□ 기술해설 □

분산 운영체제 연구동향

포항공과대학교 김치하*

● 목 차 ●

1. 서 론

최근 우리는 정보화 물결에 휩쓸리면서 범람하는 정보 속에서 살고 있다. 이 많은 양의 정보를 처리하기 위해서 과거의 단일 컴퓨터에 의한 중앙 집중형 데이타 처리 시스템에서 분산처리 시스템으로의 전환이 요구되고 있다. 그리고 최근 반도체 기술의 발전에 의해 고 성능 개인용 컴퓨터, 워크스테이션, 일꾼 컴퓨터(server computer) 등의 컴퓨터 시스템들이 보급되고, Ethernet, Token Ring, FDDI 등과 같은 고속 컴퓨터 네트워크들이 개발됨에 따라서 기존의 장애요인들이 해소되어 분산처리 시스템의 개발이 더욱 현실성을 갖게 되었다. 이러한 분산 시스템은 컴퓨터 네트워크에 의해 서로 연결되고, 이를 통한 상호작용을 전제로 하는 무른모(software)를 갖춘 컴퓨터들의 집합으로 정의된다. 그러므로 분산 시스템은 공유 메모리가 없으며, 공유하는 물리적 시계가 존재하지 않는 것으로 특징지을 수 있다. 분산 시스템의 무른모는 컴퓨터들이 그들의 작업을 수행하기 위해 서로 협력하고, 굳은모(hardware)와 무른모의 시스템 자원들을 공유할 수 있도록 해 준다. 분산 시스템의 궁극적 목표는 사용자들이 서로 다른 장소에 위치한 컴퓨터들을 하나의 통합된 컴퓨팅 장비로 인식하여, 시스템 자원의 위치에 관계없이 사용할 수 있는 환경을 제공해 주는 것이다[14].

분산 시스템은 사용자 수준의 분산 시스템이나 또는 분산 운영체제에 의해 실현될 수 있다. 사용자 수준의 분산 시스템은 각기 다른 운영체제를 갖는 이기종의 컴퓨터 시스템들로 구성된 분산 환경을 제공한다. 이것은 운영체제위에서 실행되는 분산 환경을 지원하는 분산 무른모 계층을 통해 이루어 진다. 이러한 분산 시스템의 예는 Open Software Foundation (OSF)의 Distributed Computing Environment (DCE)이다[27]. DCE는 분산 시스템에 필요한 핵심 서비스를 정의하는 규격이다. DCE 서비스는 스레드 (thread), RPCs(Remote Procedure Calls), 디렉토리 서비스, 보안 서비스, 데이타 공유 서비스 등으로 구성된다. 현재 IBM, HP, DEC, DG 등의 대형 컴퓨터 업체에서 DCE를 지원하는 무른모를 제공하고 있다. 반면에 분산 운영체제는 커널 수준에서 분산처리에 관한 지원을 하므로, 통합 설계된 운영체제를 갖는 시스템들로 구성된 분산 시스템을 지향한다. 이 경우 분산 시스템을 구

*종신회원

성하는 컴퓨터들은 모두 동일한 분산 운영체제가 탑재되어 있어야 한다. 분산 운영체제에 관한 분야는 지난 10년간 순수한 연구로부터 시작하여 Amoeba[34], Angel[37], Chorus[30], GLUnix[36], Mach[1], Maruti[31], Masix[10], MOSIX[3], Plan9, QNX[20], Spring System[28], Spring Project[33] 등의 많은 분산 운영체제가 개발되어 왔다. 이들 분산 운영체제의 개발에 있어서 항상 투명성(transparency)이 주요 목표이다. 즉, 투명한 자원공유를 제공하는 것이다. 분산 시스템의 사용자들은 그들의 화일이 어디에 위치해 있으며, 그들의 프로세스가 어느 컴퓨터에서 수행되고 있는지를 알 필요가 없다. 이런 시스템에서 사용자는 네트워크 통신, 작업량의 분배, 자원 할당등과는 관계없이 가상의 단일 컴퓨터 시스템을 사용하는 환상을 갖게 된다. 이 논문은 분산 운영체제에 대한 소개와 연구동향 및 분산 운영체제의 개발사례에 대하여 간략히 살펴본다.

2. 분산 운영체제

초기의 컴퓨터에는 운영체제가 존재하지 않았다. 모든 일은 사용자가 수동으로 패널 스위치를 조작하여 이루어졌다. 그후 사용자가 편리하고 효과적인 방법으로 프로그램을 실행할 수 있는 환경을 제공하는 운영체제가 개발되었다. 이러한 운영체제는 지난 수십년간 그 기능과 성능이 개선되어 왔고, 이것들은 단일 사용자, 멀티유저, 네트워크 운영체제, 그리고 분산 운영체제로 발전하고 있다.

단일 사용자 운영체제는 한명의 사용자만을 위한 단일 컴퓨터 시스템에서 실행되는 운영체제이다. 모든 자원은 한명의 사용자가 사용한다. MS-DOS와 OS/2가 이 부류에 속한다. 멀티유저 운영체제는 단일 컴퓨터 시스템에서 많은 사용자들이 동시에 작업할 수 있도록 지원하는 운영체제이다. 사용자들은 대부분 단말기을 통해 컴퓨터와 연결되어 있다. 이들을 보통 중앙집중형 운영체제라고 부른다. 이 시스템에서는 굳은모, 무른모 자원이 여러명의 사용자에게 적절히 배분된다.

네트워크 운영체제는 사용자가 다른 컴퓨터와 쉽게 통신할 수 있는 기능을 제공하고, 원격사용등록(remote login)과 화일 전송을 수행할 수 있게 해준다. 네트워크 운영체제를 사용하는 사용자는 다른 컴퓨터의 네트워크 주소를 사용하여 그 컴퓨터에 접근할 수 있다. 원격사용등록이나 화일전송의 경우에 상대방 컴퓨터의 네트워크 주소를 네트워크 접속 명령어에 명시하여야 한다. 그러므로 네트워크 운영체제는 제한된 자원공유를 제공하고, 지역적으로 분산되어 있는 이 기종 컴퓨터 시스템들을 연결하기 위하여 사용한다. 그러나 노벨사의 Netware, 마이크로소프트사의 윈도우 NT 일꾼(server) 등과 같은 최근의 상용 네트워크 운영체제들은 디렉토리 서비스를 지원하여 자원의 위치에 대한 투명성을 지원하고 있다.

분산 운영체제는 상호연결된 컴퓨터들을 사용자가 하나의 컴퓨터 시스템으로 인식할 수 있도록 완전한 투명성을 제공한다. 분산 운영체제는 데이타 위치에 대한 투명성을 제공하므로 최근의 네트워크 운영체제보다 한 단계 진일보 한 것이다. 예를 들어서, 여러 프린트 일꾼이 네트워크상에 설치된 분산 컴퓨팅 환경을 생각하자. 사용자가 출력을 요청할때 어떤 프린트 일꾼은 매우 바쁜 반면, 다른 프린트 일꾼은 출력 요청을 기다리고 있을 수 있다. 분산 운영체제는 네트워크상의 한가한 프린트 일꾼를 찾아서 출력 요청을 전달하여 출력 작업의 부하에 대한 균형을 유지한다. 이렇게 분산 운영체제는 사용자에게 분산 시스템을 구성하는 모든 시스템 자원들을 자원들의 실제 위치에 관계없이 하나의 컴퓨터를 사용하듯이 사용할 수 있는 환경을 제공할뿐만 아니라 자원들을 효율적으로 운용할 수 있도록 한다. 그래서 사용자는 자신의 작업이 어느 컴퓨터에서 수행되고 있는지, 자신의 화일이 어느 컴퓨터에 저장되어 있는지 또는 어떻게 프로세스들이 통신을 하는지 알지 못할 것이다. 즉, 입출력 장치, 메모리, 화일, 프로세서들로 구성된 하나의 가상 컴퓨터를 사용하는 셈이다. 이러한 가능을 제공하는 분산 운영체제는 사용자가 입력한 명령어의 해독 및 실행, 시스템 자원의 위치에 투명한 접근서비스 제공, 시스템 자원에 대한

사용자 접근통제, 분산 컴퓨팅 환경의 최적 자원의 할당, 프로세서간 작업량 분배, 프로세스 관리, 자원 관리등을 수행하는 무른모로 구성된다. 분산 운영체제의 개발시 투명성(transparency), 융통성(flexibility), 신뢰성(reliability), 성능(performance), 확장성(scalability) 등의 주요 설계상의 문제들이 함께 고려되어야 한다[35].

3. 분산 운영체제의 설계 문제

3.1 분산 운영체제의 구조

Carnegie Mellon대학의 Mach, Vrije 대학의 Amoeba, Chrous System의 Chrous [30], Maryland 대학의 Maruti등과 같은 대부분의 분산 운영체제들은 모두 마이크로 커널을 기반으로하는 구조를 사용하고 있다. 마이크로 커널은 기존의 UNIX, MS-DOS, VMS등과 같은 단일(monolithic) 커널에 상반되는 운영체제 구조이다. 마이크로 커널은 기존의 단일 커널중 프로세스와 스레드 관리, 메모리 관리, 프로세스간 통신 및 동기, 최소한의 디바이스 관리, 각종 시스템 인터럽트 처리등과 같은 아주 기본적인 부분만을 커널로 유지하고 나머지 기능들은 마이크로 커널의 응용 프로세스로 취급한다. 그림 1은 마이크로 커널구조를 갖는 분산 운영체제 구조의 예를 보여준다. 프로세스 관리자는 프로세스의 생성, 제거에 관련된 기본적인 작업을 수행한다. 스레드 관리자는 스레드의 생성, 제거, 동기화, 순서배정(scheduling)에 관련된 작업을 수행한다. 통신 관리자는 서로 다른 프로세스에 속한 스레드들간의 통신, 그리고 서로 다른 컴퓨터에서 실행되는 스레드들간의 통신과 관련된 작업을 수행한다. 메모리 관리자는 메모리 자원, 메모리 관리 단위(memory management unit), 굳은모 캐쉬(hardware cache)를 관리한다. 그리고 감독자 프로그램(supervisor)은 인터럽트, 시스템 호출, 예외상태(exception)등을 처리한다.

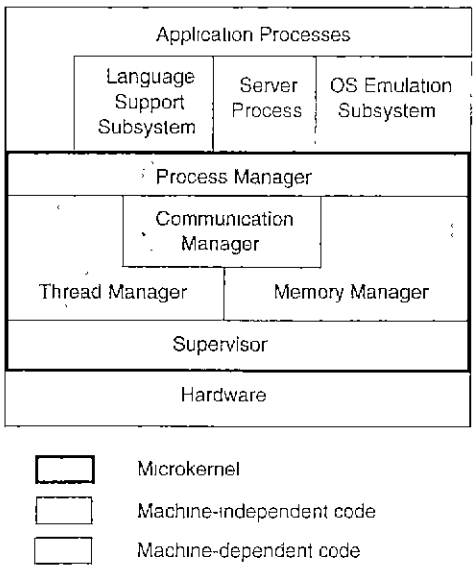

그림 1 마이크로 커널 구조의 예

마이크로 커널의 특징은 커널의 단편화(modularization)로서 커널을 여러개의 단편(module)들로 분리하고, 각 단편들간의 통신은 메세지 전달(message-passing)방식을 사용한다. 이 같은 커널의 단편화는 시스템 기능의 확장성을 향상시켜주며, 컴퓨터 시스템별로 필요로하는 기능만을 갖출 수 있도록 재구성(reconfigurability)을 가능케 한다. 그리고 마이크로 커널중 굳은모에 의존적인 부분을 제외한 대부분이 고급 언어로 작성됨으로서 높은 이식성을 제공한다. 단일 운영체제 커널에 비해 크기가 작고 단편화되어 있으므로 완성후에는 결함이 적고 유지하기도 수월하다. 그러나 단일 운영체제에서 커널모드로 수행되던 부분들이 마이크로 커널구조에서 사용자 영역과 커널모드에서 수행되고, 단편간의 메세지 전달을 처리해야 하는 등의 작업이 수반되므로 성능이 감소된다는 단점이 있다. 마이크로 커널의 개요 및 마이크로 커널 구조를 갖는 운영체제들의 비교는 [13]에 자세히 소개되어 있다.

3.2 프로세스와 스레드

대부분의 단일 운영체제에서 각 프로세스는 하나의 주소공간과 하나의 실행관련 제어(control), 즉 스레드를 갖는다. 스레드는 일련의 처리행위를 나타내는 추상적인 개념이다. 이 같은 프로세스 개념에 기반을 둔 운영체제는 높은 동시성(concurrency)를 요구하는 응용 프로그램이나, 고 성능의 분산 시스템을 구축

하는데 부적합하다. 대부분의 분산 운영체제에서 이에 대한 해결방법으로 하나의 프로세스가 여러개의 스레드들을 가질 수 있게 다음과 같은 프로세스의 확장된 개념을 지원하고 있다. 프로세스는 하나 또는 그 이상의 스레드를 갖는 실행환경(execution environment) 으로 구성된다. 각 스레드는 자신과 관련된 프로그램 계수기(program counter), 통(stack), 레지스터 세트, 자식 스레드(child thread), 상태정보들을 갖는다. 그리고 프로세스와 연계된 실행환경은 자원관리의 한 단위로써 주소공간, 전역변수들, 스레드 목록, 스레드들이 접근하는 자원들 목록, 타이머들, 통신 인터페이스, 세마포(semaphore) 등을 포함한다. 각 스레드들은 실행환경내의 모든 자원들을 공유한다. 여러개의 스레드들을 실행하는 목적은 관련된 작업들간의 동시실행을 극대화하는 것이다. 예를 들면, 여러개의 스레드를 가질 수 있는 일꾼 프로세스(server process)는 고객(client)으로부터의 요청을 처리하기 위해 하나의 스레드를 할당한다. 일꾼이 여러개의 고객 요청을 처리해야 할 경우에 하나의 스레드가 시스템 호출의 결과를 기다리거나, 봉쇄(block) 되었을때 다른 스레드를 실행시킴으로써 요청 처리율을 높여 빠른 서비스를 제공할 수 있다. 고객의 요청 수신시 그것을 처리하기 위한 새로운 프로세스를 생성하는 것보다 스레드를 생성하는 것이 훨씬 경제적이며, 스레드들간의 문맥전환(context-switching)이 프로세스들간의 문맥전환보다 더 경제적이다. 또한 프로세스들간에 자원을 공유하는 것보다 하나의 프로세스내에서 스레드들간에 자원을 공유하는 것이 더 간편하고 효과적이다. 그러나 스레드들은 프로세스내의 모든 자원들을 접근할 수 있으므로, 프로그래밍 잘못으로 한 스레드에 의해서 다른 스레드가 사용하는 데이타가 수정될 경우 에러가 발생할 수도 있으므로 스레드 프로그래밍에 많은 주의를 기울여야 한다.

분산 운영체제들은 스레드 순서배정(thread scheduling)을 위해 preemptive와 non-preemptive 두가지 방법을 사용하고 있다[1, 34]. Preemptive 순서배정에서 스레드가 실행가능한 상태일때에도 다른 스레드를 먼저 실행시키기 위해서 정지(suspend)될 수 있다[4]. Non-preemptive 순서배정에서는 스레드가 스스로 정지하도록 호출(call)하지 않는한 그 스레드의 수행은 계속된다. 즉, 다른 스레드가 현재 실행중인 스레드를 멈추게 할 수 없다. Non-preemptive 순서배정의 장점은 구현하기가 비교적 쉽다는 점이며, 단점은 장시간의 계산만을 수행하는 스레드가 존재할 때 이로 인해 다른 모든 스레드들의 실행이 중지될 수 있다. 그러므로 프로그램의 성능을 높이기 위해서 적절한 스레드 교환이 필요하며, 이를 위해 프로그램 작성시 많은 주의를 기울여야 한다. Preemptive 순서배정은 시간분할 또는 우선순위 방식을 이용하여 스레드의 순서배정을 수행할 수 있다. Preemptive 순서배정 방식이 non-preemptive 방식보다 더 공평하게 스레드들이 CPU를 사용할 수 있게한다.

스레드는 커널 수준에서 구현되기나 사용자 수준에서 구현될 수도 있다. 사용자 수준의 경우에 스레드는 응용 프로그램과 연결(link) 될 수 있는 모음(library) 으로 구현된다. 이 같은 구현 예는 SunOS 4.1 경량 프로세스 꾸러미(LWP : Lightweight Process Package),[1] Mach 운영체제를 위해 개발된 C Thread 꾸러미, IEEE Computer Society가 스레드 설계간의 호환성을 위해 스레드 표준으로 제정한 P Thread 꾸러미, Free Software Foundation에서 개발한 GNU Thread 꾸러미가 있다. 이 경우 커널은 사용자 레벨에서 지원되는 스레드들에 대한 정보가 없으므로 그것들을 독립적으로 순서배정할 수 없다. 대신 스레드 실행시간 모음(run-time library)이 스레드의 순서배정을 담당한다. 그러므로 멀티 프로세서를 이용할 수 없고, 서로 다른 프로세스들에 속한 스레드들을 스레드들에 주어진 실행 우선순위에 따른 순서배정을 할 수 없다. 이 방법이 커널 수준의 구현에 대해 갖는 장점은 하나의 스레드에서 다른 스레드로 제어가 넘어갈 경우에 커널 시스템 호출이 수반되지 않으므로 같은 프로세스내에서 스레드간의 문맥교환이 경제적이고,

1) 스레드의 생성, 제거, 그리고 스레드들간의 동기화에 필요한 기본요소(primitive) 들을 종합하여 스레드 꾸러미(thread package) 이라 한다.

스레드 순서배정 단편(thread scheduling module)이 커널밖에서 구현되므로, 응용 프로그램 고유의 요구에 맞추어서 스레드 순서배정 방법을 조정할 수 있게되어 실시간 응용에 매우 적합하다는 것이다. 최근 이 두가지 접근방법의 장단점 분석 및 이들의 성능을 개선하기 위한 연구가 진행되고 있다[5, 9, 12, 15].

3.3 프로세스간 통신

분산 운영체제는 멀리 떨어져 있는 프로세스(또는 스레드) 들간에도 통신을 할 수있는 안정적이고 효율적인 기능을 제공해야 한다. 분산 시스템과 단일 시스템과의 가장 큰 차이중 하나는 프로세스간 통신이다. 단일 시스템에서의 프로세스들은 공유 메모리로의 입출력을 통해 서로 통신할 수 있으나 분산 시스템은 이와 같은 공유메모리가 없으므로 다른 통신 기법이 사용되어야 한다. 분산 환경에서의 프로세스간 통신은 고객 프로세스와 일꾼 프로세스간의 요청/응답(request/reply) 형태로 가장 잘 모델링되며, 이러한 모델을 기반으로 구현된 기법이 RPC(Remote Procedure Call)이다. RPC는 분산 운영체제에서의 프로세스간 통신을 위해 가장 많이 사용되고 있으며, 또한 범용 고객-일꾼 처리를 위한 기본 통신방식으로도 널리 사용되고 있다.

분산 시스템에서 수행되는 대부분의 프로세스간 통신도 같은 컴퓨터에서 실행되는 프로세스들 사이에서 이루어질 것이라고 예상하고 있다[7]. 분산 시스템에 있어서 이러한 동일 컴퓨터내에서 실행되는 프로세스간 통신의 성능이 매우 중요하므로, 이에 대한 성능개선을 위한 많은 연구가 수행되었다. 동일 컴퓨터내의 프로세스간 통신 성능을 개선하기 방법으로 수정시 복사 방법(copy-on-write)과 공유영역(shared region)을 사용하는 방법이 제안되었다. 같은 컴퓨터내에 있는 프로세스들간에 큰 메세지 전송시의 성능을 개선하기 위해 일꾼 프로세스는 메세지를 수신할 때 메세지의 저장을 위한 영역만을 할당받고, 그 영역으로 메세지를 복사하지 않는다. 그리고 할당받은 영역에 대한 페이지 테이블은 고객 메세지가 저장된 장소를 가르키도록 만든다. 일꾼이 메세지를 읽을 때에는 고객 메세지를 읽고, 메세지 내용을 수정할때만 관련 페이지를 일꾼 메세지 영역으로 복사하여 내용을 수정하고, 일꾼은 복사된 영역의 페이지 주소로 자신의 페이지 테이블을 수정한다. 이러한 방법을 수정시 복사방식이라 하며 Mach와 Chrous에서 사용한다.

하나의 컴퓨터에 있는 프로세스간 RPC의 성능 역시 분산 시스템에서 매우 중요하다. 두개의 서로 다른 컴퓨터에서 실행되는 프로세스간의 RPC는 다음과 같은 단계로 수행된다. 고객 조각(client stub)은 원격절차(remote procedure)에게 전달하는 매개변수(parameter)들을 메세지로 만들고(marshalling), 메세지는 통신 관리자에 의해 일꾼에게 전송된다. 일꾼에 도착된 메세지는 일꾼 조각(server stub)에게 전달되고, 일꾼 조각은 메세지로부터 매개변수들을 분리하여(unmarshalling) 절차(procedure)를 호출하고 매개변수들을 전달한다. 절차의 수행 결과는 다시 메세지로 만들어진후 (marshalling) 고객에게 전송된다. 이 같은 RPC 수행단계에 있어서 RPC 지연시간의 주요 요인들은 매개변수의 메세지화, 메세지로부터 매개변수 분리, 메세지 송수신시 사용자 주소공간과 커널 주소공간간의 메세지 복사, 각 통신규약 계층(protocol layer) 간 메세지 복사, 스레드 순서배정과 프로세스 문맥전환등에 소요되는 시간들로 분석된다. 하나의 컴퓨터에서 실행되는 프로세스간의 RPC는 메세지 전송 부분을 제외한 나머지 RPC 수행단계와 동일하며, 경량 RPC(Lightweight RPC)는 같은 컴퓨터내에서 수행되는 고객와 일꾼간의 RPC 통신의 성능을 공유영역(shared region)을 사용하여 개선하였다[7]. 고객와 일꾼은 공유영역(shared region) 내의 스택을 통하여 호출의 매개변수와 실행결과를 주고 받으므로, 고객의 매개변수는 한번만 복사된다. 반면에 RPC에서는 네번의 복사가 이루어진다 : 1) 고객 조각의 통(stack)에서 메세지, 2) 커널 완충역역(buffer)으로 메세지 복사, 3) 커널 완충영역에서 일꾼 메세지, 4) 일꾼 메세지에서 일꾼 조각의 스택. 하나의 고객 프로세스에 속하는 여러개의 스레드들이 동시에 일꾼을 호출

할 수 있으므로, 공유영역에는 하나 이상의 통들이 존재할 수 있다.

RPC는 두개의 프로세스간의 통신 방식이나, 분산 시스템에서는 두개 이상의 프로세스들이 서로 통신을 해야할 경우가 자주 발생한다. 예를 들면, 하나의 장애 감내형(fault-tolerant) 화일 서비스를 제공하기 위해 서로 협력하는 화일 일꾼(file server)의 집단을 생각해 보자. 고객는 화일 일꾼중의 하나가 고장나는 경우에도 자신이 원하는 서비스를 받을 수 있도록 모든 일꾼들에게 서비스 요청 메세지를 전송해야 한다. 이러한 1 : N 통신을 집단 통신(group communication) 이라 한다. RPC는 각각의 화일일꾼에게 별도의 RPC들을 전송해야 하므로 비효율적이다. 집단통신은 Amoeba, V system, Chorus와 같은 여러 분산 운영체제에서 제공되고 있다. Amoeba는 신뢰성있는 메세지 전달 순서보장(totally-ordered mulicast) 서비스를 제공한다[22]. 집단통신에서 집단이 동적이어서 통신중에 새로운 프로세스가 집단에 가입하거나, 기존의 프로세스가 집단에서 탈퇴하는 경우가 발생할 수 있고, 또 한 프로세스가 동시에 여러개의 집단에 가입할 수 있다. 그러므로 그룹통신 통신규약에는 집단 참가자를 관리하기 위한 집단 참가자 관리 통신규약(group membership protocol)과 함께 집단 통신 서비스를 제공한다.

3.4 작업량 분배(Load Balancing)

분산시스템에서 불규칙적으로 발생하는 작업들로 인하여 어떤 컴퓨터는 작업량이 많은 반면, 다른 컴퓨터들은 작업을 기다리고 있는 경우가 발생할 수 있다. 이러한 경우에, 작업량이 많은 컴퓨터의 작업들을 작업량이 작은 컴퓨터로 옮겨서 처리함으로써 시스템 전체의 성능을 향상시킬 수 있다. 여기서 성능이란 작업시작부터 종료까지 경과된 시간을 나타내는 각 작업의 평균 응답시간(average response time)을 의미한다. 작업량 분배의 목적은 모든 컴퓨터들이 처리하는 작업량을 비슷하게 유지함으로써 작업의 평균응답 시간을 줄이는 것이다. 작업량 분배는 분산 운영체제가 네트워크 운영체제와 구별되는 주요 특징이며, 분산 운영체제의 가장 중요한 기능중의 하나이다.

작업량 분배 알고리즘은 작업량 정보의 유지와 전파, 프로세스 이동(process migration) 시점의 결정, 이동시킬 프로세스의 선택, 프로세스의 중지, 프로세스의 캡슐화, 프로세스가 이동될 컴퓨터의 결정, 캡슐화된 프로세스의 전송, 프로세스의 상태복구등을 고려하여야 한다. 작업량 분배 알고리즘에 있어서 가장 중요한 문제중의 하나는 어떻게 컴퓨터의 작업량 지수(load index) 를 나타내는가 하는 것이다. 지금까지 연구된 작업량 지수로는 CPU 대기열(queue) 의 길이, 평균 CPU 대기열 길이, 가용한 메모리 양, 프로세스간 문맥전환율, 시스템 호출(system call) 발생율, CPU 활용율등이 있다. 이들중 가장 효과적인 작업량 지수는 CPU 대기열 길이이다[24]. 또한 작업량 지수를 측정하는 방식은 최소의 오버헤드가 소요되며, 효과적이어야 한다.

작업량 지수 정보는 전이할 프로세스가 발생했을 때마다 작업량 정보를 요청하거나, 주기적으로 요청하거나, 또는 시스템의 상태가 바뀔 때마다 다른 컴퓨터에게 작업량 정보를 전파함으로써 수집될 수 있다. 정보수집으로 인해 발생되는 오버헤드가 작업량 분배의 장점을 약화시켜서는 안되므로 이에 대한 효율적인 수집방법은 중요한 연구과제 중의 하나이다.

작업량 분배 알고리즘은 정적(static), 동적(dynamic), 또는 적응적(adaptive)일 수 있다. 분산 시스템이 N 개의 컴퓨터로 구성되어 있을 때 정적 알고리즘은 시스템에서 발생하는 i 번째 작업을 (i mod N) 번째 컴퓨터에 할당하거나 미리 결정된 Pi 확률로 i 번째 컴퓨터에 할당한다. 확률은 모든 컴퓨터에 대해 동일하거나, 또는 각 컴퓨터에서의 작업처리율에 따라 결정될 수 있다. 적응적 알고리즘은 시스템에 작업량이 생기는 형태에 따라서 그 작업량 형태에 가장 적합한 작업량 분배 알고리즘을 적용하는 방법이다.

동적 알고리즘에서는 프로세스 이동이 결정된 시점에 각 컴퓨터 작업량 지수에 대한 정보를 이용하여 가장 작업량이 적은 시스템에게 그 작업을 할당한다. 프로세스의 이동 시점을 결정하는 방법에는 프로세스가 생성될때 마다

이동시키는 방법, 작업량 지수가 한계값(threshold)을 초과할 때 프로세스를 이동시키는 방법, 그리고 다른 컴퓨터와의 작업량 차이가 클 때 프로세스를 이동시키는 방법등이 있다. 최근의 작업량 분배에 관한 연구들은 동적 알고리즘에 집중되어 있다.

프로세스 이동이 결정되면 이동시킬 프로세스를 결정해야 한다. 이동시킬 프로세스는 수행시간이 길고, 프로세스 전송에 작은 부하가 소요되어야 한다. 이동시킬 프로세스 결정후, 그 프로세스의 수행을 정지시켜야 한다. 이때 프로세스는 그 수행이 정지된 후 다른 컴퓨터에서 다시 실행이 시작될 수 있어야 하므로 프로세스가 시스템 호출중에 있는 때나 RPC를 수행하고 있을 때 프로세스를 정지시켜서는 안된다.

프로세스가 정지된 후 다른 컴퓨터로 이동하기 전에 그 프로세스에 관한 모든 실행상태 정보를 모으고, 적당한 구조로 캡슐화하여야 한다. 캡슐화된 프로세스를 다른 컴퓨터로 전송하는 것은 작업량 분배 알고리즘에 있어서 가장 많은 오버헤드가 발생하는 부분이다. 그 이유는 프로세스의 주소공간이 함께 전송되어야 하기 때문이다.

작업량 분배의 마지막 단계는 이동된 프로세스가 다시 실행할 수 있도록 복구시켜주는 것이다. 이러한 복구작업은 프로세스가 요구하는 자원을 할당하고, 이동된 프로세스의 모든 메모리 참조들을 이동된 컴퓨터의 메모리 주소로 다시 바인딩(binding) 시키고, 그리고 메세지의 전달(forwarding) 작업들을 포함한다. 이러한 커널에 의한 재바인딩(rebinding) 은 투명하게 수행되어야 한다. 프로세스가 원래 실행되던 컴퓨터에게 그 프로세스로 전달되어야 할 메세지가 도착하면, 그 메세지는 프로세스가 이동된 컴퓨터로 다시 전달되어야 한다. 작업량 분배 알고리즘에 대한 더 자세한 사항은 [32]을 참조하기 바란다.

3.5 분산 공유 메모리 (Distributed Shared Memory)

분산 공유 메모리는 물리적으로 메모리를 공유하지 않는 컴퓨터들에서 수행되는 프로세스들간에 데이타를 공유하기 위해서 사용되는 추상적 모델이다. 분산 공유 메모리를 지원하는 시스템에서 프로세스는 자원을 그 위치와 관계없이 투명하게 사용할 수 있으며, 실제 물리적인 메모리는 분산되어 있으나 마치 프로세스들이 하나의 공유메모리상에서 데이타를 공유하는 것과 같은 환상을 갖게 한다. 분산 공유 메모리는 프로세스간 데이타 공유를 위한 동기화 방법 및 데이타의 일관성(consistency)을 유지하기 위한 관련 통신규약을 포함한다.

메세지 전달을 이용하는 프로그래밍에 대하여 분산 공유 메모리를 이용하는 프로그래밍은 다음과 같은 여러가지 장점을 갖고 있다. 메세지 전달 프로그램들보다 공유 분산 메모리의 프로그램은 보통 짧고, 프로그래밍이 쉽다. 분산 공유 메모리을 이용하여 서로 다른 컴퓨터에 있는 프로세스들은 쉽게 상태 정보(state information)을 공유하거나, 복잡한 데이타 구조들을 유지할 수 있다. 또한 공유 데이타는 프로세스들의 종료 후에도 시스템에 남아 있게 된다. 메세지 전달을 사용할 때와는 달리 사용할 데이타의 위치에 관계없이 응용 프로그램을 작성할 수 있다. 마지막으로 공유 메모리를 통한 프로그래밍은 잘 알려져 있다.

분산 공유 메모리는 다음 두가지 방식에 의해 구현될 수 있다 : 페이지 분할 분산 공유 메모리 방식(page-based distributed shared memory) 과 모음(library)를 이용해 구현한 분산 공유 메모리 방식(library-based distributed shared memory). 페이지 분할 분산 공유 메모리 방식은 모든 프로세스들의 주소공간들이 가상 메모리에 대응(mapping)된 형태로 구현되며, 가상 메모리 관리는 분산 운영체제 커널에 의해 제공된다. 모음을 이용해 구현된 분산 공유 메모리에서는 Orca와 Linda와 같은 언어에 의해 분산 공유 메모리를 제공한다 [11]. 데이타의 공유는 가상 메모리 시스템을 이용하여 수행하는 것이 아니고, 언어 모음에 제공되는 기능들간에 메세지 교환을 통하여 이루어진다. 프로세스가 분산 공유 메모리에 있는 데이타에 접근할 때에 컴파일러가 삽입한 모음 호출이 실행되고, 필요할 때 마다 모음간의 통신을 통해 메모리 내용의 일관성을 유지한다.

분산 공유 메모리의 주요 설계문제에는 분산 공유 메모리 구조, 동기화, 공유 메모리의 세분성(granularity), 메모리 일관성 모델, 메모리 수정 통신규약등이 있다. 분산 공유 메모리는 일련의 바이트, 공유 데이타들의 집합, 또는 오브젝트들의 집합으로 구성될 수 있으며, 이에 따라 분산 공유 메모리는 다음의 세가지 형태로 분류된다 : 페이지 분할 분산 공유 메모리[25], 공유변수(shared-variable) 분산 공유 메모리[6, 8], 객체 지향(object-based) 분산 공유 메모리. 분산 공유 메모리에서 데이타 공유 단위의 크기가 너무 작으면, 이에 관한 정보들을 유지하기 위해 많은 테이블이 필요하게 되므로 시스템 자원이 낭비된다. 공유 데이타에 대한 동시 접근은 동기화 방법에 의해 순서적 접근이 이루어 지도록 조정된다. 공유 단위의 크기가 조금 커지면 공유 데이타에 대한 프로세스들간의 빈번한 접근충돌로 인하여 접근을 기다리는 시간이 늘어나게 된다. 그러므로 시스템 자원의 효율적 활용과 공유 메모리 접근시간을 고려하여 적절한 메모리의 세분화(granularity)가 이루어져야 한다. 그리고 분산 공유 메모리에서 데이타 일관성을 유지할 것인가 아닌가 또는 일관성을 유지하기 위하여 어떤 방법을 사용할 것인가 등을 결정하여야 한다. 분산 공유 메모리의 일관성 모델로 엄격한 일관성(strict consistency), 순차 일관성(sequential consistency), causal consistency, PRAM (Pipelined RAM) consistency, 프로세서 일관성(processor consistency), 약한 일관성(weak consistency), release consistency, entry consistency와 같은 엄격한 일관성 부터 약한 일관성 모델까지 다양한 모델과 각 모델이 보장해야하는 일관성를 제공하는 통신규약들이 존재한다[17, 21, 26, 2, 16. 18, 8, 23]. 분산 공유 메모리에 대한 성능 모의실험 연구결과는 약한 일관성 모델이 엄격한 일관성 모델보다 좋은 성능을 얻을 수 있음을 보여준다[19, 38]. 메모리 일관성 모델에 대한 여러가지 문제를 [29]에서 다루고 있다.

4. 분산 운영체제의 개발동향

표 1은 현재까지 개발된 분산 운영체제들의

표 1 분산 운영체제의 특징 비교

분산 운영체제	개 발 기 관	특 징
Amoeba	Vrije Univ.	– 마이크로 커널 운영체제 – 그룹통신, 메세지 전달순서 보장
Angel	City Univ. of London	– 마이크로 커널 운영체제 – PC 기반 분산 운영체제 – 단일 주소공간 운영체제
Chorus	Chorus Systems	– 마이크로 커널 운영체제 – 실시간 운영체제 – 객체 지향 운영체제
GLUnix	Univ. of California at Berkeley	– UNIX에 탑재되는 운영체제
Mach	Carnegie Mellon Univ.	– 마이크로 커널 운영체제 – 실시간 운영체제 – 고장 감내 서비스 지원
MASIX	Institute Blais	– 마이크로 커널 운영체제 – 분산 다중특성 (Multi-personality) 운영체제 (UNIX, DOS, OS/2등 동시 실행지원)
QNX	QNX Software Systems	– 마이크로 커널 운영체제 – PC 기반 실시간 운영체제
Spring System	SUN Microsystems	– 마이크로 커널 운영체제 – 실시간 운영체제
Spring Project	Univ. of Massachusetts	– 마이크로 커널 운영체제 – 실시간 운영체제

특징을 간략하게 보여준다. 기존의 분산 운영체제들은 다음과 같은 세가지 개발동향을 갖고 있다. 첫째 대부분의 분산운영체제들은 마이크로 커널 구조로 개발되고 있는 추세이며, 둘째, 분산 운영체제가 객체 지향 방식으로 설계되는 경향을 볼 수 있다. 그리고 셋째, 고장 감내 기능과 실시간 응용을 지원하도록 개발되고 있다. Amoeba는 유일하게 그룹통신을 지원하며, 전송 메세지의 순서보장 서비스도 제공하고 있다. 이러한 그룹통신은 분산 시스템에 중요한 서비스로 인식되고 있으므로 다른 분산 운영체제들도 점차 그룹통신에 대한 서비스를 제공하게 될 것으로 예상된다. Angel은 단일 주소공간 운영체제(single address-space operating system) 이며, 이것은 분산 시스템을 구성하는 컴퓨터의 물리적 메모리의 주소들의 합이 가상 메모리를 형성하는 매모리 관리 방식이다. Angel은 이 방법의 장점인 공유 데이타의 접근이 쉽다는 점을 이용하여 RPC 성능을 개선하였다.

이외에도 각 시스템에 따라 필요한 기능을 쉽게 갖추고, 또 확장할 수 있는 운영체제(extensible distributed operating system)의 구조에 관한 연구도 활발하게 진행중이다.

5. 결 론

본 논문에서는 분산 운영체제의 개념, 주요 설계문제와 연구동향, 그리고 분산 운영체제의 개발동향에 대하여 기술하였다. 분산 시스템은 우리가 구축하고, 사용할 컴퓨터 시스템의 지표가 되고 있다. 분산 시스템은 사용자에게 단일 컴퓨터의 영역에서 벗어나 네트워크를 통해 연결된 모든 컴퓨터들과 그들이 갖고 있는 모든 자원들을 위치와 복잡한 구성에 투명하게 사용할 수 있는 환경을 제공하는 것을 목표로 하고 있다. 분산 운영체제는 그 목표를 달성하기 위해 필요한 모든 서비스를 제공해야 한다. 이러한 분산 운영체제에 대한 연구는 80년대 초부터 시작하여 많은 연구가 수행되어 왔으며, 최근에는 지난 10 여년간의 연구결과를 이용하여 많은 분산 운영체제들이 개발되었다. 이들 분산 운영체제들은 공통적으로 마이크로 커널구조, 객체 컴퓨팅 모델, 메세지 송수신에 의한 프로세스간 통신, 네트워크에 투명한 가상 메모리 관리, 객체 지향 프로그래밍 방법등을 채택하고 있다. 또한 다중 스레드 프로세스 모델(multi-threaded process model)을 공통적으로 채택하고 있다. 현재 분산 운영체제에 장애 감내 기능 및 실시간 응용 프로그램을 지원하기 위한 기능확장 및 성능개선에 대한 연구가 활발하게 수행되고 있다. 그리고 최근 응용 무른모에 대하여 지속적으로 제시되는 새로운 사용자 요구사항과 계속 발표되는 고속 컴퓨터들로 인한 굳은모 플랫폼의 변화가 발생하고 있다. 분산 운영체제는 이러한 급격한 변화를 맞이하여 새로운 굳은모 플랫폼으로 쉽게 이식이 가능하고, 새로운 사용자 요구를 쉽게 수용할 수 있는 무른모 구조 및 기능 확장방안을 적극 모색하고 있다. 향후 각 대학, 업체, 그리고 연구소들은 이상적인 분산시스템의 구축에 필요한 보다 완벽한 기능을 갖춘 분산 운영체제를 개발을 위해 더 많은 연구와 개발을 추진해 나아갈 것으로 예상된다.

참고문헌

[1] M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, and M. Young, Mach : A New Kernel Foundation for UNIX Development, *Proc. Summer 1986 USENIX Conf.*, pp. 321-328.

[2] M. Ahamad, R.A. Bazzi, R. John, P. Kohli, and G. Neiger, The Power of Processor Consistency, *Tech. Report GIT-CC-92/34*, College of Computing, Georgia Inst. of Technology, March 1993.

[3] A. Barak, S. Guday, and R. Wheeler, The MOSIX Distributed Operating System, Load Balancing for UNIX, *Lecture Notes in Computer Science*, Vol. 672, Springer-Verlag. 1993.

[4] A. Barak, O. Laden, and Y. Yarom, The NOW MOSIX and its Preemptive Process Migration Scheme, *Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments*, Vol. 7, No.

2, pp. 5-11, Summer 1995.

[5] F. Bellosa, Techniques for building a fast threads package on NUMA architectures, *Tech. Report TR-I4-94-06*, University of Erlangen-Nuernberg, Germany, 1994.

[6] J.K. Bennett, J.K. Carter, and W. Zwaenepoel, Munin : Distributed Sared Memory Based on Type-Specific Memory Coherence, *Proc. 2nd ACM Symp. on Priciples and Practice of Parallel Programming*, 1990, pp. 168-176.

[7] B.N. Bershad, T.E. Anderson, E.D. Lazowska, and H.M. Levy, Lightweight Remote Procedure Call, *ACM Trans. on Computer Systems*, Vol. 8, No. 1, pp. 37-55, 1990.

[8] B.N. Bershad, M.J. Zekauskas, and W.A. Sawdon, The Midway Distributed Shared Memory System, *Proc. IEEE COMPCON Conf.*, 1993, pp. 528-537.

[9] R.D. Blumofe and C.E. Leiserson, Space-Efficient Scheduling of Multithreaded Computations, *Proc. of 25th Annual ACM Symposium on the Theory of Computing*, San Diego, CA, USA, May 1993, pp. 362-371.

[10] Ry Card, Hubert Le Van Gong et Pierre-Guillaume Raverdy, Design of the Masix Distributed Operating System on top of the Mach Microkernel, *Proc. of the Internal. Conf. on Applications in Parallel and Distributed Computing, Caracas*, Venezuela, April 1994.

[11] N. Carriero and D. Gelernter, Linda in Context, Commun. of the ACM, Vol. 32, April 1989, pp. 444-458.

[12] C.-S. Chen and C.-C. Tseng, Integrated Support to Improve Inter-thread Communication and Synchronization in a Multithreaded Processor, *Proc. of the Internat. Conf. on Parallel and Distributed Systems*, 1994.

[13] 최효진, 임기욱, 마이크로 커널기술에 대한 고찰, 정보과학회지, 제 12 권, 제 10 호, 1994년 11 월, pp. 16-26.

[14] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems : Concepts and Design*, 2nd Ed.. Addison-Wesley, 1994.

[15] J.B. Dennis and G.R. Gao, *Multithreaded Computer Architecture : A Summary of the State of the Art*, Kluwer Academic, 1994.

[16] M. Dubois, C. Scheurich, and F.A. Briggs, Memory Access Buffering in Multiprocessors, *Proc. 13th Ann. Interal. Symp. on computer Architecture*, 1986, pp. 434-442.

[17] M. Dubois, C. Scheurich, and F.A. Briggs, Synchronization, Coherence, and Event Ordering in Multiprocessors, *IEEE Computer*, Vol. 21, Feb. 1988, pp. 9-21

[18] K. Gharachorloo. et al., Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, *Proc. 17th Ann. Interal. Symp. on computer Architecture*, 1990. pp. 15-26.

[19] K. Gharachorloo, A. Gupta, and J. Hennessy, Performance Evaluation of Memory Consistency Models for Shared Memory Multiprocessors, *ACM SIGPLAN Notices*, Vol. 26, No. 4, April 1991, pp. 245-257.

[20] D. Hildebrand, POSIX for realtime embedded systems, *Computer Design*, February 1995.

[21] P.W. Hutto and M. Ahamad, Slow Memory : Weakening Consistency to Enhance Concurrency in Distributed Shared Memories. *Proc. 10th Internat. Conf. on Distributed Computing Systems*, 1990, pp. 302-311.

[22] M.F . Kaashoek and A.S. Tanenbaum, Efficient reliable group communication for distributed systems, *Tech. Report IR-295*, Dept. of Math and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, May 1993.

[23] P. Keleher. A.L. Cox, and W. Zwaenepoel, Lazy Release Consistency for Software Distributed Shared Memory. *SIGARCH Computer Architecture News*, Vol. 20, No. 2. May 1992.

[24] R. Kunz, The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme, *IEEE Trans. on Software Eng.*, Vol. 17, No. 7, July 1991, pp. 725-730.

[25] K. Li and P. Hudak, Memory Consistency in Sared Virtual Memory Systems, *ACM Trans. Comput. Systems*, Vol. 7, Nov. 1989, pp. 321-359.

[26] R.J. Lipton and J.S. Sandberg, Pram : A Scalable Shared Memory, *Tech. Report CS-TR-188-88*, Dept. of Computer Science, Princeton Univ., Sep. 1988.

[27] M.D. Millikin, *DCE : Building the Distributed Future*, *Byte*, June 1994, pp. 125-134.

[28] J. Mitchell, J. Gibbons, G. Hamilton, P. Kessler, Y. Khalidi, P.Kougiouris, P. Madany, M. Nelson, M. Powell, and S. Radia, An Overview of the Spring System, *Proc. of COMPCON*, Feb. 1994.

[29] D. Mosberger, *Memory Consistency Models*, *Tech. Report TR93-11*, Dept. of Computer Science, Univ. of Arizona, 1993.

[30] M. Rozier, et al., Chorus Distributed Operating System, *Computer Systems Journel*, Dec. 1988, pp. 305-370.

[31] M. Saksena, J. da Silva and A.K. Agrawala, *Design and Implementation of Maruti-II : Principles of Real-Time Systems*, Sang Son (ed.), 1994.

[32] N.G. Shivaratri and M. Singhal, Load Distributing for Locally Distributed Systems. *IEEE Computer*, Vol. 25, No. 12, Dec. 1992, pp. 33-44.

[33] J. Stankovic and K. Ramamritham, The Spring Kernel : A New Paradigm for Real-Time Systems, *IEEE Software*, Vol. 8, No. 3, pp. 62-72, May 1991.

[34] A.S. Tanenbaum, M.F. Kaashoek, R. van Renesse, and H. Bal, The Amoeba Distributed Operating System-A Status Report, *Computer Communications*, Vol. 14, No. 4, July-Aug. 1991, pp. 324-335.

[35] A.S. Tanenbaum, *Distributed Operating Systems*, Prentice-Hall, 1995.

[36] A.M. Vahdat, D.P. Ghormley, and T.E. Anderson, Efficient, Portable, and Robust Extension of Operating System Functionality, *Tech. Report CS-94-842*, Dept. of Computer Science, Univ. of California at Berkeley, 1994.

[37] T. Wilkinson, K. Murray, Extensible, Flexible and Secure Services in Angel, a Single Address Space Operating System, *1st Internat. Conf. on Parallel Architectures and Algorithms*, April 1995.

[38] R.N. Zucker and J-L. Baer, A Performance Study of Memory Consistency Models, *SIGARCH Computer Architecture News*, Vol. 20, No. 2, May 1992.

김 치 하

1974 서울대학교 전자공학과 졸업
1974~1981 국방과학연구소 선임연구원
1984 Univ. of Maryland 전자계산학과 석사
1986 Univ. of Maryland 전자계산학과 박사
1986~1990 State Univ. of New York at Buffalo 전자계산학과 조교수
1990~1992 포항공과대학 전자계산학과 조교수
1992~현재 포항공과대학 전자계산학과 부교수
관심분야 : 컴퓨터 시스템, 네트워크, 분산시스템, 성능평가