분산환경을 위한 Sample Distributed Virtual Machine 설계 및 구현

양 일 등[†] · 이 석 희[†] · 김 성 열[††]

요 약

고성능 컴퓨터들의 대량 보급으로 인해 다양한 종류의 분산 자원이 주위에 산재해 있다. 이러한 분산 자원을 이용하는데, 각 분야별 시스템들의 개념, 모듈, 운영 및 관리를 배제하고 사용할수 있다면 최소한의 비용으로 분산 자원을 사용할수 있을 것이다. 이에 '분산환경에 적용될수 있는 DVM 및 DESPL의 제안'을 통해 제안된 내용중 그 일부를 수용하여 SDVM을 설계 및 구현한다. SDVM을 이용하는 사용자는 분산 환경의 개념없이 최소한의 비용으로 분산된 자원을 사용할 수 있다.

A Design and Implementation of Sample Distributed Virtual Machine for Distributed Environment

Il-Deung Yang[†] · Seok-Hee Lee[†] · Soeng-Ryeol Kim[††]

ABSTRACT

By large quantity supply of high efficiency computers, various kind of distributed resource has been extravagant around. Use this distributed resource, if exclude concept of systems by each field, module, operation and administration then we can use it into minimum expense. Hereupon, accommodate some among contents that is proposed through 'Proposal of DVM and DESPL that have apply in distributed environment' and designs and implements SDVM. The user who use SDVM can use into minimum expense without concept of distributed environment.

키워드 : 분산시스템(Distributed System), 분산처리(Distributed Process), 분산환경(Distributed Environment)

1. 서 론

우리는 컴퓨터 시스템을 이용하여 단순 지식 정보부터 복잡한 산술 자료등, 수많은 데이터를 처리한다. 그러한 데이터 중에는 단 시간안에 결과가 도출되는 것도 있지만 보다 많은 시간을 요하는 데이터 처리도 있다. 이에 날씨 데이타 계산에 사용되는 고성능 슈퍼컴퓨터와 같이 고가의 전용 시스템을 사용하거나 저가의 PC들을 유기적으로 연결하여 동영상 데이터 처리를 수행하는 곳도 있다. 이러한 곳들은 개념이 숙지된 운영자와 개발자, 전용 시스템, 장소, 비용 등을 소비하면서 처리를 수행한다. 좀더 깊이 보면 분야별로 병렬처리, 클러스터, 분산시스템, 분산객체, P2P, 다중에이전트 혹은 그리드 등이 있고 이러한 시스템들에서 프로그램을 제작하기위해 C, C++, JAVA등, 기본적인 프로그래밍 언어와 MPI, SOAP, CORBA, DCOM, EJB, RMI, RPC, P2P 등, 각 분야별로 개발 방법들과 모듈들을 숙지해야 한다. 이러한 시스템에서 동작하는 프로그램은 특정 하드웨어 및 운영체제에 종속적이고 다른 시스템과 연계가 쉽지 않으며, 프로그램을 실행하기 위한 정적인 자원 확충이 있어야 하고, 그리드 등을 제외한 시스템들은 컴퓨터의 특정 자원 공유만을 대상으로 한다. 이것은 사용자가 일반 프로그래밍의 개념만을 가지고 특정 시스템에 맞는 프로그램을 작성할 수 없음을 의미하며, 작성했다 하더라도 운영장소나 하드웨어가 있어야 한다.

따라서 이러한 문제를 해결하기 위해 '분산환경에 적용될 수 있는 DVM 및 DESPL의 제안'을[1] 통해 제안된 내용에 따라 그 일부를 수용한 SDVM(Sample Distributed Virtual Machine)을 설계 및 구현한다. SDVM은 분산된 각 자원들 즉 CPU, MEMORY, DISK, DEVICE등을 관리하며 각 시스템의 여유 자원을 사용자가 사용할 수 있도록 제공한다. SDVM 환경에서 프로그래머는 일반 프로그래밍의 개념을 가지고 분산된 자원을 이용할 수 있는 프로그램을 작성할 수 있으며, 작성된 프로그램은 특정 하드웨어 및 운영체제에 종속적이지 않으며, 시스템을 운영하는데 정적인 자원

† 준 회 원 : 청주대학교 대학원 컴퓨터정보공학과
†† 정 회 원 : 청주대학교 컴퓨터정보공학과 교수
논문접수 : 2004년 1월 26일, 심사완료 : 2004년 8월 11일

확보가 필요치 않다. 일반 사용자는 'Hello World'와 for문장을 이해했다면 분산된 자원을 사용할 수 있는 것이다.

2. 분산 시스템들과 SDVM 비교

병렬처리, 클러스터, 분산객체, 다중 에이전트 및 객체 관리 시스템, P2P, 그리드 등 분산 시스템들은 분산되어 있는 자원을 부분적으로 제공한다. 분산자원의 제공 범위는 각 시스템들마다 조금씩 다르며, 가장 많이 공유되는 자원은 CPU이다. 이것은 컴퓨터에 존재하는 자원중에서 높은 비중을 차지하지만 저장 장치 같은 자원 또한 사용된다면 사용자는 보다 많은 저장 공간을 확보할 수 있다. 이러한 자원의 분포는, 작게는 주위의 컴퓨터들의 자원을 포함하며 크게는 전 세계의 컴퓨터들의 자원을 포함한다. 하지만 현재의 분산된 자원을 이용하기 위해서는 기본 프로그래밍 개념 및 분산 환경에 대한 이해가 필요하고 특정 시스템의 특성 및 프로그래밍 모듈에 대해 숙지가 필요하다. 프로그램 작성 및 실행시에는 정적인 컴퓨터 자원과 이를 운영할 수 있는 비용이 발생한다.

따라서 내가 원하는 분산된 자원을 사용할 때 사용자는 반드시 시스템의 개념과 라이브러리 및 모듈을 숙지해야 작성할 수 있다. 이러한 개념 및 모듈을 감추어 주며 이 기종 하드웨어 및 운영체제 독립적이고 일반 프로그래밍 개념만을 가지고 프로그램을 작성하여 분산된 자원을 사용할 수 있다면 자신의 컴퓨터의 자원이 사용되지 않을때 다른 사람이 사용할 수 있고 자신 또한 자신의 자원을 제공할 수 있을 것이다. 그러면 특정 시스템을 위한 공간 및 관리에 드는 비용이 없을 것이고 사용자는 손쉽게 자신이 원하는 작업을 분산된 자원을 이용하여 처리할 수 있을 것이다. 이를 비교하면 <표 1>과 같다.

〈표 1〉 현재 구현 시스템들과 SDVM의 비교

	현재 구현 시스템들 (P2P제외)	SDVM
기본 프로그래밍 언어	C, C++, CORBOL, JAVA, FORTRAN, ADA	JAVA
구성형태	라이브러리	실행가능프로그램
분산자원개념	필 요	불필요
지원자원	CPU(그리드제외)	CPU, 파일 기타장치(예정)
사용자 프로그램 서버/클라이언트 코드	필 요	불필요

3. SDVM 설계 및 구현

3.1 시스템 구성

SDVM은 NN(normal node)와 SN(service node)가 존재한다. NN은 DVM이 실행되는 컴퓨터이며 SN은 DVMS가 실행되는 컴퓨터이다. DVM은 요청자의 컴퓨터에서 수행되며 작성된 프로그램을 수행하는 역할을 하고 다른 컴퓨터에 있는, SN에 요청하여 작업을 처리한다. DVMS는 해당 컴퓨터의 자원을 관리하며 NN의 요청이 있을 경우 자신의 자원을 제공해 준다. 하나의 컴퓨터는 어느 프로그램을 실행하는가에 따라 NN 및 SN이 될 수 있고 두개를 동시에 실행한다면 동시에 NN 및 SN이 될 수 있다. 그러나 중요한 개념은 네트워크에 SN이 꼭 존재해야 하는 것은 아니며 SN이 존재하더라도 NN의 자원이 더 많다면 프로그램은 NN을 통해서 수행된다. 이것은 SDVM 환경에서 프로그램을 수행하기 위한 전용 시스템을 운용하는 것이 아니고 가용자원이 있는 컴퓨터의 자원을 사용하는 것이기 때문에 해당 컴퓨터가 특정한 작업을 자신의 컴퓨터를 위해 수행한다면 다른 NN의 요청을 수행할만한 자원이 없기 때문이다. 이러한 모든 과정은 SDVM에서 처리해 주며 사용자에게는 이것에 대한 프로그래밍 작업을 요구하지 않으며 내부적으로 사용자의 프로그램에 추가적인 SDVM 운영 코드가 삽입되지 않는다. 이것은 DVM에서 사용자의 코드를 수행하면서 적절히 판단하기 때문이다. SDVM을 정의하면 다음과 같다

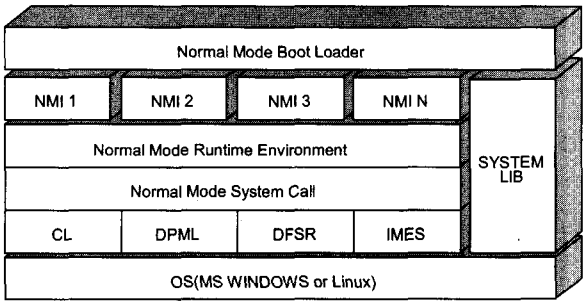
$$SDVM := DVM + DVMS + CLASSLIB$$

3.2 SDVM 설계

3.2.1 DVM 설계

DVM은 NN상에 존재하며 필요하다면 사용자의 특정 자원 요청을 각 SN들에게 전달하여 작업을 수행한다. DVM의 세부 구성은 (그림 1)에서 보는 것처럼 명령줄에서 사용자가 입력한 classpath와 실행 classfile을 통해 시스템을 기동하는 NMBL(Normal mode Boot Loader), 실행 조각별로 구분되는 NMI(normal mode interpreter), NMI의 요청을 받아 적절한 처리를 수행하는 NMRE(normal mode runtime environment), NMRE 요청을 처리하는 NMSC(Normal mode System call)가 존재하며, NMSC하부에는 classfile을 로드하여 시스템에서 사용할 수 있는 자료 구조로 변환하는 CL (Class Loader)[7], DPMS의 Listener로 DVMS의 DPMS와 연계되는 DPML(Distributed Process Manager Listener), DFS의 Requester로 DVMS의 DFS와 연계되는 DFSR(Distributed File System Requester), SN들과의 메시지 통신을 위한 IMES(Instance Message Exchange System)등으로 구성된다. 또한 내부적 각 모듈들에서 사용되는 라이브러리를 가지고 있는 SYSTEM LIB로 구성되어 있다. 가장 하부에 있는 OS는 Linux 및 Windows가 될 수 있다. DVM의 구현 범위는 new 연산자를 이용한 객체의 할당, String 처리 integer, long형의 처리, integer, long형의 사칙연산, char, byte, int, long형의 배열지원 등이다. 따라서 Exception, 부

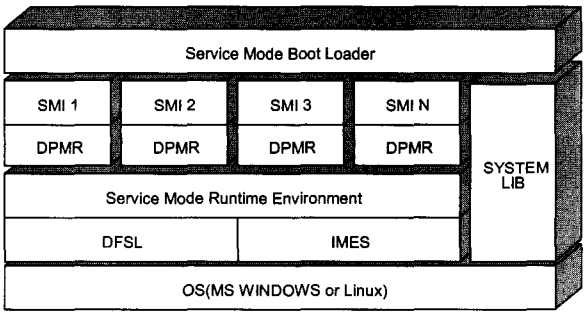
동수숫점 연산, GC, 유니코드 처리 등등의 처리와 SDVM 클래스 라이브러리에 정의되지 않는 기능은 사용할수 없다.



(그림 1) DVM SYSTEM 구성도

3.2.2 DVMS 설계

DVMS는 SN 상에 존재하며 사용자들의 요청에 적절한 자원을 제공해주는 역할을 한다. (그림 2)와 같이 세부구성은 DVMS를 로드하는 SMBL(Service Mode Boot Loader), NN 상에 존재하는 DVM의 요청에 CPU 자원을 제공하는 SMI (Service Mode Interpreter)와 DPMR(Distributed Process Manager Request), DVMS의 유지와 하부처리를 담당하는 SMRE(Service Mode Runtime Environment), 저장공간의 자원을 제공하는 DFSL(Distributed File System Listener), 시스템들 간에 메시지를 교환하는 IMES(Instance Message Exchange System), 그리고 SYSTEM LIB등으로 구성되어 있다. DVMS는 자신의 CPU 및 저장공간 자원을 유지, 관리하는 책임이 있으며 사용자의 선택에 따라 DFSL 혹은 (SMI + DPMR)은 선택적으로 실행될 수 있어야 한다.



(그림 2) DVMS SYSTEM 구성도

3.2.3 Protocol 설계

① DPMS Protocol

DPMS의 데이터 교환은 원시데이타, 배열, 문자열등을 전달할수 있어야 한다. 이는 메소드의 원격 호출시 인자값과 반환값에 있어 그 내용을 네트워크를 통해 전달해야 하기 때문이다. DPMS를 사용하는 모든 모듈들은 프로토콜을 직접 사용하지 않고 추상화된 인터페이스를 통해 작업을 수행하며 정의는 다음과 같다.

DPMS protocol format := *integer(msgid)* : *string(msgbody)*
msgbody := *string(args1)*
< *define separate char of msgbody* > *string(args2)*
args2 := *string(args3)* < *define separate char of args2* >
string(argsN)
argsN := *string(argsN + 1)* < *define separate char of argsN* >
string(argsN + 2)

② DFS Protocol

DFS는 파일의 데이터 송수신 및 파일의 삭제 등을 처리할수 있어야 한다. 실제데이타의 교환은 'remote_request'가 발생되면 일어나며 호출자는 프로토콜을 직접 다루지 않는다. 정의는 다음과 같다.

DFS protocol format :=
integer(commandType) < *String(data)* *or* *Integer(s)(data)* >

③ IMES Protocol

IMES는 동일한 Subnet상의 IMES와의 통신을 담당하며 호출자는 프로토콜을 직접 다루지 않는다. 정의는 다음과 같다.

IMES protocol format := *string(msgTitle)* : *string(msgBody)*

3.2.4 SDVM 클래스 라이브러리

SDVM에서 사용되는 클래스 라이브러리는 <표 2>와 같이 사용자가 일반 JDK로 개발면서 아래의 클래스를 사용할 경우 wrap한다. 특히 DPMS와 DFS를 사용하기 위해 wrap된 java class는 'java/lang/Thread', 'java/io/FileInputStream', 'java/io/FileOutputStream'이다. 구현되는 class의 모든 메소드는 JDK 클래스 라이브러리와 호완을 이루어야 한다. 또한 JDK Java compiler에 의해 컴파일될 수 있어야 한다.

<표 2> SDVM 클래스 라이브러리 구성

class lib	내 용
java/lang/Object	모든 Class의 부모 class
java/lang/String	문자열 관리
java/lang/StringBuffer	'System.out.println'에서 다중 데이터 처리
java/lang/System	지원 클래스
java/lang/Thread	DPMS 지원
java/io/FileInputStream	DFS 지원
java/io/FileOutputStream	DFS 지원
java/io/PrintStream	System.out.println

3.3 SDVM 구현

개발 RAD Tools은 크로스 플래폼을 지원과 Object 개념이 잘 정의된 Object Pascal을 사용하기 위해 Window용 Delphi Enterprise Version 6[8]와 Linux용 Kylix Enterprise Version 3.0을 사용하였고 SDVM 클래스 라이브러리 개발과 사용자 프로그램 test를 위해 Sun JDK 1.3, 1.4[9]을 사용하였다. 개발 플래폼은 MS-Windows XP Professional, MS-Windows 2000 Server 그리고 RedHat Linux 8.0을 하였고 기타 java ClassFile배치 및 java 소스 편집을 위해 jpcap of ej-technologies와 울트라 에디터 10.10을 사용하였다.

3.3.1 DVM 구현

DVM의 NMBL은 NMRE와 NMI를 생성하여 기본 thread인 'public static void main(String[ ])'을 시작한다. 이때 '<clinit>' static 초기화 메소드가 있으면 이것으로 점프하고 종료된다. NMRE는 하부 모듈을 초기화 하고 메인 thread의 종료를 기다린다. NMRE에는 클래스 라이브러리에서 정의한 'native' 메서드가 존재하며 SN들과의 작업을 중계한다. NMI에서는 스택을 가지고 있으며 지원하는 Java byte code를 처리한다. NMI에서 작업을 수행하기 위해 종료클래스, 종료메서드, load클래스, codeOffset, variableOffset 등을 유지한다. 초기값을 NMBL이 설정하고 그 후에는 실행결과에 따라 변경된다.

3.3.2 DVMS 구현

DVMS의 SMBL이 SMRE를 생성하고 시작한다. DPMR 및 DFSL은 DVM의 요청으로 생성된 후 별도로 실행된다. DVMS는 dvms.ini파일로 설정을 컨트롤 할 수 있다. 그 내용은 다음과 같다.

```
[DVMSMAIN]
isProcessListener = true
isDFSListener = true
allowsizeDFS = 100
```

isProcessListener는 DPMS를, isDFSListener는 DFS를 설정하며 DFS는 부가적으로 디스크의 쿼터를 적용하여 allowsizeDFS에 byte단위로 설정할 수 있다. SMBL에 의해 로드된 후 IMES에서 수신된 메시지에 의해 DPMS를 포함하고 있는 SMI를 시작시킨다. SMI는 SMRE에 시작된 후에는 독립적으로 작동한다. DFSL는 SMRE가 시작될 때 실행되며 그 후에는 독립적으로 작동한다.

3.4 SDVM 동작과정

NN 혹은 SN이 네트워크에 참여하게 되면 자신의 존재를 IMES를 통해 알린다. 알린 후 SN에서는 지속적으로 자신의 자원 상황을 갱신하여 네트워크에 알린다. 이러한 정보는 NN의 DVM에서 수집되어 관리되며, 사용자의 프로그램 수행중 CPU, 혹은 저장장치에 대한 요청이 들어오면 저장되어 있는 자원 리스트를 검색하여 연결한다. 이러한 과정은 NN이 수행중에도 SN이 추가된다면 동적으로 갱신될 수 있다. 따라서 고정된 수의 node가 정적으로 증가하는 것이 아니고 수행중에도 동적으로 증가시킬 수 있다. 동작과정은 N1 혼자 존재하는 경우, S1이 존재 후 N1이 존재하는 경우, S1, N1 존재후 S2가 존재하는 경우등의 동작과정으로 나뉠 수 있다. N1 혼자 존재하는 경우는 네트워크 상에 있는 SN등의 응답이 없기 때문에 모든 자원요청을 로컬로 처리한다. 이 경우에는 SN등의 내부 처리 때문에 약간의 오버헤드가 발생할 수 있다. S1이 존재 후 N1이 존재하면 N1은 S1의 정보를 저장하고 사용자의 자원 요청을 수락한다. 요청 자원의 여유가 S1이 더 많다면 S1과 연결을 시도하고 그렇지 않다면 로컬로 처리한다. 마지막으로 N1, S1 존재후 S2가 네트워크가 진입하면 S2는 자신의 자원 정보를 네트워크에 알린다. 이 정보는 네트워크에 있는 모든 Nn들이 저장한 후 사용준비한다. Nn중 S2에 자원요청를 하면 S2는 자신의 자원을 할당하여 작업을 처리한다.

4. SDVM 평가

SDVM의 평가는 DPMS의 평가만을 수행한다. DFS는 네트워크 대역폭 한계와 로컬 I/O와의 격차가 존재하기 때문에 분산된 저장 자원을 사용/관리하기 위한 별도의 프로그래밍이나 시스템을 구입하지 않는 것에 의미를 두어야 한다. 하나로 보는 특성을 감안하여 사용하여야 하기 때문이다. DPMS의 평가를 위해 사용된 계산식은 다음과 같다.

$$fx(A) = y$$

$$A = TN(r1), V(r2), SN(r3)$$

TN : $ThereadNumber$

V : 10^N

SN : $ServiceNodeNumber$

함수 fx에 작업량 A를 넣고 총 작업수행시간 y를 얻는다. A는 1부터 10^N개까지 덧셈하는 작업으로 TN, V, SN의 인자가 있다. TN은 Thread Number로 작업 A를 Thread 개수로 나누어 실행하며 TN이 3이면 Thread가 3개이다. V는 10^N N승으로 3이면 '10 × 10 × 10'로 1000을 나타낸다. SN은 service node의 개수를 나타내며 실제 컴퓨터에서 실행되고 있는 DVMS의 개수와 일치한다. 예를 들어 SN이 0이면 모든 작업은 local로 처리되고 SN이 1이면 SN와 작업을 나누어 실행한다. SDVM 평가를 위한 Test 시스템 구성은 다음과 같으며 각 시스템은 SDVM을 구동하기 위해 최적화 되지 않고 다른 프로그램들이 수행되고 있을 수 있다. TEST 시스템은 <표 3>에서 같이 다양한 시스템들로 구성되었으며, NN에서 프로그램 처리시간을 산출하기 위해 Linux

의 time 명령어를 사용하였다. fx(A) = y를 평가를 하기 위한 샘플 프로그램은 다음과 같다.

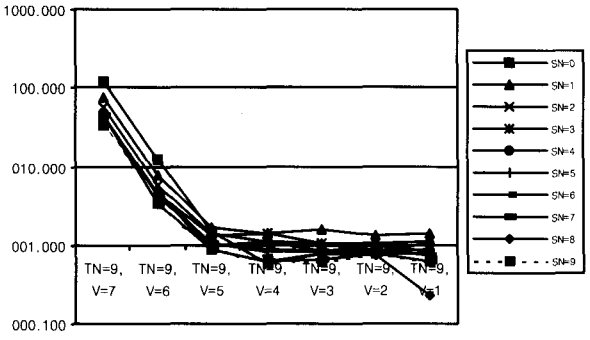
```
class adder extends Thread {
  long svalue, evalue ;
  public adder(long s, long e) {
    this.svalue = s ;
    this.evalue = e ;
    System.out.println(s + " : " + e) ;
  }
  // for jdk 1.4 java compiler public void start( ) {
    super.start( ) ;
  }
  public void run( ) {
    long e, s, t ;
    s = svalue ;
    e = evalue ;
    t = 0 ;
    for(long a = s ; a <= e ; a++) t = t + a ;
    dtsTest.totalValue = dtsTest.totalValue + t ;
    System.out.println(dtsTest.totalValue) ;

  }
}

public class dtsTest {
  public static long totalValue ;
  public dtsTest( ) { }
  public dtsTest(long maxValue, int threadNumber) {
    long start, end, incValue;
    start = 1 ;
    incValue = maxValue / threadNumber ;
    end = incValue ;
    for(int b = 0 ; b < threadNumber ; b++) {
      new adder(start, end).start( ) ;
      start = end + 1 ;
      end = end + incValue ;
    }
  }
  public static void main(String[ ] args) {
    int threadNum = 4 ;  // TN = 4
    System.out.println("process test") ;
    new dtsTest(100000000L, threadNum) ;  // V = 8
  }
}
```

4.1 1차 Test

fx(TN(9), V(7..1), SN(0..9)의 결과는 <표 3>과 같으며 데이터의 결과는 1/100초이다. 소수점이상은 초단위로 분까지 합산한 결과이다. 120.130은 2분 130(m/s)를 나타낸다. 결과를 살펴보면 작업량이 작을때는(V = 3이하) SN을 증가 시켜도 속도의 개선이 크게 나타나지 않는 것을 볼 수 있다. 하지만 V가 커지고 SN을 증가시켜면 속도의 향상이 나타난는 것을 볼 수 있다. 또한 SN을 증가시키는 것이 작업량을 1/2하는 것이 아님을 알 수 있다. 이는 NN에서 SN들과의 연계작업에서 얻는 속도의 이익과 그에 따른 오버헤드가 동반상승하기 때문이다. 이것을 그래프로 나타내면 (그림 3)과 같다.

<표 3> fx(TN(9), V(7..1), SN(0..9)의 결과

	SN = 0	SN = 1	SN = 2	SN = 3	SN = 4	SN = 5	SN = 6	SN = 7	SN = 8	SN = 9
TN = 9, V = 7	120.948	077.586	055.042	044.213	043.872	044.197	043.990	043.764	044.176	033.002
TN = 9, V = 6	012.036	007.789	005.767	004.527	004.352	004.434	004.342	003.417	004.506	003.421
TN = 9, V = 5	001.491	001.630	001.380	001.290	000.960	001.041	001.021	000.854	000.854	000.882
TN = 9, V = 4	000.597	001.405	001.079	001.408	001.090	000.851	000.874	000.608	000.620	000.660
TN = 9, V = 3	000.791	001.581	001.046	001.051	000.839	000.837	000.825	000.662	000.786	000.628
TN = 9, V = 2	000.784	001.305	001.044	000.825	000.822	000.983	000.857	000.796	000.755	000.794
TN = 9, V = 1	000.782	001.370	001.122	000.874	001.049	000.833	000.744	000.624	000.229	000.632



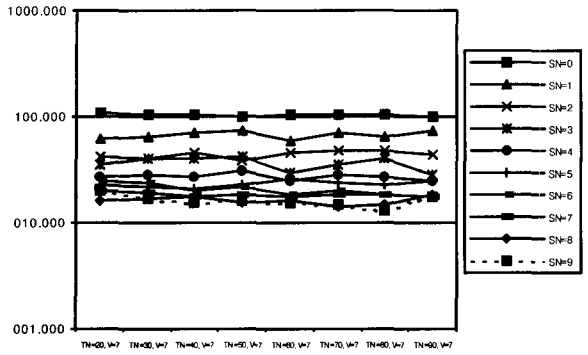
(그림 3) fx(TN(9),V(7..1),SN(0..9) 성능 비교도

4.2 2차 Test

<표 4>은 V를 7로 고정하고 TN을 20, 30..90 순서로 증가시킨 결과이다. 같은 작업에 대해 TN을 증가시키면 속도의 개선이 일어난다. 이것은 전 작업을 마치고 쉬고 있는 시스템에게 다른 작업 부여함으로서 시스템의 전반적인 운용성이 증대된 결과이며 이것을 그래프로 나타내면 (그림 4)과 같다.

<표 4> fx(TN(20..90), V(7), SN(0..9)의 결과

	SN = 0	SN = 1	SN = 2	SN = 3	SN = 4	SN = 5	SN = 6	SN = 7	SN = 8	SN = 9
TN = 20, V = 7	110.654	060.817	042.224	034.869	026.654	023.044	024.998	019.834	016.401	019.812
TN = 30, V = 7	106.503	064.138	039.895	039.850	028.393	021.636	023.539	019.469	016.959	016.496
TN = 40, V = 7	104.298	070.033	044.880	040.427	026.687	021.002	020.231	017.561	017.501	015.231
TN = 50, V = 7	101.518	072.809	038.772	042.434	030.534	022.417	021.640	018.625	015.551	015.795
TN = 60, V = 7	103.169	059.451	046.441	029.929	025.279	026.484	018.147	017.643	016.039	015.523
TN = 70, V = 7	102.572	71.794	046.992	035.299	027.957	024.154	019.932	018.115	014.433	014.909
TN = 80, V = 7	103.510	65.444	046.917	040.337	026.615	022.999	018.458	018.709	014.805	013.005
TN = 90, V = 7	101.730	74.145	043.560	028.783	024.990	024.707	017.776	017.746	018.090	017.790



(그림 4) fx(TN(20..90), V(7), SN(0..9) 성능 비교도

5. 결 론

'분산환경에 적용될수 있는 DVM 및 DESPL의 제안'에[1] 따라 DVM이 갖추어야 할 요건은 '각 DVM간에 위치 동등', 'DVM 설치후 객체 공유', '자원들의 효율적 관리', '각 분산 자원의 상대적 평가', '분산 환경 정보의 동적 갱신', '오류대처', '분산 환경 정보 보호', '각 분산 환경들의 동적인 가입/탈퇴', '플래폼 독립', '표준 프로토콜 설계', '싱글 환경 실행가능', '쉬운 설치' 등이다. 이에 본 연구에서는 제안된 내용중 일부를 수용하여 사용자가 분산환경의 자원을 어려움 없이 사용할수 있는 SDVM을 설계 및 구현하였다. 설계 구현한 SDVM의 평가 결과와 같이 다른 특정 시스템보다 효과적임을 검증하였다. 따라서, 특정 분산 시스템을 위한 전문 시스템이 아닌 일반시스템을 가지고, 작업을 처리하는 데 있어서, '공간적', '운영적' 그리고 '분산환경'에 대한 개념적인 비용이 발생하지 않는 다는 것을 증명하였다.

앞으로의 연구과제는 SDVM을 바탕으로 독립적인 DESPL과 DVM을 설계 구현하여 독립적인 시스템으로 확장해 나가는 것이다.

참 고 문 헌

[1] 양일등, 이석희, 김성열, "분산환경에 적용될수 있는 DVM 및 DESPL의 제안", 한국정보과학회 학술발표논문집(B), 제30권 제1호, pp.160-162, Mar., 2003.

[2] 김영태, 이용희, 최준태, 오재호, "초고속 네트워크를 이용한 PC 클러스터의 구현과 성능 평가", 한국정보과학회 논문지, 시스템 및 이론, 제29권 제2호, Feb., 2002.

[3] 장종현, 이동길, 한치문, "개방형 네트워크 환경을 위한 멀티쓰레드 기반 코바 설계 및 구현", 한국정보처리학회, Nov., 2001.

[4] 김경하, 김영학, 오길호, "IBM Aglets를 기반으로 하는 가상 병렬 컴퓨팅 시스템에서 작업 할당 기법과 성능 비교", 한국정보과학회, Apr., 2002.

[5] 이석희, 현정식, 이태희, 조 상, "Hybrid P2P의 그룹관리와 신뢰성을 위한 시스템 설계", 한국정보과학회, 2002.

[6] 강경우, "이기종 슈퍼컴퓨팅 자원들을 활용한 Globus 기반 그리드 구축에 관한 연구", 한국정보처리학회, 2002.

[7] Tim Lindholm, Frank Yellin, "The Java™ Virtual Machine Specification Secound Edition," Sun Microsystems.

[8] 볼랜드 코리아, "Borland Delphi 6 Part I - Delphi 프로그래밍".

[9] 정의현, 김성진, "자바2 JDK 1.4", 대림, 2002.

양 일 등

e-mail : fbiskr@hanmail.net

2002년 청주대학교 컴퓨터정보공학과 공학사

2004년 청주대학교 컴퓨터정보공학과 공학석사

2004년~현재 (주)우린정보 R&D 연구원

관심분야 : 운영체제, 분산시스템, 컴파일러, 언어론

이 석 희

e-mail : netause@cju.ac.kr

2000년 청주대학교 컴퓨터정보공학과 공학사

2003년 청주대학교 전산정보공학과 공학석사

2003년~현재 청주대학교 컴퓨터정보공학과 박사과정

관심분야 : P2P, 그리드 컴퓨팅, 홈 네트워킹, 유비쿼터스 컴퓨팅, 이동 에이전트 S/W

김 성 열

e-mail : srkim@cju.ac.kr

1982년 숭실대학교 전자계산학과 공학사

1987년 숭실대학교 대학원 전자계산학과 공학석사

1992년 숭실대학교 대학원 전자계산학과 공학박사

1982년~1984년 한국전력공사 전자계산소 근무

1984년~1990년 오산대학 전자계산과 교수

1997년~1998년 호주 QUT ISRC 객원 교수

1990년~현재 청주대학교 컴퓨터정보공학과 교수

관심분야 : 컴퓨터 네트워크, 컴퓨터 보안, 분산 객체 시스템