Goodus 기술노트 [35 회]
Open Solaris
& File-system vs Raw-device

Author	**유광복,조연철,백순성**
Creation Date	**2008-10-30**
Last Updated	**2008-11-06**
Version	**1.0**

Version	변경일자	변경자(작성자)	주요내용
1	2008-11-06	유광복,조연철,백순성	문서 최초 작성
2			
3			

Goodus
Breathing Life into Technology

Contents

1. Open Solaris

1.1. Open Solaris 의 등장

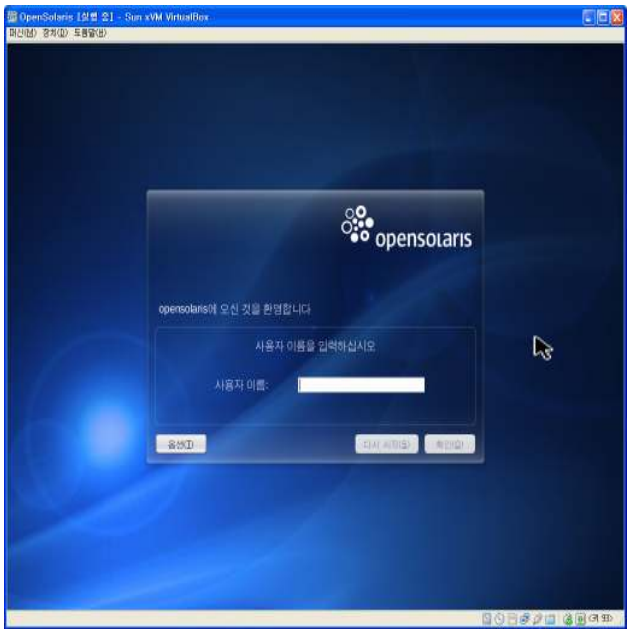
1990 년대 말, 이른바 닷컴(DOT.COM) 시대에 대부분의 백엔드 환경을 점유했던 Sun 의 솔라리스는 인터넷 환경을 주도하는 네트워크 환경에 가장 발전된 운영체제였다. 이러한 발전이 결국 Sun 의 자만심을 불러일으켰을까?

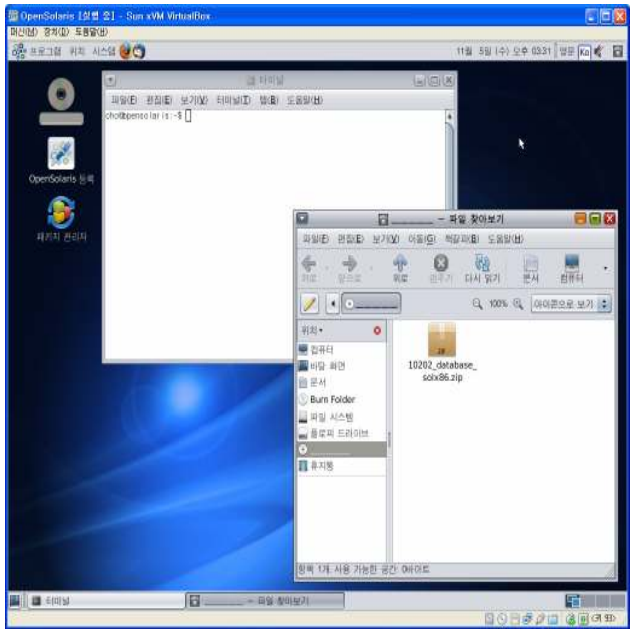
가정에서 사용되는 PC 가 윈도우와 리눅스 등의 발달로 단순한 클라이언트의 역할에서 서버로서의 기능까지 수행 가능해지면서 IT 서비스의 세계는 새로운 변화의 물결을 맞이하게 되었다. 무엇보다 리눅스의 획기적인 성장은 x86 플랫폼에 신선한 바람을 불어넣기 시작했고, 이제는 비교적 덜 중요한 업무 환경에서 많이 쓰이게 되었다.

사람들은 리눅스가 성장하게 된 가장 기본적인 배경으로 특히 소스 공개에 주목했다. 임의의 불특정 사용자들이 이용 중에 느껴지는 불편함을 참다못해 스스로 수정하게 되다 보니, 지금은 엔터프라이즈 운영체제와 맞먹는 기능들을 가지게 되었고, 이 과정에 직접 참여한 사용자들은 그 자신의 공헌을 외부에 알리기 위해 노력하면서 리눅스의 세계가 크게 성장했다고 볼 수 있다.

Sun 은 바로 이 부분에 주목했다. 끊임없이 첨단 기술을 개발해 추가하려면 엄청난 개발 비용이 필요했고, 이렇게 만들어진 솔라리스를 팔기 위해서는 역시 적지 않은 마케팅 비용이 요구되었다. 또한 첨단 기술을 개발하는 속도가 오픈 소스 계열에 점차 뒤쳐지기 시작했다는 점도 무시할 수 없었다. 물론, 오픈 소스 세계에서는 쉽게 상상할 수 없는 기능을 개발하기도 하겠지만, 그러한 기능이 시장에서 경쟁력을 가지도록 하기 위해서는 또 다른 노력이 필요했다.

이런 고민 끝에 Sun 은 마침내 자사의 지적 재산권 목록에서 큰 부분을 차지하고 있던 솔라리스의 소스를 공개 하기에 이르렀다. 이름하여 오픈 솔라리스(http://open solaris.org)가 등장한 것이다.





1.1.1. Source 공개의 힘

오픈 솔라리스는 기존의 유닉스 사용자들과 리눅스 사용자들에게 대단한 관심을 불러일으켰는데, 그

이유는 솔라리스를 보는 관점에 따라 다양했다. 유닉스의 고비용 때문에 리눅스로 옮겼던 사용자들은 오픈 솔라리스를 통해 다시금 유닉스(솔라리스)로 돌아갈 수 있게 된 점을 매우 크게 반겼다. 오픈된 솔라리스를 통해 저비용으로 엔터프라이즈 수준의 운영체제를 사용할 수 있게 되었기 때문이다. 또한 기존의 솔라리스를 사용하던 사용자들은 오픈된 솔라리스의 소스를 통해 솔라리스의 커널 구조, 디바이스 관리 구조 등을 완벽하게 이해함에 따라 보다 안정된 구조의 솔라리스 환경을 구축할 수 있게 되었다

1.1.2. 개발자를 위한 새로운 선택, 오픈 솔라리스

오픈 솔라리스는 취약한 보안성에 시달려온 리눅스 관리자들에게도 희소식이었다. 기존까지 미국방성으로부터 유일하게 인정받은 보안 기술을 가진 솔라리스를 무상으로 사용하게 됨에 따라서 전체적으로 혹은 선택적으로 솔라리스 기반의 강력한 보안 환경을 구축할 수 있게 되었기 때문이다. 아울러 리눅스 및 유닉스용 보안 툴을 개발해 제공했던 보안 개발업체들도 솔라리스의 발전된 보안 기술을 참조할 수 있게 되었고, 이는 보다 향상된 보안 솔루션을 만들 수 있는 계기로 작용했다.

1.1.3. CDDL기반의 오픈

개발자들을 위한 오픈 솔라리스의 이런 특성 외에 법률적인 관점에서도 주목할 만한 특징이 하나 있다. 오픈 솔라리스는 기존의 오픈소스 라이센스인 BSD 라이센스의 변형이자 Sun 의 새로운 배포 라이센스인 CDDL(Commnon Development and Distribution License)을 기반으로 소스를 공급한다. 이는 기존의 GPL 기반의 라이센스와는 매우 다르다. GPL 은 GPL 하의 소스 코드를 참고하여 작성된 모든 결과물은 오픈되어야 하는 반면에, CDDL 은 오픈 솔라리스의 소스 코드를 참고해 작성된 결과물이라 하더라도 그 소스를 반드시 오픈할 필요가 없다. 이는 오픈소스를 기반으로 작성한 애플리케이션을 판매하는 업체들에게 매우 유리한 조건이 아닐 수 없다. 기존의 GPLv2 기반의 소스들을 참고해 만든 것을 수출했다가 GPL 위반 사유로 소송당한 업체들이 점점 증가하는 것으로 보아 오픈 솔라리스의 CDDL 은 오픈소스를 참조해 새로운 솔루션을 개발하는 업체들에게 매우 매력적인 배포 라이센스가 아닐 수 없다.

1.1.4. 우수한 안정성

한편 리눅스 운영체제의 또 하나의 중요한 특징으로 다수의 자발적인 프로그래머들에 의해 구현 및 구성된다는 점을 꼽을 수 있다. 사실 이런 상황에서 코드의 안정성을 확보하는 것은 매우 어렵다. 따라서 리눅스는 다소 독특하게 '안정판(stable edition)' 이라던가 '베타판(development release)' 이라는 이름으로 다양한 릴리즈가 소개되고 있고, 이러한 릴리즈를 테스트해서 제공하는 배포판 업체들 역시 난립하고 있다.

이와 달리 솔라리스는 기존에 어느 정도 수준 이상으로 완성되어 있는 코드 위에서 확장하면서 호환성을 유지하도록 프로젝트 커뮤니티의 통제가 이뤄지기 때문에 매 릴리즈 사이에 새로운 기술을 추가하면서도 그 호환성을 충분히 유지할 수 있도록 운영되고 있다. 물론, 현재 3~4 개 정도인 오픈 솔라리스 배포판의 종류가 더 늘어나면 지금의 리눅스에서와 같은 문제가 없으리라고는 보장하기가 힘들지만, 리눅스 계열보다는 적을 것으로 예상된다.

1.1.5. 각 배포판의 특징

오픈 솔라리스는 Sun 에서 공급하는 3 개의 배포판을 제외하고 현재 4 개의 배포판이 제공되고 있다. Sun 에서 제공하는 3 가지는 각각 Solaris Express, Solaris Express Community Release, Sun Solaris 이다. 이 가운데 Solaris Express Community Release 가 오픈 솔라리스를 바탕으로 만드는 가장 최신 빌드로, 이는 업데이트 서비스가 없는 무상 제품이다. 반면, 여기에 업데이트 서비스를 제공하는 배포판은 Solaris Express 에 해당된다. 그리고 이 최신 빌드를 기반으로 Sun 의 추가 테스트와 별도 소프트웨어 패키지를 추가한 배포판이 바로 Sun 솔라리스(Sun Solaris)가 되는 셈이다.

1.1.6. SunOs 5.11 커널

오픈 솔라리스의 커널은 SunOS 로 명명되어 있다. 레드햇이나 수세 리눅스의 커널이 Linux 로 표시되는 것과 같은 개념이라고 할 수 있다. 현재 쓰이고 있는 SunOS 5.11 은 최신 솔라리스 10 의 기본이 되는 커널 버전으로 네트워크 서비스, 분산 컴퓨팅, SMP & NUMA, 멀티코어와 같은 최신 기술 모두에 최적화되어 있다. 이 버전은 유닉스 및 유사 유닉스 운영체제 가운데 가장 진보된 커널로 자타가 인정하고 있다. 특히 솔라리스는 아주 이른 시기부터 멀티코어에 최적화되어 있고, 심지어 AMD 의 옵테론이 구성하는 유사 NUMA 구조에도 최적화되어 있다. 최신의 저가 하드웨어 기술에도 적합한 운영체제인 셈이다.

```
# uname -a
SunOS opensolaris 5.11 snv_86 i86pc i386 i86pc
```

Sun 은 솔라리스에 컴퓨팅 이론을 구현하는 데 많은 공을 들였고 이러한 노력 덕분에 솔라리스의 명성이 유지될 수 있었다. 솔라리스 10 의 등장과 오픈 솔라리스로 변신한 이후, 그러한 솔라리스의 사고와 기술은 다른 운영체제에도 지속적으로 영향을 주었다.

솔라리스의 커널은 특히 SMP 시스템에 가장 안정적이고 최적화된 것으로 알려져 있으며 실제로 Sun 은 업계에서 유일하게 최대 144 개의 CPU 코어를 가지고 있는 단일 박스(SMP)를 솔라리스와 함께 제공하고 있다. 많은 CPU 를 운영할 수 있다는 것은 커널 자체의 병렬화가 우수함을 의미하는 것이다. 때문에 솔라리스는 우수한 멀티스레드 하부 시스템을 제공하고 있다. 웹 서버와 같은 인터넷 서비스들은 대개 멀티 스레드를 이용해 작동하도록 구성되는 것이 최근의 흐름임을 감안하면, 솔라리스가 인터넷 서비스에 매우 적합하다는 점을 쉽게 예측할 수 있다. 솔라리스 커널의 특징을 간략하게 요약하면 다음과 같다.

● 시스템 콜 인터페이스

모든 유닉스 및 유사 유닉스 운영체제가 그렇듯이 모든 사용자 프로세스는 커널과 커뮤니케이션하기 위해 시스템 콜을 써야한다.

● 프로세스 관리와 스케줄링

프로세스의 생성, 실행, 관리, 종료 등과 같은 모든 프로세스/스레드 관련 행위를 관리하며, 스케줄러는 이러한 스레드에 대해 우선순위 기반의 스케줄링을 지원한다. 완벽한 우선순위 기반의 사전 스케줄링(preemption)을 지원함으로써 실시간 애플리케이션을 지원한다.

● 가상 메모리 시스템

가상메모리 시스템을 만들어 실제 메모리를 사용자 프로세스와 커널에 제공한다. 솔라리스 메모리 관리는 크게 2 개의 계층으로 구분된다. 하나는 공통 메모리 관리 계층이고 다른 하나는 하드웨어 지원 계층이다.

● 파일시스템 인터페이스

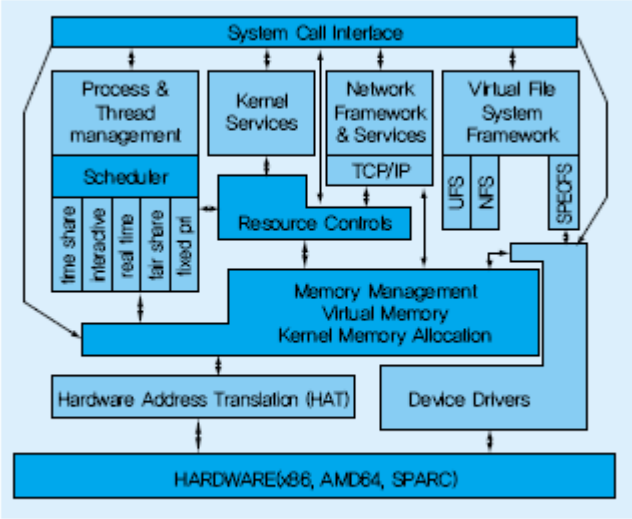
솔라리스는 가상 파일시스템 프레임워크를 구현한다. 때문에 다양한 종류의 파일시스템(ufs, nfs, hsfs, specfs, procfs, sockfs 등)이 가상 파일시스템 내부에 장착되어 사용자가 모든 형태의 파일시스템을 일관된 방법으로 접근, 제어할 수 있도록 한다.

● 구조적 I/O 버스와 장치 관리

솔라리스 I/O 프레임워크는 버스를 중심으로 연계되어 있는 장치를 구조적으로 (hiercarchical) 관리하도록 되어있다.

● 네트워킹

TCP/IP 프로토콜 관련 기능을 지원한다. SunOS 5.10 이후부터는 이전 커널과 달리 TCP 와 IP 의 드라이버를 단일 드라이버로 통합함으로써 레이어간에 발생할 수 있는 지연시간을 단축했다.



<그림 2> 솔라리스 커널 인터페이스

1.2. Open Solaris 의 주요 기능

이와 같은 SunOS 5.10 을 기반으로 작성된 Sun Solaris 최신 배포판은 다수의 새로운 기능을 가지고 있다. 그 중에 커널과 관련이 있는 주요 기능을 정리하면 다음과 같다.

- 자가 치유 기능
- 솔라리스 서비스 관리
- 솔라리스 장애 관리

- 솔라리스 존(가상서버)
- 동적 자원 풀
- 동적 트레이싱 툴(Dtrace)
- 프로세스 권한 관리
- 솔라리스 보안 확장셋(Trusted Extension)
- ZFS

1.2.1. 솔라리스 장애 관리, 서비스 관리 기능과 자가 치유 기능

솔라리스의 자가 치유 기능은 기술적으로 솔라리스 서비스 관리 기능과 솔라리스 장애 관리 기능의 구현으로 인해 자동으로 생겨난 솔라리스의 기능이라고 볼 수 있다. 솔라리스 장애 관리자는 솔라리스 내에 구성된 장애 감시 대상 오브젝트들을 실시간으로 감시하다가 장애 발생시 이에 대한 통보와 조치를 취하게 되는데, 이때 부연으로 발생하는 서비스 장애에 대응할 수 있도록 생성된 것이 솔라리스 서비스 관리 기능이다. 예를 들어, 특정 CPU 에 장애가 발생해서 운영체제가 그 CPU 를 오프라인(offline) 시켜야 하는 경우를 생각해보자.

이때 강제 오프라인을 하게 되면 그 CPU 에서 동작하던 애플리케이션은 우선 정지(kill)된다. 이때 서비스 매니저는 정지된 해당 서비스를 새롭게 자동으로 시작할 수 있도록 지원해줌으로써 서비스는 자동으로 복구된다. 덕분에 기존의 시스템과 서비스의 가용성을 크게 늘릴 수 있다. 설사 서비스가 죽었다 하더라도 빠르게 서비스를 재가동할 수 있도록 지원함으로써 관리자는 서비스 중지에 따른 피해를 최소화할 수 있다는 얘기다.

1.2.2. 솔라리스의 가상화와 존(솔라리스 컨테이너)

솔라리스의 존은 최근 강력해진 하드웨어의 성능을 바탕으로 발전하고 있는 가상화 기술의 일종이다. 흔히 알려져 있는 가상화의 대표주자인 VMWARE 의 방식과는 달리, 게스트 영역에 별도의 운영체제를 사용하지 않고 호스트 운영체제를 빌려 쓰는 기법을 사용한다. 기술적으로 하나의 운영체제를 마치 여러 개의 운영체제가 있는 것처럼 보이게 만드는데, 이는 자원관리 기능을 확장한 기술로 구현한 가상화 기법이라고 할 수 있다.

이 기법은 VMWARE 류의 가상화 기술이 가지고 있는 오버헤드를 완전히 없앰으로써 동일 운영시스템, 혹은 다른 버전의 동일 운영 시스템에서 서비스할 수 있는 애플리케이션들을 가상화 영역으로 분리해 하드웨어 활용률을 극대화할 수 있다. 뿐만 아니라 각 가상화 영역별로 관리의 독립과 보안성 개선, 안정성 개선 등을 이뤄낼 수 있다. 기존에는 단일 솔라리스 영역에 웹 서버, DB 서버, 웹 애플리케이션 서버들을 모두 설치해 사용했다면, 여기서는 존을 이용해 각각을 분리함으로써 용도별 분할 관리가 가능해지고 더 쉬어졌다. 아울러 전체적인 관리의 효율성도 증가해 관리에 소요되는 시간도 줄일 수 있게 되었다.

또한, 오픈 솔라리스 build 49 에는 x86 솔라리스만을 위한 존 기능이 추가되어 있는데, 존 영역에 솔라리스 버전뿐만 아니라 32 비트 레드햇 계열의 리눅스도 설치할 수 있게 되었다. 이 기능 덕분에 인텔이나 AMD 프로세서 환경에 설치된 솔라리스 안에는 32 비트 리눅스도 함께 설치해 운영할 수 있게 되어, 솔라리스로 이동하려는 관리자나 리눅스의 애플리케이션이 어떻게 동작하는 지를 솔라리스의

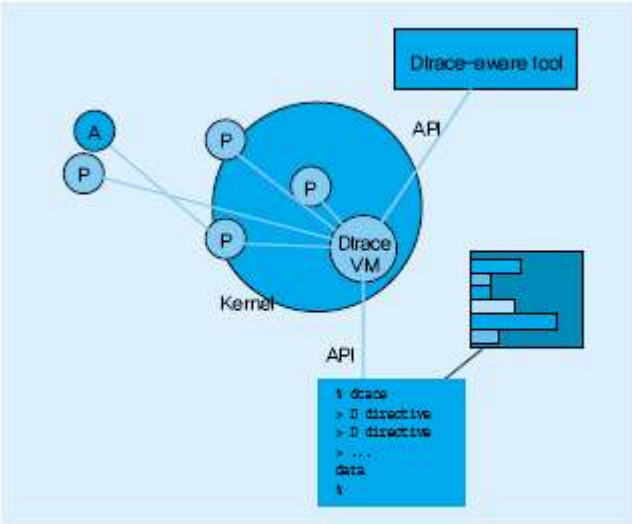
Dtrace 로 추적해보고 싶은 개발자들이 흥미를 느끼기에 충분하다.

1.2.3. 동적 자원 관리 풀(Pool)

솔라리스는 풀(pool) 개념을 도입하여 자원을 관리할 수 있는 기능을 제공한다. 필요한 작업(project, task, zone)에 필요한 양만큼의 자원(CPU, 메모리)등을 동적으로 할당해 사용할 수 있게 해준다. 즉 특정 프로젝트나 업무를 만들어 그 영역에 필요한 만큼 할당해 사용하고, 필요가 사라지면 자원을 동적으로 조절할 수 있게 하는 것이다. 존도 기술적으로는 일련의 업무(project) 모임이므로 동적 자원 관리의 대상이다. 오픈 솔라리스에서는 이러한 동적 자원 관리가 한층 지능화되어 CPU 사용률에 따라 CPU 의 자동 추가 등을 구성할 수 있다.

1.2.4. 동적 자원 관리 툴(Dtrace)

솔라리스 10 에 처음으로 추가된 동적 추적 툴인 Dtrace 는 그야말로 IT 업계에 커다란 충격을 줬다. 기존의 운영체제들이 제공해왔던 사후추적 방식의 툴이 지닌 한계를 벗어나서 실시간으로 이벤트 중심의 동적 추적을 업계에서 최초로 구현했기 때문이다.

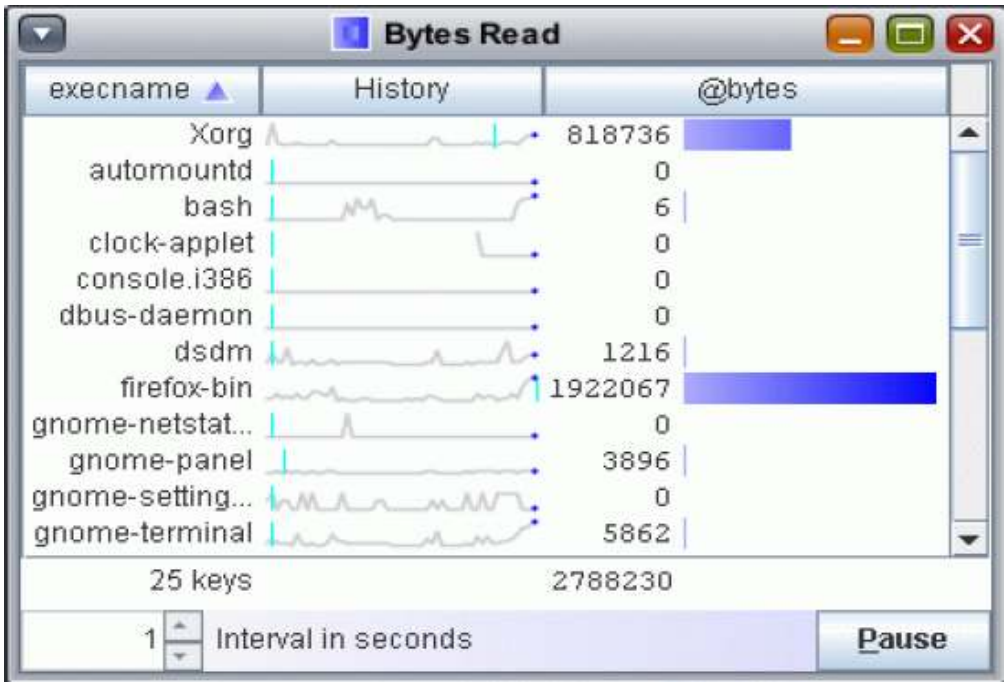


<그림 3> Dtrace 개념도

모든 시스템 및 운영체제에서 그렇듯이 정상적인 경우에는 문제될 것이 없다. 늘 비정상적인 경우에 문제가 발생하며, 그 대부분은 예측하지 못했을 때다. 따라서 주로 업무가 마비된 이후에 문제를 알게 되므로 그 비정상적인 경우를 재생, 추적하기란 결코 쉽지 않다. 이때 Dtrace 는 예상된 영역에 덫(?)을 놓아 특정 이벤트가 실행되면 보고할 수 있게 한다. 이 기술을 이용하여 솔라리스 커널 자체의 버그, 시스템 튜닝, 애플리케이션의 메모리 릭(memory leak)이나 병목 등을 예전과 달리 훨씬 쉽게 추적할 수 있다.

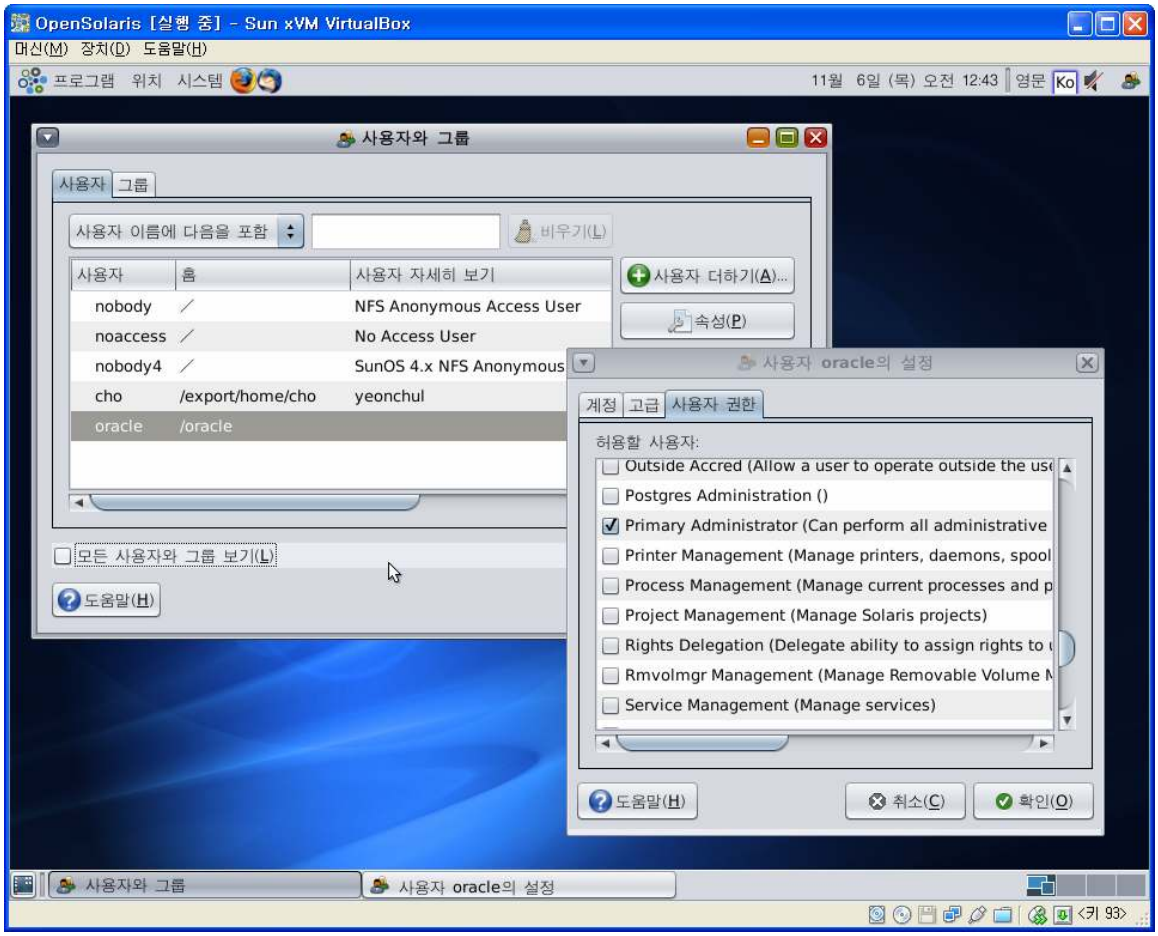
```
# dtrace -l | head
   ID   PROVIDER            MODULE                          FUNCTION NAME
    1     dtrace                                                 BEGIN
    2     dtrace                                                 END
    3     dtrace                                                 ERROR
                                        .
                                        .
66585     syscall                                          umount2 entry
66586     syscall                                          umount2 return
66587     profile                                                profile-97
66588     profile                                                profile-199
```

Dtrace는 그 기능을 통해 이미 타 운영체제 개발자들에게 많은 영감을 주었다. 앞에서 언급한 바와 같이 애플이 Dtrace 기능을 MaxOSX의 차기 개발 버전(Machintosh OSX Leopard 버전)의 Xcode에 포팅해둔 것만 봐도 얼마나 유용한 기능인지를 짐작할 수 있다. 이처럼 Dtrace는 커널 중심이 아니라 프로세스 중심이면서 데이터와 사용자 중심의 관점으로 접근하는 객체 지향적인 기술을 이용해 처리의 흐름을 파악한다. 이를 통해 비정상 문제를 야기하는 근본적인 원인을 빠르게 추적할 수 있다



1.2.5. 프로세스 권한 관리

전통적인 유닉스 계열에서는 root, 즉 슈퍼 유저이면 모든 것을 할 수 있었다. 실제로 슈퍼 유저는 특정 업무를 위해서만 사용되어야 함에도 불구하고, 많은 곳에서 root를 노출해 사용하거나 공유하고 있다. 이렇게 되면 자연스레 시스템은 보안에 매우 취약해질 수밖에 없다. 솔라리스는 기본적으로 root가 할 수 있는 모든 업무들을 일정한 범주의 역할로 구분하고 사용자들에게 역할을 할당한다.



이로써 root 로 해야 하는 업무를 일반 사용자로 수행하면서도 root 가 할 수 있는 실수를 원천적으로 방지할 수 있고, 동시에 이러한 시스템의 이용을 통해 실질적인 root 사용자를 완전히 없앨 수도 있다. 이 기능은 솔라리스 하부에 있는 RBAC 와 함께 이용되어 특정 프로세스에 대한 권한을 조정할 수도 있다. 이처럼 특정 사용자나 특정 역할에 대해 권한을 조정할 수 있게 됨으로써 한층 강화된 보안과 관리가 가능하다.

1.2.6. ZFS

Sun 이 솔라리스를 오픈하기 이전부터 해결하고자 했던 하나의 프로젝트가 있었는데, 그것은 바로 지난 30 년간 솔라리스의 기본 파일시스템이었던 UFS 를 대체할 만한 파일시스템을 만드는 것이었다. 이 프로젝트의 결과는 오랜 검토와 코딩, 그리고 테스트를 거쳐 등장하게 되었는데, 마침 솔라리스의 오픈화와 시기가 맞물려 오픈소스의 세계에 모습을 나타내게 되었다. 그것이 바로 ZFS 이다.

- 자동 데이터 손상 방지
- 적극적 체크섬
- 최초의 128 비트 파일 시스템
- 최대용량 : 10 억 TB
- 즉각적인 스냅샷

- 0 에 가까운 오버헤드
- 손쉬운 롤백

이 파일시스템은 시스템 관리자의 디스크/스토리지 관리에 들어가는 수고를 대폭 줄여주면서도 서비스 장애 및 데이터 장애 발생률을 현격하게 낮춰준다. 24 시간으로 일주일 내내 수행되어야 하는 인터넷 서비스나 중요 서비스 업무에 사용될 수 있도록 구성되어 있어 미션 크리티컬한 상황에서 최적의 파일 시스템이라고 할 수 있다. 또한 ZFS 는 솔라리스의 실시간 장애 관리 오브젝트로 등록되어 있어서, 관리자가 언제든지 파일시스템의 장애를 실시간으로 관측 및 관리할 수 있다. 이 ZFS 도 Dtrace 와 함께 애플에 채택되어 Mac OSX 버전에 적용될 전망이다.

Sun 은 운영체제를 만드는 입장에서 관리자의 작업 종류와 난이도, 소요 시간 등을 분석했다. 그 결과, 시스템 관리자는 스토리지 관리와 파일시스템 구축 및 변경, 그리고 복구에 매우 많은 시간을 보내고 있다는 흥미로운 결과를 얻었다. ZFS 이전의 모든 파일시스템들은 스토리지(디스크)에 문제나 변화가 발생하면 시스템과 서비스가 모두 중지되어야 했기 때문이다. 기존 데이터는 모두 백업 및 복구되어야 하고 새로운 스토리지로 새로운 구성을 마치고 나서야 새로운 파일시스템을 구축할 수 있었던 것이다. 물론 대체 시스템으로 서비스를 제공하고 있는 기업도 있겠지만, 이는 곧 적지 않은 추가 투자를 의미하고, 나아가 사람과 서비스가 기계에 종속되어 있음을 반증하는 결과이다.

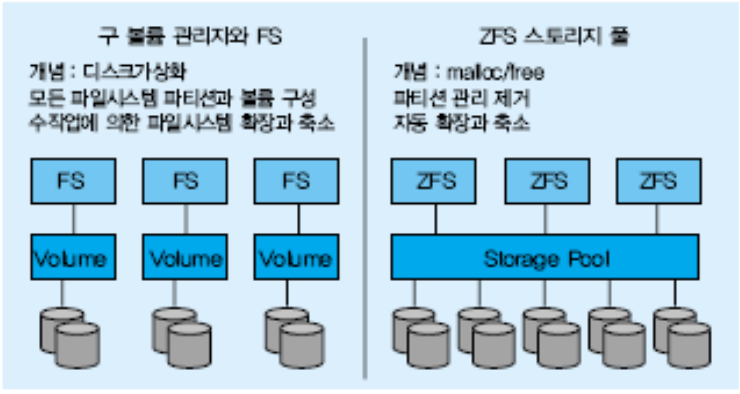
따라서 솔라리스의 파일시스템 개발자는 관점을 바꿔 디스크나 스토리지 중심이 아니라 사람과 서비스를 중심으로 하는 파일시스템에 초점을 맞췄다. 시스템에 디스크와 관련되어 어떠한 일이 발생해도 서비스는 절대 멈추지 말아야 하며, 서비스 가동 중에 디스크나 스토리지는 얼마든지 추가 및 제거할 수 있어야 한다. 또한 얼마든지 이전 구성으로도 되돌아갈 수 있는(filesystem undo), 그런 환상적인 파일시스템을 고려하게 된 것이다. 사실 이런 기능에 가장 근접한 기술은 이미 존재했는데, 그것은 바로 솔라리스의 메모리 관리 기술이었다. 시스템 사용자들은 메모리를 장착하면서 메모리를 어떻게 나누고 구성할지에 대해 전혀 신경 쓰지 않는다. 운영체제는 전체의 메모리를 하나의 커다란 풀로 운영하면서 필요한 요청에 대응한다. ZFS 는 바로 이점에 착안해 만들어졌다.

```
# zpool list
NAME     SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
oracle  7.75G   110K  7.75G     0%  ONLINE  -
rpool   14.9G  11.5G  3.42G    77%  ONLINE  -

# zfs list
NAME                          USED  AVAIL  REFER  MOUNTPOINT
oracle                        105K  7.63G    18K  /oracle
rpool                        11.5G  3.18G    55K  /rpool
rpool@install                  16K      -    55K  -
rpool/ROOT                   11.5G  3.18G    18K  /rpool/ROOT
rpool/ROOT@install               0      -    18K  -
```

```
rpool/ROOT/opensolaris                  11.5G  3.18G  11.4G  legacy
rpool/ROOT/opensolaris@install          53.4M      -  2.24G  -
rpool/ROOT/opensolaris/opt              5.03M  3.18G  5.00M  /opt
rpool/ROOT/opensolaris/opt@install        33K      -  3.60M  -
rpool/export                            584K  3.18G    19K  /export
rpool/export@install                       0      -    19K  -
rpool/export/home                       564K  3.18G   546K  /export/home
rpool/export/home@install                18K      -    21K  -
```

ZFS 역시 모든 스토리지를 하나의 거대한 풀로 구성 운영한다. 구성된 풀에서 필요한 만큼의 파일시스템을 만들고 변경하며 운영할 수 있도록 하는 것이다. 따라서 새로운 디스크를 추가할 때 어떻게 파일시스템을 나눌지, 혹은 파티션을 어떻게 나눌지에 대해 전혀 고민할 필요가 없다. 풀이 허용하는 한 파일시스템의 크기는 마음대로 바꿀 수 있고, 필요에 따라서는 특정 상태 저장(snapshot)을 다수 해두었다가 원하는 상태로 복구(undo)할 수 있게 되었다. 따라서 이론적으로 볼 때 풀의 용량이 허용하는 한다면 백업도 필요 없게 되었다.



<그림 4> ZFS 아키텍처

또한 ZFS는 데이터뿐만 아니라 메타데이터에 대한 Checksum을 유지함으로써 데이터 장애시의 복구율을 혁신적으로 높였다. 즉, 데이터의 장애로 인한 데이터 분실을 고민할 필요가 사라진 것이다. 더불어, ZFS는 미러 방식으로 구성된 환경에서 한 디스크에 장애가 발생하면 문제없는 디스크로부터 정상적인 데이터를 가져와 장애가 발생한 디스크를 치료하는 기능까지 제공한다. 이런 점에 비춰볼 때 ZFS는 특히 용량은 크지만 관리의 수고로움이 많았던 저가의 스토리지에 유용하게 적용될 수 있다.

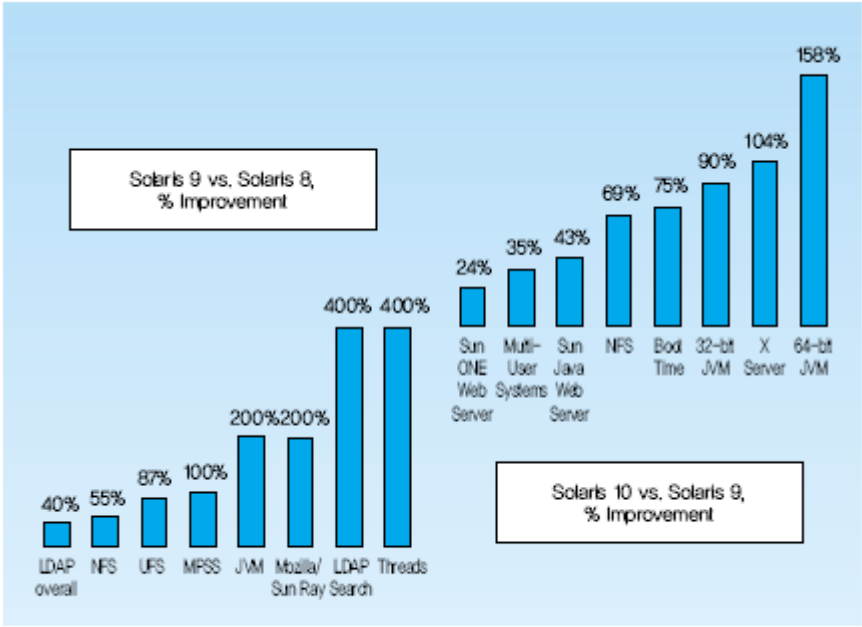
1.2.7. 솔라리스 트러스티드 익스텐션

최신 오픈 솔라리스 b54 및 Sun 솔라리스 업데이트3(11/06)에는 '솔라리스 Trusted Extension(이하, TE)'이라는 솔라리스의 확장된 보안 기능이 추가되었다. 솔라리스 TE는 선택적으로 구현되는 규칙 기반의 다층구조와 데이터에 권한 표식을 붙여 보안을 고차원적으로 확장시킨 기술이다. 미 국방성과의 협력으로 만들어진 Trusted Solaris 8의 기능을 오픈 솔라리스에 구현한 셈이다. 이 솔라리스 TE가

활성화된 환경은 보안등급 B2 에 준하는 등급으로 LSPP, RBAC 를 준수할 뿐 아니라 기존에 DAC, MAC 으로 알려져 있는 보안 정책들을 지원한다.

1.2.8. 솔라리스의 성능과 안정성

운영체제를 논하면서 성능을 논하지 않을 수는 없다. 솔라리스는 오랜 기간 서버용 운영 체제로 많이 사용되면서 안정된 서버로서 작동할 수 있도록 많은 부분에서 가감이 이뤄져왔다. 일반적으로 데스크탑 상의 작동을 우선시 하는 운영체제들은 사용자 동작(사용자와의 인터페이스)에 더 높은 우선 순위를 할당하면서 백그라운드 모드로 동작하는 서비스에는 신경을 덜 쓰는 구조를 지닌다. 하지만 솔라리스는 상대적으로 백그라운드 서비스에 더 많은 기회를 주도록 구성되어 있다(물론 이를 변경할 수는 있다). 이러한 특징은 솔라리스가 네트워크와 스레드 기반의 서비스에 최적화되어 있음을 감안할 때 쉽게 짐작할 수 있는 부분이다.



〈그림 5〉 솔라리스의 성능 변화

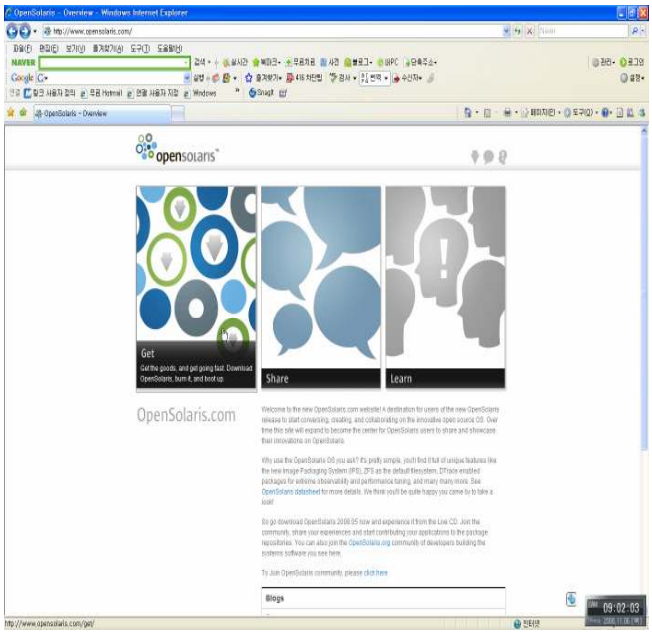
또한, 서버로서 전원 장애나 시스템 장애가 발생했을 때 데이터의 손실을 최소화하기 위해 성능 감소에도 불구하고 쓰지 않는 기능들도 있다. 대개 마이크로소프트의 윈도우 계열 서버나 리눅스와 같은 운영체제는 SCSI 디스크를 이용할 경우 디스크 내에 내장된 캐시를 자동으로 활성화함으로써 디스크 쓰기가 좋아지도록 구성한다. 그러나 솔라리스는 기본 설정상 읽기에 한해서만 캐시를 구성하고, 쓰기에 대해서는 캐시를 사용하지 않도록 한다. 이로써 전원 이상이나 시스템 장애가 발생했을 때 데이터의 손실을 최소화하는 것이다. 물론 디스크 쓰기 성능을 향상시키고 싶다면 쓰기 캐시를 활성화 하는 게 좋다.

이처럼 운영체제를 설계하는 몇 가지 관점의 차이 탓에 솔라리스는 멀티스레드와 네트워크 부분의 우수성에도 불구하고, 리눅스나 윈도우 계열에 비해 특정 부분에서 성능이 뒤처지는 것으로 평가되기도 한다. 하지만 솔라리스는 시스템을 멈추게 할 만큼 서비스 부하가 걸릴 때 그 진가를 발휘한다.

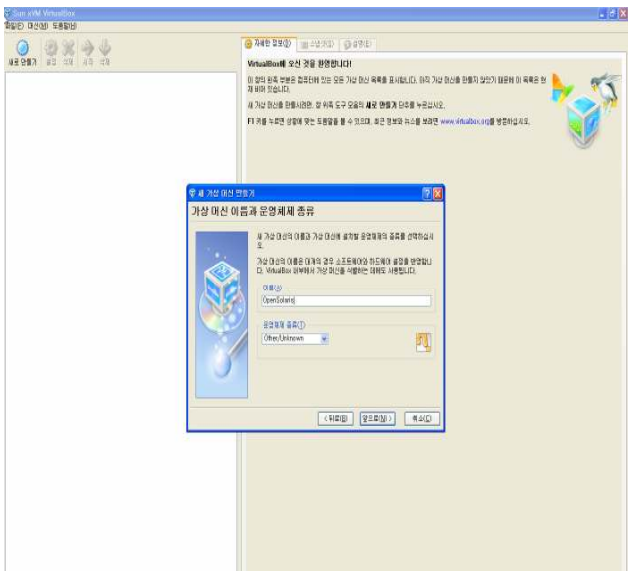
감당하기 어려울 만한 엄청난 부하가 주어졌을 때 타 운영체제는 묵묵부답이겠지만, 솔라리스는 다소 느리더라도 응답을 할 것이다. 때문에 관리자들이 부하에 대응할 수 있는 시간을 벌 수 있다. 이처럼 안정된 서비스가 우선이라면 솔라리스를 염두에 둘 필요가 있다.

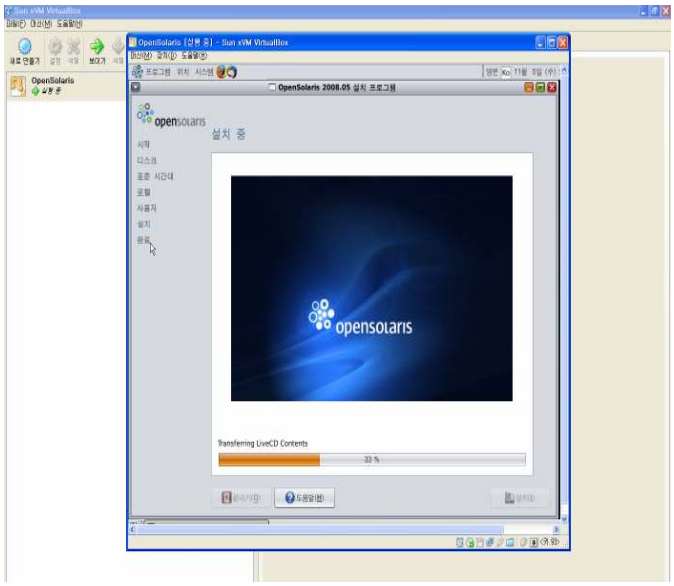
1.3. Open Solaris 다운로드 및 설치

오픈 솔라리스의 설치/사용을 원하는 경우, 설치파일은 홈페이지(http://www.opensolaris.com)에서 무상으로 다운로드를 받을 수 있으며, 오픈 솔라리스 패키지가 들어있는 LiveCD 를 신청하여 받아 볼 수도 있다.(Mail 화면에서 좌측의 "get" 선택 -> 다음 화면에서 "Download" 또는 "Free CD order"선택)



오픈 솔라리스는 기존의 방식과 마찬가지로 GUI 환경에서 손쉽게 설치가 가능하다. CD-ROM 으로 부팅하면 언어선택과 하드디스크, 시간대 등을 선택하고 User 를 생성하면 바로 설치가 된다. 아래 그림은 SUN 의 Virtual Machine 인 "VirtualBox"를 이용하여 설치하는 화면이다. VirtualBox 를 이용하면 LiveCD 의 x86 버전을 손쉽게 설치하여 OpenSolaris 를 사용할 수 있다.(http://www.opensolaris.com 참조)



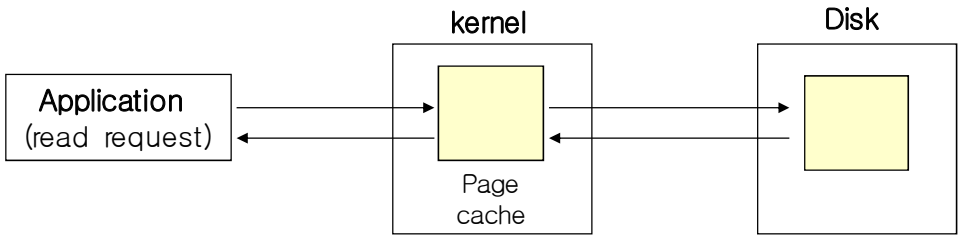


2. File-system vs Raw-device 성능 비교

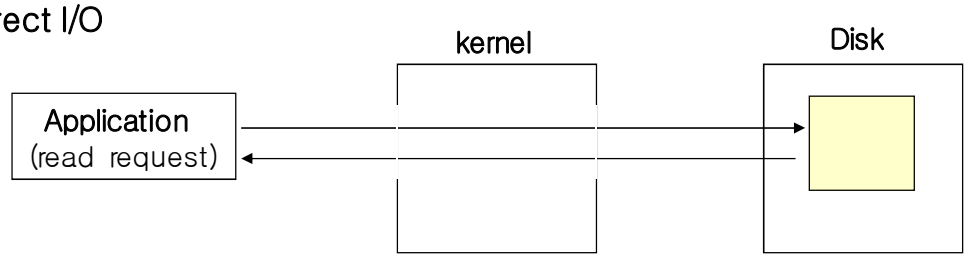
2.1. Buffer I/O(file system)과 Direct I/O(raw device) 비교

2.1.1. Buffer I/O, Direct I/O 개념도

Buffer I/O (page cache)



Direct I/O



2.1.2. Buffer I/O (page cache)

화일 시스템을 통한 I/O 는 커널의 Page Cache 를 거치며, 그 데이타는 Page Cache 알고리즘에 의해 커널에 보관된다.

```
# mount  /dev/dsk/c0t1d0s0  /oradata  (default)
```

2.1.3. Direct I/O (forcedirectio, raw device)

파일 시스템에서 커널의 Page Cache 의 기능을 사용하지 않는 방법과 raw device 를 사용하는 방법이 있다.

```
# mount  -o forcedirectio  /dev/dsk/c0t1d0s0  /oradata
```

화일 시스템을 UFS Direct I/O 로 마운트하면, Page Cache 기능을 사용하지 않는다.

Block device 가 아닌 character 방식으로 Disk I/O 가 전송이 되며 file system 방식이 아니다.

- 오라클에서 Raw device 사용 가능

  : data file, control file, redo log file

- 오라클에서 Raw device 사용 불가능

  : archived log file

* Os level 에서 raw device 를 먼저 생성한 후 oracle user 가 소유권을 갖고 있어야 사용이 가능하다.

* dd 명령어로만 control 가능 (cp, tar 사용 불가)

2.1.4. File system vs Raw device

	File system	Raw device
장 점	- Datafile 에 대한 Resizing 이 가능하다 - 디렉토리 구조를 사용하기 때문에 관리의 편의성이 있음 - 사용명령어 : cp, rm, mv	- 양쪽 Node 에서 ACTIVE, ACTIVE 로 Mount 한다. - 향후 RAC 전환이 용이하다. - File system 에 비해 성능이 우수 - 사용명령어 : dd
단 점	- Raw Device 를 사용할 경우 보다 성능이 떨어진다. - 시스템에 대한 Over head 가 발생 된다 (CPU,Memory)	- Data File 을 미리 나누어 놓고 소유권을 Oracle 로 지정해야 함 - Extents Size 에 맞는 적절한 Datafile 추가가 어렵다 - File system 에 비해 관리가 어려움

2.2. File system 과 Raw device 성능 비교

2.2.1. 테스트 환경

- H/W Spec	Model : Sun Ultra 60 CPU : 400M * 2ea Memory : 1.5 G O S : Sun Solaris 10
- Disk management	Raw device, File system with Veritas Volume Manger
- Oracle Version	Oracle Enterprise 10.2.0.4
- Stress Tool	Swingbench

2.2.2. Swingbench 란

Oracle 사의 Dominic Giles 라는 개발자가 만든 Oracle 전용 (TimesTen 포함) Benchmark Tool 이며 특히 Cluster-RAC 지원이 용이하며 Swinbench 의 GUI Tool 과 Text-Base 인 charbench 을 통해 다양한 환경에서 해당 Server 의 TPMC 값을 산정할 수 있으며 물론 DB 성능향상을 위해 DB Tuning 을 병행한다.

설치방법 및사용법은 http://www.dominicgiles.com/index.html 를 참고 바란다.

2.2.3. 테스트 환경

	File system	Raw device
parameter	shared_pool_size=200m db_cache_size=200m log_buffer=1m Archive log mode	
Data file	TABLESPACE_NAME FILE_NAME --- --- USERS /oradata/user_120m SYSAUX /oradata/sysaux_500m UNDOTBS1 /oradata/undo01_500m SYSTEM /oradata/system_500m EXAMPLE /oradata/examples_160m UNDOTBS2 /oradata/undo02_500m SOE /oradata/soe TUNE /oradata/mail_500m NONG /oradata/nong_1 NONG /oradata/nong SOE_IDX /oradata/soe_idx	FILE_NAME --- /dev/vx/rdsk/10gRAC/raw_user_120m /dev/vx/rdsk/10gRAC/raw_sysaux_500m /dev/vx/rdsk/10gRAC/raw_undo01_500m /dev/vx/rdsk/10gRAC/raw_system_500m /dev/vx/rdsk/10gRAC/raw_examples_160m /dev/vx/rdsk/10gRAC/raw_undo02_500m /dev/vx/rdsk/10gRAC/raw_soe_1g /dev/vx/rdsk/10gRAC/raw_mail_500m /dev/vx/rdsk/10gRAC/nong_1 /dev/vx/rdsk/10gRAC/nong /dev/vx/rdsk/10gRAC/soe_idx
	Log file /oradata/log12_120m /oradata/log11_120m /oradata/log21_120m /oradata/log22_120m /oradata/log13_120m /oradata/log23_120m	/dev/vx/rdsk/10gRAC/raw_log12_120m /dev/vx/rdsk/10gRAC/raw_log11_120m /dev/vx/rdsk/10gRAC/raw_log21_120m /dev/vx/rdsk/10gRAC/raw_log22_120m /dev/vx/rdsk/10gRAC/raw_log13_120m /dev/vx/rdsk/10gRAC/raw_log23_120m
	Controlfile /oradata/control01_110m /oradata/control02_110m	/dev/vx/rdsk/10gRAC/raw_control01_110m /dev/vx/rdsk/10gRAC/raw_control02_110m

2.2.4. Stress Test

Swingbench 툴을 사용하여 동시 접속자수 5 세션으로 3 분간 DML 를 수행하여 평균 TPM 와 TPS 를 체크한다.

$ pwd

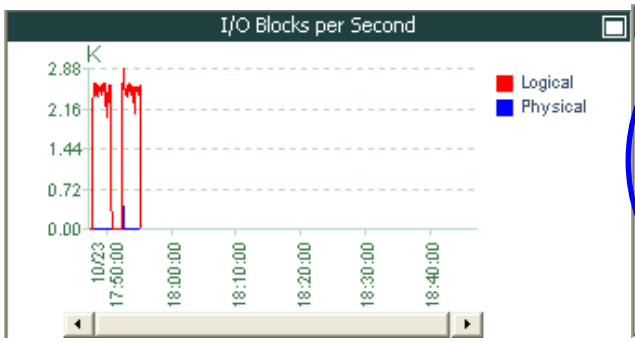
/ora10/swingbench/bin

$ ./charbench -c sample/soeconfig.xml -cs lkb2:1521:ORA10g3 -dt thin -cpuloc lkb2 -uc 5 -rt 00:03 -a -v users,tpm,센

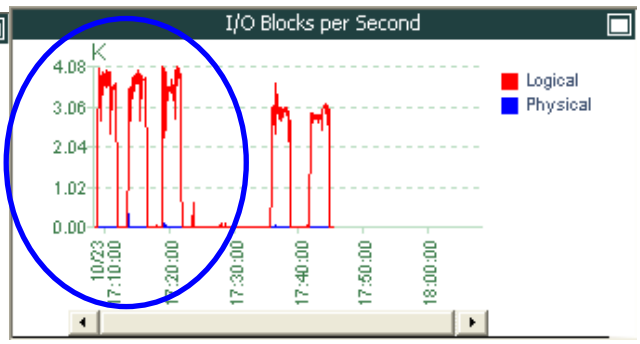
Stress	File system		Raw device	
Insert 15	TPM	TPS	TPM	TPS
Select 40	1 차 : 13,919.21	279.8914	1 차 : 21674.91	435.9483
Update 30	2 차 : 13993.71	278.6158	2 차 : 21503.44	436.1839
Delete 10				
Insert 30	TPM	TPS	TPM	TPS
Select 50	1 차 : 19517.74	392.1395	1 차 : 22996.07	456.9831
Update 10	2 차 : 18269.11	365.0514	2 차 : 23415.44	468.6514
Delete 10				
Insert 10	TPM	TPS	TPM	TPS
Select 85	1 차 : 22939.1	459.0852	1 차 : 22914.36	460.8655
Update 3	2 차 : 22856.63	457.6034	2 차 : 23053.41	461.2171
Delete 2				

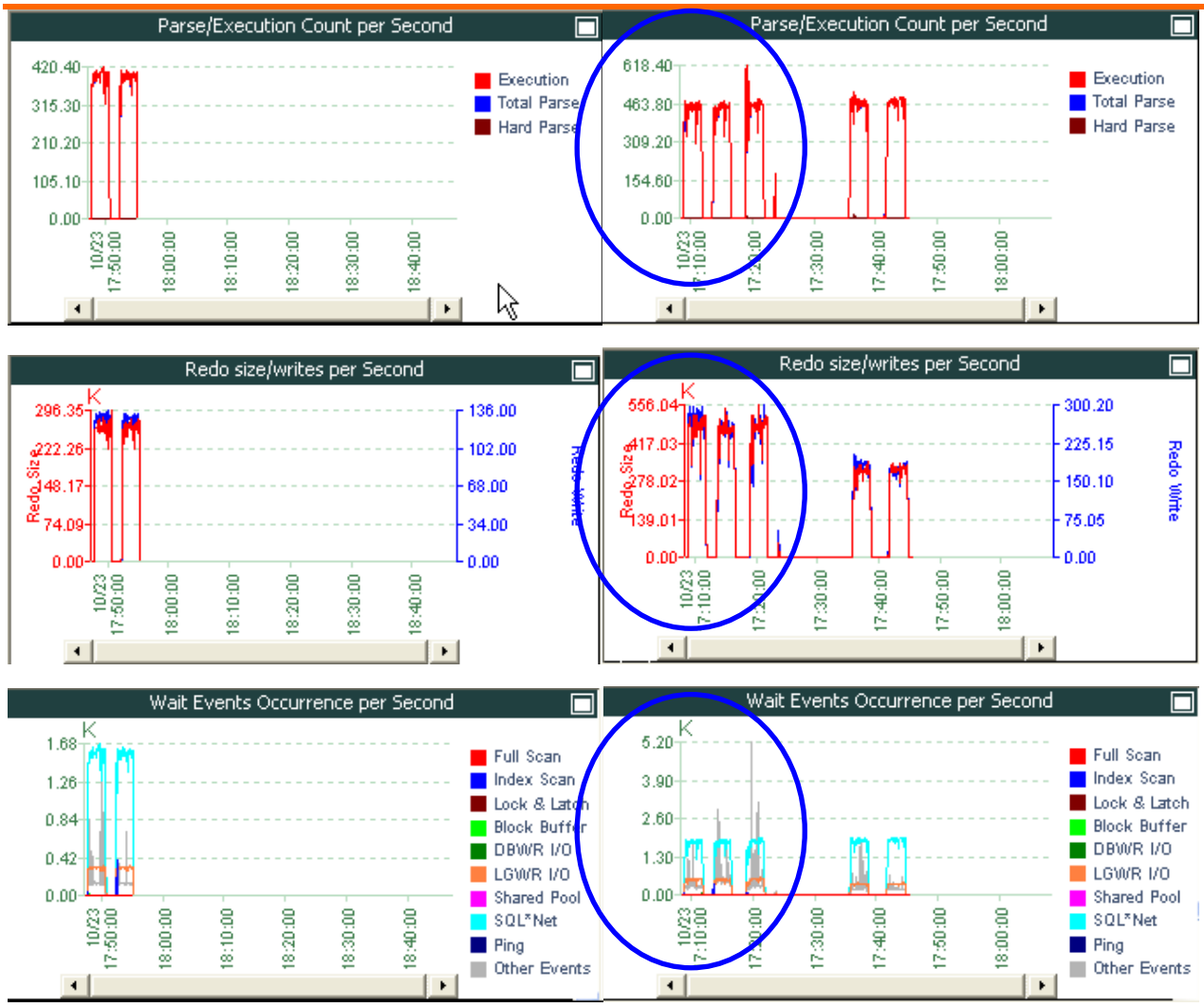
2.2.5. Stress Test 시 Orange 의 Instance Monitor

⊙ File System



⊙ Raw Device





2.2.6. Stress Test 시 iostat, vmstat 측정 결과

File system

```
#  iostat -zxcn 1

   cpu
us sy wt id
 27 14  0 59
                    extended device statistics
    r/s    w/s   kr/s    kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0  176.7    0.0 1413.7  0.0  0.6    0.0    3.1   0  52 c1t4d0
     cpu
 us sy wt id
 27 13  0 60
                    extended device statistics
    r/s    w/s   kr/s    kw/s wait actv wsvc_t asvc_t  %w  %b device
    6.0  159.1   48.0 1272.9  0.0  0.6    0.0    3.4   0  56 c1t4d0
     cpu
 us sy wt id
 28 13  0 58
                    extended device statistics
    r/s    w/s   kr/s    kw/s wait actv wsvc_t asvc_t  %w  %b device
    6.0  165.2   56.1 1321.8  0.0  0.8    0.0    4.6   0  65 c1t4d0
```

Raw device

```
#  iostat -zxcn 1

cpu
 us sy wt id
 54 33  0 12
                    extended device statistics
    r/s    w/s   kr/s    kw/s wait actv wsvc_t asvc_t  %w  %b device
    1.0  233.9   16.0   570.9  0.0  0.1    0.0    0.3   0   6 c1t4d0
     cpu
 us sy wt id
 54 42  0  4
                    extended device statistics
    r/s    w/s   kr/s    kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0  267.9    0.0   648.3  0.0  0.1    0.0    0.2   1   6 c1t4d0
     cpu
 us sy wt id
 48 16  0 36
                    extended device statistics
    r/s    w/s   kr/s    kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0  280.0    0.0   671.0  0.0  0.1    0.0    0.2   0   6 c1t4d0
```

File system

```
# vmstat 1
r b w   swap  free  re  mf pi po fr de sr f0 s0 s2 s6   in   sy   cs us sy id
 0 0 0 2836072 175432 40 17 47  0  0  0  0  0  0  0  0 2042 9231 2641 29 14 57
 1 0 0 2836072 175440 40 18 16  0  0  0  0  0  0  0  0 2025 8299 2629 26 13 61
 0 0 0 2836072 175448 38 16 16  0  0  0  0  0  0  0  0 1943 8167 2499 26 12 62
 0 0 0 2836000 175376 41 18 23  0  0  0  0  0  0  0  0 1975 7917 2586 26 14 61
 0 0 0 2836064 175432 40 36 16  0  0  0  0  0  1  0  0 2036 8813 2660 27 13 59
```

Raw device

```
# vmstat 1
r b w   swap  free  re  mf pi po fr de sr f0 s0 s2 s6   in   sy   cs us sy id
 3 0 0 2840824 206256 0   9  0  0  0  0  0  0  0  0  0 2281 12495 3220 41 17 42
 2 0 0 2840824 206256 0   9  0  0  0  0  0  0  2  0  0 2416 14046 3400 45 16 39
 1 0 0 2840824 206256 0   9  0  0  0  0  0  0  0  0  0 2418 13249 3472 43 16 42
 0 0 0 2840824 206256 0  11  0  0  0  0  0  0  0  0  0 2471 13154 3504 43 16 42
 0 0 0 2840824 206256 0  11  0  0  0  0  0  0  0  0  0 2537 13536 3550 44 16 40
```

2.2.7. 결론

Oracle DBMS 는 별도의 SGA 영역으로 Physical Memory 를 사용하기 때문에 File system 보다 Buffer Cache 를 사용하지 않는 Raw device 를 사용하는 것이 performance 측면상 대략 5 ~ 30% 의 효과가 나타남을 알 수 있다.

단 업무의 특성상 (select 의 비중도)를 고려하여야 겠지만 특히 주로 물리적인 Disk Access 가 많고 insert 가 많은 history 관련 log 를 관리하는 table 의 경우 raw device 를 전환을 고려해 볼 만 하다.

3. 부록

3.1. Sun 시스템 장애 분석

3.1.1. /(root) 파일 시스템이 90% 이상 되었을 경우 조치법

파일 시스템이 90% 이상을 넘을 때 방치할 경우, Disk 의 읽기/쓰기 속도 저하 현상 및 시스템 다운 현상이 일어 날 수 있으므로 아래와 같이 조치 해야 한다.

- /(root) 파일 시스템에 사용자가 임의로 만들어 준 디렉터리가 있는지 점검

- /dev 디렉터리 밑에 일반파일이 있는지 점검

```
# find /dev -type f | -exec ls -l {} ₩
```

◆ 일반파일이 있을 경우

모두 지우면 된다. 특히 테이프에 백업을 받을 경우에 사용자가 잘못된 디바이스 명을 지정하여 테이프에 백업되지 않고 파일에 저장되는 경우가 있다.

- core 파일 제거

```
# find / -name core -exec rm {} ₩ -print
```

- /var 밑의 Sub 디렉터리 용량(KB) 점검

```
# du -sk /var/* | sort -nr
```

◆ /var 밑에 있는 서브 디렉터리 목록

/var/adm
-> messages, wtmp, wtmpx, pacct 파일들의 사이즈가 클 경우 **# cp /dev/null `파일명`** 명령어를 사용하여 크기를 0 으로 만듦

/var/mail
-> 메일 데이터가 보관되는 디렉터리. 사이즈가 큰 파일이 있으면, 해당 사용자에게 그 메일을 정리하도록 조치

/var/log
/var/preserve
/var/spool
/var/crash
-> 기타 디렉터리(Application Log)에 대해서도 조사하여 불필요하게 사이즈가 큰 파일이 있을 경우 어떠한 파일인지 정확히 파악 한 후 지우도록 한다.

3.1.2. 다른 시스템과 Networking 이 안되는 경우

- flag 상태 확인 및 ip address/ netmask/ broadcast를 확인

```
# ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
inet 127.0.0.1 netmask ff000000
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
inet 61.250.99.209 netmask ffffff00 broadcast 61.250.99.255
ether 8:0:20:b7:13:73
```

- LED 가 있는 LAN card 인 경우 연결했을 때 LED 가 Green 으로 변하는지 확인
- LAN cable 이 LAN card 와 HUB / SWITCH 간에 연결이 제대로 되어 있는지 확인
- route table 에 local network 에 대한 routing 이 맞게 잡혀 있는지 확인

```
# netstat -rn
Routing Table:
Destination       Gateway           Flags    Ref     Use     Interface
-----------       ---------------   -------  ------  -------  ---------------
61.250.99.0       61.250.99.209      U       3       323      le0
224.0.0.0         61.250.99.209      U       3       0        le0
default           61.250.99.1        UG      0       546
```

- ping 명령을 사용하여 LAN 상의 다른 시스템으로부터의 응답을 받는지 확인

```
# ping -s 61.250.99.209
  PING 61.250.99.209: 56 data bytes
  64 bytes from new (61.250.99.209): icmp_seq=0. time=0. ms
  64 bytes from new (61.250.99.209): icmp_seq=1. time=0. ms
  64 bytes from new (61.250.99.209): icmp_seq=2. time=0. ms
  64 bytes from new (61.250.99.209): icmp_seq=3. time=0. ms
  64 bytes from new (61.250.99.209): icmp_seq=4. time=0. ms
```

3.1.3. 시스템이 갑자기 reboot/down 되었을 경우 점검/조치 방법

- **reboot 증상**

운영 중인 시스템이 자동 reboot 되어 정상적으로 reboot 되었고, 시스템 LED 또한 정상적으로 나타남

◆ 장애처리 절차

▪ /var/adm/messages 에서 panic(error) 메시지 여부 확인

```
# grep panic /var/adm/messages*
```

▪ /usr/platform/`uname -m`/sbin/prtdiag -v 명령어로 하드웨어 이상여부 확인

◎CPU 장애

예) 다음 메시지가 /var/adm/messages 파일에 존재함

에러 메시지:
panic[cpu7]/thread=0x6401e040: CPU7 Ecache SRAM Data Parity Error:
AFSR 0x00000000 00400008 AFAR 0x00000000 00200000
syncing file systems... [14] 18 [14] [14] [14]panic[cpu7]/thread=0x30037ec0: panic sync timeout

원인 : Ecache SRAM Data Parity Error로 인한 시스템 다운
조치사항 : 해당 CPU 교체 (CPU#7)

- **Down 증상**

운영중인 시스템이 자동으로 shutdown 되어 reboot 되어 있지 않음

◆ 장애처리 절차

- 현재 시스템 전원이 켜져 있는지 확인(꺼져 있으면 전원에 해당하는 부품 불량)
- ok prompt 상태라면 sync 명령어로 현재 운영상태를 core dump 받음
- 시스템 부팅을 진행하여 진행 중인 메시지를 관찰
- 정상 진행 된다면 /var/adm/messages 파일에서 panic 메시지 여부 확인
- /usr/platform/`uname -m`/sbin/prtdiag -v 명령어로 하드웨어 이상여부 확인

◎ Power Supply 장애

증상: 시스템 **전원부(AC Power Supply) LED가 황색** 상태

원인: 황색 상태의 전원 부품(AC Power Supply)이 불량
조치사항: 해당 부품(AC Power Supply) 교체

◎ 높은 가동 온도로 인한 서버 운영 장애

증상: 시스템이 shutdown 되어 있고 콘솔화면에 메시지가 존재하나 화면에 밀려 정확하게 분석하기 힘듦

원인: sync 명령어로 core dump 받으면서 시스템 재부팅 시킨 후, 메시지 파일에 쌓인 에러메시지 점검결과 전산실 온도상승으로 인해 자동 shutdown 된 상태임
조치사항: 담당자에게 통지 후 **시스템 가동 온도를 60도 이하로 낮춤**

3.1.4. 시스템 부팅을 시켰으나 Maintenance 모드로 진행됨

- **I/O Error 증상**

사용자가 특정 파일시스템 I/O가 되지 않아 시스템을 재부팅 시켰지만 정상부팅 안됨

◆ 장애처리 절차

- 정상부팅 안 되는 시점의 콘솔 메시지를 확인
- root 패스워드 입력 하여 maintenance 모드로 들어감
- 시스템 메시지 분석하여 파일시스템 I/O 가 발생되지 않는 원인을 찾음
- format 명령어로 디스크의 상태가 정상인지 확인
- /etc/vfstab 파일을 점검하여 사용해야 할 파일시스템을 check

▪ File system check 에러 발생 여부 확인

◎ Disk 장애

예) Internal 디스크 2번으로 사용중인 /data 파일시스템이 check 되지 않아 maintenance 모드로 진행됨

증상 : /data 파일시스템 check 하면 I/O error가 발생함
format 명령어로 디스크 상태 확인 시 c0t2d0 **디스크가 <drive not available: formatting>** 상태임
원인 : c0t2d0 디스크가 물리적으로 정상동작 하지 않음
조치 : 해당 디스크 교체 및 파일시스템 원복(백업 본)

3.1.5. Telnet "connect refused remote hosts"

- **telnet refuse 증상**

  remote system 으로 telnet 접속 시도 시 remote system 에서 telnet request 를 거부하는 경우

  "telnet Unable to remote host: Connection refused" 와 같은 에러가 발생

- **원인 및 해결방법**

  telnet request 를 받으면 "inetd" daemon 에 의해서 in.telnetd 가 수행되고 telnet 서비스를 요청해 온 시스템에 login prompt 를 띄워준다. 따라서 위와 같은 에러 메시지가 발생하면 telnet request 를 받은 remote system 에서 아래와 같은 사항들을 점검

◎ inetd daemon 설정

"inetd" daemon이 실행 중인지 확인
```
# ps -ef | grep inetd
```
inetd daemon이 실행 중이 아니면 아래와 같이 inetd daemon을 실행한다.
```
# /usr/sbin/inetd -s
```

◎ telnet service 설정

/etc/inetd.conf 파일에 "telnet" 에 관해서 아래와 같이 정의되어 있는지 확인
```
# grep telnet /etc/inetd.conf
telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd
```
위와 같이 정의가 되어 있지 않으면, 위의 내용을 추가한 후 "inetd" daemon을 re-start 시킨다.
```
# ps -ef | grep inetd
root 162 1 0 11월 28 ? 0:01 /usr/sbin/inetd -s
```
"inetd"의 process 번호
```
# kill -HUP 162
```