자발적 컴퓨팅 환경에서 P2P기반 작업분배기법

권경희[O1], 최장원[1], 이규철[2]
[1]한국과학기술정보연구원, [2]충남대학교 컴퓨터공학과
{khkwon[O1], jwcho[1]}@kisti.re.kr, khlee[2]@cs.knu.ac.kr

Work Distribution Mechanism based on P2P in Volunteer Computing

Kyounghee Kwon[O1], Jangwon Choi[1], Kyuchul Lee[2]
[1]Korea Institute of Science and Technology Information,
[2]Dept. of Computer Engineering, Chungnam National University

요 약

자발적 컴퓨팅은 인터넷의 연결된 가정 혹은 사무실의 PC를 집적하여 고성능 컴퓨팅을 창출하고자 하는 것이 목표이다. 이는 많은 자원제공자가 참여할 수 있도록 확장성이 보장되어야 한다. 하지만 기존의 자발적컴퓨팅 시스템에서는 중앙관리서버에 의해서만 작업분배가 이루어지므로 서버의 병목현상이 발생하게 된다. 본 논문에서는 결함포용에 사용되는 중복수행기법을 이용하여 P2P기반의 중복작업분배기법을 제안한다. 제안하는 P2P기반 중복작업분배기법은 서버의 병목현상을 줄이고, 서버의 결함 시에도 일정시간 서비스를 유지할 수 있는 강건함을 보였다.

1. 서 론

최근 네트워크의 성능향상과 고사양의 PC 보급으로, 이를 이용하여 저비용·고성능 컴퓨팅 능력을 창출하여 대용량의 연산처리를 필요로 하는 기상, 천문학, 바이오 등의 응용문제를 풀기 위한 움직임이 활발히 이루어지고 있다. 대표적인 인터넷 기반의 고성능 컴퓨팅 시스템으로 그리드 컴퓨팅(Grid Computing)[1-2]과 자발적 컴퓨팅(Volunteer Computing)[3-5]을 들 수 있다. 그리드 컴퓨팅은 조직내의 다양한 컴퓨팅 자원(슈퍼컴퓨터, PC, 데이터 등)을 가상조직으로 구성함으로써 고성능의 컴퓨팅 파워를 제공하는 컴퓨팅 패러다임이라면, 자발적 컴퓨팅은 인터넷에 연결되어있는 일반 대중의 PC를 대상으로 유휴(idle) 컴퓨팅 자원을 이용하여 고성능의 컴퓨팅 성능을 창출하고자 하는 패러다임이다. 그리드 컴퓨팅과 자발적 컴퓨팅은 대규모의 분산된 자원을 활용하여 고성능·대용량 컴퓨팅 파워를 얻어내고자 하는 목적은 동일하나, 그리드 컴퓨팅은 각 자원의 소유가 확실하며 명확하게 관리 잘 관리되는 고성능의 컴퓨팅 및 네트워킹 자원 표준화 추진되어지고 있는 반면, 자발적 컴퓨팅은 병렬 처리만을 위해 사용되는 PC들이 아니라 각각 나름대로의 목적에 이용되는 개인 소유의 PC들이므로 제어가 불가능하고 기종 및 성능 면에서 매우 큰 다양성을 가진다.

자발적 컴퓨팅은 인터넷에 연결된 PC를 대상으로 함으로 많은 PC가 참여할수록 잠재적인 컴퓨팅 파워는 높아진다. 하지만, 기존의 SETI@home[6], Folding@Home[7], Korea@Home[8]과 같은 자발적 컴퓨팅 시스템은 작업 분배 및 자원제공자 관리가 중앙관리서버에 의해 수행되어지기 때문에 서버의 병목현상과 함께 확장성(scalability)에 제약사항이 발생한다. 또한 서버의 결함(fault)시 전체 시스템이 중단되는 단점이 존재한다. 이러한 문제를 해결하기 위해 Bayanihan[9], Javelin[10] 같은 연구들은 복수개의 지역 서버를 두어 해결하려 했으나 여전히 확장성과 병목현상 및 서버의 결함에 대해서 연산의 지속적인 수행을 보장하지 못하는 한계가 발생한다.

본 논문에서는 자발적 컴퓨팅 환경에서 P2P를 이용한 작업 분배 기법을 제안하고자 한다. 이는 중앙집중식의 자발적 컴퓨팅 환경에서 중앙관리서버만 작업 분배를 함으로써 발생하는 서버의 병목현상을 줄여 많은 자원제공자가 참여할 수 있도록 확장성을 향상시키며, 중복 작업 데이터에 빠르게 접근함으로서 전체 연산 수행 시간을 줄일 수 있다. 또한 중복된 작업에 대하여 서버의 중개 없이도 P2P기반의 작업 분배가 이루어지기 때문에 서버의 결함 시에도 일정시간 서비스를 유지할 수 있다.

2. 관련연구

자발적 컴퓨팅 환경은 크게 클라이언트, 자원제공자, 중앙관리서버로 구성된다. 클라이언트는 자원제공자에게 작업을 요청하는 주체이며, 자원제공자는 자신의 유휴 컴퓨팅 자원을 제공해주는 자발적 자원 제공자이다. 중앙관리서버는 인터넷에 연결된 자원제공자들과 작업을 통제하는 관리자이다. 일반적으로 클라이언트는 서버에 병렬처리가 필요한 작업을 위탁한다. 중앙 관리 서버는 응용작업을 각자 자신의 입력을 가진 코드와 데이터로 구성된 세부작업(sub-job)으로 나눈다. 본 논문에서는 세부작업을 단위작업(WU: Work Unit, 이하 WU)라고 한다. 서버는 이렇게 나눠진 WU을 스케줄링 알고리즘에 따라 자원제공자들에게 분배한다. 자원제공자는 할당받은 WU을 수행하고 그 결과를 서버에게 반환한다. 마지막으로 서버는 자원제공자로부터 받은 결과들을 취합하여 클라이언트에게 최종 결과를 되돌려준다. 아래와 같이 자발적 컴퓨팅에서의 전체 연산 수행 과정은 크게 6단계로 수행된다.

①등록 단계: 서버로 자원제공자의 정보를 등록하는 단계
②작업 위탁 단계: 클라이언트가 서버에게 응용작업을 위탁하는 단계
③작업 분배 단계: 서버가 스케줄링 알고리즘에 의해 자원제공자들에게 WU을 할당하는 단계
④연산 실행 단계: 자원 제공자가 할당 받은 WU을 실행하는 단계

⑤작업 결과 반환 단계: 자원제공자들이 연산 결과를 서버에게 반환하는 단계
⑥최종 결과 반환 단계: 자원 제공자 서버가 클라이언트에게 최종 작업의 결과를 반환하는 단계

자발적 컴퓨팅 환경에서는 많은 자원제공자 PC를 확보하여 고성능의 컴퓨팅 파워를 창출하는 것이 목표이지만, 모든 자원제공자 PC가 동시에 작업 요청이 이루어진다면 ③ 단계에서 서버에 병목현상이 발생하게 된다. 이러한 병목현상을 줄이기 위한 기존의 자발적 컴퓨팅 시스템의 사례를 알아본다.

SETI@Home[6], Korea@Home[8] 프로젝트에서는 중앙관리서버로의 트래픽 집중을 줄이기 위해 스케줄링서버와 데이터서버를 물리적으로 분리하여 병목현상을 줄였지만, 여전히 많은 자원제공자가 작업 요청을 한다면 확장성 문제에 직면하며, 서버 결함에는 시스템 전체가 중단되는 단점이 존재한다.

Bayanihan[9]은 필리핀의 공동체 정신("*bayanihan*")을 인터넷 기반 병렬 컴퓨팅 분야에 적용시킨 프로젝트로 웹 기반 자발적 컴퓨팅 시스템이다. 중앙관리서버의 부하를 줄이기 위해 지역성을 고려한 지역 서버를 사용하지만 지역 서버에 대한 선정 방법과 결함이 발생했을 때 포용할 수 있는 방법에 대해서 다루지 않았다.

Javelin[10]은 자바로 구현된 자발적 컴퓨팅 시스템으로 브로커를 이용한 확장성과 결함 포용 기법을 지원한다. Javelin은 작업 연산에 대한 작업 트리를 구성한 다음 작업 스케줄링을 트리에 기반을 두어 수행한다. 만약 작업을 끝마친 자원 제공자의 부모 노드의 작업이 남아 있다면 해당 자원 제공자는 부모노드의 작업을 가로채어 작업을 수행하게 된다. 브로커의 결함과 부하를 분산시키기 위해 복수개의 브로커로 구성된다. 그러나 Javelin에서는 브로커의 결함 발생 시 중간 연산 결과에 대한 지속성을 보장하지 못하고 자원 제공자의 빈번한 참여와 이탈에 의해 트리 구조를 재구성해야하는 오버헤드가 크다.

3. P2P기반 중복 작업 분배 기법

기존의 중앙집중식 작업분배기법과 제안하는 P2P기반 작업 분배 기법에 대해 비교 설명한다. 표1은 시스템 설계 및 성능 분석에 사용되는 기호들이다.

표 1 용어 정의

기호	의 미
Peer	자발적인 자원제공자 PC
App	응용 연구자가 자발적 컴퓨팅을 통해 해결하고 하는 대상 (예: 신약후보물질탐색, 암호학 등)
Project	동일한 응용수행파일을 이용하여 연산할 수 있는 단위작업들의 집합
WU(Work Unit)	단일 피어에서 수행되는 작업의 최소 단위
RWU(Result Work Unit)	피어로부터 수행되어 서버로 보내지는 결과 단위작업
R(Redundancy)	단위작업마다 중복 수행되는 수

3.1 중복 수행 기법

자발적 컴퓨팅 환경에서 자원제공자는 자발적으로 자원을 제공하는 불특정 PC로 연산 수행 시간을 예측하기 어렵고, 휘발성으로 인하여 할당된 RWU를 되돌려주지 않아 전체 연산의 완료를 보장하지 못한다. 혹은 RWU는 정확하지만 결과 작업을 너무 늦게 되돌려주어 전체 연산의 성능을 지연시킨다. 또한, 악의적인 외부 공격자, 악의적인 자원제공자, 조작의 실수 등으로 인해 WU이 변경되거나 위·변조에 대한 RWU에 대한 검증이 필요하다. 따라서 자발적 컴퓨팅 시스템에서는 전체 연산의 성능향상과 자원제공자의 결함 극복, 결과 작업의 정확성 검증을 위해 동일한 WU을 중복(redundancy) 수행한다.[11]

작업 중복수는 보통 홀수(3, 5, 7..)로 이루어지며, 연산 결과에 대한 정확성 검증을 위해 다수 투표법(majority voting)을 활용한다. 다수 투표법[12]은 최소 두개 이상의 여분을 통해 수행된 결과와 비교하는 투표 방식으로 과반수이상 RWU가 같으면 채택한다. 따라서 서버에서 작업 분배시 PC의 결함으로 일정 시간동안 RWU를 받지 못하면 WU의 재분배가 이루어지며, 정확성 검증에 의해 잘못된 RWU에 대해서도 재분배된다.

3.2 작업 분배 기법

기존의 중앙집중식 작업분배는 중복되는 WU을 포함하여 모든 WU이 중앙관리서버로부터 분배되기 때문에 수많은 피어가 WU을 동시에 요청하게 되면 한정된 대역폭으로 인하여 작업분배가 지연되게 된다. 제안하는 P2P기반 중복작업분배기법에서는 중복되는 WU을 P2P를 이용하여 이웃 피어에게 WU을 할당받으므로 서버의 부하를 감소시킨다. 또한, 서버의 결함이 발생한 경우 다른 피어로부터 복제된(replicated) WU을 분배함으로 전체시스템이 중단되지 않는다.

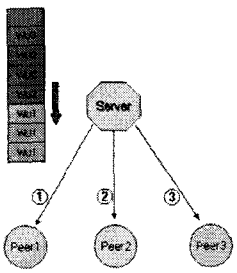

그림1. 중앙집중식 작업분배기법(기존기법)

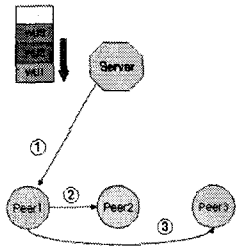

그림2. P2P기반 중복작업분배기법(제안기법)

자발적 컴퓨팅 시스템에서는 중복수행기법을 이용하기 때문에 각 WU의 중복수를 3으로 가정한다. 그림 1에서 보는 것과 같이 기존의 중앙집중식 작업분배기법은 WU1를 분배하는 서버에서 3번 분배가 이루어지는 반면, 제안하는 P2P기반 중복작업분배기법은 그림 2와 같이 서버가 WU1를 Peer1에게 분배하면 Peer1이 중복되어야 할 WU1의 수만큼 Peer2, Peer3에게 분배한다. 예를 들어 전체 WU이 10개, 각 WU의 중복수가 3이라 한다면 중앙집중식 작업분배기법에서 서버의 작업분배 횟수는 10(WU의 개수)×3(WU의 중복수)+α(피어의 휘발성에 의한 WU 재분배)이며, 제안하는

중복작업분배기법에서는 10(WU의 개수) + β(피어의 휘발성에 의한 WU 재분배)이다. 즉, 아래와 같이 정의할 수 있다.

· 중앙집중식 분배기법에서 서버의 작업 분배 횟수
=(총 WU 수 × 중복수(R))+ 휘발성에 의한 재분배(α)
· 중복작업분배기법에서 서버의 작업 분배 횟수
= 총 WU 수 + 휘발성에 의한 재분배(β)

만약 Peer의 휘발성에 의한 재분배를 배제한다면, 제안하는 중복작업분배기법에서는 '1/R'로 서버의 부하를 줄일 수 있다.

4. 시스템 설계 및 구현

자발적 컴퓨팅 시스템의 피어는 분산컴퓨팅을 위한 플랫폼이 설치된 PC이며, 유휴(idle)상태가 되면 작업을 요청하거나 받아놓은 WU을 수행한다. 모든 는 P2P통신으로 위한 이웃 피어의 정보 포함하는 피어리스트를 가지며, 피어리스트를 기반으로 WU요청 및 WU광고 메세지를 멀티캐스팅 한다. 피어리스트는 최초 참여시 서버로부터 받으며, 다른 피어의 참여(login) 및 이탈(logout) 메시지를 받으면 갱신된다.

4.1. 작업 요청 및 수행

기존 기법에서는 서버에게만 작업을 요청하는 반면, 제안하는 기법에서는 이웃 피어에게 WU을 요청하거나 혹은 서버로 WU 요청하여 WU을 다운로드 받고 수행한다.

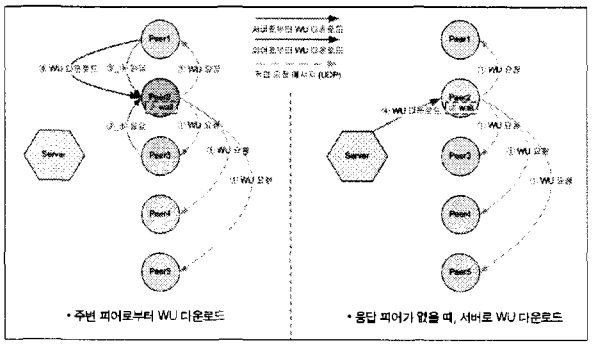

그림 3. P2P기반 중복작업분배기법의 작업 요청 과정

피어가 연산에 참여하기 위해 WU을 요청하는 과정을 그림 10에서 설명하고 있다. Peer2는 수행할 WU을 할당받기 위해 서버로부터 받은 피어리스트의 피어들에게 ① 'WU 요청 메시지(UDP)'를 멀티캐스팅 한다. ② 요청 메시지를 뿌린 후 일정시간 동안 응답 메시지를 기다린다. ③_ⓐ Peer1의 응답메시지와 ③_ⓑ Peer3의 응답 메시지를 받았지만, 가장 먼저 응답 메시지가 도착한 ④ Peer1과 TCP 연결을 맺고 WU를 다운로드 한다. 만약 일정시간동안 응답 피어가 없을 경우, ④ 서버로부터 수행할 WU을 다운로드 받는다.

하지만 이러한 작업 요청 과정에서 WU의 수행완료 후 다음 WU을 요청하고 다운로드받을 때까지 대기시간이 발생한다. 따라서 본 논문에서는 각 피어마다 작업큐(Work Queue)를 두고 WU을 미리 받아 놓는 방안을 제안하였다. 예를 들어 작업큐가 10이라면 WU 수행 후 남은 WU 개수가 3개일 때 WU 요청을 통해서 작업큐의 빈공간인 7개의 WU를 다운로드받아 다시 채운다. 그러나 자발적 컴퓨팅 시스템의 특성상 피어의 휘발성이 크므로 작업큐에 미리 받아놓은 WU이 언제 수행될지 예측할 수 없다. 따라서 서버는 피어의 결함포용을 위해 분배된 WU이 일정시간동안 RWU이 돌아오지 않으면, 동일한 WU을 다른 피어에게 재분배한다. 작업큐를 이용해 미리 받아놓은 WU이 오랫동안 지연되면, 그 WU은 이미 다른 피어에게 분배되어 불필요한 수행이 될 수도 있다. 따라서 피어는 WU을 수행하기 전에 서버에게 수행할 WU의 수행시작메시지(WUStartMsg)를 요청하여 혹시 다른 피어로부터 이미 수행된 WU이라면, 해당 WU을 처리하지 않도록 설계하였다.

표 2 WU 요청 및 수행 알고리즘

```
If NumberOfRemained WU < MinimumSize then
    For NumberOfPeerlist do
      Send WURequestMsgToPeer(ProjectID, UDP);
    Endfor
Endif

While WaitTime(s) do
     Store ResponseQueue(RespondedPeer);
Endwhile

If NumberOfResponseQueue > 0 then
      /*응답메시지가 있으면 피어로부터 WU 요청*/
    For WorkQueue != Full do
        Download WU From RespondedPeer(WU);
        ResponseQueue(Next RespondedPeer);
     End for
else
   Send WURequestMsg To Server(ProjectID, TCP)
Endif

/*미리 받아 놓은 작업으로 인한 불필요한 수행을 막기 위해*/
Send WUStartMsg To Server(WUid, TCP);
While WaitTime(s) do
    If ResponseMsg == fail do
      DeleteWorkQueue(WUid);
       WorkQueue(NextWU);
    else
       executeWU(WUid);
    endif
endWhile

executeWU(WUid)
```

표2는 작업큐를 이용한 WU 요청 및 수행 알고리즘이다. 작업큐의 남은 WU의 수가 미리 설정해 놓은 최소치(minimum size)가 되면 피어는 피어리스트를 통해서 이웃 피어들에게 자신이 수행하고자 하는 ProjectID를 UDP를 통해서 WU요청메시지(WURequestMsg)를 멀티캐스팅한다. WU요청메시지를 멀티캐스팅한 후 일정시간동안 기다린다. 한번 뿌린 요청메시지로 작업 큐의 부족한 개수만큼 WU을 할당 받을 수 있도록 응답메시지가 오면 응답큐에 응답피어를 순차적으로 저장한다. 만약 응답큐의 개수가 0보다 크다면 즉, 응답피어가 있다면 작업큐의 빈공간만큼 WU을 다운로드 받는다. 응답큐에 가장 먼저 저장된 응답피어에게 TCP 연결을 맺고 해당 WU을 다운로드 받는다. 가장 먼저 응답메시지를 보낸 응답피어는 WU를 전송할 때도 빠른 전송속도를 보장할 것으로 가정하고 응답큐에 저장된 순서대로 작업을 요청한다. 만약 응답큐의 개수가 0보다 크지 않다면 응답피어가 없으므로 서버에게 WU을 요청한다. 피어는 작업을 수행하기 전에 서버에게 해당 WUid의 수행시작여부메시지(WUStartMsg)를 보낸다. 해당 WU에 대한 응답메시지가 "Fail"이 수신되면 다른 피어에 의해 이미 완료된 WU을 말한다. 따라서 해당 WU을 처리하지 않고 작업큐에서 지운다. 만약 "Fail"아니라면 즉, "OK" 이거나 혹은 응답메시지를 받지 못할 경우에는 일정시간

서버의 응답이 없더라도 작업을 수행하도록 하여 서버의 결함 시에도 서비스가 일정시간 동안 유지되어진다.

4.2. 작업 분배

제안하는 중복작업분배기법에서는 서버뿐만 아니라 피어간에도 중복수를 가진 WU을 복제하여 분배한다. 그림 4는 Peer1이 서버로부터 WU1를 다운로드하여 Peer2와 Peer3에게 WU1의 복제하여 분배하는 그림이다. 각 피어에는 WU를 관리하기 위한 로컬DB를 가지며, 각 WU의 ProjectID, WUid, R(중복수) 등의 정보를 포함하고 있다. 다른 피어에게 WU를 분배할때마다 해다 WUid의 R값에서 1을 뺀 값으로 갱신하여 중복수만큼 복제하여 전송한다.

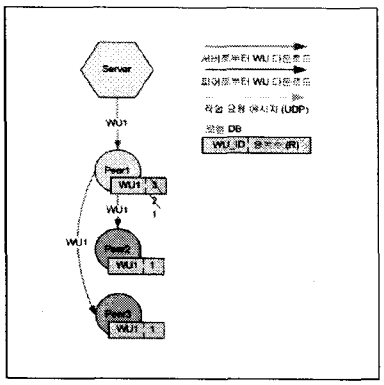

그림 4 P2P기반 중복 작업 분배 기법의 작업분배 과정

본 논문에서는 요청피어에게 WU를 복제하여 전송하는 피어를 Super-Peer라고 한다. 하지만 자발적 컴퓨팅 환경에서는 휘발성이 큰 피어들로 구성되므로 WU을 분배해주는 Super_Peer의 가용성 문제가 고려되어야 한다. Super_Peer가 중복된 WU을 다른 피어에게 복제하여 주지도 못하고 이탈해버리거나 혹은 네트워크가 단절이 되어버린다면 서버의 부하는 중앙집중식 작업분배와 동일하게 된다. 따라서 Super_Peer가 가능하면 빨리 작업을 복제하여 줄 수 있는 방안이 필요하다.

본 논문에서는 Super_Peer가 작업요청메시지만을 기다리지 않고 서버로부터 받은 WU에 대해서 광고메시지(AdvertisementMsg)를 UDP를 통해 멀티캐스팅한다. 응답메시지가 오는 피어에게 중복된 WU의 수만큼 복제하여 전송하도록 설계하였다. 이와 같은 작업분배는 광고메시지를 통한 분배와 작업요청메시지에 의한 작업분배로 구성되어 진다. 자세한 과정은 표 4를 통해 설명하였다.

서버로부터 중복수가 2이상인 WU을 다운로드 받으면, 피어리스트의 피어들에게 UDP를 통해 광고메시지(AdvertisementMsg)를 멀티캐스팅한다. 그리고 일정시간동안 응답메시지를 기다리며, 응답메시지를 보낸 피어를 응답큐에 순차적으로 저장한다. 일정시간 이후에 저장된 응답큐에서 첫 번째 저장된 응답피어에게 WU을 복제하여 전송한다. 그리고 해당 WUid의 R에서 1를 뺀 값을 저장한다. WU의 중복수가 1이 될 때까지 응답큐에 저장된 피어들에게 WU을 복제하여 전송한다.

다른 피어로부터 작업요청메시지(WURequestMsg)가 수신되면 메시지에 포함된 ProjectID와 로컬의 ProjectID가 같고, 로컬에 저장된 WU 중에 중복수가 있는 1보다 큰 WU이 있다면, 요청피어에게 응답메시지(ResponseMsg)를 보낸다. 응답메시지를 받은 요청피어는 응답피어에게 TCP 연결을 요청하고 해당 WU을 다운로드 받는다. 그리고 응답피어는 복제해 준 WU의 중복수를 1을 뺀 값을 로컬에 저장한다.

표 4 WU 분배 알고리즘

```
If DownloadWUFromServer then
   For Peerlist do
      SendAdvertisementMsgToPeer(ProjectID, UDP);
   EndFor

   While WaitTime(s) do
        StoreResponse-Queue(Response-peer);
   EndWhile

   If NumberOfResponse-Queue > 0 then
        For NumberOfRedundantWU do
            ConnectResponse-peer(TCP);
            TransmitReplicateWU(WUid)
            UpdateLocalDB(NumberOfRedundantWU-- )
        EndFor
   EndIf

EndIf


If Receive(WURequestMsgFromPeer) then
   If ProjectIDFromRequest-peer == LocalProjectID then
        If NumberOfRedundantWU > 1 then
            SendResponseMsgToRequestedpeer(WUid)
            TransmitRedundantWU(WUid)
            UpdateLocalDB(NumberOfRedundantWU-- )
        Endif
   Endif
Endif
```

5. 성능평가

본 논문에서 실험을 위해 Korea@Home에서 수행 중인 신약후보물질탐색 응용의 CYP51 프로젝트 일부를 수행하였다. 신약후보물질탐색은 가상 탐색(Virtual Screen) 기술을 이용하여 약이 체내에서 특정 작용점(수용체, 효소 등)에 결합하여 약효가 발휘되는 것을 PC를 이용하여 시뮬레이션 한다. 다양한 화합물들의 활성과 구조를 비교함으로써 타겟 단백질에 대한 신약후보 물질들을 도출해 낼 수 있으며, CYP51은 항진균제를 개발하기 위한 가상탐색 응용이다. 성능 평가를 위한 실험환경으로 전체 작업의 WU 개수는 100개, WU의 중복수는 3으로 설정하였다.

자발적 컴퓨팅 환경에서는 유휴상태에 WU를 수행하기 때문에 사용자의 PC 사용여부에 따라 연산 시간에 많은 영향을 받는다. 특히 본 실험에서는 PC가 15대밖에 되지 않기 때문에 PC 사용여부에 의해 결과 값이 좌우될 수 있다. 따라서 본 실험에서는 PC의 사용여부에 상관없이 WU을 처리하도록 설정하였다. 또한 기존의 중앙집중식 작업분배 기법에서는 WU을 1개씩 서버로부터 받아오는 방식이며, 제안하는 중복작업분배에서는 작업큐를 5개로 설정하고 남은 WU이 3개 이하가 되면 작업을 요청하도록 설정하였다.

실험을 통해서 제안하는 중복작업분배기법과 중앙집중식 작업분배기법의 서버에서의 outbound 트래픽을 측정하였으며, 서버 결함시 전체 작업의 수행 완료되는 시간을 비교하였다.

5.1. 서버에서의 트래픽 측정 및 분석

기존의 작업분배기법과 제안하는 중복작업분배기법에서 피어의 증가에 따라 서버의 outbound 트래픽이 얼마나 증가되는지에 대해 측정한다. 피어의 수가 증가하면 전체 연산

수행 완료 시간은 감소하지만, 서버에서의 작업 분배 트래픽은 증가한다. 피어의 수가 많아지면 동시에 서버로 작업을 요청하는 피어가 집중되기 때문에 서버의 네트워크 대역폭은 일정한데 분배할 작업이 많아지면서 작업 분배의 지연 원인이 된다. 즉, 서버의 병목현상이 일어나게 된다. 이러한 서버의 집중을 줄이고자 제안된 중복작업분배기법에서 발생하는 서버의 outbound 트래픽을 측정한다. inbound 트래픽은 결과작업에 대한 업로드이므로 기존의 기법과 제안하는 기법이 동일하게 나타나므로 비교하지 않았다.

서버의 트래픽을 측정할 수 있도록 RRDTool을 기반으로 하는 Cacti[13]를 설치하여 피어의 수에 따라 작업에 1시간 동안 참여하였을 때 서버로부터 outbound되는 데이터량을 비교하였다.

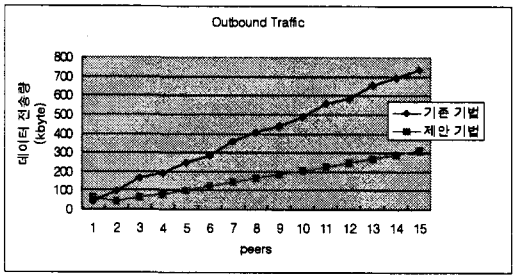

그림5. 1시간 동안 서버에서의 outbound 트래픽

그림5의 그래프에서 볼 수 있듯이 제안하는 기법의 트래픽이 기존의 기법보다 적은 량의 데이터를 내보냈음을 알 수 있다. 피어의 수가 1대일 경우에는 기존기법이 제안기법보다 더 적은 량의 데이터를 전송하는데, 기존기법은 피어가 WU을 1개씩 요청하기 때문이다. 제안한 기법의 경우는 한번 가져오는 WU의 개수를 5개씩 다운로드 받기 때문에 피어 1대만 참여할 경우엔 비효율적일 수 있다. 하지만 피어의 수가 2대이상일 경우에는 제안기법의 전송된 데이터양이 훨씬 적음을 알 수 있다. 제안 기법에서 피어의 수가 1대에서 2대로 증가하였을 때 전송량이 감소한 이유는 피어가 1대일 때는 P2P통신을 통해 이웃 피어에게 작업을 전송받지 못하고 서버로부터만 작업을 요청하기 때문에 발생한다.

비록 피어간의 메시지양은 증가하였지만 서버의 부하는 감소하였다. 본 실험에서는 피어의 휘발성을 고려하지 않아 그림 5와 같은 결과를 보여주었지만 만약 실제적인 자발적 컴퓨팅 환경이라면, 피어에 대한 결함도 고려하여할 것이다.

5.2. 연산 수행 시간 측정 및 분석

본 실험에서는 기존의 작업분배기법과 제안하는 중복작업분배기법에서 전체 WU의 분배시점부터 서버로 되돌아오는데 걸리는 시간을 비교하였다. 그림 6은 참여 피어를 증가시켜가면서 전체 연산 수행 시간을 비교한 것이다. 제안 기법은 기존 기법에 비해 전체 연산 수행 시간이 평균 약 6%의 감소를 하였다. 이는 본 실험이 LAN환경에서 이루어져 서버와 피어간의 전송시간과 피어간의 전송시간에 차이를 보이지 않았으며, 참여피어의 수가 적어 서버의 병목현상이 발생하지 않았기 때문에 큰 기존 기법과 제안기법의 연산 수행 시간이 큰 차이를 보이지 않았다.

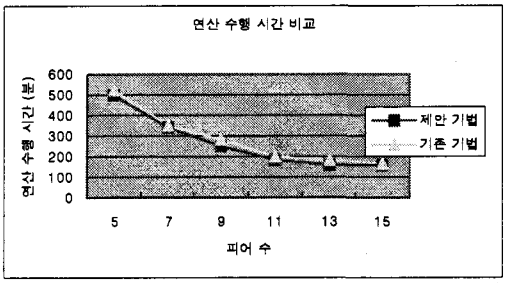

그림 6. 연산 수행 시간 비교

일반적으로 LAN 환경에서의 피어간의 연결은 WAN 환경에서의 피어간의 연결보다 더 빠르다고 가정[18]한다. 이 가정은 WAN 환경을 기반으로 하는 Volunteer Computing에서도 적용된다. 제안 기법에서는 WU 요청 및 광고 메시지를 멀티캐스팅하고 응답시간이 가장 빠른 피어에게 WU을 전송받거나 복제해 주는 기법이다. 하지만 본 실험에서는 모든 피어가 LAN 환경 내에 있었으므로 피어간의 전송시간에 차이를 보이지 않았다. 따라서 실제 Volunteer Computing에서는 제안 기법이 기존 기법보다 전체 연산 수행시간이 줄어들 것이다.

5.3. 서버 결함 시 성능 평가

본 논문에서 제안한 P2P기반 중복작업분배기법은 서비스 중에 서버의 결함이 발생하였을 때 얼마나 서비스를 유지하여 지속적인 연산을 수행할 수 있는지를 측정하였다. 서버의 결함 시에도 연산을 지속하여 전체 수행시간을 줄이는 것이 목적이다. 따라서 기존 기법과 제안하는 기법에서 서버의 결함이 발생했을 때에 전체 수행시간을 비교 분석하였다. 본 실험에서는 서버의 네트워크를 2시간 동안 중단시켰다가 다시 활성화시켜 서버의 결함으로 가정하였다.

그림 7은 기존 기법에서 정상 실행되었을 때와 서버의 결함이 2시간 지속되었을 때의 전체 수행시간을 비교한 것이다. 서버와의 네트워크가 끊기면서 모든 피어는 로컬에서 수행하던 작업만 마치고 서버의 연결만을 기다리기 때문에 피어의 수와 상관없이 지연된 수행시간은 거의 비슷함을 알 수 있다.

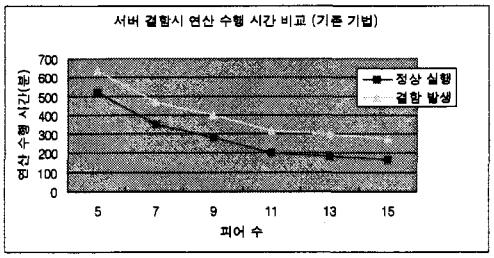

그림 7. 기존기법에서의 수행시간 비교

그림 8은 제안하는 기법에서는 정상실행 되었을 때와 서버 결함이 2시간 지속되었을 때의 전체 작업 수행시간을 비교한 것이다. 각각의 피어는 작업큐에 여분의 WU을 미리 저장하고 있기 때문에 서버와의 네트워크가 단절되더라도 연산을 지속할 수 있으며, 주변 피어에게 WU을 요청하기 때문에 서버와의 통신이 없어도 피어로부터 중복된 WU을 전송받아 일정시간동안 WU의 연산을 지속할 수 있다.

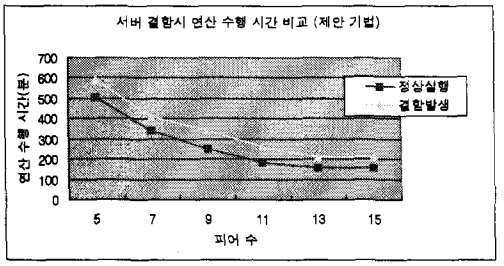

그림 8. 제안기법에서의 수행시간 비교

기존 기법에서는 피어의 수와 상관없이 지연되는 시간이 비슷했지만 제안하는 기법에서는 피어의 수가 증가할수록 지연시간이 줄어드는 것을 알 수 있다. 이는 피어의 수가 많아지면 중복될 WU을 가지고 있는 피어의 수가 많아지고 서로 다른 피어에게 WU을 복제하여 전송함으로 작업수행이 지속적으로 이루진다. 즉, 결함에 의한 지연 시간이 감소한다. 피어의 수뿐만 아니라 작업큐의 크기, WU의 중복수가 증가할수록 지연시간은 더욱 감소한다.

표5는 서버결함시 기존 기법과 제안한 기법의 지연시간을 비교한 표이다. WU의 중복수와 피어의 작업큐의 크기가 커질수록 서버 결함시 연산 수행의 지연 시간이 감소됨을 볼 수 있다.

중복수	결함 시간	중앙집중식 (기존기법)	P2P기반 중복 작업 분배 (제안기법)	
			작업 큐 5	작업 큐 7
3	120분 (100%)	평균 115분 (95.8%) 지연	평균 72분 (60%) 지연	평균 31분 (25.8%) 지연
5	120분 (100%)	평균 115분 (95.8%) 지연	평균 56분 (46.7%) 지연	평균 24분 (20%) 지연

표 5. 서버 결함시 연산 수행 시간 비교 결과

6. 결론

본 논문은 자발적 컴퓨팅 환경에서 피어의 결함 허용을 위해 사용되는 중복 작업을 이용하여 P2P기반의 작업 분배 기법을 제안하였다. 기존의 작업 분배 기법은 중앙집중식으로 서버에서만 작업 분배가 이루어지기 때문에 서버의 병목현상 발생 및 서버 결함 시 전체 시스템이 중단되는 현상이 발생하였다.

제안하는 기법은 각각의 피어에서 WU 광고 및 WU 분배 그리고 WU 요청을 통해서 중복된 WU에 대해서 복제하고 전송하여 준다. 서버에서 작업 분배로 인한 병목현상을 줄일 수 있으므로, 결국 많은 피어가 동시에 작업을 요청하더라도 피어간의 작업을 분배하기 때문에 확장성을 증가시킬 수 있었다. 또한 작업큐를 통한 미리받기 기능과 중복된 WU에 대하여 서버의 중개없이도 P2P기반의 작업분배가 이루어지기 때문에 서버의 결함 시에도 일정시간 동안 서비스를 지속할 수 있는 강건함을 보였다. 비록 피어간의 메시지양은 증가하지만 작업요청 및 작업광고 메시지는 UDP를 이용하여 전송되기 때문에 전체적인 트래픽에는 큰 영향은 주지 않았다.

본 논문의 실험에서는 자발적 컴퓨팅 시스템의 특징인 피어의 동적인 특성이 배제된 소수의 피어로 실험의 한계가 있었다. 연산에 참여하는 피어의 동적인 특성을 고려하여 보완할 수 있는 연구가 필요하다. 또한, P2P기반의 통신을 이용하여 종속관계에 있는 작업들을 분배하고 피어간의 상호 작용을 통해서 최종결과만을 서버에게 전달함으로써 outbound 트래픽뿐만 아니라 inbound 트래픽도 줄일 수 있도록 P2P를 이용한 전송 기법에 대한 연구가 요구된다.

7. 참고문헌

[1] I.Foster and A.Iamnichi, "On Death, Taxes, and The Convergence of Peer-to-Peer and Grid Computing", 2nd Int.Workshop on Peer-to-Peer System, LNCS 2735, Feb.2003

[2] F. Berman, G. C. Fox, and A. J. G. Hey, "Grid Computing : Making the Global Infrastructure a Reality", Wiley, 2003

[3] BOINC : http://boinc.berkeley.edu/volunteer.php

[4] Luis F. G. Sarmenta, "Volunteer Computing", Massachusetts Institute of Technology,June 2001

[5] DP Anderson, G Fedak, "The Computational and Storage Potential of Volunteer Computing", ieeecomputersociety, CCGRID 06. Sixth IEEE 2006

[6] SETI@Home : http://setiathome.berkeley.edu/donate.php

[7] Folding@Home : http://folding.stanford.edu/

[8] Korea@Home : http://www.koreaathome.org

[9] L. F. G. Sarmenta, S. Hirano. "Bayanihan: Building and Studying Volunteer Computing Systems Using Java", Future Generation Computer Systems Special Issue on Metacomputing, Vol. 15, 1999.

[10] M. O. Neary, A. Phipps, S. Richman. "Javelin 2.0: Java-Based Parallel Computing on the Internet". Proceedings of Euro-Par2000. Munich, GERMANY, Aug. 28 - Sep. 1, 2000.

[11] D.P. Anderson, Eric Korpela, Rom Walton, "High-Performance Task Distribution for Volunteer Computing", First International Conference on e-Science and Grid Computing, 2005

[12] L. LAMPORT, R. SHOSTAK, M.PEASE. "The Byzantine Generals Problem", ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982.

[13] CACTI : http://forums.cacti.net/

[14] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage", 5th IEEE/ACM Int. Workshop on Grid Computing, Nov. 2004.

[15] D.S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing", Technical Report HPL-2002-57, HP Labs. 2002

[16] David P. Anderson, Carl Christensen, Bruce Allen, "Designing a runtime system for volunteer computing", ACM/IEEE SC2006 Conference on High Performance Networking and Computing, November. 2006

[17] David P. Anderson, Eric Korpela, Rom Walton, "High-Performance Task Distribution for Volunteer Computing", First International Conference on e-Science and Grid Computing, 2005

[18] Dwight Deugo, "Mobile Agent Messaging Models", In Proc. 5th International Symposium on Autonomous Decentralized Systems, 2001.