WWW 기반 자바 병렬 처리 시스템에서 적응적 태스크 할당 기법

최광희*, 한연희*, 정영식**, 황종선*
*고려대학교 컴퓨터학과, **원광대학교 컴퓨터공학과

An Adaptive Task Allocation Scheme in a Java Parallel Processing System based on the WWW

KwangHee Choi*, YounHee Han*, YoungSik Jung**, Chong-Sun Hwang*
*Dept. of Computer Science and Engineering, Korea University,
**Dept. of Computer Science and Engineering, WonKwang University

요 약

WWW 에서 의뢰인-병렬처리 서버-작업자 구성을 이용하여, 작업자 애플릿을 임의의 호스트에 분산시키고, 대량의 연산수행을 지닌 작업을 배분하여 수행시킨 뒤, 그 결과를 의뢰인에게 보여주는 WWW 기반 자바 병렬 시스템이 기존의 LAN 상에서의 병렬 시스템보다 확장성 및 이용 용이성 면에서 크게 주목을 끌고 있다. 이러한 WWW 기반 자바 병렬처리 시스템에서 서버가 주어진 태스크들을 작업자들에게 할당하는 효율적인 기법이 크게 요구된다. 본 논문에서는, 이미 구현된 WWW 기반 자바 병렬 시스템 원형(prototype)에서 효율적인 적응적 태스크 할당 기법을 제시한다. 제안하는 적응적 태스크 할당 기법이 WWW 에서 여러 호스트들의 성능이 시간의 흐름에 따라 크게 변화하는 상황에 크게 이점이 있음을 성능 분석 및 평가를 통해 보여준다

1. 서론

WWW는 수 백만 대의 컴퓨터를 연결한 가장 큰 가상 시스템이 되고 있다. 게다가, 플랫폼 독립적이며, 네트워크 관련 API 및 쓰레드 처리가 용이한 자바 때문에 클라이언트-서버 스타일 인터넷 프로그램 개발이 용이해 졌다[1, 2]. 임의의 작업 한 개를 여러 개의 태스크들로 나누어서, 병렬적으로 여러 호스트에서 수행시키면, 수행하는데 있어서 총시간을 줄일 수 있어서 매우 효율적이다. 하나의 작업을 하나의 호스트에서 수행할 때 걸리는 시간이 T라고 하면, 동일한 성능을 가진 N개의 호스트에서 수행을 할 때는 T/N 이 되는 것이 이상적이다 따라서, 병렬 처리를 수행하는 호스트의 수가 많으면, 수행 시간을 줄일 수 있음은 자명하다 특히, WWW에 있는 호스트들을 병렬 처리의 대상으로 삼는다면 매우 큰 능력을 얻을 수 있다[3]

WWW의 호스트들은 그 수가 많은 만큼 그 성능도 다양하다 따라서, 각 호스트에 수행 능력에 맞게 태스크를 할당해 주는 일은 매우 중요하다. 병렬처리 시스템에서 태스크들의 할당의 목적은 모든 작업자들이 같은 시간동안 수행을 하여 결과를 구하도록 각 작업자들의 성능에 맞게 알맞은 양의 태스크들을 할당하는 것이다[4]

그러나, 주시해야 할 점은 시간의 흐름에 따라서 호스트의 성능이 변한다는 것이다 특히, 웹 상에 있는 호스트들은 사용자들에게 개방되어 있어서 더욱 영향을 받는다 여러 사용자가 컴퓨터를 사용함에 의해 프로세스의 수가 증가될 수도 있고, 사용중인 프로세스를 종료시킴에 의해 프로세스의 수가 감소될 수도 있다[5]

따라서, 시간이 흐름에 따라 변하는 각 호스트의 수행능력을 인식하여 태스크를 적응적으로 할당하여 준다면, 더욱 효율적인 성능을 기대할 수 있다. 본 논문에서는 적응적으로 각 호스트의 수행능력을 인식하여 태스크를 분배하는, 적응적 태스크 할당 기법을 제안한다

또한, 적응적 태스크 할당 기법에 대응하는 정적 태스크 할당 기법과 함께 수행 성능 분석을 하고, WWW에서 적응적 태스크 할당 기법이 정적 태스크 할당 보다 더 성능이 좋은 타당성을 제시한다 그리고, 실제 구현된 WWW 기반 자바 병렬 처리 시스템에서 제시한 알고리즘을 적용한 성능 평가를 보인다.

† 이 연구는 정보통신부의 대학기초연구 지원사업에 의하여 수행 되었음

2. 제안하는 적응적 태스크 할당 기법

앞에서 언급하였듯이 웹 상의 호스트들은 시간에 따라 성능이 부하에 의해 변한다. 따라서, 웹 기반 병렬 처리 시스템에서는, 시간에 따라 변화하는 작업자들의 성능에 대처할 수 있는 적응적 태스크 할당 기법이 필요하다. 다음은 적응적 태스크 할당 기법을 설명하기 위한 기호들이다

Ω : 작업자들의 집합
Wi : 작업자 i
Wc : Complete 메시지 보낸 작업자
Γ : 태스크들의 집합
Γi : 작업자 i에 할당된 태스크들의 집합
Πi 작업자 i의 미수행 태스크의 집합
Complete : 작업자 i가 Γi를 다 수행시 보내는 메시지
Stop : 작업 수행을 종료하라는 메시지

그림 1은 서버에서 수행하는 적응적 태스크 할당 기법을 보여준다. 의뢰자가 서버에게 작업을 요청하면 서버는 주어진 작업을 일정 크기의 태스크들로 나누고 이 작업을 위해 참여한 모든 작업자들의 성능 비에 따라 태스크들을 할당하여 준다 작업자는 자신에게 할당된 모든 태스크를 마치면, 서버에게 Complete 메시지를 보낸다 서버는 Complete 메시지를 받으면 이 메시지를 보낸 작업자를 제외한 모든 작업자들에게 Stop 메시지를 보낸다.

Server

```
∀Wi∈Ω, allocate Γi to Wi;  Γ← NULL;
isReceiveAll = true;
while (true) {
   receive(Complete);
   ∀(Wi∈Ω and Wi≠Wc), send(Stop) to Wi;
   ∀Wi∈Ω,  Γ ← Γ∪ ⋃_{i=0}^{N-1} Πi;
   while (isReceiveAll) {
        reset Γi;
        ∀ Wi∈Ω,  allocate Γi to Wi;
        isReceiveAll = false;
   }
}
```

그림 1 적응적 태스크 할당 기법

Stop 메시지를 받은 작업자는 작업을 중단하고 아직 처리하지 못한 태스크 집합을 보내준다. 서버가 모든 작업자들로부터 이 정보를 받으면, 아직 수행하지 못한 태스크들을 작업자들이 지금까지 수행한 태스크의 개수의 비에 근거해서 태스크 재 할당을 수행한다

3. 태스크 할당 기법 분석

태스크 할당의 기본 정책은 다음과 같다 임의의 의뢰인이 수행하고자 하는 작업 J를 서버에게 전달한다 서버는 그 작업을 일정한 크기의 태스크들로 나누고, 그 작업 수행에 참여 가능한 임의의 작업자 W_i들에게 서로 다른 수의 태스크들을 할당한다. 각각의 작업자 W_i들은 할당된 태스크 단위로 작업을 처리하여 의뢰인에게 결과를 준다

태스크 할당 기법 분석을 위한 인자들은 다음과 같으며, 이 인자는 정적과 적응적 태스크 할당 기법에서 공통으로 쓰인다.

- $TotalNmTask$: 서버에 의해 나누어진 총 태스크 수
- NmW : 주어진 작업 J을 처리하는 작업자들의 수
- U_J : 주어진 작업 J의 총 연산량
- $NmTaskW(t)_i$: 시간 t에서 W_i에 할당되는 태스크 수
- $P(t)_i$: W_i의 시간 t에서의 단위 시간당 연산 수행량
- $TimeCS$: 의뢰인에서 서버까지 메시지 전송 시간
- $TimeSW_i$: 서버로부터 W_i까지 메시지 전송 시간
- $TimeW_iS$: W_i로부터 서버까지 메시지 전송 시간
- $TimeW_iC$: W_i로부터 의뢰인까지 메시지 전송 시간

3.1 정적 태스크 할당 기법 분석

모든 작업자들은 서버에게 등록을 할 때, 자신의 벤치마크 수행결과를 함께 전해준다 그것을 태스크 할당 시간 t_0에 $P(t_0)_i$로 가정하여 $TotalNmTask$를 $P(t_0)_i$ 의 비율로 각각 나누어 각 작업자들에게 한번만 할당한다 즉, $\left(P(t_0)_i \;/\; \sum_{i=0}^{NmW-1} P(t_0)_i \right) \cdot TotalNmTask$ 로 $NumTaskW(t_0)_i$ 이 결정된다. 이 때, 각 W_i가 할당받는 모든 태스크를 처리하는 시간을 $StaticTimeW_i$ 서버가 정적 태스크 할당 스케줄링을 행하는 시간을 $TimeS_S$라고 가정하자

[정의 1] 정적 태스크 할당에서, 의뢰인이 작업개시 후 모든 결과를 받을 때까지의 총 수행 시간 $TotalStaticTime$은 다음으로 정의된다.

$$T_k = TimeSW_k + StaticTimeW_k + TimeW_kC .$$

$$TotalStaticTime = TimeCS + TimeS_S + max\{T_0, T_1, T_2, \cdots, T_{Num\,W-1}\} \quad \square$$

[정의 1]에 의해 정적 태스크 할당에서 $TotalStaticTime$을 줄이려면, $max\{T_0, T_1, T_2, \cdots, T_{Num\,W-1}\}$을 줄여야 함을 알 수 있다 가정에 의해 $TimeSW_i$ 및 $TimeW_iC$ 은 상수이므로, $StaticTimeW_k$가 $TotalStaticTime$의 가장 중요한 요소가 된다

한편, 주어진 상수들에서, $\frac{U_J}{TotalNmTask}$는 단위 태스크 당 할당되는 연산량이므로, 임의의 시간 t_0에서 임의의 W_i에 할당되는 태스크들이 지닌 총 연산량은 $\frac{U_J \cdot NmTaskW(t_0)_i}{TotalNmTask}$ 이다 그러므로, $StaticTimeW_k$는 다음과 같이 결정된다.

$$StaticTimeW_k = \frac{U_J \cdot NmTaskW(t_0)_i}{TotalNmTask \cdot P(t_0)_i} = \frac{U_J}{\sum_{i=0}^{NmW-1} P(t_0)_i} \qquad (1)$$

(1)에 의해 정적 태스크 할당 기법은 모든 작업자들에게 동일한 시간에 할당된 태스크들을 처리하는 것을 기대한다. 이것은 1장에서 정의한 태스크 할당 목적에 부합된다. 그러나, (1)은 작업자의 성능이 처음에 고정되어, 작업 수행중에는 변하지 않을 때에 의미가 부여된다 따라서, 시간에 따라 작업자의 성능이 크게 변하는 웹 환경의 호스트들에게는 (1)의 $StaticTimeW_k$ 에 가변적인 ΔT_i가 더 추가된다.

[정의 2] 정적 태스크 할당에서 임의의 W_i 에 할당되어진 모든 태스크를 처리하는 시간 $StaticTimeW_i$는 다음으로 정의된다.

$$StaticTimeW_k = \frac{U_J}{\sum_{i=0}^{NmW-1} P(t)_i} + \Delta T_i \quad \square$$

그러므로, 정적 태스크 할당에서 $TotalStaticTime$은 기본적으로 예측되는 시간이외에 각 호스트의 성능이 변함으로써 얻어지는 ΔT_i 에 크게 영향받는다 호스트의 고장을 고려해 보았을 때, ΔT_i의 범위는 0에서 무한대까지이다.

3.2 적응적 태스크 할당 기법 분석

제시된 기법에 의하여 서버에 의해 태스크를 재 할당하는 수 ε는 호스트의 성능변화율에 따라 달라진다 한편, 서버가 할당 스케줄링을 하는 시간을 $TimeS_a$라고 가정하고, 서버와 작업자 W_k, 작업자 W_k와 서버와의 메시지 교환 횟수인 τ_1 와 τ_2는 그림 1의 알고리즘에 의해 다음과 같이 정해진다.

$$\tau_1 = \begin{cases} \varepsilon & (W_i\text{가 작업이 끝났을 때}) \\ 2 \cdot \varepsilon & (W_i\text{가 작업을 수행중이었을 때}) \end{cases}$$

$$\tau_2 = \begin{cases} \varepsilon & (W_i\text{가 작업이 끝났을 때}) \\ \varepsilon & (W_i\text{가 작업을 수행중이었을 때}) \end{cases}$$

[정의 3] 적응적 태스크 할당에서 의뢰인이 작업개시 후 모든 결과를 받을 때까지의 총 수행 시간 $TotalAdaptiveTime$은 다음으로 정의된다

$$T_k = \tau_1 \cdot TimeSW_k + \tau_2 \cdot TimeW_kS + AdaptiveTimeW_k + TimeW_kC .$$

$$TotalAdaptiveTime = TimeCS + \varepsilon \cdot TimeS_a + max\{T_0, T_1, T_2, \cdots, T_{Num\,W-1}\} . \quad \square$$

[정의 3]과 [정의 1]을 비교해 볼 때, $TotalAdaptiveTime$에 정적 태스크

할당 보다 적응적 태스크 할당에서 추가되는 시간이 있다. 우선 각 작업자 W_k 마다 서버와의 메시지 교환에서 $(\tau_1 - 1) \cdot TimeSW_k + \tau_2 \cdot TimeW_kS$이 더 필요하며, 서버에서는 태스크 할당 스케줄을 위하여 $\varepsilon \cdot TimeS_a - TimeS_S$ 가 더 필요하다.

하지만, 적응적 할당 기법은 [정의 3]의 $StaticTimeW_k$에 추가되는 예측 불가능한 ΔT_i를 없애려는 방법이기 때문에, $AdaptiveTimeW_k$가 $TotalAdaptiveTime$를 결정하는 가장 중요한 요소가 됨을 알 수 있다

임의의 재 할당 z번째에, 처리되는 작업 J의 연산량을 $U(z)_J$. 작업자 W_i가 지니게 되는 단위 시간당 연산 수행량을 $P(t)_i$ 라고 가정하자.

[정의 4] 적응적 태스크 할당에서 임의의 W_i 에 할당되어진 모든 태스크를 처리하는 시간 $AdaptiveTimeW_i$는 다음으로 정의된다

$$AdaptiveTimeW_k = \sum_{z=0}^{\tau-1} \frac{U(z)_J}{\sum_{i=0}^{NumW-1} P(t_z)_i} \quad \square$$

[정의 4]의 $AdaptiveTimeW_i$는 [정의 3]의 $StaticTimeW_k$과는 다르게, 호스트의 성능변화에 따라 추가되는 예측 불가능한 시간이 추가되지 않는다. 즉, $(\tau_1 - 1)TimeSW_k + \tau_2 \cdot TimeW_kS + \varepsilon \cdot TimeS_a - TimeS_S$를 $MoreTimeW_k$ 이라고 가정하면, $StaticTimeW_k - AdaptiveTimeW_k$가 $max(\ \forall k,\ 0 \le k \le N-1, MoreTimeW_k)$ 보다 크다면, 적응적 태스크 할당 기법이 정적 태스크 할당 기법보다 우수하다 하지만, 각 호스트가 성능의 차이가 심한 웹에서는 $StaticTimeW_k - AdaptiveTimeW_k$ 가 큰 값을 가질 때가 대부분이다

4. 성능 평가

본 논문에서 제시하는 적응적 할당 기법의 성능을 평가하기 위해 구현되어진 시스템에서는, 프랙탈 이미지 처리를 위한 태스크를 작업자에게 할당하여 처리된 후 그 결과를 의뢰인에게 전달해준다. 서버로서 UltraSPARC 10을 사용하고, 작업자 및 의뢰인을 위한 호스트로는 UltraSPARC 10과 Pentium 200을 함께 사용한다.

WWW에 상주하는 호스트들의 성능이 시간에 따라 변한다는 특성을 반영하기 위해 작업자의 애플릿에 임의의 시간동안 중단하였다가 일정한 양의 루프(loop)를 돌아 CPU의 시간을 빼앗는 쓰레드를 임의의 수만큼 생성시킨다

작업자의 수는 4대로 고정하고, 작업자 3의 성능을 1로 설정하였을 때, 작업자 0, 1, 2는 각각 1.35, 2.28, 1.33의 성능을 보인다 시스템에 총 256라인의 256 픽셀 그림을 리는 프랙탈 작업을 수행시키는데, 태스크의 크기를 2라인, 8라인과 32라인으로 나누어 수행한다.

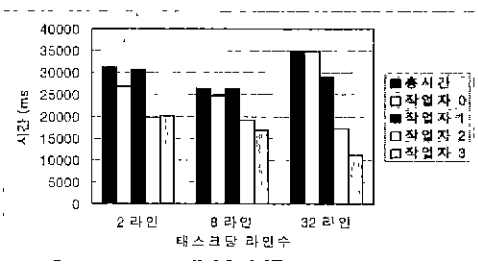

그림 2. 정적 할당의 수행 시간

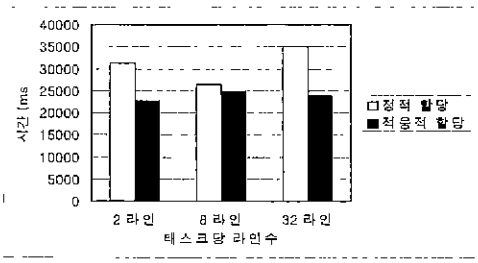

그림 3 적응적 및 정적 태스크 할당의 수행 시간 비교

그림 2에서는 각 작업자간의 성능 비율에 맞게 정적으로 태스크를 할당하여 그 작업자들이 동일한 시간동안 일할 것을 기대하지만, 시간이 흐름에 따라 변하는 각 작업자의 성능 때문에, 서로 다른 시간동안 일하는 모습을 보여준다 또한, 가장 늦게까지 일하는 작업자의 수행 시간에 총 수행 시간이 절대적으로 영향 받음을 알 수 있다.

그림 3에서는 적응적 태스크 할당이 정적 태스크 할당 보다 약 22%수행시간을 단축시키는 효과를 보여준다. 이러한 단축 효과는 작업자 호스트의 성능 변화에 따라 항상 다른 값을 보여준다.

5. 결론과 향후과제

본 논문에서는 웹 기반 병렬 처리 시스템에서, 시간에 따라 성능이 변하는 작업자의 상황을 극복하기 위해 작업자들의 작업 수행 시간에 적응적으로 태스크를 할당하여 병렬 처리의 성능을 향상시키는 기법을 제안하고, 이 기법을 분석하여 모든 작업자가 동일한 시간동안 작업을 수행하게 한다는 것을 보여주었다. 성능 평가를 통해서 이 기법이 성능을 향상시킴을 보여 주었다. 향후 과제로 동적으로 변화하는 작업자에 대해 병렬성을 극대화시키는 방법이 남아 있다

참고 문헌

[1] K M Chandy, B Dimitrov, H Le, J Mandleson, M. Richardson, A Rifkin, P. A. G. Sivilotti, W. Tanaka, and L Weisman. "A World-Wide Distributed System Using Java and the Internet," *the Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, Syracuse, NY, August 1996.

[2] Bernd O. Christiansen, Peter C. Japello, Mihai F Ionescu, Michael O. Neary, Klaus E Shauser, and Danieal Wu. "Javelin' Internet-based parallel computing using java," *the ACM Workshop on Java for Science and Engineering Computation*, 1997

[3] Jerrell Watts, Stephen Taylor "A Practical Approach to Dynamic Load Balancing," *IEEE Transaction on Parallel and Distributed Systems*,Vol 9. NO.3, March 1998

[4] Timothy Brecht, Xiaotie Deng, Nian Gu. "Competitive Dynamic Multiprocessor Allocation for Parallel Applications," *the Seventh Symposium on Parallel and Distributed Processing(SPDP'95)*, October 25-28, pp448-455,1995

[5] Xiaotie Deng, Nian Gu, Tim Brecht, KaiCheng Lu "Preemptive Scheduling of Parallel Jobs on Multiprocessors," *the Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, January. 1996, pp. 159-167