Server Cooling 알고리즘을 이용한 서비스 객체 이주시의 동기화 알고리즘

이 준 연† · 김 창 민††

요 약

본 논문에서는 분산 시스템에서 서비스 객체 이주시 발생하는 동기화 문제를 해결하는 알고리즘을 제시하였다. 분산 시스템에서는 각 노드에 할당되는 부하의 양의 차이로 인하여 부하 불균형이 발생한다. 이를 해결하기 위하여 과부하 노드에서 저부하 노드로 서비스 객체가 이주되어야 한다. 이러한 과정에서 서비스 객체가 완전히 이주되기 전에 클라이언트의 요청을 받았을 때 동기화 문제가 발생할 수 있다. 따라서 본 논문에서는 서비스 객체 실행 환경인 importer/exporter/trader 모델을 기술하고, 서비스 객체 이주시 비동기가 발생할 수 있는 상황을 분석하여 적절한 이주 알고리즘을 제안하였다. 또한 모의실험을 통하여 본 논문에서 제안한 알고리즘의 성능을 측정하였다.

A synchronization algorithm of migrating service object using Server Cooling algorithm

Jun-Yeon Lee† · Chang-Min Kim††

ABSTRACT

In this paper, we propose the algorithm which solve synchronization problem happened migrating service objects. Load imbalance is occurred due to allocate different amount of load to be proposed to each node in distributed system. To solve this problem, the service objects have to be migrated from heavily loaded node to lightly loaded node. And in such a process, the synchronization problem may occur when client request a service object migrated incompletely. Therefore, we describe the environment of executing service objects, the importer/exporter/trader model compatible with migration of service objects, and appropriate migration algorithm. Finally, we analyze the conditions of problems, and propose the solution of each situations. Also, the performance advantages of using proposed algorithms are quantified through a simulation study.

1. 서 론

최근 통신 기술의 발달로 지역적으로 분산된 여러 개의 워크스테이션들이 컴퓨팅 자원을 공유하기 위하여 하나의 네트워크로 서로 연결하는 것이 가능해졌다. 분산 운영 체제는 효율적인 사용과 좋은 성능을 보장하도록 각 자원을 할당해야 한다. 분산 시스템에서 가장 중요한 문제중의 하나가 바로 가용 자원의 할당이다. 일반적으로 각 노드의 부하를 발생시키는 자원의 할당량은 동일하지 않으므로 어떤 노드는 유휴 상태인 반면에 어떤 노드는 과부하 상태가 된다. 따라서 컴퓨팅 자원의 효율적인 사용을 위하여 가용 자원

† 준 회 원 : 동명정보대학교 멀티미디어공학과 교수
†† 정 회 원 : 성결대학교 컴퓨터학부 교수
논문접수 : 1999년 3월 19일, 심사완료 : 1999년 12월 21일

들의 부하 분산이 필요하다. 이러한 부하 분산의 목적은 시스템의 부하의 불균형을 피하고, 따라서 더 좋은 성능을 얻는 것이다[8].

클라이언트/서버 시스템에서의 부하분산은 상당히 어렵다. 클라이언트/서버 시스템에서 클라이언트는 일련의 서버에서 제공되는 일련의 서비스를 요청한다. 서비스는 서버에서 사용 가능한 서로 다른 종류의 자원을 나타낸다. 각 클라이언트는 일련의 오퍼레이션을 수행하는데, 각 오퍼레이션은 특정 서비스를 요청할 수 있다. 모든 서버의 모든 서비스가 이용가능하지 않기 때문에, 클라이언트는 현재 요청을 처리할 수 있는 서버를 찾아야 한다. 이러한 목적을 위하여 클라이언트와 서버 사이의 서비스를 중재하기 위하여 trader라는 노드가 사용된다. 본 논문에서는 문제를 간단히 하기 위하여 단일 종류의 작업만을 고려하였고, 이러한 작업은 시스템의 모든 노드에서 처리가 가능하다. 그러므로 실제 클라이언트/서버 시스템에서는 자원의 할당에 있어서 여러 종류의 작업이 고려된 방법이 사용된다.

클라이언트/서버 시스템에서의 또다른 차이점은 서비스를 제공하는 기계가 이기종이라는 것이다. CORBA 환경에서는 서버로의 접근이 투명하게 수행되므로 이기종 기계로의 접근에 추가적인 비용이 없다는 것이다. 그러나 부하 분산에 있어서 사용되는 기계의 특징을 고려하는 것은 매우 중요한 것이다[4, 5].

CORBA 환경에서 부하 분산이 수행됨에 있어 몇가지 추상적인 계층이 있다. 이러한 계층은 부하 분산 정책을 여러 부류를 나누어 실제 시스템에서 이들을 사용하도록 한다. 첫 번째 계층은 바로 객체 계층이다. 이 계층에서 시스템은 CORBA 객체들의 집합으로 구성된 것으로 간주되는데, 각 객체는 객체 레퍼런스로 구분된다. 객체를 호출할 때 객체 레퍼런스가 사용된다. 이 정책은 클라이언트의 요청을 사용 가능한 서버에 분산할 때 사용된다. 특정 서비스를 호출하는 클라이언트는 객체 레퍼런스를 얻기 위하여 trader에게 문의하여야 한다. 이 정책에서 부하 분산은 적절한 객체 레퍼런스를 선택하여 클라이언트에게 돌려주는 trader에 의하여 수행되거나, 혹은 trader가 모든 객체 레퍼런스를 돌려주고 이들중 하나를 선택하는 클라이언트에 의하여 수행된다[5].

부하 분산의 두 번째 계층은 구현 단계이다. 이것은 운영체제가 사용 가능한 객체 항목의 서버를 어떻게 아는가를 고려하여야 한다. 객체는 특정 프로세스의 객체를 나타내는 객체 인스턴스로 구현된다고 가정한다. 이 정책에서는 사용 가능한 서버 중에서 서비스 객체 인스턴스의 분산으로 설명된다. 이 경우, 부하 분산은 서비스를 제공하는 객체 인스턴스를 사용 가능한 서버로 할당하는 것이다. 부하 분산을 실행하기 위하여 객체 인스턴스는 서버 사이에서 서로 이주될 수 있다.

세 번째 부하 분산 단계는 시스템 단계이다. 이 방법은 프로세스를 전통적인 방법으로 노드에 할당하고, 자원에 대한 요청은 각 자원에 할당한다. 이 방법은 본 논문의 부하 분산 시스템에서는 사용되지 않기 때문에 여기서는 더 이상 언급하지 않는다.

본 논문에서는 두 번째 부하 분산 정책에 관하여 연구하고자 한다. 이 정책은 클라이언트가 적절한 서비스 객체들 중에서 하나를 요청하고, 해당 서비스를 제공할 수 있는 서버에게 서비스 객체를 할당한다.

첫 번째와 두 번째 정책은 서로 독립적이고, 동시에 적용될 수 있다. 이 두 정책의 주요 차이점은 분산 항목의 입자성, 즉 분산되는 항목의 크기이다. 요청 할당의 입자성은 상대적으로 적은데, 그 이유는 클라이언트 요청의 순서는 일련의 클라이언트의 요청이 매우 크지 않다. 반면에, 이주되어야 할 상태 정보를 요약한 객체는 큰 항목이므로 서비스 객체의 크기는 상대적으로 훨씬 더 크다.

서비스 객체의 이주를 제어하는 것은 감독자(supervisor)의 책임이다. 감독자는 각 서버로부터 상태정보를 수집하고, 객체의 이주가 적합한지를 검사하여야 한다.

만약 클라이언트가 이주가 완성되지 않은 서비스 객체를 요청할 때, 동기화 문제가 발생한다. 이 때 객체가 이주되는 동안 제기되는 클라이언트의 요청은 정확한 객체의 위치로 전달되어야 한다.

본 논문에서는 이러한 문제를 해결하기 위한 알고리즘을 제기하였다.

본 논문은 다음과 같이 구성되었다. 제 2장에서는 서비스 객체 이주시의 동기화가 필요한 이유에 대하여 기술하였고, 제 3장에서는 시스템 모델과 이주 정책에 관하여 알아보았다. 제 4장에서는 비동기 상황이 발생하는 조건을 알아보고, 이 문제를 해결하기 위한 알고리즘을 제안하였다. 제 5장에서는 동기화 알고리즘의 성능을 측정하기 위한 모의실험에 대하여 기술하고, 마지막으로 6장에서는 결론을 맺고자 한다.

2. 동기화 이유

서비스 객체 이주의 과정과 관련하여 많은 문제가 있다[1]. 첫 번째 문제는 서비스 객체 상태의 정의, 관련 상태 정보의 전송, 그리고 새로운 환경에의 접속과 관련된 것이다.

서비스 객체의 상태는 프로세스 주소 공간에 의해서 뿐만 아니라 지역 메시지 큐, 그리고 원격 서비스 객체의 통신 상태 등에 의하여 정의되므로 서비스 객체의 상태를 결정하는 것은 간단하지 않다. 그러나 모든 메시지의 교환을 기록할 필요는 없다. 이것은 서비스 객체의 이주를 나타내는 모든 메시지와 그 순서를 기록할 필요가 없다는 것이다.

importer/exporter/trader 모델에서는 클라이언트의 요청이 어떤 상태인지를 결정할 수 있다. 이것은 import 명령을 수행함으로서 가능하다. 이 명령을 수행하면 trader는 서비스 객체의 상태를 제공한다. 이러한 정보는 trader에 의해서 유지되는 디렉토리에 저장된다.

두 번째는 이주 정책 문제인데, 이것은 언제 이주하고 어느 서비스 객체를 어디로 이주하여야 가장 좋은 성능을 얻을것인가에 관한 것이다. 그리고 어떤 경우에는 현재 원격 컴퓨터로부터 서비스 객체를 언제 선점할 것인가를 결정하는 것도 필요하다. 이주 정책은 제 3장에서 다루기로 한다.

세 번째 문제는 다른 서비스 객체에 투명하게 서비스 객체의 새로운 위치를 지정하고, 서비스 객체의 이주에 아무런 영향을 미치지 않고 전송을 수행하여야 한다는 것이다. 이러한 문제는 CORBA 표준에 의하여 간단하게 해결될 수 있다. CORBA 표준에 의하여 제공되는 투명성의 종류는 다음과 같다.

접근 투명성 : 클라이언트가 객체 구현을 요청할 때는 서버가 클라이언트와 동일한 주소공간에서 수행되거나 혹은 다른 기계에서 수행될 때 문법적 차이는 없다.

위치 투명성 : 클라이언트는 자신이 사용하고자 하는 객체의 실제 위치를 알 필요가 없다.

구현 투명성 : 서버 객체를 구현하는데 사용된 언어는 자신을 사용하는데 아무런 영향을 미치지 않는다.

이러한 문제는 chooseServer 함수로 구현된다. 이 함수는 특정 부하 분산 정책에 따라import 리스트에서 서버를 선택한다.

다음 문제는 서비스 객체 이주시의 동기화 문제이다. 어떤 정책에 따라 감독자가 한 서버에서 다른 서버로 서비스 객체의 이주를 결정하여야 한다면 이주 프로세스를 시작하여야 한다. 즉, 대상 서버에게 주어진 원본 서버로부터 특정 서비스 객체의 이주를 알려야 한다.

〈표 1〉 서비스 객체 이주 과정

1) 감독자 -> 대상 서버	: 이주 시작
2) 대상 서버 -> 원본 서버	: 새로운 객체 생성
3) 대상 서버 -> 원본 서버	: 객체 상태정보 요청
4) 원본 서버 -> 대상 서버	: 객체의 상태정보 송신
5) 대상 서버 -> trader	: 객체 이주 예약
6) trader -> 대상 서버	: 원본 서버에게 알림
7) 대상 서버 -> 원본 서버	: 객체 복사본 삭제
8) 대상 서버 -> 감독자	: 이주 완료

이와 반대로 대상 서버는 특정 객체를 받기 위하여 원본 서버와 접촉하여야 한다. 이러한 것은 대상 서버에 동일한 형태의 새로운 객체를 생성하고, 원본 서버로부터 객체 상태를 전달받는다. 객체 상태가 전송된 후, 대상 서버는 trader에게 서비스 객체의 이주를 알리고 원본 서버의 객체 복사본을 삭제하여야 한다.

서비스 객체의 이주 과정은 <표 1>과 같다.

만약 클라이언트가 이주가 완전히 끝나지 않은 서비스 객체를 요구할 때 동기화 문제가 발생하게 된다. 서비스 객체가 이주하는 동안 클라이언트의 요청이 발생할 경우, 이 요청은 정확한 객체의 위치로 전달되어야 한다.

본 논문에서는 비동기가 발생할 수 있는 상황을 3가지로 나누고, 이 때 각각의 문제를 해결할 수 있는 알고리즘을 제안하고자 한다.

3. 시스템 모델과 이주 정책

3.1 시스템 모델

Exporter/Importer/Trader 모델은 trader에 의하여 클라이언트/서버 모델로 확장할 수 있는데, 여기서 trader는 클라이언트와 서버 사이에서 서비스의 중재 역할을 한다. 이 모델에서 클라이언트는 요청된 서비스가 가장 적절한 서버를 동적으로 찾을 수 있다.

서비스를 중재하는데 필요한 기본적인 과정은 다음과 같다.

1. 클라이언트에게 서비스를 제공하려는 서버는 이러한 서비스들을 export하여야 한다. 즉, trader에게 자신의 함수를 예약하여야 한다. 따라서 서버는 exporter가 된다.
2. 만약 클라이언트가 특정 서비스를 사용하려 한다면, 클라이언트는 trader에게 import명령을 호출하여야 한다. 그 결과 요청한 서비스에 대한 적절한 서버를 얻게 된다. 따라서 클라이언트는 해당 서비스의 importer가 된다.
3. 서비스를 사용하기 위하여 클라이언트는 정보를 받은 서버로 직접 요청을 전송한다.

만약 서버가 자신이 제공하는 서비스를 더 이상 제공하지 못한다면 trader에게 withdraw 명령어를 실행하여 해당 서비스를 더 이상 사용할 수 없음을 알린다. 여기서 중요한 것은 trader에게 서비스를 어떻게 예약하고, 클라이언트가 자신이 import하고자 하는 서비스의 종류를 어떻게 기술하는가이다. 이러한 이유로 인하여 서버는 export하고자 하는 서비스의 종류에 관한 정보를 제공하여야 한다. 서비스의 기능에 관한 속성을 기술함으로서 이러한 정보를 얻을 수 있다. 이것은 서비스 형태의 이름만을 기술함으로써 구현된다. 그리고 서버가 export하는 각 객체에 대하여 이 서비스를 접근하는 방법도 제공된다. CORBA에서는 서비스가 객체에 의하여 제공되므로 서비스의 객체 레퍼런스를 예약하여야 한다. 따라서 클라이언트가 원하는 서비스를 import하려 한다면 서비스 형태의 이름을 기술하여야 한다. trader는 이 형태를 적절한 객체 레퍼런스로 매핑시키고, 이 레퍼런스로 서버에 직접 접근할 수 있는 클라이언트에게 레퍼런스를 보낸다. 이 방법은 간단하고 객체에 대하여 유일한 접근 방법을 보장한다[3].

CORBA환경에서 trade 기능은 투명하게 ORB(Object Request Broker)에 통합되어 있으며, CORBA 객체 서비스의 한 요소로서 제공된다[4].

3.2 객체 이주 정책

클라이언트/서버 응용은 서버로의 요청을 생성하는 많은 수의 클라이언트로 구성되어 있다. 또한 서버는 다른 서버의 클라이언트처럼 동작하기도 한다. 이러한 시스템은 하나의 클라이언트와 병렬 서버로 구성된 소프트웨어 시스템처럼 간주되기도 한다. 반면에 서비스는 서비스 객체로 구성되어 있다[7].

동적 부하 분산을 이해하는데는 두가지 기본적인 메커니즘이 있다.

- 요청 할당 : 클라이언트가 서비스를 요청할 때 부하 분산 정책은 서비스를 구현한 서비스 객체를 선택할 수 있다.
- 서비스 객체 할당(이주) : 서비스 객체가 동적으로 서버에 할당될 수 있다. 본 논문에서는 서버에 새로 생성된 서비스 객체의 동적 할당 정책에 관해서는 고려하지 않는다. 그 이유는 서비스를 구현한 서비스 객체를 영속하는 것으로 가정하기 때문이다. 대신 서버들간의 서비스 객체의 이주를 고려한다.

이 모델에 따르면 두가지 부하 분산 정책이 있는데, 하나는 요청 할당이고 다른 하나는 이주이다. 모든 부하 분산 정책을 연구하는 것은 너무나 방대한 것이므로, 본 논문에서는 서비스 객체의 이주와 이주시 발생하는 객체들의 동기화에 관해서만 연구하고자 한다.

본 논문에서 언급된 서비스 이주 정책은 [2]와 [6]에서 제안한 "server cooling" 알고리즘에 동기화 방안을 연구한 것이다.

이 논문들은 분산 파일 시스템에서의 파일 할당에 관련된 문제를 처리하였다. 이 알고리즘을 간략하게 살펴보면 많은 수의 디스크에 파일을 할당하여 과부하 상태가 되는 디스크가 없도록 하는 것이다. 즉 각 디스크에서 접근하는 블록 수의 균형을 유지하는 것이다. 이 논문에서는 단위 시간당 디스크의 접근 횟수를 사용하는데 이것은 디스크의 "heat"라 부른다. 따라서 과부하 디스크에서 저부하 디스크로 파일을 할당하는 알고리즘을 "disk cooling" 알고리즘이라 부른다[2, 6].

이 상황은 CORBA 응용프로그램과 매우 유사하다. 디스크는 부하 분산을 수행하는 서버로 대치하고, 블록 접근은 서버에 의하여 처리되는 클라이언트의 요청으로 대체하여 생각할 수 있다. 부하 분산을 수행하기 위해서는 디스크에 할당되는 파일 대신에 사용 가능한 서버들에 서비스 객체를 분산하는 것으로 대체하면 된다.

그러므로 "Server Cooling 알고리즘"이라 불리는 부하 분산 정책을 제안하고자 한다.

이 알고리즘은 "hot"(과부하) 서버에서 "cool"(저부

하) 서버로 적절한 서비스를 이주하게 된다. 다음 (그림 1)은 기본적인 "server cooling" 알고리즘을 설명하고 있다.

부하가 가장 많은 서버에서 시작하여 일정 비율(p) 이상의 평균 부하를 넘는 모든 서버가 이주가능한 서비스를 가지고 있는지를 검사한다. 모든 서버에서 먼저 서버에 가장 많은 부하를 발생시키는 서비스 객체는 다른 서버로 이주를 시도할 것이다. 이렇게 하기 위하여 서버는 다른 서비스 객체에 의하여 발생되는 부하의 양을 유지하여야 한다. 이 알고리즘에서 $SV_{i,j}$가 의미하는 바는 서버 i의 서비스 객체 j를 의미한다.

```
Get load information from all known servers.
Calculate average server load index L_avg.
Sort the selected servers by descending load index
For every server S_i (i = 1, ..., n) :
   Sort all services within the server by
    descending load quotas.
FORALL servers S_i (i = 1, ..., n) with a load
 index L_i > p*L_avg DO
   FORALL services SV_ij (j = 1, ..., n_i) DO
      FORALL other servers S_k
       with k = n, ..., i+1 DO
         IF (Sk may provide service SV_ij )
         AND (S_k does currently not provide
              SV_ij)
         AND (load of S_k would not rise above
              current load of S_i)
             THEN (migrate SV_ij from S_i to S_k)
             STOP algorithm
          ENDIF
      ENDDO
   ENDDO
ENDDO
```

(그림 1) Server Cooling 알고리즘

이주가 가능한 서버 중에서 부하가 가장 적은 서버가 대상 서버로 선택된다. 그리고 진동 현상(oscillating)[1]을 피하기 위하여, 대상 서버의 부하가 원본 서버의 현재 부하량보다 더 커지지 않을 때 서비스 객체를 이주하게 된다. 또한 "server cooling" 알고리즘이 한번 실행될 때 하나의 서비스 객체만이 이주될 수 있다. 그 이유는 서비스 객체 이주시 상대적으로 많은 비용이 소요되므로 이주를 많이 허용하지 않는다.

"server cooling" 알고리즘은 감독자에 의하여 정기적으로 수행된다. 이 알고리즘이 수행될 때 연속하는 두 개의 시간 간격의 길이는 서버의 부하의 양의 변화를 부하 분산 시스템에 얼마나 빨리 반영하는가를 결정한다.

감독자 기반의 서비스 이주로 인하여 발생하는 자원의 비용은 상당히 높다. 매번 실행될 때마다 모든 이용 가능한 서버로부터 부하 정보를 수집하므로, 통신 비용은 서버의 수에 비례하여 증가한다. 또한 객체를 이주할 때 소요되는 비용도 상당히 높다. 따라서 "server cooling" 알고리즘은 가능한 한 부하 분산의 오버헤드를 최소한으로 유지하기 위하여 너무 자주 실행되어서는 안된다.

4. 동기화 알고리즘

서버간의 서비스 객체의 이주는 감독자에 의하여 제어된다. 만약 감독자가 원본 서버에서 대상 서버로 한 객체를 이주하기로 결정하였다면, 대상 서버에게 migrateService 명령을 실행한다. 이 때 대상 서버로 전달되는 매개변수는 원본 서버의 이름과 객체 레퍼런스, 그리고 원본 객체의 이름과 객체 레퍼런스이다.

제 2장에서 기술한 객체 이주 과정에서 동기화 문제가 발생할 수 있는 부분을 크게 3개로 나눌 수 있다. 첫 번째는 <표 1>의 1) 에서 4)까지의 과정 수행중 객체 접근이 발생하는 경우이다. 이 부분은 trader가 감독자에게 이주를 알린 후 클라이언트가 import 명령을 실행하는 부분이다. 이 부분에서는 동기화 문제는 간단하게 해결될 수 있다. 여기서는 단지 trader가 새로운 객체 레퍼런스를 돌려주고, 이주가 종료될 때까지 기다리게 된다. 이 과정이 <표 2>에 나타나 있다.

두 번째는 <표 1>의 5)에서 6) 사이에 클라이언트의 요청이 발생하는 경우이며, 이 때에는 다음의 (그림 3>에서 그 처리과정을 보여준다. 원본 객체의 상태가 이미 대상 서버로 전송되었으나, 원본 객체는 아직 삭제되지 않은 경우이다. 원본 객체는 클라이언트로부터 여전히 요청을 받고 있으나 객체의 상태가 이미 전송되었기 때문에 이 객체의 처리를 더 이상 해서는 안된다. 이 경우 원본 객체는 대상 서버에 생성된 객체의

1) 서비스 객체 이주시 과부하 서버는 이주후 저부하가 되고, 저부하 서버는 과부하가 된다. 따라서 과부하가 된 서버는 다시 이주받은 서비스 객체를 원래 위치로 이주시키며 이러한 과정은 반복하여 계속하게 되는데, 이 현상을 진동현상이라 한다.

레퍼런스를 클라이언트에게 돌려주고, 클라이언트는 새로운 서버에 다시 요청하여야 한다. 이 방법을 사용하면 클라이언트는 원본 서버로부터 대상 서버로 자신의 요청을 보내야 한다.

```
if (migration flag is set)
   {
   if (client knows the destination server name)
      {
       update server information dictionary;
       return new destination server name;
      }
      else {
         give up this service;
         re-import from trader;
      }
   }
lookup server in the dictionary;
if (Server Information dictionary is null)
   {
    return error (no server for request service is
     known)
   }
send request to the specified object;
if (request service object is migrated)
   then  call re-import at the trader;
if (unknown migration)
   then  system exception;
```

(그림 2) 객체 정보 전송 전의 동기화 알고리즘

```
if (migration flag is set  and  service object
    still remain)
    {
     source server stops requested service;
     source object returns new object reference;
     client re-transmit request to the new
      server;
     supervisor forwards the client request from
      source to the target server;
    }
```

(그림 3) 객체 정보 전송후 객체 복사본 삭제 전의 동기화 알고리즘

세 번째는 <표 1>의 7)에서 8) 사이에서 클라이언트의 요청이 발생하는 경우이다. 원본 객체는 이주된 후 삭제되고, 클라이언트는 삭제된 이전의 객체 레퍼런스를 요청하는 경우이다. 여기서 클라이언트는 더 이상 원본 서버에 존재하지 않는 객체를 가리키게 되므로 즉시 에러 메시지를 받게 된다. 이 경우, 클라이언트는 서비스를 다시 import받아야 한다. 즉, trader에게 다시 필요한 원본 서비스 객체의 새로운 위치를 요청하여야 한다. 이렇게 재시도되는 요청에서 클라이언트는 trader로부터 얻은 새로운 객체 레퍼런스를 사용한다. 이 과정이 (그림 4)에서 표현되었다.

```
if (migration flag is set and object does not
    exist)
{
   the source server returns exception error;
   re-import the service object;
   client request service object to new server;
}
```

(그림 4) 객체 복사본 삭제 후 이주 종료전의 동기화 알고리즘

이와 같은 세가지 경우를 모두 종합하면 서버가 이주를 시작할 때부터 이주를 완료하기 전까지 발생하는 클라이언트의 요청을 모두 해결할 수 있으며, 따라서 동기화 문제는 더 이상 발생하지 않는다.

다음의 제 5장에서는 동기화를 고려하지 않은 부하 분산 알고리즘과 동기화 문제를 해결한 부하 분산 알고리즘의 성능을 모의실험을 통하여 알아보고자 한다.

5. 성능 평가

본 논문에서 제안된 알고리즘의 성능을 평가하기 위하여 K. T. S. Co., Ltd의 SIMAN IV[11]를 사용하였으며 실험 환경은 Pentium III 400MHz, 운영체제는 Windows98이다.

먼저 클라이언트 프로그램의 발생과 서비스 객체 요청은 정규 분포를 따른다고 가정하였다. 그리고 서버에서 사용 가능한 서비스를 trader에게 알려주는 export명령의 실행은 균일 분포(uniform distribution)를 따른다고 가정하였다. 네트워크 지연 시간은 고정값으로 가정하였다.

이 시뮬레이션에서 사용되는 주요 매개 변수는 다음과 같다.

〈표 2〉 시뮬레이션 매개변수 값

매개변수	값
프로그램의 서비스 요구량	1
프로그램 발생 확률	$\lambda = 0.1$
export 실행 비용	3
네트워크 지연	0.5

또한 시뮬레이션에 사용되는 CPU의 종류와 각각의 성능은 iCOMP Index 2.0 Intel 마이크로프로세서 성능 비교표[11]와 SUN사의 CaffeineMark Sun 프로세서 성능 측정 비교표[12]에 근거하여 CPU의 성능 수치를 할당하였다.

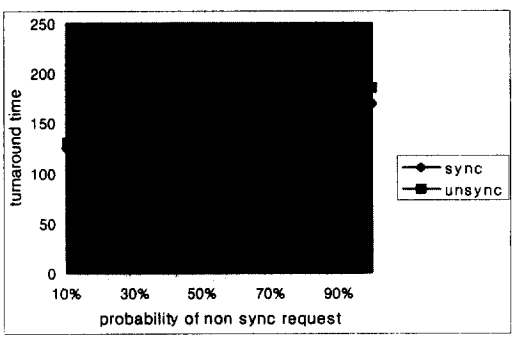
이 모의실험 환경에서, 이주 기간중의 상황을 기술하기 위하여 사용 가능한 서버들 중에서의 서비스 객체 이주와 클라이언트에 의하여 실행되는 명령어들이 사용되었다.

클라이언트는 여러 기계에서 발생할 수 있고, 클라이언트 프로그램을 실행할 수 있다. 이 프로그램은 일련의 두가지 명령으로 구성되어 있는데, 이는 import와 export 명령이다. 클라이언트는 import 명령으로 요청한 서비스 형태의 새로운 객체 서비스를 찾을 수 있고, export 명령으로 이전에 이미 import된 서비스로 요청할 수 있다.

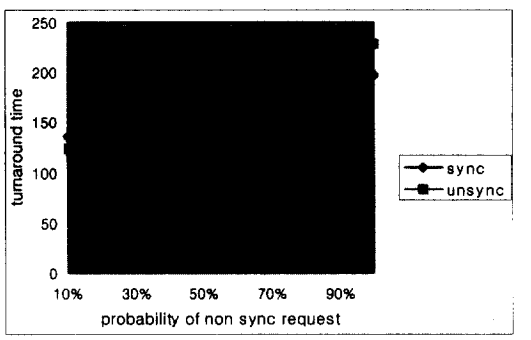
요청에 의하여 필요한 자원들은 전송될 데이터의 양, 서비스 객체, 그리고 요청이 사용하는 CPU 시간에 의하여 모델링된다. 서버에서의 CPU 소비 시간은 서비스 객체 만큼의 수학적인 더미 명령어를 실행함으로써 측정될 수 있다.

본 실험 모델에서는 이주되는 서비스 객체의 크기와 비동기 요청 확률을 증가하면서, 무조건 import 명령을 다시 실행하는 Server Cooling알고리즘과 본 논문에서 제안한 동기화 알고리즘을 포함한 Server Cooling 알고리즘의 성능을 측정하였다.

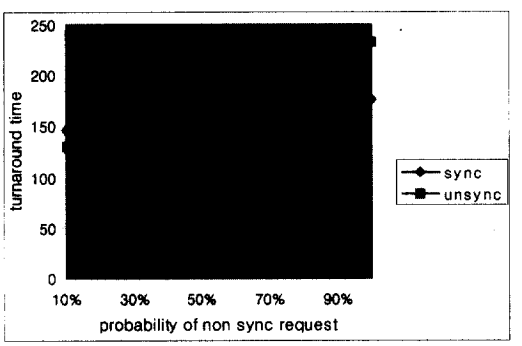
다음 그림은 서비스 객체의 크기를 변경시키면서 평균 반환 시간을 측정한 결과이다.



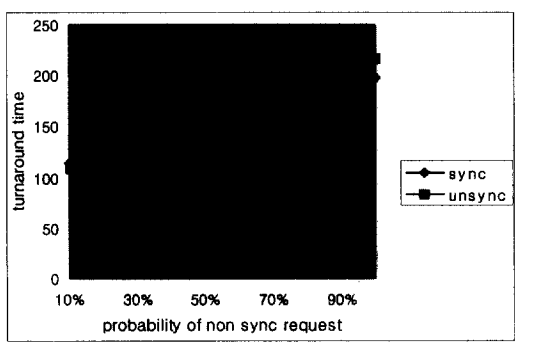
(그림 5) Object size = 5KB



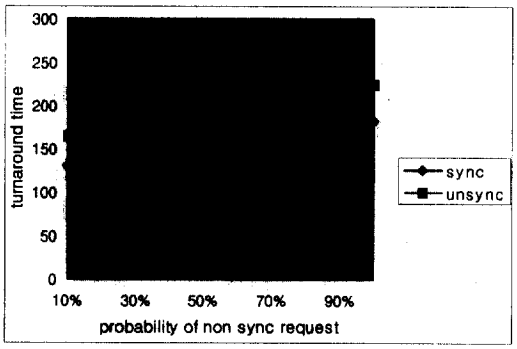
(그림 6) Object size = 10KB



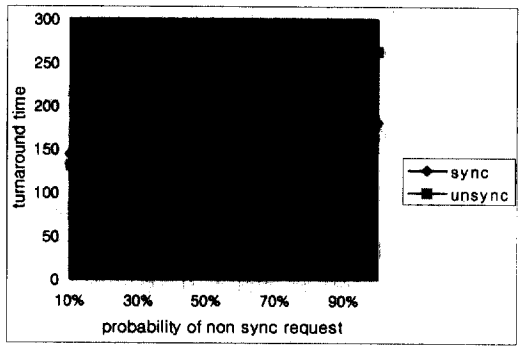
(그림 7) Object size = 20KB



(그림 8) Object size = 50KB



(그림 9) Object size = 100KB



(그림 10) Object size = 1MB

5.1 결과 분석

성능 측정 결과에서는 비동기 요청이 빈번하게 발생할 때 특히 동기화 알고리즘의 성능이 향상됨을 알 수 있다. 특히 서비스 객체 이주의 발생 확률이 40% 이상일 경우 비동기 알고리즘에 비해 훨씬 더 좋은 성능을 보인다.

비동기화 알고리즘이 동기화 알고리즘에 비해 더 좋은 성능을 보이는 구간이 일부 있는데, 이것은 동기화의 오버헤드로 인한 것이다. 그러나 이러한 성능 저하는 너무 미약하므로 무시할 수 있다.

6. 결 론

본 논문에서는 서비스 객체 이주시 발생할 수 있는 동기화 문제를 해결하기 위한 알고리즘을 제안하였다.

분산 시스템에서는 각 노드에 할당되는 부하량이 서로 달라 부하의 불균형이 발생할 수 있다. 이러한 문제를 해결하기 위하여, 과부하 노드에서 저부하 노드로 서비스 객체가 이주되어야 한다. 그러나 이러한 과정에서 서비스 객체의 이주가 완료되기 전에 클라이언트로부터의 요청이 발생한 경우 동기화 문제가 발생한다.

따라서 본 논문에서는 이러한 동기화 문제가 발생할 수 있는 상황을 분석하고, 이들 각각의 상황에서 동기화 문제를 해결하기 위하여 3가지 알고리즘을 제시하였다. 그리고 이 알고리즘의 성능을 검증하기 위하여 모의실험을 통하여 성능을 측정하였다. 본 논문에서 제안한 알고리즘을 사용할 경우 비동기 요청이 40%이상 발생할 경우에는 비동기 알고리즘에 비하여 훨씬 더 좋은 성능을 얻었음을 알 수 있다.

본 논문에서는 이주되는 서비스 객체를 단일 종류로 한정하였지만, 실제 시스템에서의 서비스 객체는 매우 다양하다. 향후 이러한 다양한 서비스 객체에 대한 고려와, 다양한 부하 분산 정책에서 적용이 가능한 동기화 알고리즘에 관한 연구가 계속 진행되어야 할 것이다.

참 고 문 헌

[1] Yuri Breitbart, Radek Vingralek, and Gerhard Weikum, Load Control in Scalable Distributed File Structures, Technical Report 254-95, Univ. of Kenturky, 1995

[2] Peter Scheuermann, Gerhard Weikum, and Peter Zabback, Disk Cooling in Parallel Disk Systems, IEEE Data Engineering Bulletin, 17 : 29-40, 1994

[3] Thomas Schnekenburger, Automatic Load Distribution for CORBA Applications, In Component Users Conference, Munich, Germany, 1996

[4] OMG(Object Management Group), CORBA service s : COSS, Technical Report 95-3-31, March, 1995.

[5] IONA Technologies Ltd., Orbix 2 Programming Guide, Version 2.1, Technical Report, Oct. 1996.

[6] Peter Scheuermann, Gerhard Weikum, and Peter Zabback. Disk Cooling in Parallel Disk Systems. IEEE Data Engineering Bulletin, Vol.17, No.3 :

29-40, Sep. 1994

[7] Andrew S. Tanenbaum. Distributed Operating Systems. Prentice Hall, Amsterdam, The Netherlands, 1995.

[8] Andrzej Goscinski, Distributed Operating System : The Logical Design. Addison Wesley, New South Wales, Australian, 1991.

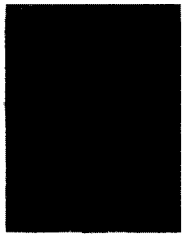
[9] Behrooz A. Shirazi, Ali R. Hurson, and Krishna M. Kavi. Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamitos, California, 1995

[10] Thomas Kunz. The Influence of Different Workload Description on a Heuristic Load Balancing Scheme, IEEE Computer Society Press, Los Alamitos, California, 1995

[11] http : //www.intel.com

[12] http : //www.sun.com

[13] 장성용, 컴퓨터 시뮬레이션, 서울산업대학교 출판부, 1993.



이 준 연

e-mail : jylee@tmic.tit.ac.kr

1990년 중앙대학교 전산과 졸업(학사)

1992년 중앙대학교 대학원 전산과 졸업(석사)

1992년~1994년 (주)Microsoft

2000년 중앙대학교 대학원 전산과 졸업(박사)

2000년~현재 동명정보대학교 멀티미디어공학과 전임강사

김 창 민

e-mail : kimcm@hana.sungkyul.ac.kr

1977년 서강대학교 수학과 졸업(학사)

1985년 중앙대학교 대학원 전산과 (공학석사)

1991년 중앙대학교 대학원 전산과 (공학박사)

1989년~1994년 관동대학교 전산과 조교수

1994년~현재 성결대학교 컴퓨터학부 조교수

관심분야 : 동기화, 부하분산, 분산 시스템, 운영체제