P2P 컴퓨팅에서 중복 수행 결과의 정확성 검증 기법

박찬열†

요 약

인터넷을 기반으로 독립적인 장치들이 참여하는 P2P 컴퓨팅은 원하는 목적을 달성하는데 있어서 참여 장치들의 이탈, 고장, 네트워크 상태, 익명성 등으로 인해 잦은 접속단절과 보안 공격을 겪는다. 여러 연구와 구현에서 이러한 문제들을 해결하기 위해 공유되는 자원의 중복 기법을 사용한다. 이 논문에서는 컴퓨팅 자원의 공유를 목적으로 하는 P2P 컴퓨팅에서 수행되는 작업의 중복 수행을 통해 접속단절과 보안 공격에도 올바른 결과를 얻어내는 정확성 검증 기법을 제안한다. 제안하는 기법에서는 종속성이 존재하는 단위작업들에 대해 시스템 전체의 전역적인 메시지 교환 없이 주기적으로 정확성을 검증하고, 검증된 결과는 검사점이 되어 복귀 회복이 가능한 결함 포용이 가능하다.

키워드 : P2P 컴퓨팅, 분산 컴퓨팅, 결함 포용, 중복

A Verification of Replicated Operation in P2P Computing

Chan Yeol Park†

ABSTRACT

Internet-based P2P computing with independent machines suffers from frequent disconnections and security threats caused by leaving, failure, network diversity, or anonymity of participated machines. Replication schemes of shared resources are used for solving these issues in many studies and implementations. We propose an operational replication scheme in P2P computing to share computing resources, and the scheme verifies the correctness of operation against faults and security threats. This verifications are carried out periodically on replicated and dependent working units without global message exchanges over the whole system. The verified working units are treated as checkpoints, and thus they could be put to practical use for fault-tolerance with rollback recovery.

Keywords : P2P Computing, Distributed Computing, Fault-tolerance, Replication

1. 서 론

분산컴퓨팅은 지역적으로 흩어져 있는 자원들을 활용하여 고성능의 컴퓨팅 파워를 얻어내고자 하는 시도로서 알고리즘, 아키텍처, 운영체제 등의 여러 영역에서 오랫동안 연구되어 왔다. 최근에는 인터넷과 PC 자원의 성능 향상으로 인해 인터넷을 기반으로 하는 P2P 컴퓨팅이 새로운 패러다임으로 자리 잡고 있다. 파일 공유 등의 목적을 가진 P2P 프로그램들은 이미 널리 사용되고 있으며, 확장성(scalability), 성능(performance)과 신뢰성(reliability)의 향상 등의 P2P 특성들을 보다 확장하고자 하는 관련 연구들이 지속적으로 이루어지고 있다[1,2].

† 정 회 원: 한국과학기술정보연구원 선임연구원
논문접수: 2004년 4월 19일, 심사완료: 2004년 5월 6일

국내에서도 P2P형 인터넷기반 분산컴퓨팅 기반 환경을 구축하고자 하는 목적으로 Korea@Home 프로젝트[3]가 진행 중에 있다. SETI@Home[4]이나 distributed.net[5]과 같이 수년 전부터 다수의 자발적인 참여로 활성화된 분산컴퓨팅 프로젝트들이 특정 응용 수행을 위해 진행된 반면, Korea@Home은 고성능/대용량 컴퓨팅 파워를 필요로 하는 여러 응용에 적용이 가능하도록 인터넷에 연결된 PC 자원의 유휴 컴퓨팅 자원 제공을 이용하는 기반 플랫폼을 제공하는 것을 목적으로 하고 있다.

하지만 자발적인 자원제공과 유휴자원의 활용, 인터넷을 기반으로 하는 환경으로 인하여 잦은 접속단절(disconnection)이나 보안 위협 등의 해결해야할 이슈들이 존재한다. 자원을 제공하는 PC들은 소유자의 원래 사용 목적에 방해 받지 않으면서 유휴컴퓨팅 자원을 제공하기 때문에 사용자의 의도에 따라 언제든지 자원제공을 멈출 수 있으며, xDSL 등 다양한 인터넷 연결 환경으로 인해 고장(failure)의 발생이 아니더라도 일시적인 접속단절이나 인터넷 주소의 변경 등 잦은 접속단절이 발생한다. 그리고 자원을 제공하는 PC들은 인터넷에 연결된 임의의 사용자이므로 일관된 정책이나 제어를 갖지 못하기 때문에 외부에 노출되어 보안 관련 공격에 취약하다. 또한 정당한 절차에 따라 참여한 자원제공자들도 악의적인 의도를 가지고 컴퓨팅의 정확한 실행을 저해하는 경우가 발생할 수 있다.

이 논문에서는 중복(replication)을 기반으로 시스템 전역적인 부하 없이 자원제공PC의 접속단절, 고장, 참여와 이탈 등으로 인한 지속적인 연산 방해를 해결하고, 내외부 공격에도 불구하고 정확한 연산이 수행되는 것을 보장하는 기법을 제안한다.

2. 관련 연구

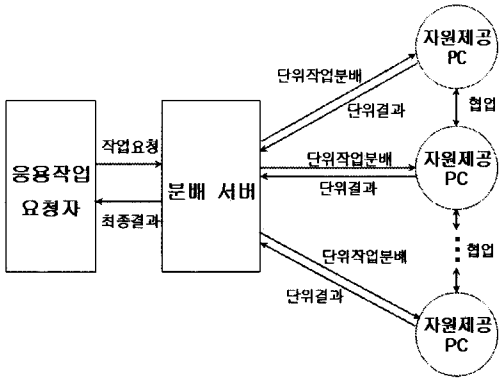
중복 기법은 전통적으로 사용되어 오던 방식으로서 중복 실행을 통해 고장이나 접속단절에 대처하여 응용 수행의 정확성을 높이거나 데이터 중복을 통해 가용성을 증가시킨다[6,7]. P2P 컴퓨팅에서는 명명(naming)이나 노드의 검색(searching & discovery)과 위치지정(locating & addressing), 경로 설정(routing)들이 기본적으로 갖추어야할 기능이므로 많은 연구들[8,9,10,11,12]이 이 기본 기능들에 대해 진행되어 왔으며, 이들 기본 기능의 목적 달성을 위한 결함 포용 기법들이 함께 연구되고 있다. 하지만 [9]는 노드의 고장으로 인한 대처 방안이 존재하지 않으며, Chord[11] 와 CAN[12]은 중복과 결함 포용을 다룰 수 있는 반면 노드의 악의적인 행동에 대한 대처 방법이 없다.

Napster[13]나 SETI@Home과 같은 P2P 시스템은 중앙 서버가 중심적인 역할을 수행하기 때문에 이 서버의 고장에 취약하다. Entropia의 DCGrid[14]의 경우 역시 자원제공PC의 참여와 이탈 등을 중앙 관리함으로써 제어와 관리가 용이하지만, 이로 인해 P2P 컴퓨팅의 자율성이 부족하다. 협업을 목적으로 하는 Groove[15]는 모든 네트워크 연결은 간헐적임을 가정하고 메시지의 목적지가 접속불가하게 되면 relay server라는 특정 노드를 활용하여 메시지들을 저장해두었다가 나중에 전달한다. 이러한 기법들은 특정 노드들에 종속적이기 때문에 P2P의 자율성과 결함 포용 특성을 충분히 활용하지 못한다. 또한 분산 메시징과 데이터 관리를 목적으로 하는 Escher Group의 WebRiposte[16]은 모든 메시지를 중복시킴으로써 노드의 이탈이나 고장이 발생하면 중복된 메시지를 재구성한다. 그리고 Legion[17]은 재실행을 바탕으로 고장에 대처하고 있다.

제안하는 기법은 중복실행이나 재실행, 암호화된 해싱기법 등 이들이 각각 취하고 있는 방식의 일부들을 포함하고 있으며 그 장점들을 취하고 있다. 하지만 명명법에 관련된 기본 기능만을 포함하고 있는 제안 기법은 검색, 위치지정, 경로설정 등에 대해서는 기존 기법이 존재한다고 가정하고 있으므로 완전한 P2P 컴퓨팅을 위해서는 우선 기본 기능들과 연계되어 보다 향상된 기법으로 연구될 필요가 있다.

3. 시스템 모델 및 가정

자발적인 유휴컴퓨팅 PC자원을 제공받아 이를 거대 응용의 수행에 활용하는 Korea@Home 플



<그림 1> Korea@Home 플랫폼 기본 구조

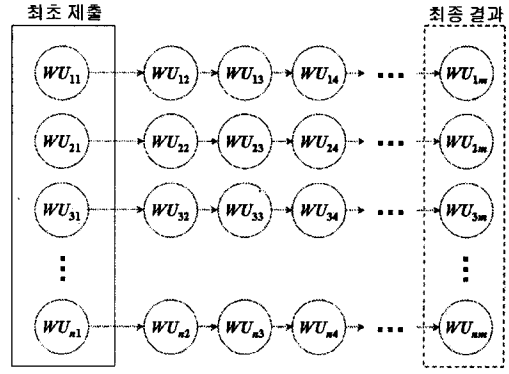
랫폼은 <그림 1>과 같은 기본 구조를 갖는다. 전체 시스템은 Korea@Home을 활용하고자 하는 응용작업요청자(Application Provider; *AP*)와 유휴컴퓨팅 자원을 제공하는 자원제공PC(Resource Provider; *RP*), 그리고 응용작업의 분배, 스케줄링, 관리 및 제어 등을 수행하는 Korea@Home 서버들로 구성된다. 고성능 컴퓨팅 자원을 필요로 하는 응용작업(Application; *App*)을 *AP*가 서버에 제출하고 최종적으로 필요로 하는 결과를 서버로부터 되돌려 받는 것이 기본적인 실행 흐름이다.

파일 공유 등을 위한 P2P 시스템에서는 *RP*들이 서버의 존재가 필요 없이 보다 자율적으로 기능을 수행하는 경우가 대부분이지만, 이는 시스템을 '활용하는 수혜자'와 '*RP*의 소유자'가 같은 경우에 가능하다. Korea@Home의 경우, 시스템 활용의 수혜자는 *AP*인 반면, 자발적으로 자원을 제공하는 *RP*의 소유자들과 달라 이 둘을 중재하고 시스템의 관리와 제어를 위한 서버의 존재는 필수적이다. 하지만 서버의 부하를 줄이기 위해 보안 관련 기능 등 최소한의 역할만을 담당하도록 하며 자율적으로 *RP*들 간의 상호작용을 통해 실행 흐름이 이루어지도록 한다.

Korea@Home에서 수행되는 모든 응용 *App*는 *WU*들의 집합으로 구성되어 있으며, 각 *WU*는 하나의 *RP*에서 수행하는데 필요한 최소한의 실행파일과 데이터로 구성된 파일들의 집합이다. 수행되는 응용 *App*는 인터넷 연결 형태 및 속도의 다양성, *RP*의 일정치 않은 성능 등으로 인하여 가능하면 작은 크기의 조각들로 나누어져 서로 종속성이 없는 형태가 바람직하다. 즉, 각 *RP*에 이 조각들이 분배되고 가용한 유휴시간 중에 빠르게 수행되어 결과를 내놓을 수 있도록 한다. 서로 독립적이고 크기가 작은 조각들인 단위작업(Working Unit; *WU*)들이 각 *RP*들에게 분배되고, *RP*들 간에 상호작용이 없다면 비교적 시스템의 운영과 관리가 단순해질 수 있다. 현재 Korea@Home에서 수행되는 응용 중 하나인 '신약후보물질탐색'이나 SETI@Home의 경우가 각 *RP*에서 수행되는 작업들이 서로 간에 전혀 종속성이 없는 형태로 이루어진 경우이다. *WU*간에 종속성이 강한 경우에는 *RP*들 간에 상호작용이 많아 연결 형태와 속도, 연결 상태가 일정하지 않고 불안정한 인터넷 환경에서 작업 중단의 비율이 상대적으로 높아져서 안정적인 응용 수행에 적합하지 않다. 작업의 중단은 *RP* 소유자의 자발적인 중단, 네트워크의 일시적인 단절, *RP*의 고장 등으로 인해 발생하는 모든 경우를 포함한다.

이 논문에서는 *WU*가 하나의 파일로 존재하는 객체로서 단순화된 형태로 가정하며, 실행 중 생성되는 파일 형태의 *WU*가 실제 생성되어야하는 내용과 달라지는 경우 옳지 않은 실행이라고 가정한다.

그리고 제안하는 기법은 현재 Korea@Home 플랫폼에서 수행되는 '질병단백질 변이해석(protein folding analysis)'에서 보이는 *WU* 간 종속성을 모델로 하여 설명한다. '질병단백질 변이해석'은



<그림 2> 응용 수행의 단위작업 간 종속성 예

응용은 $App=\{WU_{ij} \mid 1\le i\le n,\ 1\le j\le m\}$이고 <그림 2>와 같은 형태의 종속성을 갖는다. AP는 초기 단위작업들의 집합으로 $\{WU_{i1} \mid 1\le i\le n\}$를 제출하고, 최종적으로 필요로 하는 결과는 $\{WU_{im} \mid 1\le i\le n\}$이다. $\{WU_{ij} \mid 1<i<n,\ 1<j<m\}$는 AP에게는 필요 없지만 각 $WU_{i(j+1)}$은 WU_{ij}로부터 생성된다. 즉, 임의의 j와 k에 대해 i가 같다면 WU_{ij}와 WU_{ik} 간에는 반드시 순차적인 종속성이 존재하고, i가 다르다면 WU_{ij}와 WU_{ik}는 서로 독립적이다. 실제 Korea@Home에서 수행되는 '질병단백질 변이해석'은 하나의 단백질에 대해서 $n=2^{15}$, $m=100$이 사용된다.

이러한 종속적 관계를 가진 실행에서 WU_{ij}로부터 $WU_{i(j+1)}$의 정상적인 생성은 결정적(deterministic)이며, 옳지 않은 $WU_{i(j+1)}$이 생성되는 확률은 매우 적고 비결정적이어서 올바른 WU_{ij}로부터 옳지 않은 $WU_{i(j+1)}$이 항상 동일하게 생성되지는 않는다고 가정한다. 이에 따라 중복 수행에서 옳지 않으면서도 모두 같은 $WU_{i(j+1)}$을 생성하는 경우는 없다고 가정한다.

각 단위작업들은 r개의 RP들에서 중복 수행되는데, 가용한 전체 RP들의 개수 N에 대해 $N>>r$이라고 가정한다.

4. 중복 수행의 정확성 검증 기법

4.1. 초기 단위작업

AP가 서버에 제출하는 초기의 단위작업들인 $\{WU_{i1} \mid 1\le i\le n\}$는 위·변조 가능성을 제거하기 위해 서버의 개인키(private key)로 서명이 이루어진 형태로 준비된다. 일반적으로 사용되는 PKI 방식의 서명 절차에 따라 각 WU_{i1}은 해싱되고, 이 해시값 X는 다시 서버의 개인키에 의해 서명이 이루어져 비교적 작은 크기의 Enc(X)를 얻는다. 이 Enc(X)는 RP의 요청에 의해서 서버에서 RP에게 직접 전달하며 다른 RP를 경유하여 전송되지 않는다.

Korea@Home에 참여하는 RP들은 자신이 유휴상태가 되어 자원을 제공할 수 있는 상태가 되면 다른 RP들에게 수행할 WU를 요청한다. 적절한 WU를 받지 못한 경우, RP는 서버에게 WU_{i1}을 요청하여 작업을 수행하기 시작한다. RP는 여러 WU_{i1}을 캐싱하고 있다가 다른 RP의 요청에 따라 전달해 주거나, 중복 수행을 위해 실행중인 WU_{ij}의 복사본을 전달하기도 한다.

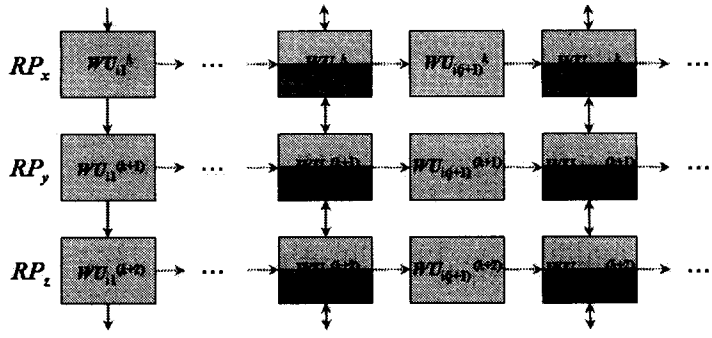
다른 RP로부터 WU_{i1}을 받은 RP는 서버에게 WU_{i1}에 대한 Enc(X)를 요청하고 이로부터 $hash(WU_{i1})$을 추출한다. 그리고 받은 WU_{i1}로부터 $hash(WU_{i1})$을 생성해내어 비교함으로써 위·변조되지 않은 올바른 WU_{i1}을 받았는지 확인한다.

4.2. 중복 수행

중복(replication)은 빠른 수행 또는 결함을 극복하고자 할 때 많은 경우 사용하는 기법이다. 데이터의 중복을 통해 가까운 곳에서 필요한 데이터에 빠르게 접근함으로써 보다 높은 성능을 얻고자 활용되기도 하고, 고장이 발생한 경우 중복된 데이터를 이용하여 수행하던 연산을 지속시킨다.

Korea@Home을 구성하는 RP들은 인터넷 환경에서 외부에 노출되어 있으므로 악의적인 외부 공격자, 악의적인 RP, 또는 조작의 실수 등으로 인해 WU이 변형되거나 위·변조될 수 있으며 이로 인해 올바른 결과를 얻지 못할 수 있다. 따라서 하나의 WU_{i1}에 대해 r개의 독립적인 RP에서 각각 WU_{i1}^1, ..., WU_{i1}^r을 수행하고, 그 결과를 비교함으로써 실행의 정확성을 얻는다.

하지만 임의의 i에 대해 AP가 필요로 하는 최종 결과인 WU_{im}을 얻기 위해 WU_{i1}부터 r개의 중복 수행이 이루어졌다고 할 때, 작업 수행의 중단없이 r개의 WU_{im}^1, ..., WU_{im}^r을 모두 얻었다 하더라도 이들 중 다른 값을 가진 임의의 WU_{im}^k가 존재하는지 확인하는 과정이 필요하다. r개의 결과 중 적어도 하나 이상이 다른 결과들과 같지 않다면 정확성을 의심하지 않을 수 없다. 따라서 $\{WU_{im}^1, ..., WU_{im}^r\}$의 크기가 1이 아닌 경우 정확성 검증의 과정이 수행되어야하는데, 값이 다른 WU_{im}^k에 대해 다시 수행하거나, 과반수 투표 등의 방법을 이용해 수행 결과가 다른 WU_{im}^k를 버리는 방법이 있을 수 있다. 첫번째의 경우는



<그림 3> 중복 수행의 실행 흐름 예

결과가 다른 $WU_{im}{}^{k}$를 $WU_{i1}{}^{k}$부터 다시 수행하는 데 오랜 시간이 소요되어 시간 비용이 크고, 두 번째의 경우는 확률적으로 적지만 과반수 투표가 항상 정확한 결과를 얻는 것을 보장하지는 못한다.

따라서 제안하는 기법에서는 과도한 재실행 시간 비용을 줄이기 위해 임의의 i에 대해 WU_{im}을 얻기 이전에 주기적으로 정확성 판단을 수행한다. 그리고 정확성 검증 과정 중에 서로 다른 결과가 나타나는 경우 특정 결과가 과반수를 넘었다 하더라도 어느 것이 옳은 수행인지 확신할 수 없으므로 과반수 여부와 관계없이 서로 다른 결과를 가진 단위작업들에 대해 각각 재수행하도록 한다.

4.3. 정확성 검증

<그림 3>에서는 WU_{i1}이 r개의 RP들에서 중복 수행되는 실행 흐름의 일부를 도식화하였다. r개의 RP들 중 서로 독립적인 RP_x, RP_y, RP_z에서 WU_{i1}의 복사본인 $WU_{i1}{}^{k}$, $WU_{i1}{}^{(k+1)}$, $WU_{i1}{}^{(k+2)}$ 등이 각각 실행되며, $WU_{i1}{}^{(k+1)}$은 RP_x로부터 $WU_{i1}{}^{k}$를 복사함으로써 중복 수행이 시작된다. 이런 방식으로 WU들을 수행하는 RP들 간에는 가상의 원(ring)형 구조가 형성된다. 그리고 RP_y는 단위작업들의 실행을 진행하면서 생성되는 $WU_{ij}{}^{(k+1)}$들에 대해 원형 구조의 이웃인 RP_x와 RP_z의 대응되는 $WU_{ij}{}^{k}$와 $WU_{ij}{}^{(k+2)}$들과 비교함으로써 현재까지 진행된 작업 수행이 서로 같은지 비교하고 정확성을 검증한다. 이 때 각각의 비교는 각각의 j에 대해 $WU_{ij}{}^{(k+1)}$이 생성될 때마다 수행되는 것은 메시지 전송 등의 부하를 유발할 수 있으므로 일정한 주기마다 비교될 수 있다. <그림 4>는 RP_y에서 중복 수행 중 이웃인 RP_x 및 RP_z와 WU 비교를 통해 정확성 검증 절차를 수행하는 알고리즘이다.

(1)에서 알고리즘이 시작되는 것은 새로 생성된 $WU_{i(j+2)}{}^{(k+1)}$가 정확성 검증 대상인 경우임을 볼 수 있는데, 매 s번의 WU 생성 후 자신이 실행한 결과가 정확한지 검증하기 위한 과정을 시

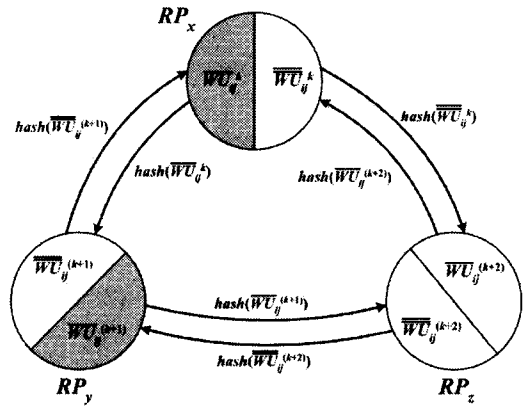
(1) 정확성 검증 대상인 $WU_{i(j+2)}{}^{(k+1)}$를 생성하면,
 (가) $WU_{i(j+2)}{}^{(k+1)}$를 WU_a와 WU_b로 분할
 (나) 분할된 WU_a와 WU_b에 대해 각각 $hash(WU_a)$와 $hash(WU_b)$ 생성
 (다) $WU_{i(j+2)}{}^{(k+1)}$의 ID를 $hash(WU_a)+hash(WU_b)$로 ID 부여
 (라) $hash(WU_a)$는 이웃 RP_x에게, $hash(WU_b)$는 이웃 RP_z에게 전송
 (마) 이웃 RP_x와 RP_z로부터 받은 해시값이 존재한다면, 각 해시값을 조합하여 ID와 비교
  (a) 같으면, $WU_{i(j+3)}{}^{(k+1)}$ 생성을 진행하고, 최근에 비교 성공한 $WU_{ij}{}^{(k+1)}$ 이전의 WU들 삭제
  (b) 다르면, 가장 최근에 비교 성공한 $WU_{ij}{}^{(k+1)}$로부터 재시작
(2) 정확성 검증 대상이 아닌 WU을 생성하면,
  다음 WU 생성을 위해 계속 진행

<그림 4> 각 RP에서 수행하는 WU의 정확성 검증 과정

작한다.

(나)와 (다)에서는 정확성 검증 과정 중 RP_x 및 RP_z들 간에 교환하는 메시지 전송량을 줄이기 위해 $WU_{i(j+2)}{}^{(k+1)}$의 해시값을 얻어 이용한다. 그리고 이 값을 ID로 활용함으로써 별도의 ID 부여체계를 갖출 필요없이 실제로 사용되는 내용물(content)을 기반으로 ID를 부여한다. 해싱의 특성에 따라 내용물이 변경되면 ID도 변경되며 서로 다른 내용물에 대해 서로 다른 ID를 부여할 수 있다. 부여되는 ID는 새로 생성된 $WU_{i(j+2)}{}^{(k+1)}$를 두 개로 분할하고 이들의 각 해시값을 조합한 것으로 한다.

(라)에서는 $WU_{i(j+2)}{}^{(k+1)}$의 분할된 두 부분에 대한 각 해시값을 서로 다른 이웃 RP_x와 RP_z에게 각각 전송한다. 이 값을 전송받은 RP_x와 RP_z는 (마)에 따라 각각 비교할 대상인 $WU_{i(j+2)}{}^{k}$와 $WU_{i(j+2)}{}^{(k+2)}$의 ID와 비교한다. 이 과정을 통해 "한번의 비교 연산"으로 이웃한 세 RP_x, RP_y, RP_z들이 독립적으로 생성한 $WU_{i(j+2)}$의 동일성 여부를 비교할 수 있다. 즉, 분할되지 않은 해시값을 이용하는 경우에 RP는 자신이 가진 해시값과 이웃으로부터 받은 두 해시값을 서로 비교하여 동일성 여부를 판단하여 모두 같거나, 두 개만 같거나, 모두 틀린 상황을 판별하게 된다. 이는 비교의 목적이 자신이 재실행을 수행할 것인가를 판단하는데 있지만, 자신의 WU 외에 두 WU 간의 동일성 여부까지 판별하는 불필요한 의미까지 포함하여 수행하는 셈이 된다. 이웃 RP로부터 받은 해시값을 조합한 값과 자신의 ID가 다른 경우, 자신의 수행이 맞건 틀리건 이웃 RP의 수행과 다름을 판단할 수 있고, 이를 통해 자신이 재실행되어야 함을 파악한다. 이웃 RP 역시 같은 방식으로 동작하기 때문에 각 RP마다 한번의 비교 연산을 통해, 세 WU 중 두 개만 같은 경우에는 서로 다른 WU를 가진 두 개의 RP만 재실행을 결정하고, 세 WU가 모두 다른 경우 세 RP 모두 재실행을 스스로 결정하게 된다. 즉, RP들 각자가 한번의 비교 연산을 통해 자신이 재실행할지를 판단하고, 이는 쌍을 이루는 세 WU들 중에서 하나씩의 서로 다른 결과만을 재실행하는 효과를 갖는다. 물론 r=2인 경우에도 서로 다른 경우에 재실행을 결정하게 된다.



<그림 5> 정확성 검증을 위한 해시값 교환 예

(b)에서 RPy는 $WU_{i(j+2)}{}^{(k+1)}$의 ID와 이웃 RP_x 및 RP_z로부터 받은 값들로 조합한 해시값이 다른 경우 재실행을 결정한다. 재실행은 가장 최근에 정확성 검증이 완료된 지점에서부터 시작되며 재실행이 두차례 이상 반복된다면 가장 최근에 정확성 검증이 끝난 WU가 나고 유지되고 있던 중에 위·변조된 것으로 간주하여 이 단위작업에 대한 수행을 포기한다.

(a)에서 $WU_{i(j+2)}{}^{(k+1)}$의 ID와 이웃 RP들로부터 받은 해시값으로부터 조합한 결과가 같은 경우에 이후 수행을 계속 진행하며 더 이상 필요하지 않은 개체들은 삭제한다. 새로 생성한 $WU_{i(j+2)}{}^{(k+1)}$가 이웃 RP들의 $WU_{i(j+2)}{}^{k}$와 $WU_{i(j+2)}{}^{(k+2)}$를 비교한 결과 동일한 것으로 판단하였더라도 재실행 가능성이 존재하기 때문에 $WU_{ij}{}^{(k+1)}$는 즉시 삭제할 수 없다. r>2인 경우 옳지 않은 결과를 내놓은 두 RP가 인접하였을 때에는 임의의 WU_{ij}가 재실행하지 않도록 판단하였다 하더라도 그 다음 $WU_{i(j+s)}$의 정확성이 검증되기 전에는 재실행 가능성이 있다.

<그림 5>을 통해 그러한 상황을 설명하는데, 하나의 WU에 대해 중복 수행 중인 세 개의 RP들이 분할된 해시값을 교환하는 예를 보여준다. WU_{ij}를 중복 수행에 따라 RP_x, RP_y, RP_z가 정확성 검증을 수행 중일 때, RP_x의 분할된 결과값은 $\overline{WU_{ij}}{}^{k}$와 $\overline{WU_{ij}}{}^{k}$이고, RP_y의 분할된 결과값은 $\overline{WU_{ij}}{}^{(k+1)}$과 $\overline{WU_{ij}}{}^{(k+1)}$, RP_z의 분할된 결과값은 $\overline{WU_{ij}}{}^{(k+2)}$와 $\overline{WU_{ij}}{}^{(k+2)}$라고 하자. 이 때 RP_x와 RP_y의 결과값은 서로 같지만 실제로는 옳지 않은 실

(3) (고장 또는 이탈의 인식) RP_x가 일정 기간동안 이웃 RP_y로부터 해시값을 받지 못하면,
(바) RP_x는 가용한 새로운 RP_y'을 선정하고 중복 수행 중인 다음 RP_z에게 통보
(사) RP_x는 정확성 검증이 끝난 최근 단위작업의 분할된 한 부분인 $\overline{WU_{ij}}^{k}$과 ID를 RP_y'에게 전송
(아) RP_z는 정확성 검증이 끝난 최근 단위작업의 분할된 다른 부분인 $\overline{\overline{WU}_{ij}}^{(k-2)}$과 ID를 RP_y'에게 전송
(자) RP_y'는 두 ID 간 비교 및 $hash(\overline{WU}_{ij}^{k})$+$hash(\overline{\overline{WU}}_{ij}^{(k-2)})$과 ID를 비교
(c) 같으면, 분할된 두 부분을 조합하여 $WU_{ij}^{(k+1)}$을 완성하고 $WU_{i(j+1)}^{(k+1)}$ 생성을 진행
(d) 다르면, 서버로부터 WU_{il}을 받아 재실행

<그림 6> *RP*의 접속단절, 이탈 또는 고장에 대한 결함 포용 알고리즘

행 결과라고 가정하자. RP_x는 $hash(\overline{WU}_{ij}^{(k+2)})$를 받고 자신과 결과가 다름을 알고 재실행을 결정하며, RP_z는 $hash(\overline{WU}_{ij}^{(k+1)})$을 받고 재실행을 결정한다. 즉, 각각은 자신의 결과가 옳은지 그른지 전역적으로 판단할 수 없으므로 옳건 그르건 모든 결과 중에서 서로 다른 하나씩의 결과가 재실행된다. RP_y는 옳지 않은 결과를 생성함에도 불구하고 재실행을 결정하지 않는다. 따라서 RP_y도 옳지 않은 수행을 했기 때문에 재실행되어야 하는데, 이는 RP_x가 재실행되고 다시 올바른 결과를 내놓았을 때 정확성 검증을 수행한 후 해결된다. RP_y는 이웃 RP_x 및 RP_z와의 해시값 교환을 통해 현재 자신이 생성한 결과가 중복 수행된 다른 것들과 같은 것으로 나타났어도 이후 재실행을 위해서는 정확성이 검증된 $\overline{WU}_{i(j-s)}$와 $\overline{\overline{WU}}_{i(j-s)}$를 유지하고 있어야한다.

4.4. 결함 포용

인터넷 환경의 자발적인 *RP*들의 자원제공을 기반으로 하는 P2P 컴퓨팅에서는 접속단절과 *RP*의 참여 및 이탈이 자주 발생한다. 따라서 이에 적절히 대처하고 연산의 지속성을 보장하기 위한 결함 포용(fault-tolerance) 기법이 반드시 포함되어야 한다.

앞 절에서 제안한 기법에서 r개의 중복 수행을 기본으로 하고 있기 때문에 자연스럽게 *RP*의 고장이나 접속단절, 참여와 이탈에 대한 대비가 가능하며, 주기적인 정확성 검증 과정을 통해 각 *RP*들은 검사점(checkpoint)를 갖게 된다. 즉, 정확성을 검증 받은 최근의 WU이 검사점이 되며, 임의의 *RP*에 발생한 고장에 대해 이 WU으로 복귀(rollback)하여 회복(recovery)될 수 있다.

<그림 6>은 <그림 3>의 상황을 바탕으로 수행되는 결함 포용 알고리즘이다.

(3)에서 RP_x는 이웃 RP_y로부터 정확성 검증을 위한 해시값이 일정 기간동안 도착하지 않으면 RP_y의 고장으로 간주하여 결함 포용 알고리즘을 시작한다. (바)에서 RP_x는 RP_y가 아닌 새로운 RP_y'을 선정하여 RP_y가 수행하던 작업 수행을 계속하도록 한다. 원형 구조에서 연속된 *RP*에 고장이 발생한 경우는 그 다음 *RP*를 선정한다. (사)와 (아)에서는 정확성 검증이 끝난 최근의 단위작업을 RP_x와 RP_z의 분할된 단위작업을 얻어 조합함으로써 생성한다. 이렇게 새로 시작할 단위작업을 얻었지만 이의 정확성을 (자)에서 다시 확인한다. RP_x 또는 RP_z가 WU_{ij}를 유지하는 동안 대한 외부의 공격이나 RP_x 또는 RP_z의 악의적인 행위로 인해 WU_{ij}가 위·변조되었는지의 여부를 다시 확인하여 보안성과 정확성을 강화한다. 위·변조되었다면 (d)에서 최악의 상황으로 간주하여 서버로부터 WU_{il}을 받아 재실행하도록 한다. 하지만 RP_y'를 제외한 나머지 *RP*들 간에 다시 (1)을 수행하는 등의 과정을 진행하여 처음부터 재실행하는 비용을 절감하는 기법 등도 있을 수 있다.

5. 제안방법의 평가

5.1. 정당성

중복 수행 중인 임의의 *RP*가 이탈했거나 고장이 발생한 경우 <그림 6>의 알고리즘에 따라 새로운 *RP*를 선정하여 이탈한 *RP*의 수행을 지속하도록 하므로 중복의 개수는 항상 r을 유지하게 된다.

또한 임의의 j에 대해 $R_j=\{WU_{ij}^k \mid 1\le k\le r\}$에서 가정에 따라 올바른 WU_{ij}^k는 적어도 하나 존재하며, 임의의 k'이 존재해서 $WU_{ij}^{k'}$가 옳지 않은 결과라면 WU_{ij}^k와 $WU_{ij}^{k'}$은 내용이 달라 $|R_j|>1$이 된다. $|R_j|>1$이라면 서로 다른 WU_{ij}^k와 $WU_{ij}^{k'}$이 존재하여 (마)에 의해 반드시 재시작하게 된다. $|R_{(j-1)}|=1$이고 이것이 올바르다면 또다시 옳지 않은 WU_{ij}^k가 생성될 확률은 적다는 가정에 따라, 또다시 옳지 않은 WU_{ij}^k가 생성되면 재시작을 반복하게 되어 올바른 WU_{ij}^k로 수렴하며, 결국 $|R_j|=1$이 된다.

R_1은 3.1의 암호화 기법을 통해 $|R_1|=1$이고 올바르다는 것이 보장되므로, 연속적으로 적용되어 최종적으로 $|R_m|=1$이 된다. 마찬가지로 가정에 의해 옳지 않은 단위작업을 가진 R_i는 항상 $|R_i|>1$이므로, R_m에 속한 하나의 WU_{im}^k는 올바르다.

5.2. 성능 논의

정확성 검증을 위한 <그림 4>는 각 RP가 매 s번의 단위작업 수행마다 시작되는데, 이 s의 크기를 어느 정도로 할 것인가를 고려해볼 필요가 있다. t는 하나의 RP에서 하나의 WU 실행을 마치는데 필요한 평균 시간이고, t'는 하나의 RP가 다른 RP들과의 비교를 위한 처리와 메시지 교환을 마치는데 필요한 시간, 그리고 f는 r개의 중복 수행 중 정확하지 않은 결과를 내놓는 확률이라고 가정하자. 그러면 <그림 4>의 과정으로 인해 추가되는 시간 비용은 $r\cdot f$개의 RP에서 각 s번의 재실행과 그 재실행 중 m/s번의 정확성 비교 메시징으로부터 발생한다. 따라서 추가적인 시간 비용은 $r\cdot f\cdot s\cdot t + (r\cdot m/s)\cdot t'$가 된다. s가 작으면 정확성 비교 횟수가 증가하여 메시징 비용이 늘어나지만, t가 t'에 비해 상당히 큰 경우가 일반적이므로 s가 커지면 재실행 횟수가 늘어날 가능성이 높아 추가 시간 비용 부담이 작은 s에 비해 더 커질 수 있다.

그리고 제안하는 기법이 갖는 장점으로 인하여 성능향상 효과를 얻을 수 있다. 각 RP들의 실행 속도는 매우 다양할 수 있기 때문에 최종적으로 r개의 중복 수행 결과를 얻는데는 시간 차이가 발생하여, 불필요하게 중복된 다른 결과를 기다려야하는 상황이 발생할 수 있다. 그러나 제안하는 기법에서는 최종 결과를 얻기 전 매 s번의 단위작업 수행마다 서로 정확성을 검증하고 정확성이 검증되지 않으면 진행하지 않으므로 불필요한 기다림으로 인해 발생하는 시간낭비를 사전에 방지할 수 있다.

또한 정확성 검사를 위한 각 비교 중에 분할된 단위작업의 해시값을 이용함으로써 과반수 투표와 같이 전체 단위작업들을 서로 비교할 필요없이 자체적으로 한번의 비교만으로 중복된 다른 실행들과의 비교를 수행할 수 있다.

생성되는 단위작업 개체에 대해 내용물을 기반으로 ID를 부여함으로써 별도의 명명법(naming scheme)을 구현하지 않아도 유일성을 가진 ID를 부여할 수 있다.

6. 결 론

이 논문에서는 인터넷을 기반으로 자발적으로 제공되는 컴퓨팅 자원을 공유함으로써 거대 응용을 수행하는 환경에서 수행 결과의 정확성을 검증하는 기법을 제안하였다. 시스템에 참여하는 독립적인 컴퓨팅 자원들은 네트워크 상태의 다양성으로 인한 접근 불가나 자원 소유자의 의도에 따른 이탈 등이 자주 발생하며, 보안 위협에도 노출되어 있다. 여러 연구에서 이러한 문제들을 해결하기 위해 중복 기법을 사용하는데, 이 논문에서도 연산의 중복을 통해 자원의 이탈과 접속 단절, 고장 등을 견디고 결과의 정확성을 보장하는 기법을 제안하였다. 제안하는 기법은 시스템 전체의 전역적인 메시지 교환이 없이도 중복된 연산 결과의 동일성 여부를 판단하고 정확한 연산을 수행하였는지 검증한다.

참고 문헌

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

2002
[2] David Barkai, *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*, Intel Press, 2002
[3] Korea@Home, http://www.KOREAatHOME.org
[4] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing", *Comm. of the ACM*, 45(11), pp.56-61, Nov. 2002
[5] distributed.net: Node Zero, http://www.distributed.net/
[6] W.Bolosky, J.Douceur, D.Ely, and M.Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs", *Proc. of the ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems*, pp.34-43, Jun. 2000
[7] A.I.T.Rowstron and P.Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", *Proc. of the 18th ACM Symposium on Operating System Principles*, pp.188-201, Oct. 2001
[8] A.I.T.Rowstron and P.Druschel, "Pastry: Scalable, Decentralized Object Location, Routing for Large-Scale Peer-to-Peer Systems", *IFIP/ACM Int'l Conf. on Distributed Systems Platforms (Middleware2001)*, pp.329-350, Nov. 2001
[9] C.G.Plaxton, R.Rajaraman, and A.W.Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment", *Proc. of the 9th ACM Symp. on Parallel Algorithms and Architectures*, pp.311-320, Jun. 1997
[10] B.Zhao, J.Kubiatowicz, and A.Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing", *TR UCB/CSD-01-1141*, C.S. Devision, U.C. Berkeley, Apr. 2001
[11] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", *Proc. of the ACM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication(SIGCOMM 2001)*, pp.149-160, Aug. 2001
[12] S.Ratnasamy, P.Francis, M.Handley, R.M.Karp, and S.Schenker, "A Scalable Content-Addressable Network", *Proc. of the ACM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication(SIGCOMM 2001)*, pp.161-172, Aug. 2001
[13] Napster, http://www.napster.com
[14] DCGrid, Etropia, http://www.entropia.com
[15] Groove Networks, http://www.groove.net
[16] WebRiposte, Escher Group, http://www.eschergroup.com
[17] A. Grimshaw, W.A.Wulf, The Legion Vision of a World Wide Virtual Computer, *Comm. of ACM*, 40(1), pp39-45, Jan. 1997

박 찬 열

1993 고려대학교 수학과 (이학사)
1995 고려대학교 컴퓨터학과 (이학석사)
2000 고려대학교 컴퓨터학과 (이학박사)
2002~현재 한국과학기술정보연구원 슈퍼컴퓨팅 센터 선임연구원
관심분야: 분산시스템, P2P 컴퓨팅, 결함포용
E-Mail: chan@kisti.re.kr