \cup \{v_{x_j}\})$이고 또한 $v_{x_i} \notin N_{FT}(P_1 \cup \{v_{x_k}\})$, $\forall k (1 \le k (\ne j) \le C(P_X))$ 이 만족되면 v_{x_j}는 FT-essential 노드임을 선언한다. P_X로 부터 모든 FT-essential 노드 v_{x_j}를 추출하여 P_1로 넣는다. (즉, p_{x_j}를 1로 만든다.)

Step 4. 조건 (1) 식을 만족하는 t의 가장 작은 값 t_{min}을 찾아서 $\hat{h}(n)$ 값으로 설정한다.

특성 3과 4에 의해서, P_X에 있는 모든 trivial 노드들과 모든 FT-essential 노드들을 추출하여 각각 P_0와 P_1에 넣음으로써 탐색 시간을 더욱 단축시킬 수가 있다. Step 3에서의 정렬된 정보를 사용하여Step 3.1과 Step 3.2에서 모든 trivial 노드와 모든 FT-essential 노드들을 찾을 수 있다.

위에서 구한 $\hat{h}(n)$ 값이 모든 n에 대해서 $h(n)$의 lower-bound이므로, 아래의 알고리즘을 사용하면 최적 고장 허용 자원 배치를 찾는 것이 보장된다.

알고리즘 PLACEMENT: 최적 고장 허용 자원 배치를 찾는다.

Step 1. 위에서 설명한 초기 상태 s를 *OPEN*이라 불리는 리스트(list)에 넣고, 알고리즘 Comp_LB으로 계산한 t_{min} 값을 $\hat{f}(s)$값으로 설정한다.

Step 2. 리스트 OPEN에서 가장 작은 $\hat{f}$값을 가지는 node n을 추출하여 *CLOSE*라고 불리는 리스트에 넣는다.

Step 3. 만약 node n이 위에서 정의한 목적 상태를 만족하면, 그 node에 해당하는 자원 배치 P가 최적 고장 허용 자원 배치가 되며 알고리즘은 끝이 난다. 그렇지 않을 경우, 알고리즘은 Step 4로 계속된다.

Step 4. Node n에 오퍼레이터를 적용하여 node 확장을 하고, 각 상속자 node n'에 대해서 $\hat{f}(n') = g(n') + \hat{h}(n')$값을 계산한다. 각 node n'에 해당하는 배치가 P'라고 하자. 그러면, $g(n') = C(P')$이 되고,

$\hat{h}(n')$값은 알고리즘 Comp_LB를 통해 계산한다. 그리고 모든 상속자 node를 리스트 OPEN에 넣는다.

Step 5. Step 2로 간다.

Step 4에서, 만약 $\hat{h}(n')$값이 항상 0이면, 알고리즘 PLACEMENT는 uniform-cost 탐색이 되며, 그렇지 않을 경우 A^*알고리즘이 된다.

5. 고장 허용 자원 배치 예제

이 절에서는 그림 1에서 보인 그래프 상에서, 본 논문에서 제시한 상태 공간 탐색 방법을 사용하여 최적 고장 허용 배치를 찾는 것을 예로 보인다 (그림 2 참조). 초기 상태 s에서, v_1은 essential 노드이고, v_2는 v_3와 v_4에 의해서 covered되었으므로 trivial 노드가 된다. 그러므로, 특성 3과 4에 의해서 초기 상태는 $s=\{1,0,X,X,X,X,X,X,X,X\}$이 된다. 불완전 배치 s의 cover number는 $\alpha(\{v_1\})=3$이다. 또한, 노드 인접 정보를 이용하여 불완전 배치 s 하에서의 각 candidate 노드들의 cover number를 다음과 같이 구할 수 있다.

$$\alpha(v_3|s)=\alpha(\{v_1,v_3\})-\alpha(\{v_1\})=10-3=7,$$
$$\alpha(v_4|s)=\alpha(\{v_1,v_4\})-\alpha(\{v_1\})=10-3=7,$$
$$\alpha(v_5|s)=\alpha(\{v_1,v_5\})-\alpha(\{v_1\})=9-3=6,$$
$$\alpha(v_6|s)=\alpha(\{v_1,v_6\})-\alpha(\{v_1\})=9-3=6,$$
$$\alpha(v_7|s)=\alpha(\{v_1,v_7\})-\alpha(\{v_1\})=9-3=6,$$
$$\alpha(v_8|s)=\alpha(\{v_1,v_8\})-\alpha(\{v_1\})=9-3=6,$$
$$\alpha(v_9|s)=\alpha(\{v_1,v_9\})-\alpha(\{v_1\})=8-3=5, \text{ and}$$
$$\alpha(v_{10}|s)=\alpha(\{v_1,v_{10}\})-\alpha(\{v_1\})=8-3=5.$$

즉, 초기 상태 s에서, $N_1(P_1)=N_1(v_2)=\{v_1,v_2,v_3,v_4\}$. 여기에서 lower-bound 추정치인 $\hat{h}(s)$ (즉, 수식 (1)을 만족하는 $t_{\min}$) 값은 $\alpha(\{v_1\})+\alpha(v_3|s)+\alpha(v_4|s)+\alpha(v_5|s)=3+7+7+6=23\geq 20\ (=2|V|)$이므로 3이 된다. 따라서, 초기 상태에서의 node 비용은 $\hat{f}(s)=g(s)+\hat{h}(s)=1+3=4$. 이러한 초기 상태 s에서 node v_3를 각각 P_0와 P_1으로 넣음으로써 새로운 상속자 node $n1$과 $n2$를 확장하여 리스트 OPEN에 넣는다. 이와 같은 방법으로 본 알고리즘을 수행한 결과 나온 상태 공간 탐색 트리가 그림 2에 나타나 있다. 이 예제에서 최적 고장 허용 자원 배치는 $\{1,0,0,1,0,1,0,0,1,0\}$ 이고 (즉, 자원의 복사가 v_1,v_4,v_6, 그리고 v_9에 각각 배치되어 있음), 따라서 전체 배치 비용은 4가 된다. Node $n4$에서 v_6이 essential 노드이고, node $n2$ 에서 v_5,v_6,v_9, node $n4$에서 v_5,v_7,v_{10}이 모두 trivial 노드임에 유의하라.

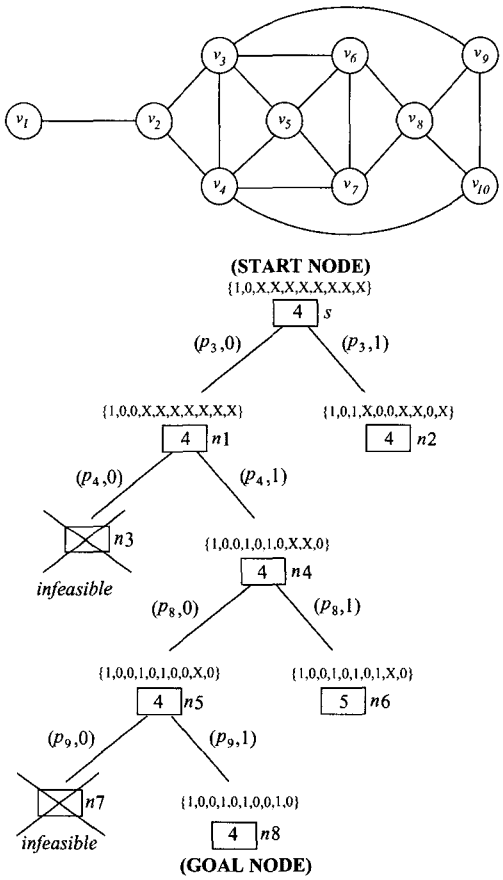

그림 2 예제 상태 공간 탐색 트리(네모 안의 숫자는 node 비용이고, $(p_i,0)$과 $(p_i,1)$은 각각 노드 v_i를 P_X에서 P_0와 P_1로 옮기는 node 확장을 의미한다.)

그림 2에서 보인 바와 같이, 목적 node를 찾을 때까지 전체적으로 4 개의 상태 공간 node가 확장되었고, 7개의 node가 만들어 졌다. 일반적으로, 탐색 트리의 높이는 그래프의 전체 노드 수와 일치한다. (이 예제에서는 10이 된다.) 그러나, 특성 3과 4에 의해서, 탐색 트리의 높이가 10보다 훨씬 작은 4로 줄었다. 이들 특성들의 효율성을 알아 보기 위해서, 알고리즘 PLACEMENT를 특성 3과 4를 사용하지 않고, 모든 node에 대해서 $\hat{h}(n)$ 값을 0으로 놓고 (즉, uniform-cost 알고리즘으로) 수행시켜 보았다. 그 결과 본 논문에 수록하기에는 너무 큰

탐색 트리가 생성되었으며, 그 트리 상에서 전체적으로 52개의 상태 공간 node가 확장되었고, 87개의 node가 만들어 졌다.

6. 실험 결과

고장 허용 자원 배치 문제에 대해, SUN/UltraSparc 상에서 정형(regular) 그래프와 임의(random) 그래프에 대해 각각 실험을 하였다. 본 논문에서 제안한 휴리스틱 정보(특성 3과 4)들의 효율성을 측정하기 위해 알고리즘 PLACEMENT를 수정하여 다음과 같이 3 가지 알고리즘을 고려하였다; Algorithm-1) 특성 3과 4를 모두 고려한 알고리즘, Algorithm-2) 특성 3은 고려하고 특성 4는 고려하지 않은 알고리즘, Algorithm-3) 특성 3 과 4를 모두 고려하지 않은 알고리즘. 그러나, 이 세 가지 알고리즘 모두 특성 1과 2는 고려하였다. (즉, feasible 한 노드들만 만들고, 각 node에서는 $h(n)$에 대해서 동일한 lower-bound 추정치 t_{min}을 사용한다.) 먼저 정형 그래프의 경우, $n \times n$ 메쉬 $(3 \le n \le 8)$와 깊이 n인 2진 트리 $(3 \le n \le 7)$상에서 각각 고장 허용 자원 배치 실험을 하였다. 각 실험에 대한 결과는 테이블 1과 2에 나타나 있으며, 여기에서 $\#_{gen}$와 $\#_{exp}$은 각각 만들어진 전체 node의 수 및 확장된 node의 수이다. 트리의 경우 Algorithm-3 의 깊이 7일 때 계산 시간이 너무 많이 걸려 결과를 얻지 못하였다. 메쉬의 경우 Algorithm-1에 의해 만들어 진 node 및 확장된 node의 수가 Algorithm-2의 그것에 대하여 46%-70% 정도이며, 또한 Algorithm-2에 의해 만들어 진 node 및 확장된 node의 수도 Algorithm-3에 대하여 60% 정도임을 알 수 있다. 그리고, 트리인 경우는 Algorithm-1이 Algorithm-2에 대하여 80%-18% 수준으로 문제의 크기가 커질 수록 성능이 좋아짐을 볼 수 있으며, Algorithm-2도 Algorithm-3 에 대하여 생성되는 node 및 확장 되는 node의 갯수가 30%-0.15% 수준으로 역시 문제의 크기가 커질 수록 그들 사이의 성능 차이는 더욱 커짐을 알 수 있다. 이로써, 정형 그래프 상에서 최적 고장 허용 배치를 찾는 데에 있어서 특성 3과 4를 이용함으로써 상당한 시간 단축을 시킬 수 있음을 알 수 있다.

두 번째 실험은 임의 그래프 $G = (V, E)$ $(|E| \simeq |V|(|V|-1)/x)$ 상에서 고장 허용 배치 문제에 대해서 행해졌다. 각 임의 그래프는 x 값에 따라서 dense (x=10)와 sparse (x=20) 그래프로 구분하여 실험을 하였다. 각 실험 마다 10 개의 임의 그래프를 사용하였으

표 1 $n \times n$ 메쉬 상에서의 실험 결과.

n	Algorithm-1		Algorithm-2		Algorithm-3	
	$\#_{gen}$	$\#_{exp}$	$\#_{gen}$	$\#_{exp}$	$\#_{gen}$	$\#_{exp}$
3	5	3	9	6	15	9
4	62	39	102	66	180	112
5	213	136	306	196	524	327
6	8114	5198	17434	11134	29279	13255
7	350559	203550	754358	438072	1266577	734498
8	17598061	8569455	31683036	18355216	54462811	31216165

표 2 깊이 n인 2진 트리 상에서의 실험 결과.

n	Algorithm-1		Algorithm-2		Algorithm-3	
	$\#_{gen}$	$\#_{exp}$	$\#_{gen}$	$\#_{exp}$	$\#_{gen}$	$\#_{exp}$
3	4	2	5	3	13	7
4	8	4	12	8	38	20
5	11	7	40	34	632	331
6	76	59	329	293	212364	113789
7	1064	856	5264	4688	-	-

표 3 Dense 임의 그래프 상에서의 실험 결과.

n	Algorithm-1		Algorithm-2		Algorithm-3	
	A_{gen}	A_{exp}	A_{gen}	A_{exp}	A_{gen}	A_{exp}
10	2.9	1.2	3.7	2.2	8.5	4.6
20	5.9	3.3	8.4	6.1	30.0	17.8
30	143.1	89.6	547.1	390.1	1816.0	1122.6
40	18880.7	11997.7	39605.9	26642.3	134877.3	80650.9
50	343547.8	199483.2	738872.2	428130.6	1241453.5	719778.0

표 4 Sparse 임의 그래프 상에서의 실험 결과.

n	Algorithm-1		Algorithm-2		Algorithm-3	
	A_{gen}	A_{exp}	A_{gen}	A_{exp}	A_{gen}	A_{exp}
10	-	-	-	-	-	-
20	4.9	2.5	7.8	5.7	20.9	11.8
30	7.3	3.9	12.8	9.5	51.8	29.7
40	10.6	6.4	29.0	22.2	286.8	172.1
50	294.3	196.7	705.7	535.4	37162.0	22649.1
60	11772	11605.3	34367.59	23878.84	9290500	5888766

며, 그 결과는 dense와 sparse 그래프에 대해 각각 테이블 3과 4에 나타나 있다. 여기에서 A_{gen} 및 A_{exp}는 각각 전체 만들어 진 node들 및 전체 확장된 node들의 평균치이다. Sparse 그래프의 경우 n=10일 때, 연결(connected) 그래프를 구할 수가 없으므로 실험을 생략하였다. 테이블 3과 4에서도 알 수 있듯이, 노드의 수가 30 이상인 경우, Algorithm-1에 의해 만들어 진 node 및 확장된 node의 수는 Algorithm-2의 그것 보다 약 50% 이내임을 알 수 있으며, 또한 Algorithm-2에 의해 만들어 진 node 및 확장된 node의 수는 Algorithm-3의 그것 보다 약 30% 이내 임을 알 수 있다. 그리고, 동일한 노드 수를 가지는 그래프 상에서는 엣지 수가 적을수록 만들어 진 node의 수가 훨씬 적어짐을 알 수 있다. 이것은 엣지 수가 적을수록 각 불완전 배치에서 trivial 노드와 essential 노드들이 더욱 많이 생기기 때문이다. 이 모든 실험 결과들을 통해 볼 때, 특성 3과 4를 사용하여 최적 배치를 찾는 데에 걸리는 시간을 훨씬 단축시킬 수 있음을 알 수 있다.

7. 결 론

본 논문에서는 병렬 및 분산 시스템에서 최소한의 자원 복사를 사용하여, 임의의 프로세서나 링크의 고장이 나더라도 각 프로세서들이 거리 1 이내에서 적어도 하나의 자원 복사를 접근할 수 있도록 하게 하는 고장 허용 자원 배치 문제에 대해서 고려하였다. 먼저 이 문제는 그래프 이론에서의 FT-dominating set 문제로 변환이 되었으며, A^*알고리즘을 사용하여 상태 공간 탐색 방법으로 최적 해를 구하였다. 또한, 최적 해를 구하는 데에 걸리는 시간을 줄이기 위하여 FT-dominating set의 특성을 분석하여 유용한 휴리스틱 정보들을 도출하였다.

일반적으로, $|V|$개의 프로세서를 가지는 배치 문제에서는 $2^{|V|}$개의 가능한 배치가 있고, 따라서 최적 해를 구하는 데에는 $O(2^{|V|})$ 만큼의 시간이 소요된다. 그러나, 여러 그래프 상에서의 실험 결과를 통해서, 본 논문에서 제안한 휴리스틱 정보들을 사용하면 최적 해를 구하는 시간을 상당히 단축시킬 수 있음을 보였다. 앞으로 수행해야 할 연구 과제로는, 알고리즘 Comp_LB를 통해 제안된 $h(n)$의 lower-bound 추정치인 $\hat{h}(n)$값에 대해서 보다 향상된 추정치에 대한 연구와, 탐색 시간을 더욱 줄일 수 있는 FT-dominating set에 대한 보다 효율적인 특성들을 도출하는 것이다. 또한, 본 논문에서는 노드들 간의 거리만을 고려한 자원 배치 문제를 다루었지만, 실제 상황에서는 자원의 사용 빈도에 따라 각 통신 채널들의 충돌 정도가 차이가 나므로, 이러한 통신 채널의 용량을 고려한 배치 문제가 앞으로 수행되어야 할 연구 과제이다.

참 고 문 헌

[1] O. Kariv and S. L. Hakimi, "An algorithm approach to network location problems I: the p-centers," *SIAM J. Appl. Math.*, vol. 37, no. 3, pp. 513-538, Dec. 1979.

[2] R. Cole and U. Vishkin, "The accelerated centroid decomposition techniques for optimal parallel tree evaluation in logarithmic time," *Algorithmica*, vol. 3, pp. 329-346, 1988.

[3] X. He and Y. Yesha, "Binary tree algebraic computation and parallel algorithms for simple graphs," *J. Algorithms*, vol. 9, pp. 92-113, 1988.

[4] X. He and Y. Yesha, "Efficient parallel algorithms for r-dominating set and p-center problems on trees," *Algorithmica*, vol. 5, pp. 129-145, 1990.

[5] S. Kutten and D. Peleg, "Fast distributed construction of small k-dominating sets and applications," *J. Algorithms*, vol. 28, pp. 40-66, 1998.

[6] A. L. N. Reddy, P. Banerjee, and S. G. Abraham, "I/O embedding in hypercubes," *Proc. Int'l Conf. on Parallel Processing*, pp. 331-338, Aug. 1988.

[7] M. Livingston and Q. F. Stout, "Distributing resources in hypercube computers," *Proc. The 3rd Conf. on Hypercube Concurrent Comp. and Appl.*, pp. 222-231, Jan. 1988.

[8] M. Livingston and Q. F. Stout, "Perfect dominating sets," *Congressus Numerantium*, vol. 79, pp. 187-203, 1990.

[9] G. -M. Chiu and C. S. Raghavendra, "Resource allocation in hypercube systems," *Proc. Dist. Memory Computing Conf.*, pp. 894-902, April 1990.

[10] H. -L. Chen and N. -F. Tzeng, "Efficient resource placement in hypercubes using multiple-adjacency codes," *IEEE Trans. on Computers*, vol. 43, no. 1, pp. 23-33, Jan. 1994.

[11] P. Ramanathan and S. Chalasani, "Resource placement in k-ary n-cubes," *Proc. Int'l Conf. on Parallel Processing*, vol. 2, pp. 133-140, Aug. 1992.

[12] 이철훈, "병렬 및 분산 시스템에서 자원의 최적 k-한도 배치", 정보과학회논문지(A), 제 23 권, 제 7 호, pp. 681-690, July 1996.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[14] N. J. Nillson, *Problem Solving Methods in Artificial Intelligence.* New York: McGraw-Hill, 1971.

김 종 훈

1996년 충남대학교 컴퓨터공학과(학사). 1998년 충남대학교 컴퓨터공학과(석사). 1998년 ~ 현재 충남대학교 컴퓨터공학과 박사과정. 관심분야는 병렬 처리, 운영 체제, 결함 허용, 실시간 시스템 등임.

이 철 훈

1979년 ~ 1983년 서울대학교 전자공학과 학사. 1983년 1986년 삼성전자 컴퓨터개발실 연구원. 1986년 ~ 1988년 한국과학기술원 전기및전자공학과 석사. 1988년 ~ 1992년 한국과학기술원 전기및전자공학과 박사. 1992년 ~ 1994년 삼성전자 컴퓨터사업부 선임연구원. 1994년 ~ 1995년 Univ. of Michigan 객원연구원. 1995년 2월 ~ 현재 충남대학교 컴퓨터공학과 조교수. 관심분야는 운영체제, 병렬처리, 결함 허용 및 실시간 시스템임.