이종 분산 환경에서 UNIX 커널 성능 측정 방법에 관한 연구

박 윤 용† · 박 정 호† · 임 동 선††

요 약

본 논문에서는 네트워크를 통해 연결되어 있는 PC 기반의 UNIX 커널인 Linux, PC_Solrais 그리고 FreeBSD들의 성능을 측정하고 비교할 수 있는 모델을 제시하였다. 중요한 UNIX 커널들의 성능 지표들인 시스템 호출 수행 시간, 명령어 수행 시간과 디스크 입·출력 시간등을 μs 단위로 측정하였다. 또한, 각각의 UNIX 커널들의 네트워크의 성능을 비교하기 위해 TCP, UDP 그리고 RPC 방법에 의한 메시지 전송 시간을 측정하였고, 네트워크파일시스템(NFS : Network File System)의 성능을 비교하였다. 그리고 측정된 데이터들을 인터넷 상에서 시각화할 수 있도록 하였다.

A Study on the Method of the Performance Measurement for UNIX Kernel in Heterogeneous Distributed Environment

Yoon-Young Park† · Jung-Ho Park† · Dong-Sun Lim††

ABSTRACT

In this paper, we propose the model to measure and compare the performances of the Linux, PC_Solaris and FreeBSD which are the heterogeneous PC based UNIX kernels connected by network. Using the stop-watch timer with the μs unit, we measure the important performance indices of UNIX kernels which are the primitive and command execution time and disk I/O time. Also we measure the message transfer time using the TCP, UDP and RPC methods to compare the network performace of UNIX kernels. And we compare the performance of NFS for each UNIX kernels. And we display the measured data on internet.

1. 서 론

최근에 PC 기반의 UNIX 커널에 관한 관심이 증가하면서, Linux 시스템에 관한 많은 연구와 Linux 시스템을 이용한 프로그램 개발이 활발이 이루어 지고 있다. 본 논문에서는 PC 기반의 UNIX 커널인 PC_Solaris와 FreeBSD시스템과 Linux 시스템의 성능을 비교하였다. 이러한 컴퓨터 시스템의 성능은 과거의 독립적 지역시스템에서부터 최근의 클라이언트/서버 모델을 기반으로 하는 분산시스템에까지 주요한 관심사가 되고 있다. 시스템의 성능을 측정, 분석하려는 이유는 시스템의 각 자원이 사용되는 현황을 측정하여 시스템을 효율적으로 운용되도록 하여 시스템의 성능을 전체적으로 향상시키기 위함이다. 또한, 새로운 시스템 도입과 시스템 확장 등에 중요한 참고 자료로 반영하기 위해 시스템의 성능을 측정한다[1-3].

컴퓨터 시스템의 처리 속도 및 성능은 그 컴퓨터에서 실행되는 응용 프로그램의 수행시간으로 알아볼 수 있다. 이와 같이 컴퓨터 시스템의 성능을 측정하는 응

† 종신회원 : 선문대학교 컴퓨터정보학부 교수
†† 정 회 원 : 전자통신연구원 실시간OS연구실 선임연구원
논문접수 : 1998년 11월 12일, 심사완료 : 1999년 9월 28일

용 프로그램을 벤치마크 프로그램이라 한다. 특히, 운영 체제와 같은 커널의 성능을 측정할 수 있는 벤치마크 프로그램들이[3, 4, 5] 있지만, 측정 내용이 실질적으로 커널들의 성능을 비교하기 어렵고, 사용자에 친숙하지 못한 측정 단위를 사용하는 등의 문제점을 지니고 있다. 또한, 일부의 커널 성능 측정 방법은 커널의 소스 코드를 수정하여 측정하기 때문에 커널의 소스 코드를 구입할 수 없거나 커널 소스 분석이 어려운 경우에는 사용할 수 없다.

따라서 본 논문에서는 기존의 커널 벤치마크 프로그램의 단점을 보완하고, 측정하기 쉽고, 측정된 데이터를 쉽게 이해할 수 있는 커널 성능 측정 방법을 제시하고 구현하였다. 특히, 현재 Linux와 같이 PC기반의 UNIX 커널들이 많이 사용되고 있기 때문에, PC 상에서 수행되는 대표적인 UNIX 커널인 Linux, PC-Solaris 그리고 FreeBSD의 성능을 비교하였다. 또한, 이러한 이종의 UNIX 커널들이 네트워크 성능을 측정하고 비교하기 위한 모델을 제시하였다. 그리고 UNIX 커널들의 성능을 표시할 수 있는 주요한 성능 지표들에 관하여 μs 단위로 측정하였다.

이러한 성능 지표로는 주요한 프리미티브와 명령어 실행 시간, 프로세스 증가에 따른 성능 변화 그리고 디스크 입·출력 시간등이 있다. 또한, 이종의 UNIX 시스템의 네트워크 성능을 분석하기 위하여 TCP 또는 UDP 그리고 RPC(Remote Procedure Call) 방법을 사용하여 메시지를 전송하는 경우의 메시지 전송 시간을 비교하였다. 또한, 이종의 UNIX 커널들을 NFS(Network File System)으로 구축한 후 NFS를 이요하는 파일 입·출력의 성능을 비교하였다. 그리고 측정된 자료들을 인터넷 상에 도시하였다.

제안하는 성능 측정 프로그램은 커널 내부 구조에 관한 사전 지식이 필요 없고, 커널 내부를 변경하지 않는 방법을 사용하여 UNIX 커널들의 성능을 μs 단위까지 측정하였다. 그러나 워크스테이션 급의Solaris계열의 UNIX 커널(SPARC Solaris, X86 Solaris)등에서는 정확한 μs 단위까지 측정 가능한 타이머가 제공되지만, PC기반의 Linux와 FreeBSD에서는 정확한 μs 단위의 수행시간을 측정할 수 없고, 프로세스 생성과 같은 시스템 호출의 수행시간 측정 시에 Alarm Clock 과 같은 오류가 발생하는 등의 문제가 있다[6, 7, 8, 9].

위와 같은 문제를 해결하기 위하여 μs 단위까지 측정 가능한 타이머를 Linux와 FreeBSD등에 직접 구현하지 않고 Solaris계열의 컴퓨터에 구현하고, 타이머 구현되어 있지 않은 Linux와 FreeBSD 커널의 성능은 원격 프로시져 호출(Remote Procedure Call : RPC)을 이용하여 성능을 측정하고 비교하였다[10, 11, 12, 13, 14, 15, 16].

본 논문의 구성은 다음과 같다. 2절에서는 시스템 소프트웨어의 성능 측정 방법에 대하여 간략히 기술하였다. 3절에서는 분산 환경에서의 성능 측정 모델과 방법등에 관하여 기술하였다. 4절에서는 RPC를 이용한 시스템 소프트웨어의 성능 측정 결과를 설명하였고 5절에서는 본 논문의 결론 및 향후 연구 방향에 대해 기술하였다.

2. 시스템 소프트웨어의 성능 측정 방법

하드웨어 또는 소프트웨어의 성능을 측정하는 프로그램을 벤치마크 프로그램이라 한다. 이러한 벤치마크 프로그램은 측정하고자 하는 대상의 동작 환경, 평가 대상과 목적 그리고 평가 척도 및 평가 기준에 따라서 종류가 다양하다. 또한, 단순히 성능을 측정한다고 해서 모두 벤치마크 프로그램이 되는 것은 아니고, 측정 방법과 결과등 일련의 과정들을 공인받을 수 있는 프로그램이어야 한다. 따라서 본 논문의 최종 목표는 커널의 성능을 측정하는 벤치마크 프로그램의 구현에 있지만, 중간 단계로서 커널의 성능 측정 방법에 관하여 기술하였다.

컴퓨터 시스템의 성능을 측정하는 기법으로는 크게 두 가지 방법으로 나눌 수 있는데 동적 성능 측정과 정적 성능 측정이 그것이다. 동적 성능 측정은 모니터링(monitoring)도구를 이용하는데, 이는 다시 시간 구동형(time-driven)기법, 이벤트 구동형(event-driven)기법과 추적 구동형(trace-driven)기법으로 나누어질 수 있다. 벤치마크를 이용하는 정적 성능 측정 방법은 CPU의 처리속도, 평균 디스크 접근시간과 같이 항상 일정한 결과를 수집하기 위한 방법으로 사용된다. 이러한 벤치마크를 이용한 정적 성능 측정은 우리가 흔히 쓰는 개인용 컴퓨터에서부터 Cray와 같은 슈퍼컴퓨터에까지 다양하게 사용되지만, 한 종류의 벤치마크 프로그램으로 모든 컴퓨터 시스템의 성능 측정에 적용할 수는 없다[3, 4, 5]. 따라서 측정하는 대상과 목적에 따라서 다양한 종류의 벤치미크 프로그램이 존재하게 된다.

벤치마크 프로그램의 종류에는 평가 대상 및 적용 범위에 따라 여러 형태로 분류된다[3]. 평가 대상을 중심으로 벤치마크 프로그램의 형태를 구별하면 기본 연산 성능 측정 벤치마크, 애플리케이션 실행 성능 측정 벤치마크, 그래픽스 처리 성능 측정 벤치마크, 플랫폼 성능 측정 벤치마크의 형태로 나누어 진다. 기본 연산 측정 벤치마크 형태의 프로그램들은 주로 프로세서의 성능 또는 캐쉬 성능등을 측정 대상으로 하고, 정수 연산이나 부동 소숫점 연산의 성능을 측정한다. 이러한 형태의 대표적인 벤치마크로는 Dhrysyone, Whetstone, Linpack 그리고 Livermore Loops등이 있다.

애플리케이션 실행 성능 측정 벤치마크는 데이터베이스, 네트워크등의 애플리커이션의 성능을 측정하는 벤치마크 프로그램이다. 이러한 형태의 벤치마크에는 분산 파일 시스템의 성능을 측정하는 NFS, 데이터베이스의 성능을 측정하는 TPC-B(TPI), Winconsin등이 있다. 그리고 데이터베이스와 네트워크의 성능을 측정하는 TPC-A(=Debit-Credit)등이 있다. 그리고 그래픽스 처리 성능 측정 벤치마크에는 Ghraphstone, 튜두초, X11perf와 GPC/PLB등이 있다.

마지막으로 플랫폼 성능 측정 벤치마크는 시스템의 전반적인 성능 평가를 목적으로 개발된 프로그램으로서 DOS 또는 UNIX 시스템 평가용 벤치마크들이다. 이러한 형태의 벤치마크로는 워크스테이션을 주 대상으로 하는 SPEC과 SPEC의 개정형인 Cint92, Cfp92 등이 있다. 또한 MS-DOS용 시스템 평가용 벤치마크인 New Byte Magazine과 UNIX 시스템 평가용 벤치마크인 Byte Magazine, AIM Suite Ⅰ·Ⅱ·Ⅲ, N.N. Business 그리고 SPEC SDM등이 있다. 그리고, 기본 연산 성능이나 I/O 성능등을 포함한 대표적인 PC/워크스테이션 시스템의 종합 성능 평가용 벤치마크로서 khorner-stone등이 있다. 본 논문에서 제안하는 성능 측정 도구도 플랫폼 성능 측정 벤치마크 형태로 분류될 수 있는 벤치마크 프로그램으로 이들의 특징들을 정리하면 <표 1>과 같다[3].

대부분의 플랫폼 성능 측정 벤치마크 프로그램은 전

〈표 1〉 플랫폼 성능 측정 벤치마크 제품 비교

벤치마크 이름	SPEC R1.0 (SPEC'89)	Cint92 Cfp92	Khorner-stone	N.N. Business	AIM Suite Ⅲ	SDM(SDET/KENBUS1)	제안하는 성능 측정도구
특 징	워크스테이션의 사실상의 표준 벤치마크이며 항상 최신의 보고서가 공표되고 있다.	구 SPEC의 문제점을 개선. 벤치마크 프로그램의 샘플 수가 많다(싱글 프리시전의 처리를 추가).	I/O 성능의 평가에 실적이 있다. 역사가 길며 풍성한 비교 데이터(각사 머신의 결과)도 갖고 있다.	결과를 그래프로 디스플레이할 수 있어 분석이 쉬움. 이용형태별 성능비교 가능.	실적이 많고 결과의 그래프 디스플레이로 분석이 쉬움.	표준벤치마크이므로 각 사의 공표치를 믿을 수 있다. 또 성능동향을 알기 쉽다.	이종의 분산 환경에서 UNIX 커널의 성능을 상대적으로 비교.
개발원	SPEC	SPEC	Workstation Laboratories	Neal Nelson Associates	AIM Technology	SPEC	제안중
평 가 내 용	정수연산 애플리케이션과 6개의 애플리케이션이 실행시간을 측정하고 기준 성능치로부터 각 성능치를 계산.	Cint92: 6개의 정수연산 애플리케이션 실행시간 측정. Cfp92: 14개의 부동소숫점 연산 애플리케이션 실행시간. 성능치의 산출방식은 구 SPEC과 같음.	아래의 네 관점에서 모두 21개의 테스트를 실시 ①정수연산 ②부동소수점연산 ③파일 엑세스 ④상기의 결과에 총실행시간을 가미하여 득점을 결산	멀티유저 환경에서의 정수/부동소숫점의 연산. 메모리, 파일 조작, 함수 call 등.	부하프로세스를 늘렸을 때의 응답시간/스루풋의 변화.	부하프로세스 수를 늘렸을때의 스루풋 변화.	UNIX 커널의 각종 프리미티브, I/O, 명령어 실행 시간, 프로세스 생성 실행 시간, 네트워크 성능 측정, NFS I/O 성능 측정.
기 술 언 어	정수: C 부동: 포트란	정수: C 부동: C/포트란	C와 포트란	C/포트란	C	C	C와 JavaApplet
평 가 척 도	종합: SPECmark 정수: SPECint 부동: SPECfp	종합: SPECmark 정수: SPECint92 부동: SPECfp92	Khorner-stone 값	테스트별 처리시간	실행시간 완료 프로세스수/초	피크 스루풋	us 단위의 시간.
기 준 성 능	VAX11-780에서의 벤치마크 결과	VAX11-780에서의 벤치마크 결과	불명	임의로 설정가능. 디폴트는 VAX-11/780에서의 결과	VAX-11/780에서의 벤치마크 결과	-	비교 대상의 커널을 상대적으로 비교.

체 컴퓨터 시스템에 걸쳐 적용되기보다는 연산, 프로세스 수행 등과 같은 특정한 서브시스템의 성능을 측정하는 것이 보통이다. 또한, 기존의 벤치마크 프로그램의 성능 평가 척도가 VAX11-780에서 측정한 성능 평가 값과 비교하여 성능 평가 값을 표시하기 때문에 일반 사용자들이 쉽게 성능 평가 척도를 이해하지 못한다. <표 1>의 마지막 행에서 보듯이 기존의 성능측정 벤치마크들이 제시하는 기준성능이 일반적인 사용자가 쉽게 이해하기 어려운 단위들을 사용하고 있다. 그리고, 평가 내용면에서도 커널의 성능을 비교할 수 있는 내용들을 측정하지 못한다. 본 논문에서 제시하는 성능측정프로그램은 이와같은 문제점을 해결하고, 사용자들이 쉽게 이해할 수 있는 측정방법과 측정단위를 사용하여 이종의 UNIX 시스템의 성능을 비교하였다.

또한, 본 논문에서 제안하는 성능 측정 프로그램은 특정한 서브시스템의 성능을 측정하는 것이 아니고, UNIX 커널의 전체적인 성능을 측정할 수 있고, 측정 단위도 μs의 시간으로 표시함으로써 사용자들이 쉽게 이해할 수 있도록 구현하였다.

그리고 플랫폼 성능 측정 벤치마크 중에서 커널의 성능을 측정하기 위해서는 커널에 관한 소스 코드를 분석해서 성능을 측정하는 방식과 커널 소스를 전혀 모르는 상태에서 커널의 성능을 측정하는 방식이 있다. 전자는 커널 소스 코드를 알아야 하는 어려움이 있기 때문에 일반적인 커널의 성능을 분석하기에는 적합하지 못하다. 본 논문에서는 후자와 같이 커널 소스 코드를 모르는 경우에도 커널의 성능을 측정할 수 있는 방법을 제시하였다.

3. 성능 측정 도구의 설계

본 절에서는 성능 측정을 위해 μs단위까지 측정 가능하도록 구현한 스톱워치 타이머를 이용하여 실행 시간을 측정하는 방법에 관하여 기술하였다. 또한, 분산 환경에서 스톱워치 타이머를 이용한 성능 측정 방법을 설명하고, 성능 측정 도구의 모델에 관하여 기술하였다.

3.1 스톱 워치 타이머

일반적으로 UNIX 커널내에 한 개의 프리미티브를 실행하는 시간은 수 μs에 해당되기 때문에 실행 시간을 측정하기 위해서는 μs 단위까지 측정 가능한 고성능의 타이머가 요구된다. 따라서 본 논문에서는 Solaris2.5.1커널에서 제공되는 시스템 프리미티브인 setitimer()와 getitimer()를 이용하여 μs단위까지 측정 가능한 스톱 워치 타이머를 구현하였다. 본 논문에서 구현한 스톱워치 타이머에서 제공하는 기본적인 프리미티브는 아래와 같다.

- set_timer() : 타이머 수행에 필요한 초기화를 수행하고, 타이머 값을 조정한다.
- start_timer() : 타이머를 동작 시킨다.
- ret = stop_timer() : 타이머를 정지시키고, start_timer() 동작 시점에서 현재 까지의 타이머 값을 μs단위로 복귀한다.
- read_timer() : 현재 타이머 값을 복귀한다.

스톱워치 타이머는 측정하고자 하는 대상의 시작 부분과 끝 부분에서 start_timer()와 stop_timer() 함수를 삽입함으로써 측정되어진다. 이때, 스톱워치 타이머 자체의 실행 시간 간섭을 배제해야 되기 때문에 실제로 측정된 값은 스톱워치 타미어 동작 시간을 제외시켜야 한다. 따라서 측정 대상의 실행 시간은

$$\text{time(측정\_대상)} = \text{TIMER\_time(측정\_대상)} - \text{TIMER\_time(스톱워치\_타이머\_실행시간)} \quad (1)$$

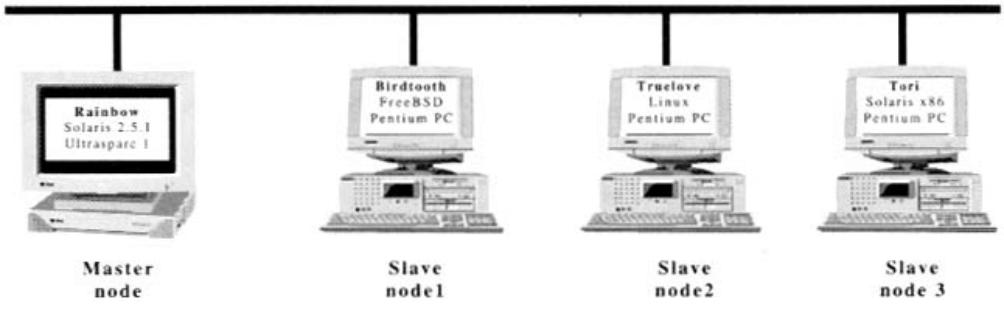
과 같이 표기된다.

- time(측정_대상) : 측정 대상의 실행 시간,
- TIMER_time(측정_대상) : 스톱워치 타이머 실행 시간을 포함한 측정 대상의 실행 시간,
- TIMER_time(스톱워치_타이머_실행시간) : 스톱워치 타이머 타이머 실행 시간.

3.2 이종의 분산 환경에서 스톱워치 타이머를 이용한 성능 측정

본 논문에서 제안하는 성능 측정 도구는 (그림 1)과 같은 분산 환경에서 이종의 UNIX 커널의 성능 측정을 하고자 한다. 분산 환경내의 각 노드들은 서로 다른 운영 체제와 CPU들로 구성되어 있다. 이러한 노드 중에서 운영 체제와 CPU가 μs 단위의 타이머를 제공할 수 있는 노드를 분산 환경에서 성능 측정의 주체가 되는 마스터 노드라 하고, 마스터 노드로부터 RPC를 이용하여 측정되는 노드들은 슬래브 노드라 한다.

따라서 본 논문에서는 마스터 노드를 중심으로 하여 분산 환경으로 연결된 슬래브 노드들의 UNIX 커널



(그림 1) 이종 분산 환경

성능을 측정하였다. 기본적인 원리는 마스터 노드에서 슬래브 노드로 RPC를 이용하여 슬래브 노드의 각종 커널 성능을 측정하게 된다. 즉, 측정하고자 하는 대상을 RPC를 이용하여 측정한 시간에서 널 메시지를 RPC를 이용하여 측정한 시간을 제거함으로써 측정 대상의 실행 시간을 구하게 된다. 따라서 측정 대상의 실행 시간은

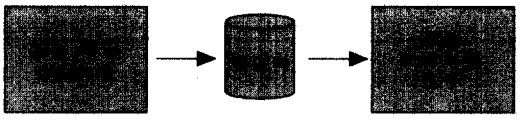
$$\text{Time(측정\_대상)} = \text{RPC\_Time(측정\_대상)} - \text{RPC\_Time(Null\_message)} \quad (2)$$

와 같이 표기된다.

- Time(측정_대상) : 측정 대상의 실행 시간,
- RPC_Time(측정_대상) : 마스터 노드에서 RPC를 이용하여 슬래브 노드의 측정 대상을 수행한 시간,
- RPC_Time(Null_message) : 마스터 노드에서 슬래브 노드까지 RPC를 이용하여 널 메시지를 전송하는 데 소요되는 시간

3.3 성능 측정 모델

본 논문에서 제안하는 성능 측정 모델은 (그림 2)와 같이 성능 측정 프로그램과 성능 표시 프로그램으로 구성되어 있다. 성능 측정 프로그램은 (그림 1)과 같은 환경에서 식 (1)과 식 (2)를 이용하여 측정 대상들의 실행 시간 데이터를 생성하는 프로그램들로 구성되었다. 성능 측정 프로그램은 대부분 C 언어로 작성되었고, 일부분만이 C 셸 프로그램과 awk 프로그램으로 작성되었다. 성능 모니터링 도구는 성능 측정 프로그램이 생성한 데이터를 시각화할 수 있는 도구로서 Java 애플릿으로 작성하여 인터넷 상에서 동작되어진다.



(그림 2) 성능 측정 모델

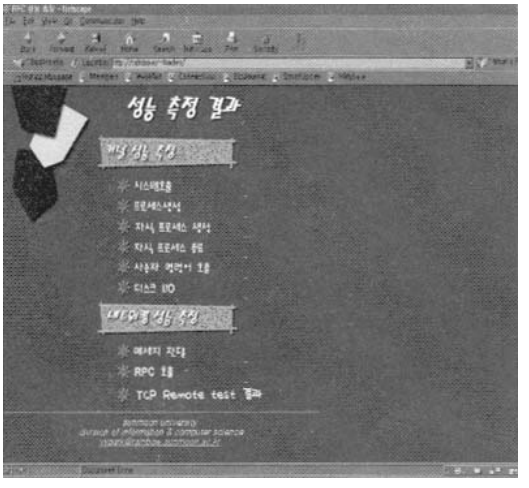
4. UNIX 커널의 성능 측정

본 논문에서는 스톱워치 타이머와 RPC 프로그램을 이용하여 (그림 1)과 같은 분산환경에서 이종의 UNIX 커널의 성능을 측정하였다. 측정 대상이 되는 UNIX는 X86_Solrais2.5.1, FreeBSD2.2.1 그리고 Linux2.0.32이고, (그림 1)과 같이 네트워크를 통해 연결되어져 있다. 각각의 커널의 성능이 CPU 속도, 메모리 용량, 캐쉬 메모리등 하드웨어적인 요소에 의해 영향을 받기 때문에, 세 개의 커널이 설치된 하드웨어의 사양을 <표 2>와 같이 동일하게 구성함으로써 하드웨어적 요소가 성능에 미치는 영향을 최소화하였다. 따라서 마스터 노드를 제외한 슬래브 노드들의 시스템 사양은 동일하게 구성하였고, 각 슬래브 노드들의 커널만을 다르게 함으로써 UNIX 커널간의 특성 및 성능을 비교하게 하였다.

〈표 2〉 시스템 사양

역할	호스트	운영체제	CPU	Memory	Disk
Master	Rainbow	Solaris2.5.1	Ultra SPARC1 170MHz	128MB	6GB
Slave	Tori	x86 Solaris2.5.1	Cyrix 6x86 133MHz	16MB	1.3GB
Slave	Birdtooth	FreeBSD2.2.1	Cyrix 6x86 133MHz	16MB	1.3GB
Slave	Truelove	RedHat5.0 (Linux2.0.32)	Cyrix 6x86 133MHz	16MB	1.3GB

본 논문에서는 UNIX 커널의 성능과 네트워크의 성능을 구별하여 측정하였다. UNIX 커널의 성능으로는 주요한 프리미티브 실행 시간, 주요한 명령어 실행 시간, 디스크 입·출력 시간 그리고 프로세스 생성 시간 등을 측정하였다. 네트워크의 성능으로는 메시지 전송 시간을 비교하였다. 특히, 메시지의 크기에 따른 TCP, UDP 그리고 RPCC 방법에 의한 전송 시간을 이종의 UNIX 시스템들을 비교하였다. 또한, 메시지 크기에 따른 이거와 레이지 방법의 RPC 실행 시간을 비교하였다. 측정된 데이터들은 성능 표시 프로그램에 의해 도시화되어 인터넷 상에서 도시하였다. 성능 측정 도구의 초기 화면은 (그림 3)과 같고, 사용자들은 메뉴 방식에 의해 측정 결과를 막대 그래프 또는 x · y 그래프 형태로 커널의 성능을 비교할 수 있다.

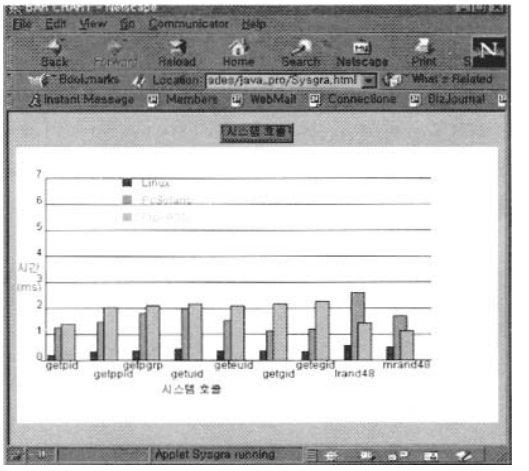


(그림 3) 성능 모니터링 도구의 초기 화면

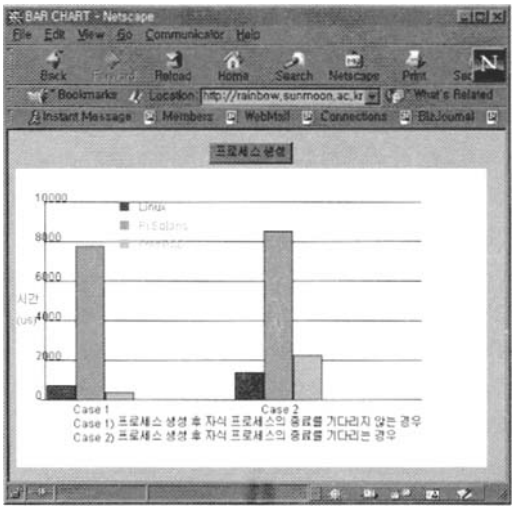
4.1 커널 성능 측정 결과

(그림 4)에서는 UNIX 시스템의 프리미티브 중에서 일부만을 측정하여 도시하였다. 대부분의 경우 Linux가 우수하게 측정되었고, PC-Solaris와 FreeBSD가 비슷한 성능을 나타내고 있다.

커널의 프로세스 관련 성능을 비교하기 위해 UNIX 시스템 호출 가운데에서 fork() 시스템 호출에 관하여 측정하였고, 결과는 (그림 5)와 같다. (그림 5)의 Case 1은 순수하게 fork() 시스템 호출의 실행 시간을 측정하였고, Linux와 FreeBSD가 PC-Solaris보다 우수하게 측정되었다. (그림 5)의 Case 2는 자식 프로세스 실행이 부모 프로세스의 실행에 어떠한 영향을 주는 지를 알아 보기 위하여 부모 프로세스가 fork() 시스템 호출 실행 후에 자식 프로세스의 실행 종료되기를 기다리는 시간을 측정하였다. 이 경우에도 Linux가 우수하게 측정되었다.



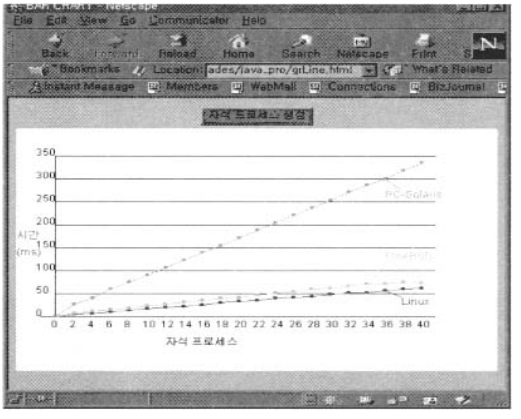
(그림 4) 시스템 호출의 실행 시간 측정



(그림 5) 프로세스 생성 시간 측정

또한 UNIX와 같이 다중 프로그램을 지원하는 경우에 프로세스의 수와 fork() 시스템 호출 시간과의 관계를 측정하기 위해 부모 프로세스가 생성한 자식 프로세스들 모두가 종료될 때까지 기다리는 경우의

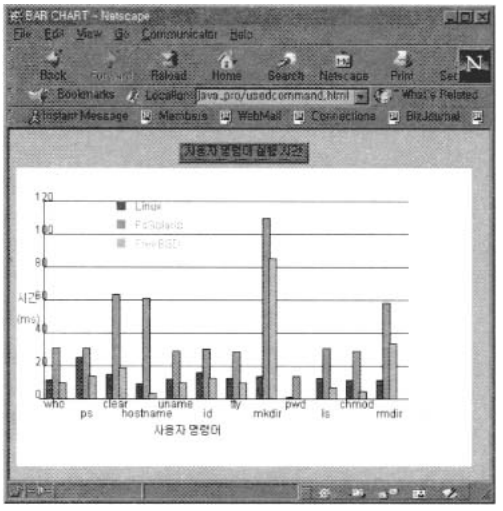
실행 시간을 측정하였다. 그러나 동시에 존재할 수 있는 프로세스의 수가 Linux 999개, PC-Solaris(x86) 229개, FreeBSD 40개로 서로 차이가 있기 때문에 (그림 6)와 같이 최대 40개의 자식 프로세스가 종료될 때까지 기다리는 경우만을 측정하여 비교하였다. 예를 들어 (그림 6)의 x축의 자식 프로세스의 값이 10이라는 것은 부모 프세스가 10개의 자식 프로세스들을 fork()를 이용해 생성하고 10개의 자식 프로세스 모두 종료할 때까지의 시간을 측정한 것이다. 이 경우 Linux와 FreeBSD가 20~30ms 정도의 시간이 소요되고, PC-Solaris가 90ms 정도의 시간이 소요된 것으로 측정되었다.



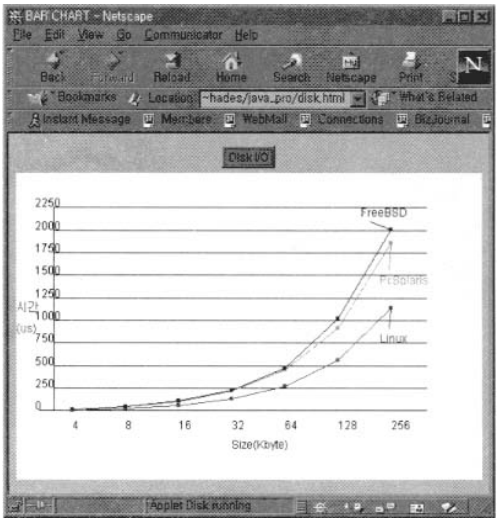
(그림 6) 프로세스의 수와 프로세스 생성 시간

(그림 7)은 UNIX 시스템에서 자주 사용하는 명령어의 수행시간을 측정한 결과이다. 명령어들은 현재 사용자 수와 현재 디렉토리의 파일의 수등 명령어들이 실행하고 있는 환경에 따라 측정값에 차이가 있기 때문에 본 논문에서는 임의적으로 세 개의 시스템에서 실행 환경이 일치하도록 하였다. 명령어 수행 시간의 경우에도 Linux와 FreeBSD가 PC-Solaris보다 우수하게 측정되었다.

마지막으로 디스크 입·출력 시간을 측정하였고 결과는 (그림 8)과 같다. (그림 8)에서는 20K 미만의 데이터를 디스크에 입·출력하는 시간은 큰 차이가 없으나 데이터의 크기가 커질수록 Linux 시스템이 우수하게 측정되었다.



(그림 7) 명령어 실행 시간

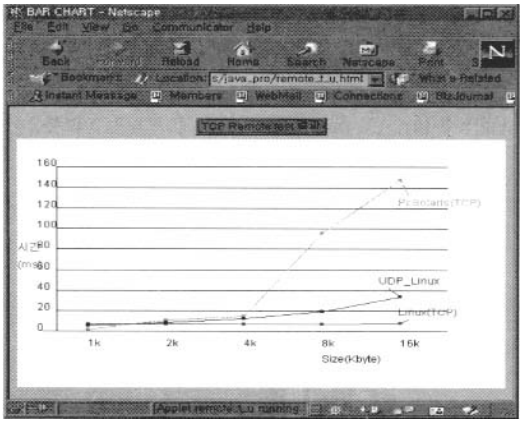


(그림 8) 디스크 입·출력 시간

4.2. 네트워크 성능 측정 결과

UNIX 시스템의 네트워크 성능을 측정하기 위해 우선 메시지 크기의 변화에 따른 메시지 전송 시간을측정하였다. (그림 9)는 Linux와 PC-Solaris에서의 TCP를 사용하여 메시지를 전송하는 시간을 비교하였다. TCP를 사용하는 경우 대체적으로 Linux가 우수하게 측정되었고, UDP를 사용하는 경우에는 두 시스템이 <표 3>과 같이 비슷하게 측정되었다.

<표 3>은 UDP를 사용하여 메시지 크기에 따른 메시지를 전송하는 시간을 측정한 내용이고, 큰 차이는 없지만 근소하게 Linux가 우수하게 평가되었다. 또한, (그림 9)에서 UDP_Linux와 Linux(TCP)를 비교해 보면, 작은 크기의 메시지 전송에서는 UDP가 유리하지만 메시지의 크기가 커지면 TCP가 유리하다는 사실을 알 수 있다.



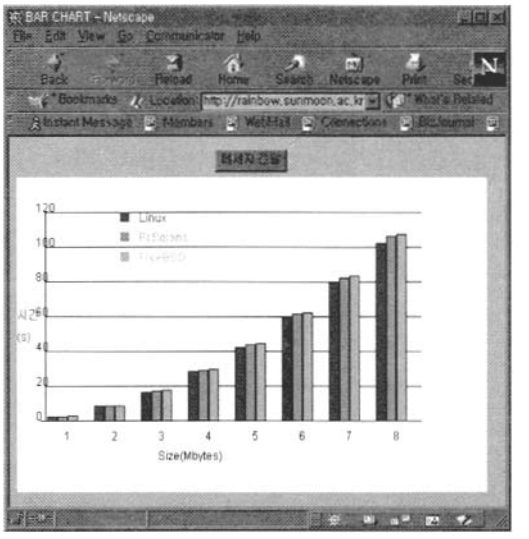
(그림 9) 메시지전송속도(TCP)

(그림 10)은 원격 프로시져 호출을 사용한 경우의 메시지 크기 변화에 따른 메시지 전송 시간을 도시한 것이다. 세 시스템의 전송시간은 큰 차이를 보이지는 않지만 비교적 Linux가 전송시간이 빨랐고, 그 다음으로는 PC-Solaris, FreeBSD의 순이었다.

원격 프로시져 호출(RPC : Remote Procedure Call)에서 포인터 연산을 수행하는 경우에 포이터 값의 변화에 따라 많은 양의 데이터들이 필요로 된다. 이 경우 서버에 존재하는 원격 프로시져에서 사용될 데이터를 RPC를 이용해 제공하는 방법에는 이거(eager)와 레이지(lazy) 방법이 존재한다[16]. 일반적으로 이거 방법의 RPC는 서버의 원격 프로시져가 처리할 데이터 전체를 한번에 메시지로 전송함으로써 추가의 데이터 요구를 하지 않도록 하는 방법이다. 이거 방법은 필요한 데이터를 모두 전송하기 때문에 결과적으로 추가의 메시지가 필요하지 않지만 많은 양의 데이터를 전송하는 오버헤드가 존재한다. 레이지 방법은 이거 방법과는 달리 데이터를 한꺼번에 전송하지 않고, 서버의 원격 프로시져가 데이터를 요구할때마다 콜백(callback) 등의 방법으로 클라이언트에게 요구하게 된다. 따라서 레이지 방법에서는 많은 량의 데이터를 전송하는 오버헤드는 없지만, 데이터를 요구할 때마다 RPC를 수행해야 되는 오버헤드가 존재한다.

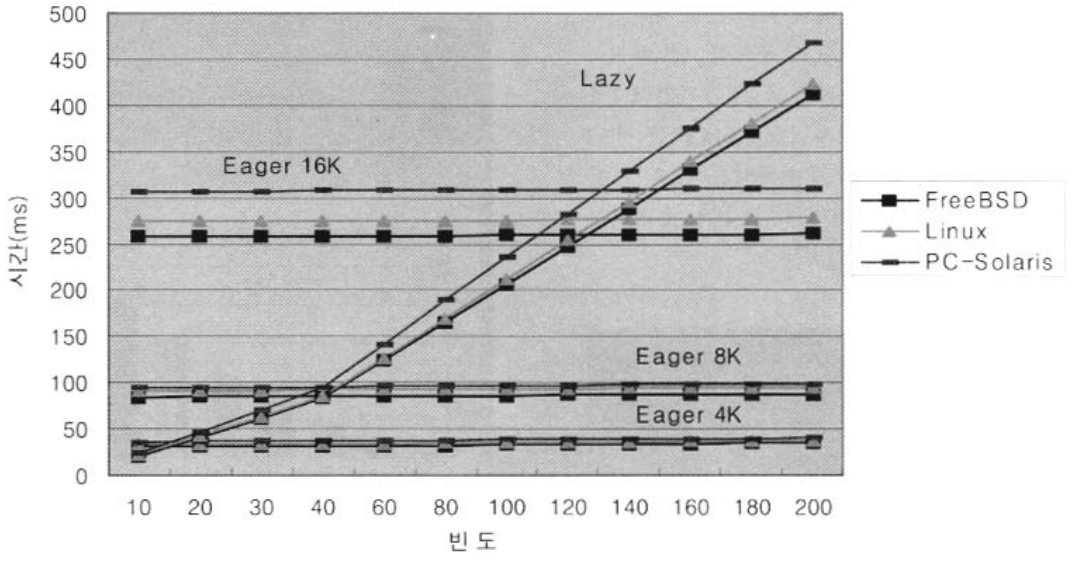
<표 3> 메시지전송속도(UDP)

	Linux	PC-Solaris
1k	5ms	6ms
2k	8ms	9ms
4k	12ms	12ms
8k	19ms	20ms
16k	34ms	35ms
32k	62ms	64ms



(그림10) 메시지 전송 시간(RPC)

본 논문에서는 RPC의 성능을 측정하기 위해 필요한 데이터의 크기와 데이터를 요청하는 빈도를 고려하여 RPC를 처리하는 시간을 측정하였다. 즉, 일정한 데이터 크기에 대해서 요청 빈도에 따른 이거와 레이즈 방법을 비교하여 측정하였고, 결과는 (그림 11)과 같다. (그림 11)에서 이거 방법은 데이터를 요청한 빈도에는 관계없이 한번에 전송하게 되는 데이터 크기에 따라서 RPC를 처리하는 속도에 차이를 보인다. 이거 방법에서는 대체적으로 FreeBSD가 우수한 것으로 측정되었다. 레이지 방법의 RPC 실행 시간은 데이터 크기와는 무관하고 (그림 11)과 같이 데이터를 요청한 빈도에 따라 증가하는 직선으로 나타났다. 레이지 방법인 경



(그림 11) 이거와 레이지 방법의 성능 비교

우에도 FreeBSD가 우수한 것으로 측정되었다.

(그림 11)에서 원격 프로시져가 요구하는 데이터 크기가 8K인 경우(Eager 8K) 이거 방법에서는 대략 100ms 정도의 처리 시간이 소요된다. 그러나 레이지 방법은 원격 프로시져가 데이터를 처리하기 위해 콜백을 몇번 수행했는가에 따라서 40번 미만인 경우는 100ms 이내에서 처리되지만, 40번 이상이 되면 이거 방법보다도 더많은 처리 시간이 소요된다. 따라서 UNIX 커널에서 RPC를 이용하여 프로그램을 개발하는 사용자들은 (그림 11)의 결과를 참조하여 데이터 크기와 요청 빈도에 따라 이거 또는 레이지 방법을 선택하여 구현할 수 있다.

마지막으로 본논문에서는 네트워크파일시스템(NFS : Network File Systems)의 성능을 비교하였다. 우선 (그림 1)의 매스터 노드가 각 UNIX시스템의 NSF 클라인언트가 되도록 NFS환경을 구축하고, 파일 입출력 시간을 비교하였다. 파일 입출력 시간에 큰 차이는 없었으나, Linux가 미세한 차이로 우수하게 측정되었다. 두 번째 열의 Solaris Local은 매스터노드 자체의 파일입출력 시간으로 NFS로 구축한 파일시스템의 성능이 로칼 파일을 접근하는 속도와 비슷한 성능을 가지고 있음을 보여준다. 그러나 (그림 8)에서 256k를 전송하기 위해 Linux인 경우에는 대략 1.2ms가 소요되고, <표 4>의 NFS인 경우에는 4.8ms가 소요되므로 대략 지역 파일 입출력보다 NFS 파일 입출력 4배 이상의 시간이 소용된다. 그러나, 파일 사용의 편리성과 네트워크 파일 시스템인 점을 고려한다면 NFS의 성능은 매우 우수한 성능으로 측정되었다.

〈표 4〉 NFS 파일 입출력

	Solaris Local	Linux	PC_Solaris
1k	0.06ms	0.06ms	0.06ms
4k	0.08ms	0.11ms	0.08ms
16k	0.29ms	0.29ms	0.28ms
64k	2.9ms	1.1ms	1.1ms
256k	4.6ms	4.8ms	5.2ms
1024k	227.6ms	224.3ms	228.1ms
2048k	444.9ms	444.2ms	447.4ms

5. 결론 및 향후 연구과제

지금까지 본 논문에서는 커널의 내부 구조와 알고리즘 등에 대한 사전 지식이 필요 없이 분산 환경에서 RPC를 이용하여 이종의 UNIX 시스템들의 성능을 측정하고 비교할 수 있는 방법을 제시하였다. 또한, 측정된 결과를 인터넷에 도시화함으로써 성능 측정 결과를 쉽게 볼 수 있도록 하였다. 본 논문에서 제안한 스톱워치 타이머는 μs 단위까지 측정 가능하며, 여러번의 측정 결과를 분석한 결과 신뢰성이 높은 것으로 판명되었다. 일반적으로 UNIX 커널의 성능 측정 결과 대체적으로 Linux2.0.32가 다른 운영체제에 비하여 비교적 우수한 성능을 보였고, FreeBSD와 PC-Solaris는

상대적으로 성능이 떨어지는 것으로 나타났다. 그리고 본 논문의 성능 측정 결과를 참조하여 RPC를 이용하는 네트워크 프로그래머들에게 이거 또는 레이지 방법의 RPC를 선택하여 구현할 수 있는 기준 정보를 제공할 수 있었다.

향후 연구 과제로는 본 논문에서 제안한 측정방법에 대한 신뢰성을 검증하여 커널의 성능을 측정하는 벤치마크 프로그램으로 개발하는 것이다. 또한, 단순히 성능측정 결과만을 제시하는 것이 아니고, 측정한 결과를 분석할 수 있는 도구를 개발하여 성능의 원인 규명을 할 수 있는 연구가 향후 연구과제가 될 수 있다. 또한, 네트워크 성능 측정 방법을 보완하여 CORBA와 DCOM등의 미들웨어들의 성능을 비교할 수 있는 방법을 연구하고자 한다. 그리고, 현재 구현된 성능 측정 요소들인 프리미티브 및 명령어 수행 시간등의 단순한 내용을 더욱 다양화하여 측정하는 방법들을 계속 연구중에 있으며 사용자가 사용하기 쉬운 사용자 인터페이스를 개발하고자 한다. 또한, 커널의 성능 측정 뿐만 아니고, 일반적인 분산 소프트웨어의 성능 측정 방법에 관하여 연구하고자 한다.

참 고 문 헌

[1] A. Sheeman, "Performance Analysis : An Introduction," Technical Report 89. 3, part No.19150, Mar. 1989.

[2] T. Bemmerl, "Distributed Monitoring Systems-A Basis for General Purpose Distributed Multiprocessors," Panel Section, Proc. of the Int'l Conf. on Distributed Computing Systems, p.380, May 1991.

[3] "오픈시스템 '95," 하이테크정보출판부, pp.241-256.

[4] Reinhold P. Weiker, "An Overview of Common Benchmarks," IEEE Computer, pp.65-75, Dec. 1990.

[5] Thomas M. Conte and Wen-mei W. Hwu, "Benchmark Characterization," IEEE Computer, pp.48-56, Jan. 1991.

[6] Steve D. Pate, "UNIX internals A Practical Approach," Addison-Wesley, 1996.

[7] Donald Lewine, "POSIX Programmer's Guide," O'Reily & Associates, Inc, 1991.

[8] W. Richard Stevens, "Advanced Programming in the UNIX Environment," Addison-Wesley, 1992.

[9] Bill O. Gallmeister, "POSIX. 4 : Programming for the Real World," O'Reilly & Assocoicates, Inc, 1995.

[10] John Bloomer, "Power Programming with RPC," O'Reilly & Associates, Inc, 1992.

[11] John Shapley Gray, "Interprocess Communications In Unix," Prentice Hall, 1997.

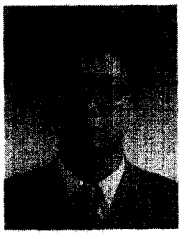
[12] SunSoft, SunOS[tm], "Nework Programming Guide," Sun Mircrosystems, 1993.

[13] IRIX[tm], "IRIX Network Programming Guide," IRIX, 1991.

[14] SunSoft, SunOS[tm], "XDR : External Data Representation," Request For Comments : 1014, Sun Microsystems, Inc, 1987.

[15] SunSoft, SunOS[tm], "RPC : Remote Procedure Call," Request For Comments : 1057, Sun Microsystems, Inc, 1988.

[16] K. Kono, K. Kato, and T. Masuda, "Smart Remote Procedure Calls : Transparent Treatment of Remote Pointers," Proceedings of the 14th International Conference on Distributed Computing Systems, June 21-24, 1994, pp.142-151.



박 윤 용

email : yypark@omega.sunmoon.ac.kr

1983년 숭전대학교 계산통계학과 졸업(학사)

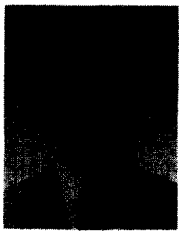
1985년 서울대학교 대학원 계산통계학과(이학 석사)

1994년 서울대학교 대학원 계산통계학과(이학 박사)

1985년~1992년 한국전자통신연구소 연구원

1992년~현재 선문대학교 컴퓨터정보학부 부교수

관심분야 : 분산처리운영체제, 분산객체지향처리시스템, 분산객체표준화



박 정 호

e-mail : jhpark@omega.sunmon.ac.kr

1980년 성균관대학교 사범대학 졸업(문학사)

1980년~1982년 성균관대학교 경영대학원 정보처리학과(경영학석사)

1985년~1987년 日本 오사까대학교 대학원 정보공학전공(공학석사)

1987년~1990년 日本 오사까대학교 대학원 정보공학전공(공학박사)

1996년~현재 한국정보처리학회 총무이사

1991년~현재 선문대학교 컴퓨터정보학부 부교수

1999년~현재 선문대학교 연구처장

관심분야 : 분산알고리즘, 원격교육, XML, 소프트웨어 공학

임 동 선

email : dslim@etir.re.kr

1986년 숭실대학교 전산학과(학사)

1996년 KAIST 정보 및 통신공학과(석사)

1986년~현재 한국전자통신연구원 선임연구원 실시간 OS팀장

관심분야 : 분산실시간처리, 멀티미디어처리