Tech Notes (vol.4)

Informix Dynamic Server 7.3 기능 분석

요약

이 글은 Informix Dynamic Server 차기 개정판인 버전 7.3의 주요 기능을 분석한 것입니다. 이 보고서는 7.3의 새로운 기능이 OLTP 환경에서 성능 향상에 미치는 효과를 알기 쉽게 상세히 설명하는데 초점을 맞추고 있으며, 이에 덧붙여 새로운 핵심적인 기능 개선들을 소개하고자 합니다. 부록에는 주요 성능 향상을 확인할 수 있는 테스트 방법을 설명합니다.

개요

Informix Dynamic Server의 차기 개정판인 버전 7.3은 DB 서버 시장 점유율에 있어 상당한 증대가 기대되는 몇가지 중요한 기능이 포함되어 있습니다. 7.3 버전에서 개선된 부분에는 다음과 같은 순위를 매길 수 있습니다.

1. 결함 보완
2. 더 한층 발전된 신뢰성, 가용성 및 지원 능력(Reliability, Availability, Support; RAS)
3. 향상된 성능
4. NT 버전과의 동일 기능 지원
5. VAR 업체를 위한 호환성 증대(Oracle 구문과의 호환성)
6. 관리 능력의 향상

7.3 개정판에 도입될 개선 사항은 50여 가지에 이르지만 이들은 모두 위의 여섯 범주로 구분할 수 있습니다. 이 보고서에서는 7.3 버전의 주요한 기능 개선을 간략히 소개하고 OLTP 성능 향상을 중점적으로 설명합니다.

기능요약

다음 내용은 7.3 버전의 주요 기능을 요약한 것입니다. 그 다음 절에서는 OLTP 성능 향상에 관련된 기능을 상세히 분석할 것입니다.

- 분할(Fragmentation)에서의 Attach/Detach

- 기존 서버의 분할 Attach/Detach 명령은 대상 테이블에 배타적 로킹을 설정하기 때문에 작동시 테이블의 가용성을 감소시키는 단점이 있습니다. 대다수의 대형 데이터웨어하우스 고객은 데이터웨어하우스 갱신을 이 명령에 의존합니다. 기존의 구조에서는 테이블이 교체되었을 때 항상 색인을 삭제/재구축하지만, 7.3 버전은 색인의 삭제 및 재구축 필요 여부를 판단할 수 있도록 개선되었습니다. 기존의 알고리즘은 색인을 무조건 삭제/재구축하지만 새로운 알고리즘은 분할의 Detach 명령을 수행할 때 색인 분할이 테이블 분할과 동일하다면(거의 동일하다면) 색인을 재구축하지 않습니다. 새로운 알고리즘은 단순히 시스템 카탈로그를 갱신하고 색인 분할을 삭제합니다(분리된 색인 분할일지라도 삭제). 또한, 분할의 Attach 명령을 수행할 때는 부착으로 인하여 기존 분할의 행을 이송해야하는지 판단하고 행 이동이 필요하지 않다면(겹쳐지지 않은 표현식) 색인 재구축을 생략합니다. 이 기능 개선은 데이터웨어하우스 관리를 위해 분할 교체를 자주 사용하는 고객들에게 상당한 도움이 될 것입니다.

- 효율적인 테이블 Alter/Modify/Drop

- 기존의 테이블 Alter/Modify/Drop 기능은 테이블에 배타적 로킹을 설정하고 교체 테이블을 생성하여 데이터를 복사한 다음 원래 테이블을 삭제합니다. 이런 방식은 DB영역 및 로그영역의 활용이 비효율적일 뿐만 아니라 배타적 로킹과 긴 트랜잭션이 발생하는 단점이 있습니다. 효율적으로 개선된 변경 기능으로는 테이블 어느 곳에서도 새로운 칼럼을 추가하거나 칼럼을 삭제할 수 있으며 칼럼의 길이, 칼럼의 데이터 형식을 변경할 수 있습니다. 여기에는 몇 가지 예외가 있는데 그 내용은 기능 명세 부분에 자세히 설명되어 있습니다.

- 개선된 oncheck

- 기존의 oncheck는 테이블에 배타적 로킹을 설정합니다. 또한 예약 페이지를 검사할 때는 물리 로그 및 논리 로그도 함께 검사하기 때문에 예약 페이지 검사 작업에는 상당한 시간을 필요로 합니다. 개선된 oncheck는 테이블의 동시 사용성을 높이기 위해 색인 및 테이블의 무결성을 검사할 때 배타적인 로킹이 아닌 행 수준 잠금만을 설정합니다. 개선된 oncheck에서 이전의 배타적 로킹 방식을 사용할 수도 있습니다. 하지만 페이지 수준 로킹이 설정된 테이블이 아니라면 기본값으로 행 수준 로킹이 설정됩니다. 개선된 oncheck는 예약 페이지 검사에 있어서 로그를 읽지 않는 것이 기본값이기 때문에 예약 페이지 검사 수행 속도가 더 빠릅니다. 이전의 배타적 로킹 방식을 사용하려면 -w 옵션을 사용합니다. 예약 페이지 검사에서 로그도 함께 검사하려면 -r 옵션이 아니라 새로운 -R 옵션을 사용할 수 있습니다.

- 개선된 예외 처리("고성능 진단")

- 이 기능은 오류가 발생했을 때 발생한 오류에 대해 충분한 정보를 제공함으로서 별도의 재진단 작업 소요를 줄이기 위한 것입니다. 또한 충분한 정보가 제공되므로 전체 프로그램을 중단시킬 필요 없이 오류가 발생한 세션만을 선택적으로 종료시킬 수 있게 됩니다. 이런 기능은 Informix 기술진의 제품 개발에 있어서도 대단히 긍정적인 효과가 있습니다.이 기능은 이벤트 경보, 오류 격리, 공유 메모리 덤프, 스택 추적, 유틸리티 옵션, 스레드 차단 루틴의 6가지 부분으로 나누어 집니다. 또한 이에 따라 6개의 새로운 구성 매개변수(afcrash, affail, afwarn, aflines, blocktimeout, sysalarmprogram)가 도입되었습니다. 이들은 오류 정보의 수준을 제어하는 매개변수이며 일반적으로 기본값이 사용됩니다. 개선된 예외 처리를 위해 onmode와 onstat에 몇 개의 새로운 옵션이 추가되었습니다. 특히 onstat -g ioq <큐명> 옵션은 상당히 유용한 옵션으로 큐 상의 다음 입출력 작업을 확인할 수 있습니다(KAIO 큐의 내용도 확인됨). 제품 지원 및 오류 진단은 서버의 안정성 향상과 고객 만족에 직접적인 영향을 미치므로 보다 나은 기능을 제공하기 위해 막대한 노력이 투입되고 있습니다.

- 외부 보관 검사

- archecker는 ontape 및 onbar의 -v(verify) 인수의 공식적으로 언급되지 않은 부분이 될 것입니다. 이것은 레벨 0, 1, 2 테이프를 읽어 전체 시스템 또는 선택된 DB영역 복원에 필요한 페이지가 모두 이상 없는지 확인하는 기능입니다. 이 확인 작업은 한번의 테이프 읽기로 수행됩니다. onbar와 함께 다른 업체의 저장관리 시스템을 사용한다면 이런 기능이 더욱 필요할 것입니다. ontape, onarchive, onbar에서 저장된 테이프를 검사할 수 있습니다.

- 재시작이 가능한 복원 UPDATE/INSERT 트리거 재진입

- 기존의 복원 유틸리티에서는 복원 작업을 진행하는 동안 디스크 또는 테이프에서 작업이 중단되는 오류가 발생하면 전체 복원 작업이 실패하고 복원 작업을 처음부터 재시작해야 했습니다. 7.3 버전에서는 오류 발생 순간에 아주 근접한 시점에서 복원 작업을 재시작할 수 있도록 개선되었습니다. 복원 작업은 2 단계의 과정, 즉 물리 복원(DB영역 및 청크)과 논리 복원 작업으로 나눌 수 있을 것입니다. 보관 일정 때문에 아직까지는 물리 복원 과정에서 오류가 발생한 경우만 재시작이 가능합니다. 복원 작업 재시작의 시점은 DB영역 단위로 결정됩니다. 즉, 오류가 발생한 복원 작업은 마지막으로 이상없이 복원된 DB영역 이후 시점부터 재시작할 수 있습니다. 이 기능 개선의 최초 대상은 고객의 오류 발생 시스템을 보다 빨리 정상화할 수 있도록 돕기 위한 기술 지원입니다.

- 다중 상주

- 하나의 Windows NT 시스템에서 복수의 OnLine 인스턴스가 구성되고 시작될 수 있도록 하는 기능입니다. 현 버전에서는 다중 상주가 허용되지 않습니다. 7.3 버전의 서버는 레지스트리 사용 방식을 개선하여 시스템에 실행중인 각각의 OnLine 인스턴스에 대한 정보가 레지스트리에 기록되도록 하였습니다. NT용 OnLine을 위한 구성 및 제어 프로그램도 복수 인스턴스를 생성하고 관리할 수 있

도록 변경될 것입니다. Unix용 OnLine에는 이미 다중 상주 기능이 구현되어 있습니다.

- 원시 장치

- 기존의 Windows NT용 7.20 버전에도 원시 장치 지원 코드가 포함되어 있지만 문서로 언급되지 않았습니다. 7.3 버전에서는 원시 장치 구현 방법이 문서로 공개됩니다. 드라이브 문자로 구성된 장치 및 드라이브 문자 없이 구성된 장치 모두에 대한 지원이 제공될 것입니다. 물리적 장치명 지정(드라이브 문자가 없는)을 통하여 26개 이상의 드라이브를 지원하는 것이 가능합니다.

- IPX / SPX지원

- Windows NT 전용. IPX(Internet Packet Exchange) 및 SPX(Sequenced Packet Exchange)는 Netware 환경에서 가장 일반적으로 사용되는 프로토콜입니다. Windows NT에서는 IPX/SPX 프로토콜로 윈도우 소켓을 구축할 수 있습니다. Informix 제품은 클라이언트에서 IPX/SPX 프로토콜로 NT 서버의 데이터베이스에 연결할 수 있도록 IPX/SPX 상의 윈도우 소켓 연결을 지원할 것입니다. 이 기능 지원은 클라이언트 접속 코드인 Informix ASF(Association Services Facility) 계층에 구현되어 있습니다. 이 기능을 사용할 수 있도록 ONCONFIG 구성 매개변수 NETTYPE에 새로운 프로토콜 형식 "socipx"를 지정할 수 있으며, SQLHOSTS 레지스트리 키에 새로운 하위 형식 "onsocipx"가 지원됩니다.

- Wolfpack(고가용성)

- Windows NT 전용. "Wolfpack"은 Windows NT 환경 고가용성 클러스터링의 코드 명칭입니다. Wolfpack의 클러스터링은 Unix 환경의 고가용성 소프트웨어인 HACMP 등과 같이 다른 장애 복구

소프트웨어와 유사한 방식입니다. Wolfpack 확장은 Windows NT 4.0용 추가 소프트웨어 패키지로 2 노드를 지원하며 공유 접속으로 장치에 연결합니다(동시 접속 불가). 또한 Wolfpack을 이용하는 경우 단일 서비스 명으로 클러스터링 시스템을 이용할 수 있기 때문에, 사용자는 실제 서비스가 제공되는 노드가 어떤 노드인지 판단할 필요없이 주어진 단일 서비스명으로 연결하면 됩니다. Informix는 Wolfpack의 응용프로그램 오류 복구를 지원할 것입니다. 한 노드에서 다른 노드의 오류가 검출되면 관리자가 사전 정의한 처리 절차가 자동 수행됩니다. 보통의 경우 이 절차는 OnLine을 시작하여 사용자 연결을 접수하고 오류를 일으킨 서버에 대한 사용자 연결을 해제한 후 다른 서버로 연결을 재구축하는 작업일 것입니다. 서버 상태에 대한 정보를 저장하지 않는 웹 검색기와 같은 소프트웨어는 재연결되지 않습니다. 또한 트랜잭션 감시기를 이용하여 내부적으로 수행되는 오류 극복을 구현하는 것도 가능합니다. 최근에는 Windows NT의 클러스터링과 Unix 시스템의 클러스터링이 대부분의 경우에 있어서 매우 유사합니다.

- Ploader(WinPload)

- WinPload(WPL)는 Windows NT 4.0 시스템용 고성능 로더의 GUI 구성요소입니다. NT 3.5.1 버전 지원되지 않았습니다. WPL은 반드시 IECC에서 실행되어야 합니다. WPL은 GUI 인터페이스와 로드/언로드 작업 작성 및 실행을 위한 "마법사"를 제공합니다. 여기서 제공되는 GUI 인터페이스는 Unix 작업 및 NT 작업 모두에 대해 사용할 수 있습니다. Unix 시스템에서 Motif를 통한 기존의 Unix용 ipload 사용도 계속 지원됩니다. 작업은 선택된 일부 테이블 또는 데이터베이스 전체에 대해 정의할 수 있습니다. 선택된 테이블에 대한 로드 및 언로드 작업은 onpload를 사용하여 수행되고, 데이터베이스 전체에 대한 로드 및 언로드 작업은 dbexport 및 dbimport로 수행됩니다. 이 기능 구현에 있어서 중요한 점은 WPL의 로드/언로드 작업이 DELUXE 모드에서는 오직 onpload만을 사용하고 테이블에 배타적 로킹을 설정한다는 것입니다.

- 광학 저장장치 지원 (Optical Support)

- Windows NT 4.0용 OnLine 7.3은 광학 플래터를 이용한 BLOB 저장을 지원할 것입니다. 이것은 NT에서 광학 저장장치를 사용할 수 있도록하는 단순한 지원 기능이 아닙니다. 7.3 버전 서버에는 광학 저장장치 관련 라이브러리가 고객이 별도로 구입해야 하는 개별 제품이 아닌 서버 자체 구성 요소로 포함되어 있습니다.

여기 분류된 기능군의 목표 대상은 Informix 환경용 소프트웨어를 공급하는 Informix의 부가가치 협력업체(PeopleSoft, SAP 등)이며, Oracle과의 호환성을 향상시켜 협력업체의 소프트웨어 개발을 돕는 것이 주 목적입니다.

- 대소문자 구별없는 검색(대문자, 소문자, 첫머리 대문자)문자열 함수(REPLACE, SUBSTR, LPAD, RPAD)

- UPPER, LOWER, INITCAP의 세 개의 새로운 SQL 함수가 제공됩니다. UPPER는 주어진 문자열의 모든 문자를 대문자로 변환합니다. LOWER는 문자열을 소문자로 변환합니다. INITCAP은 단어 또는 단어 열의 첫번째 문자를 대문자로 변환하고 나머지 문자는 모두 소문자로 변환합니다. 이 함수들은 SQL 문과 ESQL 함수로 사용할 수 있으며 SQL92 표준에 부합하는 방식으로 구현되었습니다. 이 함수를 이용한 문자열 검색 및 변환은 약간의 수행 속도 향상이 기대됩니다. 이외의 다른 문자열 작업은 성능면에서 어떠한 부정적인 영향도 받지 않습니다.또한 이에 덧붙여 문자열 함수 REPLACE, SUBSTR, LPAD(left pad), RPAD(right pad)가 지원됩니다. REPLACE는 문자열을 검색하여 검색된 내용을 주어진 값으로 대체합니다. SUBSTR은 문자열에서 주어진 길이 및 변위에 해당하는 부분 문자열을 리턴합니다. LPAD/ RPAD는 채움 문자(또는 빈칸)로 좌/우측이 채워진 문자열을 리턴합니다.

- UNION 뷰CASE 문/DECODE 함수/NVL 함수

- UNION 연산자를 사용할 수 있도록 CREATE VIEW 명령이 개선됩니다. 이것은 SQL92 표준과 일

관성이 유지되는 활용입니다. 예: CREATE VIEW uv(a,b) AS SELECT c1, c2 FROM t1 UNION SELECT c1, c2 FROM t2;7.3 버전에서는 SQL92 표준의 CASE 문이 지원됩니다.또한 7.3 버전에는 Oracle 호환을 위해 두 개의 새로운 함수 DECODE, NVL이 포함됩니다. DECODE 함수는 효율적으로 CASE 문을 사용하도록 하며 Oracle 7의 구현과 동일합니다. NVL 함수도 Oracle 7과 동일하게 구현되었습니다. NVL은 두 개의 매개변수(expr1, expr2)를 사용합니다. 만약 expr1이 널이면 expr2가 리턴되고 널이 아니면 expr1이 리턴됩니다. 이 두 함수를 이용함으로서 VAR 업체들의 응용프로그램 변환 및 이식 작업이 상당히 수월해질 것입니다.

- 날짜/시간 변환

- 두 개의 새로운 내장 함수가 지원됩니다. TO_DATE는 문자열을 DATETIME 형식의 자료로 변환하는 함수이고 TO_CHAR은 DATETIME 형식의 자료를 문자열로 변환하는 함수입니다. 7.3 이후 버전에서는 TO_CHAR 함수를 이용하여 날짜 이외의 다른 데이터 형식을 문자열로 변환하는 기능도 지원될 것입니다. 이 함수의 구현 및 사용 방식은 Oracle 7과 동일합니다. 변수를 변환할 때 포맷 문자열을 이용하여 변환된 문자열이 표시되는 방법을 지정할 수 있습니다(완전한 날짜명, 부분 날짜명, 완전한 월명, 월 숫자 등등). 포맷 문자열 사용은 선택적입니다. 이에 더불어 Oracle 소프트웨어를 좀더 손쉽게 이식할 수 있도록 Oracle 날짜 형식을 받고 대응되는 Informix 날짜 형식을 리턴하는 ifx_to_gl_datetime() 함수가 지원됩니다.ifx_to_gl_datetime() 함수는 genlib 형태로 제공되며 ESQL 프로그램에서 사용할 수 있습니다.

OnLine 관리 및 운영에 관련된 기능개선입니다.

- IECC

- IECC(Informix Enterprise Command Center)는 데이터베이스 서버 관리, 이벤트 관리, 데이터와 저장 영역 관리, 데이터베이스 오브젝트 관리를 위한 GUI 툴입니다.IECC는 Unix 시스템 GUI용이 아

니며 Unix 서버를 위한 GUI 설치를 지원하지 않습니다. IECC 사용가능 데스크탑은 Windows 뿐입니다. 웹 형식의 GUI 데모가 제공되지만 이것은 단지 데모일 뿐입니다.

- Informix 저장 관리자

- 7.3 버전에는 Informix 저장 관리자(Storage Manager), "ISM"이 제공됩니다. 이것은 Legato 사 제품인 Legato Networker의 축소된 기능 버전입니다.

- 계층적 복제(Hierarchical Replication)

- 기존의 Enterprise Replication 구현에서는 모든 노드가 서로 직접 연결되어야만 했습니다. 이런 방식으로는 노드 수가 증가할수록 확장성에 제한이 따르고 통신 설비비가 급격히 증가하게 됩니다. 따라서 노드가 복제 패킷을 네트워크 상의 다른 곳으로 전달할 수 있도록 하는 완전한 "캐스케이딩 복제"를 구현하기 위해 다양한 시도가 있었습니다. 진정한 캐스케이딩 복제가 최초로 실현된 것은 Informix의 7.3 버전으로 이것을 "계층적 복제"라고 합니다. 계층적 복제(Hierarchical Replication) 방식에서는 루트 서버로 불리는 일부분의 서버들만 서로 직접 연결되고 복제 네트워크 상의 나머지 서버들은 직접 연결할 필요없이 루트 서버로 연결하면 됩니다.이에 덧붙여 연결 해제된 서버에 관련된 보다 향상된 지원과 사용 편리를 위한 GUI가 제공됩니다.

성능특성분석

Informix는 TPC-C 벤치마크 결과로 확인된 그 동안의 기록적인 성능을 계속 유지하기 위해 7.3 버전에 막대한 노력을 투입했습니다. 여기에는 OLTP 시스템 성능 향상에 관련된 몇 가지 중요한 개선 사항이 포함되어 있습니다.

다음은 성능 관련 개선 사항들입니다. 이어지는 절에서 각각의 항목을 자세히 설명합니다.

- 첫째 행 최적화 및 정렬된 병합(First Row Optimization and Ordered Merge)
- 최적화기 지시어(힌트) (Optimizer Directives(hints))
- 최초 'N' 행 선택(Select First 'N' rows)
- 연결 다중화(Connection multiplexing)
- 부질의어 성능(Subquery performance)
- 메모리 상주 테이블(Memory Resident Tables)

이제 위의 성능 특성을 세부적으로 설명합니다. 부록에서는 이들의 실제 효과를 측정하는 방법에 대해 설명할 것입니다.

첫째 행 최적화 및 정렬된 병합(First Row Optimization and Ordered Merge)

'첫째 행 최적화'라는 이름에서 알 수 있듯이 첫째 행 최적화는 질의 결과의 첫번째 행이 리턴되는 시간의 최소화하는 것을 그 목표로 하는 새로운 최적화 전략입니다. Informix의 최적화기는 일반적으로 질의 실행에 소요되는 전체 시간을 최소화하는 방향으로 질의 최적화를 수행하기 때문에 최적화 과정을 거친 질의는 매우 빠른 속도로 실행될 수 있습니다. 하지만 대화형 응용프로그램의 사용자, 특히 PC 사용자들은 질의 결과가 화면에 신속하게 표시되는 것을 선호하며, 이것이 바로 질의 결과를 빠르게 되돌려 주는 응답 시간 최소화가 필요한 이유입니다.

최적화에 있어서 두 가지 전략, 즉 총시간 전략(total time) 대 최초 응답 시간(response time) 전략은 완전히 상반된 것이고 어떤 전략을 선택하느냐에 따라 질의 계획 또한 완전히 바뀌어야 합니다. 그렇기 때문에 첫째 행 또는 반응 시간 위주로 최적화된 질의는 처리 속도 위주로 최적화된 질의에 비해 전체 실행 속도는 더 느려지는 경향이 있습니다.

새로운 기법이 도입됨에 따라 이제는 실행할 질의의 최적화 전략을 지정할 수 있게 되었습니다. 새로운 SQL 문 "SET OPTIMIZATION "가 제공됩니다. 이전 버전의 일관된 동작을 위해 최적화 기본값은 여전히

ALL_ROWS입니다. 설정된 값은 현 세션이 종료되거나 다른 값이 지정되기 전까지 이후에 실행되는 모든 질의에 효력을 미칩니다. 새로운 방식에서도 OPTIMIZATION의 HIGH/LOW 설정은 계속 사용할 수 있습니다.(즉, FIRST ROWS 및 ALL_ROWS 설정 모두에서 HIGH/LOW 최적화가 지원됩니다.)

또한 지시어를 이용한 최적화 목표 지정이 가능합니다. 최적화 목표 설정에 있어 지시어를 이용하는 것이 지원되는 유일한 방법입니다. 지시어를 이용하여 최적화 목표를 지정할 때 지정된 목표는 지시어가 포함되는 현재 질의 블럭에 대해서만 적용됩니다. 각각의 블럭(개별적인 질의)의 최적화 수준은 --+ <목표(goal)> 형식으로 지정합니다. 자세한 지시어 설명은 다음 절을 참고하십시오.

서버 전체의 기본 최적화 목표를 조정하기 위한 새로운 ONCONFIG 매개변수 OPT_GOAL이 제공됩니다. 이 매개변수에는 ALL_ROWS를 뜻하는 -1, 또는 FIRST_ROWS를 뜻하는 0을 지정할 수 있습니다. 또한 ONCONFIG 기본 설정을 변경할 수 있도록 동일한 이름의 환경 변수 및 구문도 지원합니다.

첫째 행 최적화 지원은 다른 업체와의 경쟁을 위해서도 필수적입니다. Sybase, Oracle, Redbrick에서 첫째 행 최적화를 지원하고 있지만 업체마다 그 구현방식에는 조금씩 차이가 있습니다.

Sybase: Sybase SQL Anywhere(Watcom)의 최적화기는 첫째 행 방식 전용이며 최적화 방식을 변경할 수 없습니다.

Oracle: Oracle 7.3에서는 사용자 작성 힌트로 개별 질의에 첫째 행 방식을 지정할 수 있으며 SET 명령으로 세션 수준에 첫째 행 방식을 지정할 수 있습니다. 하지만 init.ora 파일을 변경하여 전체 인스턴스에 대해 첫째 행 방식을 지정하는 것이 가능한지는 확실치 않습니다.

Redbrick: Redbrick사 웹 사이트에 Warehaouse 버전 5.0이 첫째 행 방식을 지원한다고 언급되어 있지만 세부 내용은 확인되지 않았습니다.

첫째 행 최적화를 구현하려면 분할된 색인만으로 질의의 ORDER BY 절을 만족시킬 수 있도록 엔진 자체가 개선되어야 합니다. 7.3 버전에서는 색인의 칼럼과 질의의 ORDER BY 칼럼이 동일한 경우 분할된 색인으로

ORDER BY 절을 만족시킬 수 있습니다. 그런데 분할된 색인으로부터 얻어지는 결과가 어떤 순서인지 확인할 수 없기 때문에 얻어진 결과를 병합하는 작업이 필요합니다. 이것을 정렬된 병합(ordered merge)이라 합니다.

최적화기 지시어(Optimizer Directives(hints))

개발자들 사이에서는 질의를 작성할 때 질의 계획을 전체적으로 또는 부분적으로 지정할 수 있는 수단이 필요하다는 의견이 종종 제시되었습니다. 왜냐하면 질의 계획 선택을 최적화기에 전적으로 맡기기 보다는 질의 개발자가 특정 경로를 사용하도록 최적화기에 지시함으로서 보다 유연한 개발 환경을 획득할 수 있기 때문입니다. 뿐만 아니라 Informix로서는 이 기능을 이미 구현한 Sybase, Oracle과의 경쟁에 따른 도입 필요성 때문에 기능 구현을 더 이상 미룰 수 없었습니다.

질의는 2단계의 과정으로 처리됩니다. 첫번째는 질의 최적화 및 컴파일 과정을 거쳐 질의 계획을 수립하는 단계입니다. 2단계는 질의를 실행하여 결과를 리턴하는 단계입니다. 최적화기 지시어는 질의 계획 수립 단계에서 최적화기의 질의 계획 선택을 제어하는 수단입니다.

- 지시어 기능은 다음과 같은 필요에 따라 도입되었습니다.

- 질의 성능 조정에 소요되는 시간 단축. 재작성된 질의 실행에는 상당한 시간이 소요되는데 지시어 기능을 이용하면 질의 계획을 빠르게 수정할 수 있습니다.
- 다른 업체와의 경쟁
- 효율적인 제품 개발. 새로운 최적화 기법의 효과를 평가할 때 코드를 수정하지 않아도 질의 계획을 능동적으로 변화시킬 수 있습니다.

Informix의 최적화기 지시어 구현 방식은 구문과 작동에 있어서 Oracle의 힌트와 호환됩니다. 그러므로 VAR 업체의 Oracle 응용프로그램을 Informix용으로 이식하는 작업이 한결 수월해질 것입니다.

지시어는 주석문 형식으로 입력하며 첫번째 문자가 "+"(덧셈 기호)입니다. 다음 예를 보십시오.

```
SELECT {+ ORDERED AVOID FULL(c)}

    c.customer_num, fname, lname, order_num, order_date

    FROM customer c, orders o

    WHERE c.customer_num=o.customer_num;
```

위의 보기는 테이블이 질의에 지정된 순서로 조인되어야 하고 customer(c) 테이블은 전체 스캔으로 액세스되지 않아야 한다고 지정한 것입니다

Informix의 구현 방식은 한가지 면에서 Oracle의 구현 방식과 확연한 차이가 있습니다. Oracle은 직접 힌트만을 지원하는 반면 Informix는 부정 지시어를 지원합니다. 부정 지시어는 최적화기에 선택해야 할 것을 알리는 대신 피해야할 것을 알리는 지시어로 Informix 만의 독특한 기능입니다. 예를 들어, 성능 저하의 가능성이 존재함이 알려진 특정한 작업은 피하도록 지시하고 그 외의 테이블 및 질의 생성 이후에 추가된 것들을 포함한 모든 색인 또는 테이블 속성은 최적화기가 전부 탐색하도록 허용해야 하는 상황이라면 부정 지시어 기능의 가치가 돋보일 것입니다. Informix 제품에서는 부정 지시어 기능이 지원되기 때문에 DBA가 테이블에 색인 등을 추가할 때 최적화기 지시어 재지정을 고려하지 않아도 됩니다.

또한 최적화기 지시어를 인식하는 SET EXPLAIN 출력 기능이 지원됩니다. Informix 제품은 모든 의미적인 오류와 구문 오류를 표시합니다. Oracle은 이 기능을 지원하지 않습니다.

- 최적화기 지시어로 최적화 과정의 다음과 같은 영역을 제어할 수 있습니다.

- 액세스 방법 : 색인 대 스캔
- 조인 방법 : 강제 해시 조인, 중첩 루프 조인
- 조인 순서 : 테이블 조인 순서 지정

- 최적화 목표 : 첫째 행 또는 전체 행 (응답 시간(response time) 대 처리 속도(throughput))

현 버전의 ESQL은 주석문을 제거하기 때문에 최적화기 지시어가 담겨있는 주석문이 서버에 전달되도록 ESQL이 개선됩니다. 지시어 처리가 수행되지 않도록 하는 새로운 ONCONFIG 매개변수 DIRECTIVES가 도입됩니다. 기본값은 지시어 처리를 수행하는 것입니다. DIRECTIVES 매개변수는 1(ON) 또는 0(OFF)으로 설정할 수 있으며 기본값은 1입니다. 또한 매개변수 설정을 무시할 수 있도록 환경변수 IFX_DIRECTIVES가 지원됩니다.

프로시저, 트리거, 뷰에 나타나는 모든 SELECT, UPDATE, DELETE 문과 INSERT 문에 포함되어진 SELECT 문에 지시어를 사용할 수 있습니다. 하지만 서버가 rowid를 이용하여 데이터에 액세스하는 "현재 커서 위치 UPDATE/DELETE" 문과 분산 질의(원격 테이블 액세스 질의)에는 지시어를 사용할 수 없습니다.

지시어는 SELECT, UPDATE, DELETE 키워드에 바로 이어서 주석문으로 나타나야 합니다. 다음과 같은 주석문 기호가 지원됩니다.

```
SELECT|UPDATE|DELETE --+ 지시어

SELECT|UPDATE|DELETE {+ 지시어 }

SELECT|UPDATE|DELETE /*+ 지시어 */
```

액세스 방법 지시어(Access Method Directives)

- 다음과 같은 4가지의 액세스 방법 지시어가 지원됩니다.

1. INDEX
2. AVOID_INDEX

3. FULL
4. AVOID_FUL

INDEX에는 임의 개수의 색인을 지정합니다. 한 개의 색인이 지정된 경우 그 색인만 사용됩니다. 여러 개의 색인 목록을 지정한 경우 최적화기는 주어진 목록에서 색인을 선택하지만 테이블 스캔은 하지 않습니다. 색인을 지정하지 않은 경우 모든 색인을 조사하지만 전체 스캔은 하지 않습니다.

예제:

```
SELECT --+ INDEX (e salary_indx)

* FROM employee e

WHERE e.dept_no = 1 and e.salary > 500000);
```

위의 질의는 테이블 액세스에 오직 salary_indx 색인만 이용합니다.

AVOID_INDEX에는 테이블 액세스에 사용하지 않아야 할 임의 개수의 색인을 지정합니다. 이것은 Informix 만의 독특한 기능으로 최적화기에 성능 저하가 존재함이 알려진 색인은 피하고 질의 작성 이후에 추가된 모든 색인은 고려하도록 지시할 수 있는 편리함이 있습니다.

FULL은 최적화기가 지정된 테이블에 전체 스캔을 수행하도록 지시합니다. 반대로 AVOID_FULL은 최적화기가 주어진 테이블을 스캔하지 않도록 지시하는 것입니다.

예제:

```
SELECT --+FULL(e)
```

```
name,age FROM employee e

WHERE e.dept_no > 0;
```

위의 질의는 최적화기가 테이블 스캔 수행을 선택하도록 지시합니다.

이 지시어의 유용성은 여러 벤치마크 실험으로 확인할 수 있을 것입니다. 때때로 일부 테이블에 대규모 스캔을 수행하는 질의 계획을 얻게 되는데, 벤치마크에서 UPDATE STATISTICS를 실행하고 스캔이 아닌 색인을 사용하도록 질의 계획이 변경되면 더 좋은 성능의 질의 계획을 얻지 못하는 경우가 있습니다. AVOID_FULL 지시어는 이것 뿐만이 아니라 여러 상황에서 유용하게 사용될 것입니다.

같은 주석문 블럭에 위치한다면 복수의 지시어를 동시에 사용하는 것이 허용됩니다. 다음 예를 보십시오.

```
SELECT --+ AVOID_FULL(e), INDEX (e salary_indx)        허용됨.

SELECT --+ AVOID_FULL(e)

          --+ INDEX (e salary_indx)                    허용되지 않음.
```

조인 순서 지시어(Join Order Directives)

ORDERED 지시어는 최적화기가 FROM 절에 나열된 순서에 따라 테이블을 조인하도록 지시합니다. 다음 예를 보십시오.

```
SELECT --+ORDERED

        name,title,dept

        FROM dept,job,emp
```

```
WHERE title=*clerk*

AND loc=*Palo Alta*

AND emp.dno=dept.dno

AND emp.job=job.job;
```

최적화기는 위의 질의를 실행할 때 FROM 절에 나열된 순서에 따라, dept와 job 테이블을 조인하고 그 결과와 emp 테이블을 조인합니다.

뷰와 관련된 ORDERED 지시어 처리에는 몇가지 서로 다른 경우가 있습니다. 첫째로 지시어를 지정하지 않은 경우 최적화기는 비용을 고려하여 최적의 순서를 선택합니다. 둘째로 뷰 내부에서 지시어를 지정한 경우 뷰의 테이블의 순서는 유지되지만 나머지 테이블은 추가적인 조인의 가능성이 있습니다. 마지막으로 뷰를 사용하는 질의이지만 뷰를 생성하는 구문에는 지시어가 없고 뷰 외부에서 지시어를 지정한 경우 FROM 절에 따라 전체 순서가 결정됩니다.

다음 예는 이 세 가지 경우를 나타낸 것입니다

```
CREATE VIEW emp_job_view AS

    SELECT * FROM emp,job;

    SELECT * FROM dept,project,emp_job_view;
```

최적화기는 자체적으로 emp, dept, job, project

각각의 비용에 따라 조인 순서를 선택합니다.

```
CREATE VIEW emp_job_view AS
SELECT {+ORDERED} * FROM emp,job;
SELECT * FROM dept,project,emp_job_view;
```

최적화기는 dept, emp, job, project 테이블을 조인할 것입니다.

여기서 뷰의 조인 순서(emp, job)는 유지되지만,

그러나 전체 순서는 최적화기가 선택합니다.

```
CREATE VIEW emp_job_view AS
SELECT * FROM emp,job;
```

```
SELECT {+ORDERED} * FROM dept,project,emp_job_view;
```

최적화기는 질의 SELECT 문의 순서에 따라 조인을 수행합니다.

즉 dept, project, emp, job 순서로 조인됩니다.

조인 방법 지시어(Join Method Directives)

- 해시 조인 및 중첩 루프 조인의 사용 요부를 선택할 수 있는 지시어가 지원됩니다.
지시어는 다음과 같습니다.

- USL_NL - 중첩 루프 사용
- USE_HASH - 해시 조인 사용 - Informix 만의 독특한 기능입니다.
- AVOID_NL - 중첩 루프 회피
- AVOID_HASH - 해시 회피

중첩 루프 조인에서 외측 테이블의 각각의 행과 합치되는 내측 테이블의 행을 검사하고 조인된 행이 검사 결과로서 리턴됩니다. 내측 테이블에 대한 액세스는 스캔(카티젼 프로덕트:cartesian product) 또는 기존의 색인 또는 동적으로 구축된 색인이 될 수 있습니다. USE_NL 지시어는 인수로 테이블명을 받습니다. USE_NL 지시어에는 --+ USEL_NL(테이블1, 테이블2,...) 형식으로 테이블을 지정합니다. 일련의 중첩 루프에서 하나의 테이블은 반드시 외측 테이블로 사용되어야 하기 때문에 FROM 절의 테이블 수가 N일 때 USE_NL 지시어에 지정 가능한 테이블의 최대 개수는 N-1입니다.

다음은 중첩 루프 조인 예제입니다.

```
SELECT
```

```
name,title,salary,dname

FROM emp,dept,job

WHERE loc=*Palo Alto*

AND emp.dno = dept.dno

AND emp.job = job.job
```

/*+USE_NL (dept) 최적화기가 dept 테이블에 NL 조인을 사용하도록 함 */

위의 예제에서 USE_NL 지시어는 최적화기가 질의의 다른 테이블과 dept 테이블을 조인할 때 중첩 루프를 사용하도록 합니다. dept 테이블은 조인에서 내측 테이블이 될 것입니다.

USE_HASH는 Oracle의 힌트와 비교했을 때 Informix 만의 독특한 기능입니다. 해시 조인은 구축 및 검사 두 부분으로 구성됩니다. 구축 단계에서는 하나의 테이블의 행을 택하여 각각의 요소에 대해 해시 테이블을 구축합니다. 검사 단계에서는 두 번째 테이블의 값을 해시 테이블로 직접 해싱한 다음 조인 쌍을 조사합니다.

USE_HASH 지시어에 아무런 인수를 지정하지 않는 경우 최적화기는 테이블 조인에 해시 조인을 사용하지만 조인의 순서는 비용을 고려하여 자체적으로 선택합니다. 다음예에서처럼 무엇을 구축 테이블로 사용하고 무엇을 검사 테이블로 사용할지 지정하는 것도 가능합니다.

```
SELECT {+USE_HASH (dept /BUILD)}

name,title,salary,dname
```

```
FROM emp,dept,job

WHERE loc=*Palo Alto*

   AND emp.dno = dept.dno

  AND emp.job = job.job
```

위의 예제에서 USE_HASH 지시어는 최적화기가 해시 조인으로 dept 테이블을 조인하고 해시 테이블 구축에 dept 테이블을 사용하도록 합니다.

최적화 목표 지시어(Optimization Goal Directives)

최적화 목표 지시어의 필요성은 앞 절에서 자세히 설명했습니다. 지정된 최적화 목표는 전체 세션이 아니라 지시어가 사용된 질의에 한해서만 영향을 미칩니다.

Informix의 지시어는 Oracle의 힌트와 유사합니다. 하지만 Informix는 FIRST_ ROWS 지시어에 부가적인 기능을 덧붙였습니다. Informix의 FIRST_ROWS 지시어에서는 행 개수를 지정할 수 있기 때문에 최적화기가 질의의 의도를 좀더 명확히 파악할 수 있습니다.

예제:

```
SELECT --+ FIRST_ROWS (100)

name,age FROM employee e, department d

WHERE e.dept_no = d.dept_no;
```

위의 질의는 질의 결과의 "최초 화면"을 표시하는데 중요하다고 여겨지는 행 갯수를 최적화기에 알립니다.

FIRST_ROWS의 반대는 ALL_ROWS(질의 엔진의 기본 작동 방식)입니다. 뷰 정의 및 부질의에는 최적화 목표 지시어를 사용할 수 없습니다.

SET EXPLAIN 지원

질의와 질의 계획 분석을 용이하게 하기 위해 지시어 모드 지시어(directive mode directive)라고 불리는 또 다른 지시어가 지원됩니다. 최적화기의 SET EXPLAIN 기능이 지시어에 관련된 추가적인 정보를 처리할 수 있도록 개선될 것입니다. EXPLAIN 지시어로 주어진 질의에 대한 계획 설명이 출력되도록 설정할 수 있으며, 이것은 세션 설정에 관계없이 해당 질의에 영향을 미칩니다.

사용 방법은 다음과 같습니다.

```
SELECT --+ EXPLAIN AVOID FULL (e)

* FROM employee e, jobs j

WHERE j.title=*CLERK* and j.job = e.job;
```

위의 질의는 최적화기가 sqexplain.out이라는 이름으로 계획 설명을 출력하고 employee 테이블에 전체 스캔을 사용하지 않도록 지정했습니다. 적용된 지시어, 적용되지 않은 지시어, 지시어의 ON/OFF 여부에 대한 정보가 포함되도록 sqexplain.out 출력도 개선될 것입니다. 지시어가 OFF인 경우(ONCONFIG나 환경변수에서 OFF로 설정) 다른 지시어 정보가 sqexplain.out 출력에 포함되지 않습니다.

최초 'N'행 선택 최적화(Select First 'N' optimizations)

이것은 질의 결과를 최초 'N' 행으로 제한할 수 있는 기능으로 TPC-D 표준을 준수하고 "순위(rank)" 질의를 지원하기 위해 구현된 것입니다. 실질적으로 이 기능은 XPS에서 약간의 수정과 함께 역이식된 것입니다.

- 순위 질의는 ORDER BY와 FIRST 'N'을 이용하여 구성됩니다.
 질의 결과는 몇가지 정렬 기준에 따라 상위 N 행만 리턴됩니다.
 이 기능에는 다음과 같은 제한이 있습니다.

1. 부질의는 지원하지 않습니다 (SELECT 속의 SELECT).
2. INTO TEMP 절과 함께 사용할 수 없습니다.
3. 뷰 정의에는 사용할 수 없지만 뷰를 참조하는 SELECT 문에는 사용할 수 있습니다.
4. 내장 프로시저는 지원하지 않습니다.
5. UNION 질의는 지원하지 않습니다.

SELECT FIRST 'n' FROM의 동작은 ORDER BY 절 존재 여부에 따라 다릅니다. ORDER BY 절이 지정되지 않은 경우 임의 순서로 최초 N 행이 리턴됩니다. 부록 A에 순위 질의를 시험하는 예제와 사용 방법이 설명되어 있습니다.

XPS와 달리 7.2에서는 'first'라는 칼럼명이 지원됩니다. 따라서 first라는 단어 뒤에 양의 정수가 존재하지 않는다면 그 토큰은 키워드가 아닌 칼럼명으로 처리됩니다. N 값은 1에서 231-1까지입니다.

연결 다중화(Connection Multiplexing)

ASF SQLI 세션 계층 다중화 기능은 복수의 SQLI ASF 세션 핸들이 하나의 공통 전송 핸들을 공유할 수 있도록 ASF 계층을 확대한 것입니다. 이 개선된 세션 계층에 따라 ASF 클라이언트들은 하나의 네트워크 연결 자원만으로 동일한 ASF 서버에 대해 완전한 다중 독립 연관 핸들을 보유할 수 있습니다.

일반적인 ESQL 사용자가 얻게되는 두드러지는 혜택은 내부적으로는 최소의 통신 자원만을 사용하면서 동일한 데이터베이스 서버에 다수의 연결을 구축할 수 있게되는 점입니다. "다중화 기능이 구현되지 않은"

Informix 클라이언트 및 서버도 sqlhosts 파일과 onconfig 파일 엔트리를 적절히 조정하면 ASF 세션 계층 다중화를 이용할 수 있습니다. 다중화 연관은 사용가능한 기반 연관에 덧붙여집니다. 기존의 기반 연관에 다중화 연관이 구축되지 못할 때는 새로운 비다중화 기반 연관이 자동으로 구축되며, 이 새로운 기반 연관은 또 다른 다중화 연관이 이용할 수 있습니다.

Baan과 같은 상당 수의 전형적인 비스레드(non-threaded) SQL 클라이언트는 단일 사용자의 작업을 수행하기 위해 동일 데이터베이스 서버에 복수의 SQL 연결을 구축합니다. 따라서 1000명의 사용자를 위해 1000개의 응용프로그램 프로세스를 운용하는 응용프로그램 시스템이 하나의 프로세스 마다 여섯 개의 연결, 즉 총 6000여 개의 SQL 연결을 사용할 수 있습니다. 모든 응용프로그램 사용자가 복수의 SQL 연결을 필요로 하지만 비스레드 응용프로그램이기 때문에 실제로는 한번에 하나의 연결만이 활성이 됩니다.

많은 수의 네트워크 연결의 입력 데이터를 처리하는 데이터베이스 서버는 실제 데이터 입력이 있는 연결은 전체의 일부분일지라도 상당량의 CPU 시간을 소모합니다. 시스템 CPU 시간 소모는 네트워크 연결 수의 선형 함수이며 UNIX select(), poll() 시스템 콜의 비효율적인 구현에 기인합니다. 네크워크 연결의 수가 증가하면 네트워크 연결 폴링에 소요되는 CPU 시간도 급격히 증가합니다.

연결 다중화 기능은 sqlhosts 파일 및 onconfig 파일에 새로운 옵션을 적용함으로서 활성화됩니다. 새로운 옵션은 sqlhosts 옵션에 덧붙여지며(5번째 칼럼) 해당 DB서버명 엔트리에 SQLI 다중화가 사용되는 여부를 지정합니다. "m=0" 옵션(기본값)은 다중화를 사용하지 않는 것이며 "m=1" 옵션은 다중화를 사용하는 것입니다.

덧붙여서 연결이 다중화 ASF 계층을 통해 구축되도록 onconfig 파일의 NETTYPE 매개변수를 SQLMUX로 설정해야 합니다. SQLMUX 형식 ASF 드라이버가 실제로 존재하는 것은 아닙니다. SQLMUX는 의사 ASF 계층 형식이며 연결 다중화의 동작을 위해서 onconfig 파일에 반드시 위치해야 합니다. SHM 연결의 다중화는 지원되지 않습니다.

연결 다중화를 통한 성능 이득은 시스템의 select(), poll() 시스템 콜에 소요되는 전체 시간에 비례할 것입니다.

부질의 성능 및 색인 개선(Subquery Performance and Index Improvements)

이 기능 개선은 PeopleSoft 제품의 성능 향상을 직접적인 목표로 합니다. 물론 대다수의 상호관련된 부질의 성능 향상에도 상당한 효과를 발휘할 것입니다. 부질의 성능 향상을 위한 접근 방법 및 해결 방안에는 여러 가지가 있지만 모든 부질의에서 최적의 성능을 이끌어 내려면 하나의 부질의 최적화 전략만으로는 부족하기 때문에 최적화기는 실행시간 동안 자체 분석한 자료를 토대로 몇 가지 다양한 최적화를 수행합니다. 최적화 기법은 부질의를 풍부하게 활용하고 있는 PeopleSoft 사 벤치마크의 광범위한 연구를 토대로 구현되었습니다. 개선된 부분은 다음과 같으며 각 항목별로 설명할 것입니다.

1. 부질의 캐시(Subquery Cashing)
2. 부질의 전개(Subquery Flattening)
3. KEYONLY 색인 액세스

부질의 캐시(Subquery Cashing)

질의 처리에 관련된 최적화 이론 및 그 세부사항은 매우 복잡합니다. 세부적인 내용은 전문적인 설계 명세를 참조하십시오. 그 문서는 매우 전문적인 내용이며 상당한 인내심과 집중력이 필요합니다.

대개의 경우 부질의는 동일한 입력 매개변수로 반복 호출됩니다. PeopleSotf GL 벤치마크 질의의 실례를 살펴보면 외측 질의 블럭이 대략 260,000 행일 때 상호관련 칼럼의 가능한 값은 26개에 불과합니다. 이런 경우 부질의(함수로 간주)는 동일한 인수로 10,000번(대략 260,000 행) 가량 호출됩니다. 질의 형태가 이와 같을 때 질의 전개(중첩 해체)를 통한 최적화는 질의 수행 성능을 높이기 보다는 더 떨어뜨리게 됩니다.

7.3 버전 최적화기는 질의 형태를 인식하고 이런 형태의 질의에 대해서는 부질의 결과 캐시를 생성합니다. 위의 경우 생성된 결과 캐시의 엔트리는 단지 26개일 것이며 최적화기는 캐시 값을 조사하는 것으로 실제 질의 실행을 대신할 수 있게 됩니다. 질의를 무조건 실행하는 경우 캐시 조사는 불필요하기 때문에 실행시간 동안 캐시 적중률이 평가되고 비효율적이라고 판단되는 캐시는 폐기됩니다.

KEY ONLY 색인 액세스

하나의 칼럼에 대해 색인이 존재하는 테이블이 있고, 질의의 술어 및 결과에 그 칼럼을 이용하는 질의가 있을 때 최적화기는 이것이 KEY ONLY 참조가 될 수 있음을 인지합니다. 이 사실은 explain 출력의 질의 계획에서 확인할 수 있을 것입니다. 하지만 색인에는 없는 칼럼을 참조하거나 평가해야할 또 다른 칼럼이 존재한다면 키 전용 계획을 생성할 수 없습니다.

이 새로운 기법은 단지 질의 결과로 리턴되는 행이 없음을 확인하기 위한 테이블 페이지 액세스를 생략하기 위해 키 전용 검색을 생성합니다. 색인에 존재하지 않는 필터가 여럿 있는 경우에도 필터에 키 전용 계획을 적용할 수 있습니다. 이를 이용하면 매우 짧은 시간에 질의의 잠재적 완료에 도달할 수 있습니다. 검색 대상을 줄이는 것이 키 전용 계획의 목표입니다. 질의 결과로 리턴되는 행이 0개임이 확인되면 즉각 질의를 중단할 수 있습니다. 색인에 대한 검색은 상당히 빠르고 키 전용 색인 액세스를 이용하면 색인되지 않는 칼럼을 일단 제외시킬 수 있기 때문에 상당한 효과를 거둘 수 있습니다. PeopleSoft 질의 중 어떤 것은 기존의 시스템에서 50초 동안 실행되지만 이 기법을 이용할 경우 1초 미만의 시간에 결과를 얻을 수 있습니다. 경쟁 업체 제품에도 이 기법이 구현되어 있습니다.

메모리 상주 테이블(Memory Resident Table)

기존의 버퍼 관리기는 두 개의 LRU 큐를 이용합니다. 하나는 사용가능 페이지를 위한 것이고 다른 하나는 변경된 페이지를 위한 것입니다. 변경된 페이지가 디스크로 플러시되면 그 페이지는 사용가능 큐의 사용 시점이 가장 오래된 페이지 자리로 옮겨집니다. 변경된 페이지 큐가 비교적 작고(검사점을 줄이기 위해) 페이지 변경이 빈번한 경우 디스크 상의 한 부분으로 페이지 액세스가 집중될 수 있습니다. 어떤 페이지가 끊임없이 변경되고 디스크로 플러시된다고 했을 때 그 페이지는 디스크로 플러시된 후 곧 이어 다른 읽기 작업에 다시 사용될 것입니다. 하지만 그 페이지는 디스크 플러시 후 사용가능 큐로 옮겨질 것이고 원본 페이지에 대한 액세스는 디스크로 향하게 됩니다.

이런 문제를 해결하기 위해서는 집중적으로 사용되는 테이블을 지정하고 그 테이블의 페이지를 위해 버퍼의

일정 부분을 할당할 수 있는 수단이 필요합니다. 그러면 자주 사용되는 페이지 버퍼가 일반적인 버퍼처럼 사용가능 큐로 해제되는 일은 없을 것입니다.

새로운 ONCONFIG 매개변수 MAX_RES_BUFFPCT가 추가됩니다. 이 매개변수의 값은 메모리 상주 테이블 페이지에 사용될 버퍼의 총량을 BUFFERS의 백분율로 지정합니다. onmode -X nnn 명령(nnn은 BUFFERS의 백분율)을 이용하면 이 값을 변경할 수 있습니다.

상주 버퍼로 관리할 테이블은 SET TABLE | INDEX 테이블명 RESIDENT | NONRESIDENT [분할_옵션] [WITH INDEX ALL] 명령으로 지정합니다. 이 명령을 이용하면 특정 테이블의 상주 여부를 설정할 수 있습니다. 이와 함께 분할_옵션은 테이블의 어떤 분할 영역을 캐시할 것인지 지정합니다. 기본으로 모든 분할이 캐시되지만 필요한 경우 테이블 또는 색인의 가장 자주 사용되는 분할로 범위를 축소할 수 있습니다.

예제:

SET TABLE hotone RESIDENT

이름이 hotone 인 테이블의 모든 페이지 및

이 테이블에 생성된 색인의 모든 페이지가 상주되도록 합니다.

SET TABLE hotone RESIDENT on dbs1,dbs3

테이블의 dbs1, dbs3 분할의 페이지만이 상주되도록 합니다.

여기에는 테이블 페이지 및 생성된 색인의 색인 페이지가 포함됩니다.

SET TABLE hotone RESIDENT WITH INDEX ALL

테이블 및 그 테이블의 분리된 색인(detached index)이 상주되도록 합니다.

SET INDEX hotindex RESIDENT

분리된 색인 hotindex 의 페이지만이 상주되도록 합니다.

SET INDEX hotindex ON dbs3

분리된 색인의 dbs3 분할만이 상주되도록 합니다.

동일한 테이블 또는 분리된 색인에 복수의 SET 명령을 반복 사용할 수 있습니다. 메모리 상주 설정이 지속적으로 유지되지 않는다는 점을 주의하십시오. Online을 재시작했다면 테이블 또는 색인의 메모리 상주 여부를 다시 설정해야 된다는 뜻입니다.

- 다음은 상주 테이블 또는 색인에 추가적인 버퍼 페이지가 필요할 때
버퍼 관리기가 취하는 행동을 설명한 것입니다.

1. 상주 테이블임이 표시된 테이블 페이지에 버퍼가 필요하고 상주 버퍼로 사용되는 버퍼 양이 MAX_RES_BUFFPCT의 설정 값 이하인 상황
   이런 경우 버퍼 관리기는 단순히 상주 풀에서 페이지를 할당되고 할당된 페이지에 데이터를 위치시킵니다.
2. 상주 테이블임이 표시된 테이블 페이지에 버퍼가 필요하고 상주 버퍼로 사용되는 버퍼 양이 이미 MAX_RES_BUFFPCT의 설정 값에 도달한 상황
   이런 경우 버퍼 관리기는 일반 사용가능 LRU 목록에서 버퍼를 할당하고 할당된 페이지를 상주시키지 않습니다.

메모리 상주 테이블 기능은 테이블의 특정 부분만 변경하는 OLTP 응용프로그램 또는 몇몇의 중요 테이블을 집중적으로 액세스하는 작업에 상당한 효과를 미칠 것입니다. 이외의 다른 질의에는 부정적인 영향은 없

습니다.

맺음말

이 글에서는 7.3 버전의 주요 기능을 소개했습니다. 또한 7.3 버전의 OLTP 성능 향상을 매우 상세히 분석하였습니다. 이 개정판의 기능 개선들은 고객의 주목을 받을 것이며 나아가 Informix 제품이 데이터베이스 업계에서 최고의 자리를 차지하도록 할 것입니다.

부록 A "OLTP 성능 개선 테스트 방법"

첫째 행 최적화의 일반적인 목적은 테이블 스캔이 아닌 "정렬된 병합"이 수행되도록 색인을 이용하는 것입니다. 전체 소요 시간은 정렬된 병합이 더 길지만 질의의 최초 결과 집합은 빠르게 리턴됩니다.

다음 예제와 같은 질의를 구성합니다:

```
SET EXPLAIN ON;

    SET OPTIMIZATION ALL_ROWS;

    SELECT * FROM emp ORDER BY salary;
```

그리고 sqexplain.out 출력을 검토해 보면 emp 테이블에 대해 테이블 스캔이 수행되고 ORDER BY 절에 임시 테이블이 필요하는 것을 알 수 있습니다.

질의 목표를 재설정하고 emp 테이블의 salary 칼럼에 대한 색인이 존재하는지 확인합니다.

```
SET EXPLAIN ON;

    SET OPTIMIZATION FIRST_ROWS;
```

```
SELECT * FROM emp ORDER BY salary;
```

sqexplain.out 출력을 검토해 보십시오. 주의할 점은 질의가 수행되려면 색인이 존재해야 하고 ORDER BY 절을 만족시키기 위해 수행되는 정렬이 없다는 것입니다.

매우 많은 수의 행이 있는 테이블에 대해 위의 두가지 질의를 실행하고 질의 완료에 소요된 시간을 비교해 보면 최적화 목표에 따라 서로 다른 결과가 나타남을 알 수 있습니다. 최적화 목표는 질의의 전체 수행 시간(소요되는 시간)과 질의 결과의 첫째 행이 리턴되는 시간(반응 시간)에 영향을 미칩니다.

Informix 암호가 변경되거나 유효기간이 만료되었다면 Informix 서버 서비스를 시작할 수 없습니다. Informix 서버를 시동하려면, 서비스 창에서 Informix 서버를 선택하여 시작시킵니다. 시스템 시작시 서비스가 시작되도록 설정할 수도 있습니다.

Informix 서버 서비스는 반드시 Informix 사용자만이 시동시킬 수 있다는 것에 주의하십시오.

다른 서버 문제에 대해서는 Informix 홈 디렉토리에 위치하는 ONLINE.LOG 파일을 참고하십시오. 만약 ONLINE.LOG 파일이 생성되지 않았다면, 관리자 도구 창의 이벤트 로그에서 Informix 서비스가 시동되지 않는 원인에 대한 정보를 얻을 수 있을 것입니다.

Informix 서버는 클라이언트와 TCP/IP를 통해 정보를 교환합니다. 기본 TCP/IP 서비스는 포트 번호 1526의 turbo입니다. 모든 서비스에 대한 정의는 다음 파일에 기록되어 있습니다.

최적화기 지시어

최적화기 지시어와 관련해서 테스트해야 할 것은 여러가지가 있습니다. 지시어를 사용하여 사용할 색인을 지시하고, 스캔 사용을 조정하고 sqexplain.out 파일의 지시어 정보를 검토해 보십시오.

SET EXPLAIN ON이 수행되지 않았을 때 EXPLAIN 지시어를 사용하는 질의도 시험해 보아야 할 것입니다. 이를 통하여 실행 시간 동안 특정 질의의 설명 출력을 얻을 수 있다는 사실을 확인할 수 있습니다.

덧붙여 IFX_DIRECTIVES 환경변수를 0으로 설정하고 위와 같은 테스트로 sqexplain.out을 검사하여 기능 명세에 설명된 바와 같이 지시어 처리가 수행되지 않도록 설정할 수 있는지 확인하십시오.

순위 질의를 위한 최초 N행 지원

SELECT FIRST 'n' 문과 ORDER BY 절을 이용하면 상위 N 질의를 구성할 수 있습니다. 다음과 같은 질의로 이 기능을 시험할 수 있습니다.

```
/* 봉급이 가장 많은 직원을 10 위까지 검색 */

SELECT FIRST 10 name,salary

FROM emp

ORDER BY salary;

/* 가장 많은 배달료를 10 위까지 검색 */

SELECT FIRST 10 ship_charge, order_num

FROM orders

ORDER BY ship_charge DESC;
```

부질의 성능

이 기능 개선에 대한 확실한 테스트 방법은 PeopleSoft 벤치마크입니다. 그러나 몇 가지 질의를 수행해 봄으로서 부질의 성능 개선을 확인할 수 있습니다.

필자는 Wisconsin 데이터베이스를 작성하고 tenktup1 테이블과 tenktup2 테이블에는 10,000,000개의 행을, onektup 테이블에는 1,000,000개의 행을 저장했습니다. 데이터베이스가 구축된 이후 7.2x 엔진과 7.3 엔진에서 다음과 같은 질의를 실행하여 결과를 비교해 보았습니다.

```
SELECT tenktup1.tenPercent,

  tenktup1.twentyPercent,

  SUM ( tenktup1.unique1 ),

  SUM ( tenktup2.unique2 ),

  COUNT (*)

FROM tenktup1, tenktup2

WHERE tenktup1.unique1 = tenktup2.unique1

  AND tenktup1.unique1 BETWEEN 0 AND 50

  AND tenktup1.twentyPercent =

    ( SELECT MAX ( twentyPercent )
```

```
        FROM onektup

    WHERE onektup.tenPercent = tenktup1.tenPercent

        AND MOD ( onektup.tenPercent, 2 ) = 0 )

GROUP BY 1, 2 ;
```

최적화기가 위의 질의를 전개(중첩 해체)할 것으로 보이지만 진정한 결과는 실제로 두 버전에서 질의를 실행해 보아야만 알 수 있습니다. Informix Dynamic Server V7.23에서 실제 실행했을 때 위와 같은 질의는 임시 테이블로 조인되도록 재작성하지 않으면 완료되지 않습니다. 매우 오랜 시간이 지나도 완료되지 않습니다.) 이 테스트 목적은 질의를 재작성하지 않아도 되게끔 7.3 버전이 자동으로 최적화하는지 확인하는 것입니다.

메모리 상주 테이블

50%의 메모리가 상주 테이블에 할당되도록 서버를 구성한 다음 onstat -R 명령을 실행하여 새로운 큐 집합과 상주 페이지 크기를 확인합니다.

Wisconsin 데이터베이스에 대해 SET TABLE onektup RESIDENT 명령을 실행하고 SELECT SUM(unique1) FROM Onektup 질의를 수행합니다. onstat -R 명령을 다시 실행하여 상주 메모리 버퍼에 읽혀진 페이지의 갯수를 확인합니다.

메모리 상주 테이블 기능의 진정한 시험이 가능한 때는 상당 수의 페이지를 갱신하는 작업 소요가 발생하여 디스크 플러시가 빈번히 발생하는 상황입니다. 이런 작업이 실행 중일 때 소규모 행 집합(하나 또는 적은 수의 페이지)을 지속적으로 갱신하는 단일 프로세스를 생성할 수 있을 것입니다. 메모리 상주 테이블 기능이 없다면 갱신되는 페이지가 캐시되지 않고 디스크로 플러시된 후 또 다시 디스크에서 읽혀질 것입니다.

여기서 30분 동안 갱신 횟수를 조사하여 이 작업의 초당 갱신율을 판단할 수 있을 것입니다. 초당 갱신율을 얻었다면 이번에는 테이블을 상주시키고 위의 시험을 다시 수행하십시오. 자주 갱신되는 테이블이 메모리에 상주됨으로서 얻어지는 처리 속도의 향상이 확연히 드러날 것입니다.

HP-UX 시스템에서 대형 Informix 데이타베이스를 위한 디스크 할당

소개

이 글은 필자가 대형 통신 업체 컨설팅에서 얻은 경험을 바탕으로 작성하였습니다. 컨설팅의 일부는

나머지 부분에서 이런 장점들이 보여질 것입니다. 이 글을 읽는데는 디스크 공간 할당에 관련된 문제에 대한 약간의 지식이 요구됩니다.

논리 볼륨관리자

HP-UX LVM은 고정 파티션에서는 얻을 수 없는 여러 가지 이점을 제공합니다. HP-UX LVM 파티션(논리 볼륨이라 불리기도 함)은 여러 디스크 드라이브에 걸쳐 정의할 수 있으며 이를 통한 입출력 부하 분산과 디스크 드라이브 한계 이상의 파티션 정의를 지원합니다. HP-UX LVM과 함께 HP MirrorDisk/UX 제품을 이용하면 동일 데이터의 사본 세 개가 동시에 저장되고 갱신되는 시스템을 구축할 수 있습니다. 또한 HP-UX LVM에는 논리 볼륨을 동적으로 확장, 축소하는 기능과 다른 디스크로 논리 볼륨을 이전하는 기능이 있습니다. HP-UX LVM를 이용하면 특정 응용 프로그램에 속하는 디스크 그룹을 한 시스템에서 다른 시스템으로 내보내기 또는 가져오기를 할 수 있습니다

HP-UX LVM은 복수의 디스크를 볼륨 그룹이라 불리는 사용 가능 공간의 모음으로 결합하는 수단을 제공합니다. 이렇게 모아진 디스크 공간은 데이터베이스, 파일 시스템, 스왑 영역 등의 디스크 데이터를 수용하기 위한 논리 볼륨으로 원하는 크기로 재분할하여 사용할 수 있습니다. 볼륨 그룹으로 모아진 디스크 공간은 각 디스크의 물리적 경계에 구애받지 않고 할당할 수 있습니다. LVM 논리 볼륨을 할당할 때 특정한 드라이브를 지정할 수 있습니다. 이것은 주로 성능에 관련 이유 때문일 것입니다.

디스크 할당 작업에서 다뤄야 할 기본 요소는 네 가지가 있습니다. 첫번째는 디스크, 즉 물리 볼륨입니다. 디스크는 **pvcreate** 명령을 통해 LVM 물리 볼륨으로 전환되며 **pvcreate** 명령은 디스크 시작 부분에 LVM 데이터 구조를 기록합니다. 두 번째는 물리 볼륨이 모여서 구성되는 대규모의 가상 공간, 즉 볼륨 그룹입니다. 세 번째 요소 논리 볼륨은 볼륨 그룹에서 할당됩니다. **vgcreate** 명령은 볼륨 그룹을 생성하고 볼륨 그룹에 디스크를 배당합니다. 논리 볼륨은 **lvcreate** 명령으로 생성됩니다. 볼륨 그룹을 생성할 때는 몇 가지 매개변수를 지정합니다. 볼륨 그룹의 활용 폭은 지정된 매개변수에 의해 규정되는데 매개변수 설정을 변경하려면 전체 볼륨 그룹을 재구축해야 하기 때문에 볼륨 그룹을 생성할 때는 매개변수 값을 신중히 고려해야 합니다. 네 번째 요소는 물리 볼륨 그룹입니다. 물리 볼륨 그룹은 볼륨 그룹에 속한 디스크를 하드웨어 특성에 따라 구분한 것입니다. 미러링을 사용할 때는 미러되는 양쪽 논리 볼륨이 서로 다른 I/O 카드에 맞물리도록 물리 볼륨 그룹을 구성합니다(I/O 채널 분리). 첫번째 볼륨 그룹에는 "SCSI 카드 A"에 물린 모든 디스크

가 포함되도록 하고 두 번째 볼륨 그룹에는 "SCSI 카드 B"에 연결된 모든 디스크가 포함되도록 구성할 수 있습니다. 물리 볼륨 그룹은 **vgcreate** 또는 **vgextend** 명령으로 생성합니다.

볼륨 그룹 및 논리 볼륨의 크기를 확장, 축소하거나 설정 내용을 확인하고 일부 특성을 변경할 수 있습니다. 이에 사용되는 명령은 볼륨 그룹인 경우 **vgextend, vgreduce, vgdisplay, vgchange**이며, 논리 볼륨인 경우 **lvextend, lvreduce, lvdisplay, lvchange**입니다. 이에 덧붙여 미러 관리 명령 **lvsync, lvchange, lvmerge**가 있으며 볼륨 그룹을 다른 시스템으로 이전하거나 볼륨 그룹 구성을 관리하는 명령 **vgscan, vgcfgbackup, vgcfgrestore**가 있습니다.

LVM 명령 실행은 시스템 관리 도구(SAM)를 이용할 수 있습니다. 하지만, 대규모의 복잡한 구성이라면 LVM 스크립트를 작성하는 것이 훨씬 단순한 방법입니다. LVM 기능 중 논리 볼륨을 특정 드라이브에 할당하거나 논리 볼륨을 다수의 디스크에 분산시키는 기능은 SAM에서 수행할 수 없습니다. 이 두 가지는 대형 데이터베이스 구성에 필수적인 기능입니다.

볼륨 그룹 공간을 논리 볼륨에 할당하는 방법은 기본적으로 두 가지가 있습니다. 첫번째는 볼륨 그룹이 속하는 물리 볼륨을 고려하지 않고 단순히 볼륨 그룹에서 공간을 할당하는 방법입니다. 이런 방식의 논리 볼륨에는 볼륨 그룹에 디스크가 추가된 순서에 따라 디스크 공간이 차례대로 할당됩니다. 이러한 유형의 할당은 매체의 종류는 고려하지 않아도 된다는 점에서 데이터베이스 관리자(DBA)에게 추상적입니다. 미러링의 I/O 채널 분리는 물리 볼륨 그룹을 통하여 달성할 수 있습니다.

그러나, 대개의 경우 성능 개선의 이유때문에 또는 미러링을 위한 I/O 분리 수단으로 데이터를 특정 드라이브에 위치시킬 수 있는 두 번째 유형의 논리 볼륨 할당이 사용됩니다. Informix OnLine Dynamic Server, 버전 7.1과 같은 데이터베이스 서버의 고도로 병렬화된 질의의 처리 속도를 높이기 위해서는 드라이브별 논리 볼륨 배치가 매우 중요합니다. 볼륨 그룹 내의 특정 장치를 지정하는 논리 볼륨 할당에는 lvcreate 대신 lvextend 명령을 사용합니다. 이어지는 절에서 각각의 LVM 명령을 보다 상세히 설명할 것입니다.

Informix와 LVM

데이터베이스 환경을 위한 논리 볼륨을 구성할 때는 모든 필요 조건들이 만족될 수 있도록 DBA와 협의해야 합니다. 이를 위해서는 핵심적인 Informix 용어 및 개념을 이해하는 일이 반드시 선행되어야 할 것입니다. 이 글에서는 Informix 논리 구조의 크기 결정에 대해서는 다루지 않습니다. 세부적인 크기 결정은 DBA의 몫이며 이에 대한 내용은 Informix 설명서를 참조해야 할 것입니다.

물리적 구조

Informix 데이터베이스의 기본 구성 단위는 청크입니다. 청크는 HP-UX 논리 볼륨으로 생성됩니다. 청크를 생성할 때는 단순히 청크가 위치할 논리 볼륨의 문자 논리 볼륨 장치 파일을 지정하면 됩니다.

Informix 데이터베이스 영역, 즉 DB영역은 하나 또는 여러 개의 청크로 이루어집니다. DB영역은 LVM의 볼륨 그룹과 거의 유사한 기능을 수행합니다. DB영역은 테이블, 색인, 로그 등과 같은 다양한 형식의 Informix 논리 구조가 담겨지는 디스크 공간의 논리적 모음입니다. Informix 데이터베이스 서버는 처음 초기화될 때 하나의 청크로 루트 DB영역을 작성합니다. Informix 청크의 최대 크기는 2GB로 제한되기 때문에 Informix 데이터베이스를 위한 논리 볼륨은 2GB를 넘지 않아야 합니다. Informix가 사용하는 문자 논리 볼륨 장치 파일의 소유자 및 그룹은 Informix로 설정되어야 하고 그룹 및 소유자에 읽기/쓰기 권한이 부여되어야 합니다.

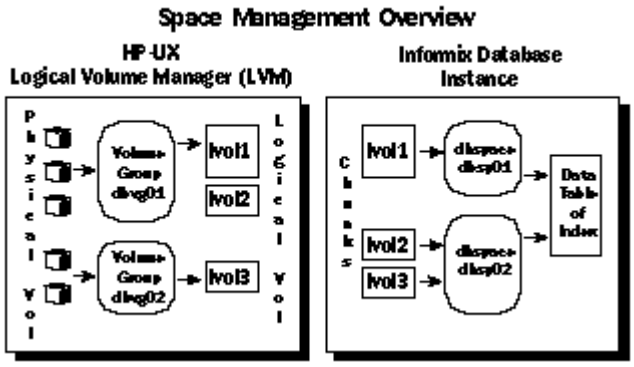


그림 1: 논리 볼륨 관리자와 비교한 Informix 데이터베이스

Informix 환경의 물리적 공간을 나타내기 위해 사용되는 Informix 용어는 이외에도 몇 가지가 더 있습니다. Informix 테이블 및 기타 논리 구조들은 2K 페이지 단위로 할당됩니다. 페이지는 DB영역에서 할당되고 페이지의 집합을 익스텐트라고 합니다. 처음에 테이블은 최초 익스텐트 크기라고 불리는 특정 크기의 페이지

집합으로 할당되며 할당된 테이블은 추가 익스텐트 크기라고 불리는 단위로 확장됩니다. Informix 익스텐트는 항상 하나의 청크에 완전히 포함되어야 하고 청크 내에서 연속된 페이지로 할당되어야 합니다.

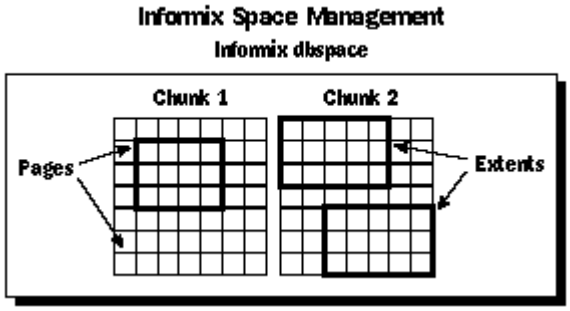


그림 2: Informix 디스크 공간 관리

Informix 물리적 공간 할당에 관련된 개념의 마지막은 변위입니다. 변위는 논리 볼륨을 재분할하여 더 작은 청크로 사용하기 위한 것입니다. HP-UX 운영 체제 관점에서 보면 청크는 이진 문자열입니다. 변위는 청크의 시작점을 나타내는 논리 볼륨 킬로바이트 수입니다. 예를 들어, 1MB 크기의 논리 볼륨을 생성하고 그 논리 볼륨을 500KB 크기 청크 두 개로 나눈 경우, 변위 0은 첫번째 청크의 시작점 그리고 변위 500KB는 두 번째 청크의 시작점이 됩니다. 변위는 DB영역을 생성할 때, DB영역에 청크를 추가할 때, DB영역에서 청크를 삭제할 때 지정해야 합니다.

이 글에서는 BLOB(이진 대형 객체), BLOB페이지, BLOB영역에 대한 내용을 다루지 않습니다. 하지만, BLOB를 위한 디스크 할당도 주의를 기울여야 합니다. 자세한 내용은 Informix OnLine Dynamic Server Administrator*s Guide를 참조하십시오.

논리적 구조

지금까지 Informix 공간 할당의 물리적 측면에 대해 설명했습니다. 이제는 논리적 Informix 구조를 다룰 차례입니다. 실질적으로 물리적 구조는 논리적 구조에 할당됩니다. 첫번째 논리적 구조는 Informix 인스턴스, 즉 Informix OnLine Dynamic Server 데이터베이스 서버(엔진이라 부르기도 함)입니다. Informix 인스턴스는 Informix 데이터베이스가 존재하는 환경입니다. Informix 인스턴스에는 하나의 루트 DB영역이 있습니다. 루트 DB영역은 Informix 환경의 내용을 저장하는 많은 수의 내부 테이블로 구성됩니다. Informix 로그는 Informix 환경의 ACID 특성을 유지하기 위한 것이고 개별 데이터베이스가 아니라 Informix 인스턴스 전체

에 관련됩니다. 하나의 시스템에서 복수의 Informix 인스턴스를 사용할 수도 있습니다.

Informix DBA가 배치해야 하는 논리적 구조는 테이블, 색인, 임시 영역, 논리 로그 및 물리 로그와 각각의 데이터베이스에 대한 정보가 기록되는 데이터베이스 카탈로그입니다. 이들을 위한 공간이 마련될 수 있도록 물리적 공간을 구성해야 합니다. Informix OnLine Dynamic Server Administrator*s Guide에는 성능, 가용성, 복구성에 대한 고려 사항을 포함한 Informix 구조 배치 작업의 윤곽을 설명하고 있습니다.

Informix 서버 제품

Informix OnLine Dynamic Server(버전 6 및 7.1)는 병렬 질의와 테이블 분할 기능을 갖춘 데이터웨어하우스에 매우 적합한 데이터베이스 서버입니다. Informix OnLine(버전 5.x) 데이터베이스의 최대 크기는 대략 130GB인 반면, Informix OnLine Dynamic Server 데이터베이스는 4테라 바이트를 초과할 수 있습니다. 당연히 Informix OnLine Dynamic Server와 Informix OnLine의 논리 볼륨 배치는 전혀 다릅니다.

Informix OnLine 데이터베이스를 위한 디스크 공간을 할당할 때는 논리 볼륨 스트라이핑을 활용하는 것이 일반적입니다. 스트라이핑은 Informix 청크에 대응되는 논리 볼륨을 복수의 디스크에 분산시키는 것으로 스트라이핑을 사용하면 프로세스의 동일 테이블에 대한 질의 또는 갱신이 병렬 I/O 수준으로 수행되므로 처리 속도가 향상됩니다. 스트라이핑은 논리 볼륨 수준에서 이루어지며DBA는 이에 대해 고려하지 않아도 됩니다.

Informix OnLine의 경우 지정할 수 있는 청크 수에 한계가 있습니다. 그리고 이 한계는 청크에 대응되는 볼륨의 완전 경로명 등의 이름의 길이로 결정됩니다. Informix OnLine Dynamic Server의 한계는 2,048 청크입니다. 또한, Informix OnLine에서 논리 볼륨명은 제한된 크기의 공간에 저장됩니다. 이러한 공간 제한 때문에 청크 수를 최대로 하려면 논리 볼륨 명을 모두 세 글자 이하(예를 들어, "/aa")로 정하는 것이 필수적입니다. 이런 짧은 볼륨명을 만들려면 /디렉토리(루트 디렉토리)에 소프트 링크를 설정하는 것이 좋습니다. 소프트 링크 생성에는 ln -s 명령을 사용합니다.

Informix OnLine Dynamic Server인 경우 DBA는 병렬 수행 성능이 최대화되도록 데이터를 위치시킬 드라이브 선정에 좀더 주의를 기울여야 합니다. 덧붙여 Informix OnLine Dynamic Server 버전 7.1은 분할화라

고 불리는 새로운 테이블 분할 기능을 지원하는데 이에 대한 내용은 다음 절에서 다룹니다.

Informix OnLine Dynamic Server 버전 6 및 버전 7.1에서는 임시 영역을 Informix 청크 또는 외부 디스크 공간에 할당할 수 있으나 Informix OnLine 버전 5의 임시 영역은 외부 디스크 공간에만 할당할 수 있습니다. 기본 설정으로 Informix OnLine은 임시 영역으로 /tmp 디렉토리를 사용합니다. Informix 매개변수를 조정하면 임시 영역으로 다른 디렉토리를 사용할 수 있습니다.

분할

분할은 알고리즘 또는 스키마에 따라 테이블 내의 행 또는 색인 키를 그룹으로 나누는 Informix의 기능입니다. 각각의 그룹 또는 분할(파티션이라 말하기도 함)은 특정 물리 디스크의 서로 다른 DB영역에 저장될 수 있습니다. 분할 생성 및 DB영역에 분할을 지정할 때는 SQL 문을 이용합니다. 분할 방법에 따라 각 테이블 행은 알맞게 그룹화됩니다. 행의 키 값에 기반하여(표현식 기반 분할; expression-based fragmentation) 또는 단순 라운드 로빈(round-robin) 할당 방식으로 분할할 수 있습니다. 테이블 분할의 저장 단위는 DB영역이며 분할된 테이블은 복수의 DB영역을 사용합니다. 다음 예를 보십시오

CREATE TABLE supplier...FRAGMENT BY EXPRESSION

supplier_id > 1000 AND supplier_id >=<= 2000 IN dbspace1, supplier_id> 2000 AND

supplier_id <= 3000 IN dbspace2, supplier_id><= 3000 AND supplier_id <<= 4000 IN dbspace3, supplier_id> 4000 IN dbspace4

코드 예제 1: 테이블을 생성할 때 Informix 표현식 기반 분할을 사용

분할화는 특히 의사결정 지원 응용 프로그램 유형 질의의 성능을 월등히 향상시킵니다. 테이블이 어떤 키 값에 따라 동일한 크기의 분할로 분할화 되었을 때 Informix OnLine Dynamic Server 버전 7.1의 병렬 데이터 질의(PDQ) 기능은 복수의 스레드를 할당하여 복수의 분할에서 병렬로 데이터를 참조하고 불필요한 분할을 제외시킵니다. 예를 들어, 어떤 질의가 supplier_id가 2100 보다 큰 모든 레코드를 요구했다면 해당 레코드가 담겨 있는 분할에 대해서만 병렬 I/O가 수행됩니다. 각각의 분할이 서로 다른 드라이브에 할당되었다면

I/O 병렬화가 더욱 향상될 것입니다. 이 기법은 SMP 기반 하드웨어와 함께 사용했을 때 더욱 효과적입니다. 분할 스키마에 관련된 DB영역을 위해 논리 볼륨과 청크를 할당할 때는 각 DB영역의 I/O 분리를 유지하는 것이 중요합니다.

디스크 드라이브 및 디스크 배열

디스크 드라이브를 논할 때 드라이브 성능과 오류 발생률이 제일 먼저 거론됩니다. (이어서 비용 문제가 고려될 것입니다.) 최근 몇 년 동안 디스크의 MTBF(평균 고장 간격) 값은 월등히 높아졌지만, 디스크 드라이브는 특히 수 백 개의 드라이브가 사용되는 시스템에서는 여전히 하드웨어 고장이 가장 일어나기 쉬운 장치입니다. RAID 배열 및 미러 사용 디스크와 같은 고가용성 기능은 대부분의 디스크 고장으로부터 시스템을 보호할 수 있습니다. HP-UX LVM 버전 10.0과 함께 제공되는 최신 기술의 디스크 배열은 인터페이스 카드, 케이블, 컨트롤러, 전원 공급장치 고장으로부터의 보호 기능도 제공합니다.

성능면에 있어서는 단독으로 사용되는 디스크가 RAID 3 또는 RAID 5 모드 디스크 배열보다 우수하다는 사실이 경험으로 입증되고 있습니다. RAID 5 배열은 임의 읽기 작업의 동시성이 높을 때 높은 성능을 나타내고 RAID 3 배열은 대규모 순차 쓰기 작업에 뛰어나지만 전체적인 성능과 가용성을 고려했을 때 최고의 선택은 미러되는 단독 디스크입니다. 비용 대 성능도 면밀히 검토해야 할 것입니다.

디스크 구성의 가용성을 높이기 위한 수단으로 디스크 배열을 선택했다면 어떤 RAID 모드가 적절한가라는 질문이 남습니다. 최신 HP 9000 서버 시스템에서는 RAID 모드 3과 모드 5를 사용할 수 있습니다. RAID 3은 대규모 순차 디스크 작업에 높은 성능을 발휘하고 읽기 보다 쓰기의 비중이 높은 환경에 적절합니다. RAID 5는 I/O 동시성이 높고 임의 액세스 환경에서 쓰기 보다 읽기의 비중이 높은 작업을 처리하는데 최적입니다. 한 시스템에서 두 가지 모드를 혼용하는 것도 좋은 방법입니다. 대규모 순차 액세스 데이터가 많은 파일 시스템에는 RAID 3 배열을 사용하고 OLTP 데이터베이스에는 RAID 5를 사용해야 할 것입니다. 의사결정 지원 데이터베이스에는 RAID 3이 적절합니다.

디스크 배열에 관련에서 종종 문제시되는 것은 인터페이스 카드와 컨트롤러의 처리 속도입니다. 하나의 연결을 통해 엄청난 양의 데이터가 액세스되기 때문에 하나의 인터페이스 카드에는 디스크 배열을 세 개에서 다섯 개 정도만 연결하는 것이 좋습니다. 각각의 배열이 많은 양의 데이터를 보유하고 데이터를 내부적으로

배열의 여러 드라이브에 분산시키지만 개별 디스크 배열 및 인터페이스 카드로 데이터를 균등히 분산시키는 일은 여전히 중요합니다. 하나의 디스크 배열은 상당한 양의 I/O 요청을 처리할 수 있습니다. A3232A Model 10, 20 디스크 배열은 초당 300 I/O를 처리할 수 있으며 각각 21, 42GB의 데이터를 저장할 수 있습니다.

디스크 배열 및 단독 드라이브 모두 내부에 메모리 캐시가 있습니다. 드라이브 캐시는 읽기 및 쓰기 성능 모두를 상당한 향상시킬 수 있습니다. HP 디스크 배열의 드라이브 캐시는 dcc 명령으로 조작됩니다. 단독 SCSI 드라이브의 쓰기 작업 즉시 보고(immediate write reporting) scsictl 명령으로 작동시킵니다. 대개의 경우 읽기 캐시는 기본값으로 사용 가능하게 되어 있지만, 쓰기 캐시 또는 즉시 보고는 데이터 일관성 문제 때문에 명시적으로 사용 가능하게 조정해야 합니다. 쓰기 즉시 보고를 사용중일 때 드라이브에 전원 공급이 끊어지면 시스템에는 작업이 성공적으로 완료되었다고 보고되지만 실제 I/O는 손실될 수 있습니다. 따라서 이 기능을 제대로 활용하려면 시스템에 UPS가 장착되어야 할 것입니다

```
dcc -won /dev/rdsk/c14d0s2
```

```
scsictl -m ir=on /dev/rdsk/c0d0s2
```

코드 예제 2: 쓰기 작업 즉시 보고를 실행.

디스크 배열에는 dcc 단독 디스크에는 scsictl 명령을 사용

시스템에 새로 설치한 디스크 배열은 적절한 RAID 모드로 전환해야 합니다. HP C2430D Cascade 디스크 배열은 보통 RAID 모드 5로 조정되어 있습니다. 현재 RAID 모드는 SAM 또는 /usr/hpC2400/bin 디렉토리에 있는 dsp 명령으로 확인할 수 있습니다. 대규모 디스크 구성 시스템에서 디스크 배열을 검사하거나 재조

정할 때는 SAM을 사용하지 않는 것이 좋습니다. SAM은 디스크 구성 메뉴로 들어가는데 5분 이상 소요될 수 있고 디스크 배열 하나의 상태를 검사할 때마다 메뉴로 다시 들어가야 합니다. 뿐만 아니라 SAM은 디스크 배열을 한번에 한 개씩만 조정할 수 있습니다. 디스크 배열 재조정 작업은 배열 하나당 30분 이상 소요될 수 있는데 동시에 여러 개에 대해 수행하려면 SAM을 사용하지 않아야 합니다. 복수 디스크 배열을 동시에 재조정할 때는 newarray 명령을 사용합니다. 이 명령은 사전 정의된 배열 구성 파일 /usr/hpC2400/etc/arraytab을 참조합니다. 다음 예제에서 Raid_3_5d를 주목하십시오.

```
newarray -NRaid_3_5d /dev/rdsk/c44d0s2 &
newarray -NRaid_3_5d /dev/rdsk/c64d0s2 &
newarray -NRaid_3_5d /dev/rdsk/c72d0s2 &
newarray -NRaid_3_5d /dev/rdsk/c79d0s2 &
```

코드 예제 3: 네 개의 배열을 동시에 RAID 모드 3으로 전환

논리 볼륨 생성

이 절에서는 논리 볼륨 구성에 사용되는 실제 LVM 명령을 설명합니다. 몇몇 중요 개념은 실제 스크립트의 내용과 함께 설명될 것입니다.

LVM 사용을 위한 디스크 초기화

디스크 구성의 첫번째 단계는 디스크를 LVM에서 사용될 수 있도록 초기화하는 것입니다. 이를 위해서는 아래 코드 예제처럼 pvcreate 명령을 사용합니다. 이것은 볼륨 그룹에 포함시키기 위해 준비되는 각각의 디스크 앞부분에 단순히 필요한 정보를 기록하는 명령입니다. pvcreate 명령이 디스크에 존재하는 이전에 기록된 정보를 덮어쓰도록 하려면 -f 옵션을 사용합니다. 시스템 부트 구조를 위한 공간을 미리 마련하는 기능의 또 다른 pvcreate 옵션이 있습니다. 이에 대한 내용은 나중에 다룰 것입니다.

```
&amp;bnsp;&amp;bnsp;&amp;bnsp;pvcreate -f /dev/rdsk/c25d0s2
&amp;bnsp;&amp;bnsp;&amp;bnsp;pvcreate -f /dev/rdsk/c26d0s2
&amp;bnsp;&amp;bnsp;&amp;bnsp;pvcreate -f /dev/rdsk/c27d0s2
```

&bnsp;&bnsp;&bnsp;pvcreate -f /dev/rdsk/c28d0s2

&bnsp;&bnsp;&bnsp;pvcreate -f /dev/rdsk/c29d0s2

&bnsp;&bnsp;&bnsp;pvcreate -f /dev/rdsk/c30d0s2

&bnsp;&bnsp;&bnsp;pvcreate -f /dev/rdsk/c31d0s2

코드 예제 4: 드라이브를 LVM 물리 볼륨으로 초기화

볼륨 그룹 구성을 위한 계획

볼륨 그룹 구성 작업은 어느 정도의 사전 계획을 필요로 합니다. 일반적으로 특정 응용 프로그램 또는 데이터베이스에서 사용되는 디스크들은 하나의 볼륨 그룹으로 묶여져야 합니다. 여러 개의 볼륨 그룹을 이용하는 구성도 고려해야 합니다. 예를 들어, 서로 다른 형식의 디스크는 서로 다른 볼륨 그룹으로 분리하는 것이 가장 좋습니다(단독 SCSI 디스크 볼륨 그룹과 디스크 배열 볼륨 그룹). **newfs** 명령의 필수 매개변수 중 하나는 디스크 형식에 대한 것이며 파일 시스템은 이 디스크 형식 정보에 따라 최적화됩니다. 따라서 논리 볼륨이 서로 다른 유형의 디스크에 걸쳐 생성된 경우 일부분에 대한 액세스 성능이 다른 부분에 비해 떨어질 수 있습니다.

데이터 스트라이핑은 단일 데이터베이스에 속하는 디스크를 여러 볼륨 그룹으로 분할해야 하는 또 다른 이유가 될 수 있습니다. 대규모의 데이터베이스를 세밀한 수준으로 스트라이핑한다면 논리 볼륨 스트라이핑 작업에만 며칠이 소요될 수 있습니다. 디스크를 여러 개의 볼륨 그룹으로 분리하게 되면 스트라이핑 작업을 병렬로 수행할 수 있습니다.(실질적으로 한번에 하나의 볼륨 그룹만 스트라이핑이 가능합니다.) 한 가지 주의할 점은 디스크를 복수의 볼륨 그룹으로 분할하게 되면 데이터가 스트라이핑되는 범위가 제한된다는 것입니다. 즉, 데이터가 더 적은 수의 디스크로 분산됩니다. 논리 볼륨은 자신이 속하는 볼륨 그룹 내의 디스크로만 스트라이핑될 수 있습니다.

복수의 볼륨 그룹이 필요한 또 다른 이유는 단순히 MirrorDisk/UX 및 LVM 명령 수행 속도에 관련된 것입니다. 디스크 미러링을 사용하고 미러 일관성을 위해 미러 쓰기 캐시(MWC) 옵션을 선택했다면 디스크 수가 적은 볼륨 그룹이 성능면에서 더 우수합니다. 볼륨 그룹 내에서 미러되는 모든 논리 볼륨은 동일한 MWC를

공유하므로 MWC가 논리 볼륨간에 쟁점이 될 수 있습니다. MWC에 대한 보다 자세한 내용은 다음 단원에서 다룰 것입니다. 끝으로 볼륨 그룹 구성을 조정하는 LVM 명령은 볼륨 그룹 내의 모든 물리 볼륨을 갱신해야 하므로 디스크가 많을수록 LVM 명령은 시간 소모적인 프로세스가 될 수 있습니다.

데이터베이스 환경을 복수의 볼륨 그룹으로 구분하는 방법은 여러 가지가 있습니다. 데이터베이스 환경에서 서로 다른 데이터베이스의 데이터를 각각 분리된 볼륨 그룹에 위치시키거나, 데이터베이스 환경의 루트 DB 영역과 로그 파일을 분리된 볼륨 그룹에 위치시킬 수 있을 것입니다. 또한, I/O 채널별 데이터 분산을 DBA 에게 좀더 명확히 할 수 있도록 I/O 채널 구성에 따라 디스크를 분리할 수도 있을 것입니다. 한 가지 염두에 두어야 할 것은 하드웨어 구성에 따른 볼륨 그룹 분할이 항상 좋은 것만은 아니며 하드웨어 구성이 자주 변경되는 환경에는 적절치 못하다는 것입니다.

볼륨 그룹 생성

우선 볼륨 그룹명을 결정합니다. 시스템 볼륨 그룹은 기본으로 vg00이라는 이름이 부여됩니다. 그 외의 볼륨 그룹은 vg01, vg02처럼 같은 패턴의 이름을 사용하던가, dbvg01, dbadmin처럼 용도에 따라 이름을 지정할 수 있습니다. **vgcreate** 명령을 사용하기 전에 먼저 **/dev** 디렉토리 밑에 볼륨 그룹명과 동일한 이름으로 디렉토리를 작성해야 합니다. 이 디렉토리에는 논리 볼륨의 원시 문자 장치 파일과 블럭 장치 파일이 저장됩니다. 덧붙여, 이 디렉토리에 group이라는 이름의 문자 장치 파일을 추가해야 합니다. LVM은 볼륨 그룹에 액세스하기 위해 내부적으로 이 장치 파일을 사용합니다. 장치 파일의 마이너 번호 여섯 자리 중 상위 두 자리는 유일한 16진수 볼륨 그룹 번호여야 합니다. 한 시스템에서 사용할 수 있는 볼륨 그룹은 최대 10개이지만 적절한 커널 매개변수를 수정하면 이 수를 늘릴 수 있습니다. 볼륨 그룹 번호를 재사용하려면 시스템을 재시작해야 합니다.

```
mkdir /dev/dbvg02
mknod /dev/dbvg02/group c 64 0x040000

vgcreate -p 32 /dev/dbvg02 /dev/dsk/c25d0s2
/dev/dsk/c26d0s2 ₩ /dev/dsk/c27d0s2 /dev/dsk/c28d0s2
```

/dev/dsk/c29d0s2 ₩

/dev/dsk/c30d0s2 /dev/dsk/c31d0s2 /dev/dsk/c73d0s2 ₩

코드 예제 5: 최대 32개의 물리 볼륨을 허용하는 새로운 볼륨 그룹 생성

vgcreate명령은 실제 볼륨 그룹을 생성하고 물리 볼륨을 할당합니다. 몇 개의 볼륨 그룹 관련 매개변수는 볼륨 그룹 생성시에만 지정할 수 있습니다. 그러므로, 볼륨 그룹 매개변수 값은 주의 깊게 결정해야 합니다. -p 옵션은 볼륨 그룹의 최대 물리 볼륨 갯수를 지정합니다. 기본값은 16개의 물리 볼륨입니다. 볼륨 그룹의 크기를 고려하여 기본 설정을 변경해야 할 것입니다. -l 옵션은 볼륨 그룹에서 할당 가능한 논리 볼륨의 갯수를 지정합니다. 기본값은 255개의 논리 볼륨이며 이 기본 설정은 변경할 필요가 없을 것입니다. -e 옵션과 -s 옵션도 중요하지만 이에 대해서는 다음 절에서 다룰 것입니다. 이 모든 매개변수들은 볼륨 그룹 생성 이후에는 변경이 불가능한 디스크 상의 데이터 구조 크기를 결정합니다. 오늘날의 대형화된 디스크 드라이브에 비하면 이 데이터 구조의 크기는 상대적으로 미미하므로 데이터 구조 크기는 넉넉하게 결정하는 것이 좋습니다. 볼륨 그룹에 디스크를 추가할 때는 vgextend 명령을 사용합니다.

논리 볼륨 생성

논리 볼륨은 lvcreate 명령으로 생성합니다. 볼륨명은 -n 옵션으로 지정하거나 lvcreate 명령이 자동으로 기본 볼륨명을 부여하도록 할 수 있습니다. 기본 볼륨명은 lvol1, lvol2, lvol3 형식으로 부여됩니다. 여러 개의 기본 볼륨명이 할당한 다음 할당된 볼륨명 중 일부를 삭제한 경우, 다음에 생성되는 논리 볼륨의 기본 볼륨명에는 삭제된 볼륨명이 사용됩니다. 논리 볼륨이 할당될 때 볼륨 그룹 디렉토리에 두 개의 장치 파일이 생성됩니다. 블럭 장치 파일의 이름은 논리 볼륨의 이름과 동일하며 원시 문자 파일의 이름은 논리 볼륨의 이름과 동일한데 앞에 r 문자가 첨가됩니다. 기본 볼륨명의 숫자는 논리 볼륨의 번호입니다. 논리 볼륨 번호는 논리 볼륨 장치 파일의 마이너 번호 하위 네 자리 16진수에도 반영됩니다.

내부적으로 논리 볼륨의 크기는 익스텐트 수로 결정됩니다. 익스텐트는 디스크 공간의 1에서 256MB 사이 크기의 디스크 공간이며 익스텐트의 크기는 볼륨 그룹이 생성할 때 vgcreate 명령의 -s 옵션으로 지정합니다. 익스텐트 크기의 기본값은 4MB이며 디스크 스트라이핑을 위해 디스크를 세분해야 하는 경우를 제외하

고 대체로 기본값이 그대로 사용됩니다. lvcreate 명령에서 -l 옵션을 사용하면 논리 볼륨의 크기를 익스텐트 수로 지정할 수 있고 -L 옵션을 사용하면 메가바이트로 지정할 수 있습니다.

논리 볼륨을 볼륨 그룹의 특정 디스크에 할당하려면 lvcreate 명령과 함께 lvextend 명령을 사용해야 합니다. lvextend 명령은 일반적으로 논리 볼륨에 할당된 공간을 증가시키기 위해 사용되지만 어떤 디스크의 공간을 할당할 것인지 지정하는데 사용할 수도 있습니다. 특정 디스크에 논리 볼륨을 할당하려면 lvcreate로 크기가 0인 논리 볼륨을 생성한 다음 lvextend 명령으로 원하는 디스크의 공간을 할당합니다. 한 가지 주의할 것은 두 가지 명령 모두에서 크기는 항상 논리 볼륨의 전체 크기로 지정해야 한다는 것입니다. lvextend 명령으로 디스크 공간을 늘리고자 할 때 추가되는 디스크 공간의 크기는 이전의 크기와 새로운 크기 매개변수의 차이로서 지정됩니다. 다음 예를 보면 디스크 34에서 10개의 익스텐트가 논리 볼륨 svcnpa에 할당되고 디스크 59에서 10개의 익스텐트가 추가로 할당되고 있습니다.

```
lvcreate -n cmpy /dev/raid3a
lvcreate -n svcnpa /dev/raid3a
lvcreate -n npa /dev/raid3a

lvextend -l 10 /dev/raid3a/cmpy /dev/dsk/c39d0s2
lvextend -l 10 /dev/raid3a/svcnpa /dev/dsk/c34d0s2
lvextend -l 20 /dev/raid3a/svcnpa /dev/dsk/c59d0s2
lvextend -l 10 /dev/raid3a/npa /dev/dsk/c71d0s2
```

코드 예제 6: 논리 볼륨을 생성하고 특정 디스크로부터 공간 할당

논리 볼륨 미러링

디스크 미러링은 디스크 고장으로부터 데이터를 보호하는 효과적인 수단입니다. 또한 디스크 미러링은 다른 디스크 보호 방법에 비해 전반적인 성능이 가장 우수합니다. MirrorDisk/UX는 별도로 구입해서 설치해야 하는 제품입니다.

MirrorDisk/UX를 이용하면 하나의 논리 볼륨에 복수의 미러를 지정할 수 있습니다. 미러 사본의 수는 논리 볼륨을 생성할 때 지정합니다. 단독 미러는 원본 더하기 사본 한 개이고 이중 미러는 원본 더하기 사본 두 개입니다. 이중 미러는 디스크 소모가 많지만 더 높은 가용성을 제공합니다. 이중 미러이면 하나의 사본은 오프라인으로 백업을 수행하고 나머지 사본을 통해 고가용성을 계속 유지하는 것이 가능합니다.

디스크 고장이 발생했을 때 미러의 일관성을 유지하는 방법은 미러 쓰기 캐시(MWC)와 미러 일관성 복구 두 가지가 있습니다. MWC를 사용하는 경우 시스템은 다른 프로세스의 수행을 고려하지 않고 최대한 빨리 데이터 일관성을 복구합니다. MWC는 물리 볼륨에 쓰여지는 모든 미러 데이터를 추적하여 시스템 장애로 쓰기가 완료되지 않은 미러 쓰기 작업의 기록을 유지합니다. 따라서 복구 과정 동안에는 입출력이 진행 중이었던 물리 익스텐트만이 복구됩니다. MWC의 쓰기 빈도는 데이터베이스 로그 파일과 같은 순차 액세스 논리 볼륨에 대해서는 낮지만 액세스가 비순차화될수록 빈도가 높아집니다. 그러므로 복구 소요시간보다 성능이 중요하다면 데이터베이스 데이터가 있는 논리 볼륨이나 대규모 파일에 대해 불규칙적으로 쓰기 작업이 발생하는 파일 시스템에는 MWC를 사용하지 말아야 합니다.

미러 일관성 복구(비MWC 복구)는 각각의 미러되는 논리 볼륨에 전체 미러 사본을 생성하는 방식입니다. 디스크 고장이 발생한 다음 시스템을 재시작하면 각각의 볼륨 그룹에 대해 백그라운드 프로세스가 시작되어 미러 일관성 복구 방식을 사용하는 모든 논리 볼륨에 완전한 미러 사본이 만들어집니다. 논리 볼륨의 일관성 복구는 차례대로 진행되며 사본이 생성되는 동안에도 논리 볼륨에 액세스할 수 있습니다. 하지만 미러 사본 구축이 완료될 때까지 고가용성 특성은 사라질 것입니다.

LVM 미러 일관성 유지 기능을 사용하지 않는 미러링도 가능합니다. 자체적으로 데이터 일관성을 유지 또는 복구하는 수단이 있는 응용 프로그램 또는 대부분의 데이터를 재구축하는 응용 프로그램을 실행한다면 이런 미러링이 유용할 것입니다. 시스템 스왑 영역도 일관성 복구를 필요로하지 않는 데이터의 한 예입니다. 논리 볼륨에 무효한 익스텐트가 있다면 논리 볼륨 동기화 작업이 필요할 것입니다. LVM에는 이런 기능의 명령 **vgsync, lvsync**가 있습니다.

대규모의 미러되는 논리 볼륨을 빠르게 생성하고자 한다면 먼저 디스크 공간이 할당되지 않은 논리 볼륨을

생성합니다. 그런 다음 미러 디스크 쌍의 첫번째 디스크에 한 개의 익스텐트를 할당하고 이어서 미러를 생성합니다. 미러 생성에는 원본 논리 볼륨의 모든 익스텐트를 신규 미러 논리 볼륨으로 복사하는 작업이 포함되지만 이 경우에는 복사되는 익스텐트가 하나 뿐일 것입니다. 마지막으로 논리 볼륨을 원하는 크기로 확장합니다. 미러가 이미 생성되었기 때문에 논리 볼륨 크기를 확장할 때 추가적인 데이터 복사가 필요하지 않습니다. 이것이 미러되는 논리 볼륨을 작성하는 가장 빠른 방법입니다. 기존 논리 볼륨에 미러를 추가하는 경우 미러 양측을 동기화 하는데 최대 30분 정도의 시간이 소요될 수 있습니다.

lvcreate -M n -c y /dev/dbvg02

lvcreate -M n -c y /dev/dbvg02

lvextend -l 1 /dev/dbvg02/lvol1 /dev/dsk/c25d0s2

lvextend -m 1 /dev/dbvg02/lvol1 /dev/dsk/c73d0s2

lvextend -l 500 /dev/dbvg02/lvol1 /dev/dsk/c25d0s2

/dev/dsk/c73d0s2

lvextend -l 1 /dev/dbvg02/lvol2 /dev/dsk/c26d0s2

lvextend -m 1 /dev/dbvg02/lvol2 /dev/dsk/c74d0s2

lvextend -l 500 /dev/dbvg02/lvol2 /dev/dsk/c26d0s2

/dev/dsk/c74d0s2

코드 예제 7: 논리 볼륨을 생성하고 비MWC 미러 일관성 방식의 미러링 적용

MWC를 사용하려면 **lvcreate** 및 **lvchange** 명령에서 **-M** y 옵션을 지정합니다. MWC를 사용하지 않는 경우 **-c** y 옵션을 지정하면 고장 발생시 비MWC 방식으로 복구가 진행됩니다. **lvcreate** 또는 **lvextend** 명령에서 **-m 1** 옵션은 단독 미러 논리 볼륨, **-m 2** 옵션은 이중 미러 논리 볼륨임을 지정합니다. 디스크 고장이 발생했을 때 미러를 관리하는 방법에 대한 설명은 다음 단원에서 다룹니다.

논리 볼륨 스트라이핑

논리 볼륨 스트라이핑은 디스크 액세스를 다수의 드라이브에 분산시키는 매우 효율적인 방법입니다. 스트라이핑의 경우 액세스 분산에 관련된 복잡한 사항이 응용 프로그램에게는 완전히 숨겨지는 장점이 있습니다. LVM은 논리 볼륨이 개별 드라이브의 한계를 넘어서 확장되는 것을 허용합니다. LVM 스트라이핑은 이를 최대한 활용하여 논리 볼륨을 작은 조각으로 다수의 드라이브에 라운드 로빈 방식에 따라 분산시키는 기법입니다. 스트라이핑된 데이터를 사용하는 응용 프로그램은 I/O 동시성의 향상으로 인하여 반응 시간면에서 높은 성능을 나타냅니다. 논리 볼륨 스프라이핑에 따른 사용자의 불편은 전혀 없습니다.

스프라이핑으로 할당되는 논리 볼륨의 조각을 스트라이프라고 합니다. 일반적으로 스트라이프는 LVM 익스텐트 하나의 크기입니다. 하지만, 스트라이프 크기가 익스텐트 하나로 제한되는 것은 아닙니다. 스트라이프의 크기는 데이터 액세스 패턴에 따라 결정되어야 합니다. 임의적이고 적절히 분산된 액세스가 예상된다면 세분화된 스프라이핑은 불필요할 것입니다. 액세스가 임의적이고 고도로 집중되는 경우라면 볼륨 그룹의 기본 익스텐트 크기를 최소값인 1MB로 줄여서라도 스트라이프의 크기를 최소화해야 할 것입니다. 사전 페치 알고리즘을 통하여 일정 수준의 I/O 병렬성이 획득된다면 순차 액세스 환경도 스트라이핑을 통한 성능 향상이 가능합니다.

numstripedisks=5

stripedisk[1]=/dev/dsk/c39d0s2

stripedisk[2]=/dev/dsk/c34d0s2

stripedisk[3]=/dev/dsk/c59d0s2

stripedisk[4]=/dev/dsk/c54d0s2

stripedisk[5]=/dev/dsk/c71d0s2

```
i=1

stripesize=50

maxlvsize=500

lvsize=$stripesize

while [[ lvsize -le maxlvsize ]]

do

   lvextend -l $lvsize /dev/vg02/custdem00

   ${stripedisk[i]}

   let lvsize=lvsize+stripesize

   if [[ i -lt numstripedisks ]]

 then let i=i+1
```

코드 예제 8: 2GB 논리 볼륨을 50 익스텐트(200MB) 크기의 스트라이프로 다섯 개의 디스크에 스트라이핑

LVM 스트라이핑의 문제는 대규모의 논리 볼륨을 작은 크기의 스트라이프로 스트라이핑할 때 소요되는 시간입니다. 각각의 **lvextend** 명령은 볼륨 그룹 내의 모든 물리 볼륨에 정보를 기록해야 하고 대규모의 논리

볼륨을 스트라이핑하는데는 lvextend 명령이 수 백회 사용되어야 하기 때문입니다. 스트라이프 크기가 4MB 이고 볼륨 그룹에 일곱 개의 드라이브가 있다면 하나의 디스크 스트라이핑 작업에 12시간 이상이 소요될 수 있습니다.

스트라이핑을 사용할 디스크를 복수의 볼륨 그룹으로 구분하면 스트라이핑에 요구되는 시간을 상당히 줄일 수 있습니다. 같은 볼륨 그룹 내의 논리 볼륨은 병렬 스트라이핑이 불가능하지만 다른 볼륨 그룹의 논리 볼륨은 병렬로 스트라이핑할 수 있기 때문입니다. 한가지 참고해야 할 사항은 볼륨 그룹 수를 늘리면 논리 볼륨이 스트라이핑 될 수 있는 디스크 갯수가 줄어든다는 것입니다.

참고사항

다른 LVM 작업이 진행되는 동안에는 볼륨 그룹 명령을 사용할 수는 없으므로 스트라이핑 스크립트를 시작하기 전에 필요한 모든 볼륨 그룹을 생성해야 합니다. 하지만 논리 볼륨 명령은 이에 해당하지 않습니다.

HP-UX, 버전 10.0 LVM의 lvcreate 명령에는 스트라이핑 기능에 포함되어 있고 따로 스크립트를 작성하지 않아도 됩니다. 앞서 설명한 스크립트도 여전히 사용 가능합니다.

LVM 스트라이핑은 디스크 배열의 스트라이핑과 차이가 있습니다. 디스크 배열은 내부 디스크로 데이터를 분산시킬 때 블럭 또는 바이트 스트라이핑을 이용합니다. 디스크 배열에 논리 볼륨 스트라이핑을 적용하여 복수의 개별 디스크 배열에 논리 볼륨 데이터를 분산시킬 수 있습니다. 이런 경우는 I/O 동시성이 디스크 배열 스트라이핑 보다 더 높은 수준으로 향상될 것입니다.

물리 볼륨 그룹 생성

논리 볼륨 미러링에 디스크를 명시적으로 지정하지 않더라도 물리 볼륨 그룹을 통하여 I/O 채널을 분리시킬 수 있습니다. 먼저 볼륨 그룹에서 한 I/O 카드(또는 I/O 카드 집합)에 부착된 모든 디스크를 하나의 물리 볼륨 그룹으로 모으고 다른 I/O 카드에 부착된 디스크는 두 번째 볼륨 그룹으로 모읍니다. 그런 다음 논리 볼

륨 미러를 서로 다른 물리 볼륨에 생성하면 I/O 채널 분리가 보장됩니다. 물리 볼륨 그룹을 생성할 때는 **vgcreate** 명령에 **-g** 옵션을 사용합니다.

```
mkdir /dev/fsvg

    mknod /dev/fsvg/group c 64 0x090000



    vgcreate -p 32 -g iocard1

        /dev/fsvg /dev/dsk/c209d0s2 dev/dsk/c210d0s2



    vgextend -g iocard2

        /dev/fsvg /dev/dsk/c225d0s2 /dev/dsk/c226d0s2
```

코드 예제 9: 물리 볼륨 그룹 iocard1과 iocard2로 볼륨 그룹을 생성

물리 볼륨 그룹은 /etc/lvmpvg라는 ASCII 파일로 관리됩니다. 물리 볼륨 그룹을 변경할 때나 물리 볼륨 그룹을 추가할 때(모든 디스크가 볼륨 그룹에 추가된 이후) 이 파일을 직접 수정할 수도 있습니다.

```
VG /dev/fsvg
PVG iocard1
/dev/dsk/c209d0s2
/dev/dsk/c210d0s2
PVG iocard2
```

/dev/dsk/c225d0s2

/dev/dsk/c226d0s2

코드 예제 10: ASCII 파일 /etc/lvmpvg 파일의 내용

논리 볼륨 그룹이 생성된 물리 볼륨 그룹을 활용할 수 있으려면 올바른 할당 방식이 지정되어야 합니다. 미러 할당 방식의 기본 설정은 strict이며 논리 볼륨의 미러 사본이 원본 익스텐트와 디스크가 분리되어 할당되는 방식입니다. 할당 방식을 un-strict로 설정할 수도 있습니다. 이런 경우 미러 사본이 원본과 동일한 디스크에 존재하게 됩니다. 물리 볼륨 그룹을 사용할 때는 할당 방식을 **pvg-strict**로 지정해야 합니다. 그러면, 미러 사본이 원본과 다른 물리 볼륨 그룹에 할당됩니다. 할당 방식을 **pvg-strict**로 지정하려면 **lvcreate** 또는 **lvchange** 명령에 **-s g** 옵션을 사용합니다.

lvcreate -m 1 -s g -M n -c y -l 1000 /dev/fsvg

코드 예제 11: 물리 볼륨 그룹을 이용하는 4GB의 미러되는 논리 볼륨 생성

파일 시스템을 위한 논리 볼륨 할당

데이터베이스 공간을 위한 논리 볼륨 뿐만이 아니라 파일 시스템을 위한 공간도 할당해야 할 것입니다. 아래 예제는 순차 액세스에 대해 최적화된 파일 시스템을 생성하는 것입니다. 디스크 I/O를 줄이기 위해 버퍼 크기를 매우 크게 지정했고 회전 지연을 0 ms로 설정했습니다. 대규모의 디렉토리를 사용한다면 예약 영역 크기를 더 작게 설정하는 것이 좋습니다. 기본 값으로 파일 시스템 공간의 10퍼센트가 시스템 영역으로 보존되며 4GB 파일 시스템인 경우 400MB가 보존됩니다. 하지만 대형 파일 시스템이라면 예약 영역으로 2에서 5퍼센트 정도만 사용하는 것이 적절합니다.

mkdir /bdata01

코드 예제 12: 파일 시스템을 생성하고 마운트.

시스템이 시동될 때 파일 시스템이 마운트될 수 있도록 /etc/checklist에 생성된 파일 시스템의 항목을 추가해야 합니다.

시스템 볼륨 그룹 미러링

대형 데이터베이스가 포함된 파일 시스템에서는 LVM 미러링을 이용하여 디스크 고장으로부터 시스템 볼륨 그룹을 보호하는 일이 중요합니다. 미러를 생성하는 절차는 다음에 보이고 있습니다. 시스템 볼륨 그룹은 시스템이 사용하는 논리 볼륨의 모음입니다. 디스크 고장이 발생했을 때 시스템이 정상적으로 작동하기 위해서는 시스템 논리 볼륨에 반드시 미러링을 사용해야 합니다.

```
pvcreate -f -B /dev/rdsk/c8d0s2

vgextend /dev/vg00 /dev/rdsk/c8d0s2

mkboot /dev/rdsk/c8d0s2
mkboot -a "hpux (2/12.6.0;0)/hp-ux" /dev/rdsk/c8d0s2
```

코드 예제 13: 시스템 볼륨 그룹에 새로운 부트 디스크를 추가

시스템 볼륨 그룹을 미러링하는 첫번째 단계는 시스템 부트 영역이 보존된 LVM 디스크를 생성하는 일입니다. 위의 예제처럼 **pvcreate -B** 옵션을 사용합니다. 시스템 볼륨 그룹에 물리 볼륨을 추가할 때는 vgextend 명령을 사용합니다. **mkboot**는 신규 생성된 부트 디스크에 부트 프로그램과 부트 명령줄을 설치하는 명령입니다. 부트 명령줄에는 반드시 새로이 추가된 물리 볼륨의 하드웨어 주소가 포함되어야 합니다.

```
/dev/vg00/lvol1 / hfs rw 0 1 #root
/dev/vg00/lvol2 swap ignore sw 0 0 #primary swap
/dev/fsvg/lvol2 /bdata02hfs rw 0 2 #
```

```
/dev/fsvg/lvol1 /bdata01hfs rw 0 2 #
/dev/vg01/lvol3 /users hfs rw 0 2 #
/dev/vg00/lvol4 /tmp hfs rw 0 2 #
/dev/vg00/lvol3 /usr hfs rw 0 2 #
```

코드 예제 14: 미러링이 필요한 시스템 논리 볼륨 네 개가 포함되어 있는 시스템 볼륨 그룹

부트 디스크를 생성하여 시스템 볼륨 그룹에 추가한 다음에는 시스템 볼륨 그룹의 각 논리 볼륨에 대해 미러를 생성해야 합니다. **bdf** 명령과 **swapinfo** 명령을 실행해 보면 또는 /etc/checklist 파일의 내용을 보면 어떤 논리 볼륨이 미러되어야 하는지 알 수 있습니다. 미러할 데이터를 판단할 때는 보조 스왑 영역도 고려해야 합니다.

```
lvextend -m 1 /dev/vg00/lvol1/ /dev/dsk/c8d0s2
lvextend -m 1 /dev/vg00/lvol2/ /dev/dsk/c8d0s2
lvextend -m 1 /dev/vg00/lvol3/ /dev/dsk/c8d0s2
lvextend -m 1 /dev/vg00/lvol14 /dev/dsk/c8d0s2
lvlnboot -r /dev/vg00/lvol1
lvlnboot -s /dev/vg00/lvol2
lvlnboot -d /dev/vg00/lvol2
```

코드 예제 15: 논리 볼륨 미러를 생성하고 시스템 볼륨 그룹 구조를 갱신

미러를 생성할 때는 lvextend 명령을 사용합니다. 원본의 모든 데이터를 미러로 복사해야 하기 때문에 미러 생성 완료에는 논리 볼륨의 크기에 따라 대략 20에서 30분 정도가 소요됩니다. 시스템 볼륨 그룹 미러링의 마지막 단계는 루트, 스왑, 덤프 볼륨의 위치가 미러에 반영되도록 갱신하는 작업입니다. **lvlnboot** 명령을 사용합니다. 이 작업은 반드시 수행해야 합니다. 시스템 볼륨 그룹이 변경되었을 때 **lvlnboot** 명령을 실행하지 않으면 시스템이 정상적으로 부트되지 않습니다.

볼륨 그룹 백업

마지막으로 볼륨 그룹 정보가 백업되도록 다음과 같은 명령을 실행해야 합니다. 디스크에 문제가 생겼을 때 이 백업된 정보를 이용하면 디스크의 논리 볼륨 구조를 복구할 수 있습니다. 이 백업에는 논리 볼륨의 데이터가 포함되지 않으며 복구는 개별적으로 수행되어야 합니다.

```
vgcfgbackup /dev/vg00
vgcfgbackup /dev/vg01
vgcfgbackup /dev/vg02
vgcfgbackup /dev/fsvg
vgcfgbackup /dev/dbvg01
vgcfgbackup /dev/dbvg02
vgcfgbackup /dev/dbvg03
vgcfgbackup /dev/dbvg04
vgcfgbackup /dev/dbvg05
vgcfgbackup /dev/dbvg06
```

코드 예제 16: vgcfgbackup 명령으로 볼륨 그룹 백업

vgcfgbackup 명령은 /etc/lvmconf 디렉토리에 하나의 이진 파일을 생성합니다. 이 파일은 드라이브의 볼륨 그룹 정보를 복원할 때 사용합니다. 볼륨 그룹 정보를 복원하는 명령은 **vgcfgrestore**입니다.

LVM 및 디스크 문제 해결

많은 수의 디스크가 구성된 시스템일수록 디스크 고장 발생 가능성은 높습니다. 드라이브의 MTBF(평균 고장 간격)가 매우 크더라도 수 백 개의 디스크가 구성된 시스템이라면 디스크 고장으로부터 시스템을 보호할 수 있는 수단이 필요합니다. 따라서 고가용성 디스크 보호 시스템 구축을 고려해야 할 것입니다. 고가용성 RAID 모드의 디스크 배열에는 자체적인 오류 극복 기능이 있습니다. 그러므로, 개별 드라이브의 고장은 운영 체제 수준에 영향을 미치지 않습니다. LVM 디스크 미러링 시스템의 고장은 명시적인 일련의 복구 절차를 필요로 하지만 응용 프로그램 사용자에게는 영향을 미치지 않습니다.

단독 미러 드라이브에 고장이 발생하여 드라이브를 교체했을 때는 **vgcfgrestore** 명령을 실행하여 교체한 드라이브에 볼륨 그룹 정보를 복원하고, **vgchange -a y** 명령으로 교체된 드라이브를 볼륨 그룹에 포함시킨 다음 끝으로 미러를 동기화해야 합니다. 논리 볼륨 미러의 자동 동기화 여부는 운영체제 버전, 패치 수준, 고장 발생 후 시스템 재시작 여부에 따라 결정됩니다. 논리 볼륨의 동기 상태는 **lvdisplay -v** 명령으로 확인할 수 있습니다. 무효한 익스텐트가 발견되었다면 동기화가 필요한 것입니다. 지금까지 설명한 절차는 단지 논리 볼륨만 복원한다는 점을 유의하십시오. 이어서 논리 볼륨 데이터를 복원해야 합니다.

다음의 도표는 LVM 문제를 분석하는데 유용한 여러 명령들을 정리한 것입니다.

명령	표시되는 정보

disk	

도표 1: LVM 문제 분석을 위한 명령

LVM은 시스템 부트 프로세스와 단단히 결합되어 있습니다. 시스템이 시동될 때 시스템 드라이브에 문제가 있으면 LVM이 시스템의 부트를 막습니다. 시스템 시동에 영향을 미치는 LVM 문제를 해결하기 위해 두 개의 특별한 부트 명령 옵션이 있습니다. 첫번째 -lm(즉, ISL> hpux -**lm** (;0)/hp-ux) 옵션은 시스템이 LVM 보수 모드로 부트되도록 합니다. 보수 모드는 정상적인 LVM 구조를 우회하도록 하는 특별한 시스템 부트 방법입니다. 이 모드는 단독 사용자 모드와 유사한 것으로 정상적인 부트에서 실행되는 프로세스 및 시스템 검사의 상당수가 생략됩니다. 보수 모드는 시스템이 LVM 데이터 구조 손상을 복구하는데 충분한 시간 동안 부트되도록 하기 위한 것입니다.

시스템 볼륨 그룹의 디스크 중 일부에 문제가 발생하여 LVM이 시스템 부트를 허용하지 않는 경우에는 볼륨 그룹 정족수가 무시될 수 있도록 부트 명령에 -lq 옵션을 사용해야 합니다. 디스크 정족수는 부트 시간에 정의되며 볼륨 그룹 드라이브 갯수 절반에 1을 더한 것입니다. 시스템 논리 볼륨을 미러하는 두 개의 드라이브로 구성된 시스템 볼륨 그룹에서 하나의 드라이브에 고장이 발생하여 시스템이 부트되지 않는다면 정족수 무시 옵션을 사용하여 미러 드라이브 없이 시스템이 부트되도록 하고 미러 드라이브를 수리할 수 있습니다.

대규모 데이터베이스 시스템의 복잡한 디스크 배치를 관리하고 디스크 문제가 발생했을 때 정확한 복구가 가능하려면 시스템의 논리 볼륨 배치에 대한 설명을 문서로 작성해 두는 것이 매우 중요합니다. 다음 절에서는 논리 볼륨 구성을 문서화하는데 도움이 될만한 지침을 제공하고 논리 볼륨 일람표 형식을 소개합니다. 보다 자세한 내용은 Hewlett Packard 설명서 Solving HP-UX Problems를 참고하십시오. LVM 관련 내용을 다시 한번 살펴보십시오. System Administration Tasks 설명서와 How HP-UX Works: Concepts for the System Administrator manuals 설명서도 참고하시기 바랍니다.

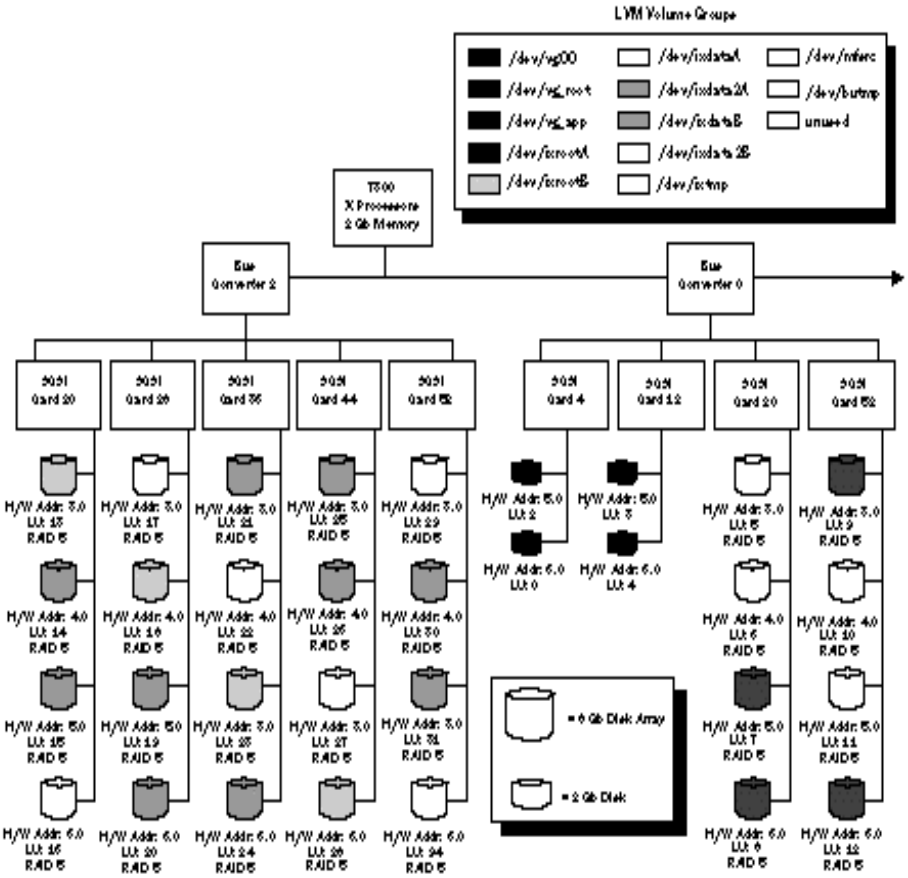
사례연구

이 절에서는 두 종류의 시스템을 통해 지금까지 다루었던 여러 개념의 실제 활용을 설명할 것입니다. 이 두 시스템은 모두 대형 통신 업체 영업 부서의 업무를 위한 것으로 고객 신상 정보 및 서비스 자료를 제공합니다. 데이터베이스의 크기는 100GB를 넘을 것이며 시스템 관리자는 데이터베이스 생성 작업의 복잡함을 극복해야만 합니다.

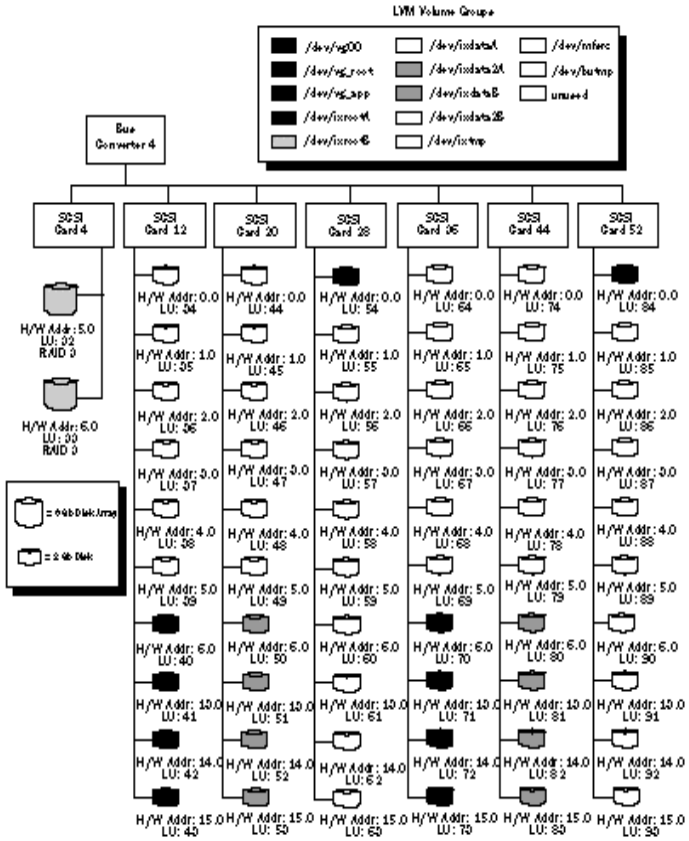
사례 연구 1

첫번째는 고객 신상정보와 요약된 서비스 정보를 관리하는 시스템입니다. 고객 데이터베이스는 메인 프레인의 요금 정보를 이용하여 구축됩니다. 이 시스템은 정기적으로 갱신되는 서비스 정보를 입력받고 고객 레코드 수정 기능을 제공해야 합니다. 이 시스템과 두 번째 시스템은 400GB 이상의 정보를 저장할 수 있도록 설계되었습니다.

여기에 선택된 시스템은 2GB 메모리의 HP 9000 T500 8이며 디스크 공간은 428GB입니다. 또한 이 시스템에는 2GB 드라이브 다섯 개로 구성된 Fast/Wide SCSI Cascade 디스크 배열이 30개, 2GB 단독 드라이브가 64개 장착되어 있습니다. 30개의 디스크 배열은 고가용성 RAID 모드로 조정되었으며 모든 단독 드라이브는 미러링을 사용합니다. 시스템의 가용 디스크 공간은 304GB입니다.



예 1: 디스크 및 볼륨 그룹 구성 (1/2)



예 1: 디스크 및 볼륨 그룹 구성 (2/2)

운영 체제와 데이터베이스 서버로는 HP-UX 버전 9.04와 Informix OnLine이 채택되었습니다. Informix OnLine의 크기 한계로 인해 각각의 데이터베이스 인스턴스는 120GB에서 130GB 사이의 자료를 보관합니다. 따라서 각 지역별 고객에 대한 모든 자료를 보관하기 위해서는 네 개의 독립적인 데이터베이스 인스턴스가 필요합니다. 하나의 Informix 인스턴스는 여러 개의 데이터베이스를 관리하고 각각의 지역에 해당하는 자료는 하나의 데이터베이스로 보관됩니다.

디스크 드라이브를 제외한 나머지 모든 하드웨어는 최적의 성능과 연결성을 제공할 수 있도록 조정되었습니다. 디스크는 세 개의 버스 컨버터로 분산되며 하나의 버스 컨버터가 초당 21MB의 입출력 요청을 처리할 수 있습니다. SCSI F/W 카드는 총 16개가 있으며 하나의 카드가 초당 7-10MB를 처리할 수 있습니다. 자세한 사항은 예 1의 일람표를 참조하십시오.

디스크 배열의 대부분은 Informix 데이터베이스를 위해 사용됩니다. 디스크 배열은 동시성이 높은 임의 읽

기 액세스에 맞춰 RAID 5모드로 조정되었습니다. 30 개의 디스크 배열 중 2개는 데이터베이스 백업 디렉토리에 할당되었고 전형적인 대규모 순차 전송인 백업 작업에 맞춰 RAID 3모드로 조정되었습니다. 모든 드라이브는 쓰기작업 즉시 보고를 사용하지 않으며 읽기 캐시만 사용합니다.

각각의 데이터베이스 인스턴스는 세 개의 볼륨 그룹으로 구분되었습니다. 첫번째 볼륨 그룹은 6개의 단독 2GB 드라이브로 구성되었으며 데이터베이스의 루트, 카탈로그, 물리 로그 및 논리 로그를 위해 사용됩니다. 이들은 모두 서로 다른 디스크에 저장되고 이 들이 담겨 있는 논리 볼륨도 각각 다른 드라이브로 미러됩니다. Informix는 루트 영역 및 로그에 디스크 배열 보다는 미러되는 단독 드라이브 사용을 권장합니다. 또 다른 볼륨 그룹은 데이터베이스 테이블 및 색인을 위해 사용됩니다. 이 볼륨 그룹은 7개의 디스크 배열로 구성되었으며 볼륨 그룹의 논리 볼륨(총 28개)은 모두 스트라이핑되었습니다. Informix 청크의 최대 크기는 2GB이므로 논리 볼륨의 크기도 2GB입니다. 모든 볼륨 그룹의 디스크는 서로 다른 I/O 카드로 균등하게 분산되었습니다. 드라이브를 배치하기에 앞서 SCSI 버스 주소의 우선순위가 고려되었습니다. 두 개의 데이터베이스 인스턴스를 구성하는 네 개의 대형 볼륨 그룹이 병렬로 스트라이핑되었고 완료되는데 대략 14시간이 걸렸습니다.

데이터베이스 볼륨 그룹 6개에 덧붙여 데이터베이스와 밀접한 관련이 있는 추가적인 볼륨 그룹이 3개 있습니다. 첫번째 /dev/mfsrc는 메인 프레임에서 다운로드 받은 파일을 저장하기 위한 것입니다. 이 볼륨 그룹은 많은 수의 단독 드라이브로 구성되었고 미러링의 I/O 채널 분리를 위해 물리 볼륨 그룹을 사용했습니다. 두 번째 /dev/butmp 볼륨 그룹은 두 개의 RAID 3 디스크 배열로 구성되었습니다. 이 볼륨 그룹은 언로드된 데이터베이스 테이블 정보를 광학 쥬크 박스로 복사하기 전까지 보관합니다. 이 볼륨 그룹에는 네 개의 4GB 논리 볼륨이 있으며 HP-UX 파일 시스템을 담게 됩니다. 세 번째 볼륨 그룹은 /dev/ixtmp입니다. 이 볼륨 그룹은 HP-UX 파일 시스템을 담게 되며 임시 작업 공간으로 사용됩니다. 이 볼륨 그룹에 대해서는 물리 볼륨 그룹이 사용되지 않으며 서로 다른 데이터베이스 인스턴스 사이의 디스크 분리가 최적화되도록 수작업으로 논리 볼륨이 할당되었습니다. 물리 볼륨 그룹을 이용하는 것만으로는 두 개의 논리 볼륨이 같은 디스크에 존재하지 않는다는 보장이 없습니다. 이 볼륨 그룹의 입출력은 상당히 임의적일 것으로 예상되기 때문에 MWC를 사용하지 않았습니다. 다른 나머지 미러되는 논리 볼륨들은 MWC를 사용합니다.

시스템은 지금까지 설명한 볼륨 그룹과 함께 HP-UX 응용 프로그램 이진 파일과 사용자 작업 영역으로 또

다른 세 개의 볼륨을 사용합니다. 시스템은 두 개의 부트 가능한 시스템 볼륨 그룹 /dev/vg00, /dev/vg_root로 구성되었으며 각각의 볼륨 그룹에는 하나의 미러되는 부트 디스크가 포함되어 있습니다. 두 개의 볼륨 그룹을 사용하는 이유는 관리자의 실수(예를 들어, 잘못된 스크립트를 실행하여 운영 체제가 삭제되는 경우)로부터 사용자를 보호하고 미러링을 통하여 드라이브 고장에 대비하기 위한 것입니다. 세 번째 볼륨 그룹 /dev/vg_app은 응용 프로그램 관련 디렉토리를 담게 됩니다.

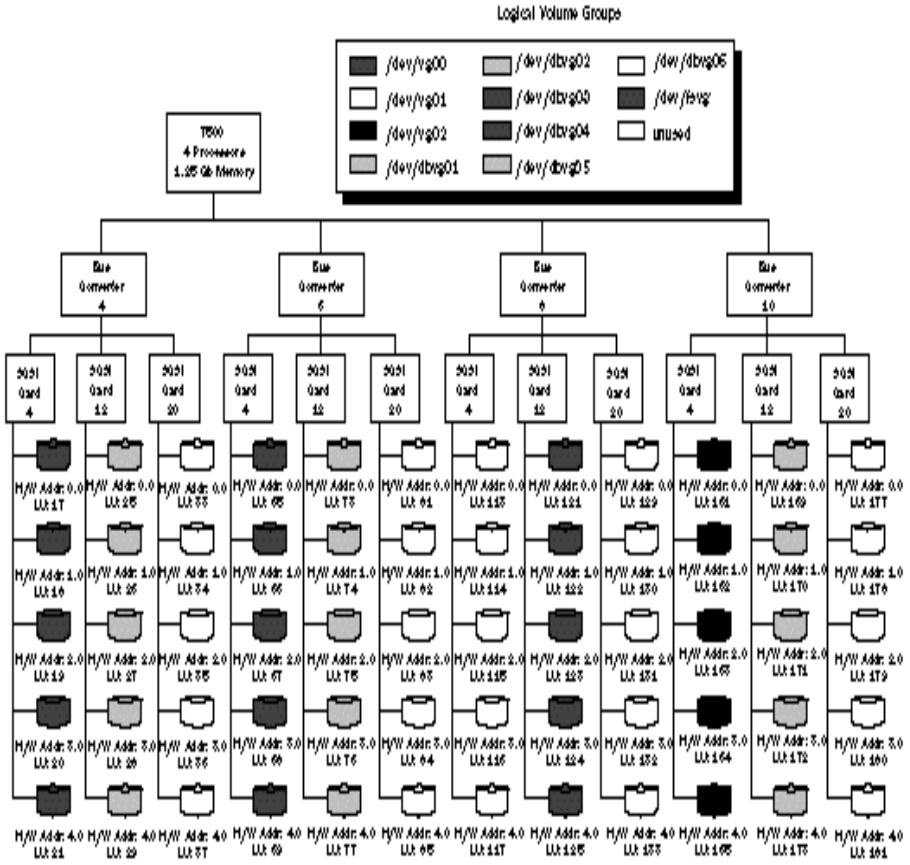
한 가지 문제시 될 만한 부분은 마지막 버스 컨버터(주소 4)에 물려 있는 입출력 장치의 갯수입니다. 이 버스 컨버터에 걸리는 입출력 부하를 검토해 보아야 합니다. 염두에 둘 것은 보통의 상황에서 이 버스 컨버터에 연결된 모든 단독 드라이브가 동시에 사용되지는 않는다는 것입니다. 대부분의 드라이브는 데이터베이스에 데이터를 로드할 때나 백업 시에만 사용됩니다. I/O 채널에 걸리는 부하를 판단할 때는 이런 요소를 고려하는 것이 중요합니다.

사례 연구 2

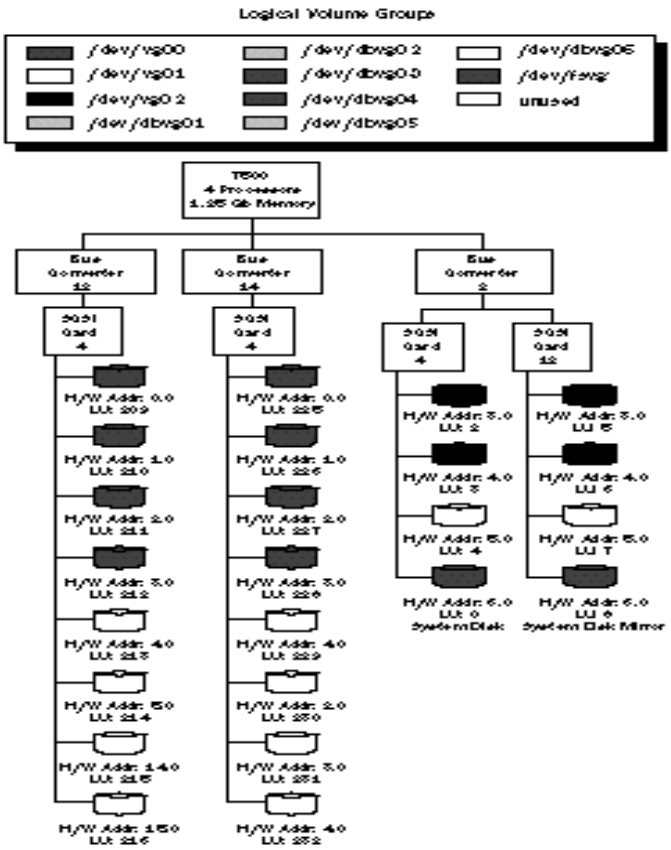
두 번째 사례 연구는 세부적인 통신 서비스 정보 분석 작업을 수행하기 위해 설계된 시스템입니다. 자료는 메인 프레임으로부터 일정한 주기로 다운로드되어 Informix 데이터베이스에 로드 됩니다. 3일분의 서비스 자료를 보관할 수 있는 충분한 디스크 공간이 필요합니다. 대규모의 의사결정 지원 유형의 질의가 데이터베이스에 대해 실행됩니다. 질의의 대부분은 지역별 서비스 변동을 표시하는 표준 집합 질의와 유사한 것이지만 일부 질의는 임의적일 수 있습니다. 표준 질의는 색인이 지원되고 Informix 병렬 데이터 질의(PDQ) 및 테이블 분할 기능을 활용합니다. 전체 데이터베이스(로그, 테이블, 색인, 카탈로그, 임시 영역)의 크기는 대략 100GB입니다.

이 시스템은 처리 성능을 높이는 데 설계의 초점이 맞추어 졌고 따라서 디스크 배열 대신 미러되는 디스크가 선택되었습니다. 또한, 가용성을 높이기 위해 SPU 스위치오버 장치와 LVM 디스크 미러링을 동시에 사용합니다. 이 두 가지 기법을 동시에 사용하는 시스템 구성은 상당히 까다로우며 LVM 스크립트 작성에는 각별한 주의가 필요합니다. 두 시스템 모두 세부적인 시스템 일람표 및 문서화가 필요하지만 이외에 논리 단위(LU) 번호 부여 방식의 통일도 필요합니다. HP-UX 설치 이후에 논리 단위 번호 부여를 다시 설정해야

할 때는 rmsf 및 insf 명령을 사용합니다.



예 2: 디스크 및 볼륨 그룹 구성 (1/2)



예 2: 디스크 및 볼륨 그룹 구성 (2/2)

선택된 시스템은 HP 9000 T500 4 way이며 메모리는 1.25GB입니다. 시스템에는 2GB Fast/Wide SCSI 단독 디스크가 120개 장착되었으며 디스크 공간 총량은 240GB입니다. 모든 디스크는 미러되므로 실제 사용 가능 디스크 공간은 120GB입니다. 8개를 제외한 나머지 모든 드라이브가 SPU 스위치오버 구성에 포함되었습니다. 스위치오버 쌍으로 사용되는 두 시스템은 I/O 구성이 동일하며 따라서 하나의 시스템에 실제 부착되어 있는 드라이브 갯수는 232개입니다. 모든 드라이브는 쓰기작업 즉시 보고를 사용하지 않습니다. 장래 확장 가능성 및 스위치오버 구성에 따른 많은 수 디스크를 지원하기 위해 디스크를 7개의 버스 컨버터로 분산하였습니다. 자세한 사항은 예 2의 일람표를 참조하십시오. 일람표에는 스위치오버 시스템에 사용되는 I/O 카드 및 채널이 표시되지 않았습니다.

데이터베이스 엔진으로는 병렬 데이터 질의 및 광범위한 데이터 분할 기능을 제공하는 Informix OnLine Dynamic Server, 버전 7.1이 선택되었습니다. OnLine Dynamic Server, 버전 7.1에서는 DBA가 데이터베이스 수준에서 테이블을 분할할 수 있기 때문에 논리 볼륨 스트라이핑을 사용하지 않기로 결정되었습니다. 따

라서 대규모 테이블의 입출력을 분산시키는 일은 DBA가 책임져야 합니다.

운영체제 수준에서 구현된 데이터 미러링과 데이터베이스 수준에서 구현된 데이터 미러링에는 서로 다른 성능 이득이 있습니다. LVM은 각각의 디스크에 대해 최단 디스크 큐를 사용하기 때문에 최소화된 입출력 처리로 미러 디스크를 선택하는, 읽기 성능 최적화의 이점이 있습니다. 또한, LVM 미러링은 데이터베이스 관리 업무의 복잡도를 감소시킵니다. 복수 사용자의 테이블 동시 액세스 처리에 뛰어난 입출력 성능을 나타냅니다. Informix OnLine Dynamic Server는 분할 읽기라고 불리는 기법을 사용하여 디스크 탐색 시간을 감소시킵니다. 데이터 페이지 읽기가 요청되면 청크의 데이터 페이지 위치를 고려하며 기본 청크 또는 미러 청크의 페이지가 액세스됩니다. 쓰기 작업은 Informix 및 LVM 구현 모두 양쪽 미러 드라이브에 병렬로 수행합니다.

이 시스템의 데이터베이스 인스턴스는 여섯 개의 볼륨 그룹으로 나누어 졌습니다. 각각의 볼륨 그룹은 동일한 F/W SCSI 카드에 연결된 디스크의 모음으로 구성됩니다. 각각의 논리 볼륨은 다른 F/W SCSI 카드에 연결된 디스크 모음으로 미러됩니다. 그러므로, 서로 다른 I/O 카드로 테이블을 분할할 때는 각기 다른 볼륨 그룹에 속하는 논리 볼륨을 선택하면 됩니다. 모든 논리 볼륨은 2GB 크기이며 단일 드라이브로 할당되었습니다. 모든 논리 볼륨이 서로 다른 드라이브를 사용하기 때문에 병렬 질의 수행에 있어서 최대의 I/O 병렬화를 획득할 수 있습니다(분할화 방식에 부합하는 공통 질의인 경우).

일곱 번째 볼륨 그룹 /dev/fsvg는 메인 프레임에서 다운로드 된 파일이 담겨지도록 구성되었습니다. I/O 채널 분리는 물리 볼륨 그룹을 통해 유지됩니다. 시스템 볼륨 그룹도 미러되도록 구성되었습니다.

데이터베이스 논리 볼륨에는 LVM 미러 쓰기 캐시(MWC)가 사용되지 않습니다. 실행 속도를 높이기 위해 비MWC로 구성되었지만 디스크 고장이 발생한 경우 6개의 볼륨 그룹에서 모든 논리 볼륨 미러가 재동기화되므로 시스템이 상당한 타격을 받을 것입니다. 하지만 재동기화 작업 동안 볼륨 그룹에 액세스할 수는 있습니다.

결론

소프트웨어 및 하드웨어 기술의 발달로 데이터베이스 크기는 날이 갈수록 증가하고 있습니다. 데이터베이스를 이용하는 업무의 증가 또한 데이터베이스의 거대화를 부추기고 있습니다. 대규모의 데이터베이스를 구축할 때 중요한 것은 안정성입니다. 데이터베이스는 신뢰성과 확장성이 보장되는 기반 위에서 전체적인 성능 목표를 달성할 수 있도록 구축되어야 합니다. 이 글에서 다룬 내용들이 대규모 Informix 데이터베이스 구축을 준비하는 분들에게 도움이 되기를 기대합니다.

Massively Parallel Processor (MPP) 시스템과 기업에서의 그들의 역할

개요

이 글은 현재 일어나고 있는 컴퓨터 구조의 주된 변화와 산업에서의 그들 각각의 역할을 간략하게 설명하고 있습니다. 그리고, 상업적 영역으로 도입된 가장 최근의 컴퓨터 구조인 Massively Parallel Processor (MPP) 시스템과 이 시스템이 솔루션으로 추천되는 이유에 중점을 둘 것입니다. 간략히 말하면, 이러한 시스템(500 GB를 초과하는)들은 대량의 데이터 사양이나 복잡한 질의, 사용자로 인하여 발생하는 과중한 부하 혹은 매우 빠른 응답 시간의 요구로 인하여 다른 구조의 컴퓨터가 적응에 실패한 곳에 사용될 것입니다. 이들은 미래에 대한 적응이 중요하고 서버의 합병이 일어날 수 있는 곳에도 역시 사용될 것입니다.

이 글은 IBM과 다른 여러 솔루션 업체들이 제공한 정보에 기초하여 작성한 것으로서 다음의 내용들을 살펴보고자 합니다.

- 단일 프로세서와 균형 잡힌 멀티프로세서 시스템
- 기업에서의 거대한 데이터베이스와 복잡한 질의
- 클러스터와 Massively Parallel Processing
- 성공적인 구축의 열쇠
- 미래

단일 프로세서와 균형 잡힌 멀티프로세서 시스템

가장 오래되고 가장 평범한 컴퓨터의 구조는 단일프로세서 방식입니다. 데스크탑과 노트북 컴퓨터로부터 메인 프레임 서버까지의 많은 범위의 컴퓨터가 이에 해당합니다. 이러한 시스템(가격이 높을수록 성능이 뛰어난)들은 프로세서의 속도(성능)와 사용 가능한 주기억장치의 양, 그리고 I/O 채널(버스)의 수로써 구별됩니다. 아래의 그림 1을 참고하십시오. 현재의 개인용 컴퓨터는 CPU와 주기억장치 그리고 표준 I/O 버스 구조를 갖고 있는 일용품이며, 상대적으로 저렴한 가격에 비하여 월등한 성능을 갖고 있습니다. 하지만, 보다 많은 사항을 요구하는 다수 사용자 환경에서는 시스템들은 보다 비싸고, 일용품적인 측면을 보다 적게 갖고 있습니다. 단일프로세서 시스템이 최대한 단순한 프로그래밍 모델을 갖고 있고 가장 인기가 있기 때문에, 대부분의 응용 프로그램은 이러한 구조에 맞추어 설계됩니다.

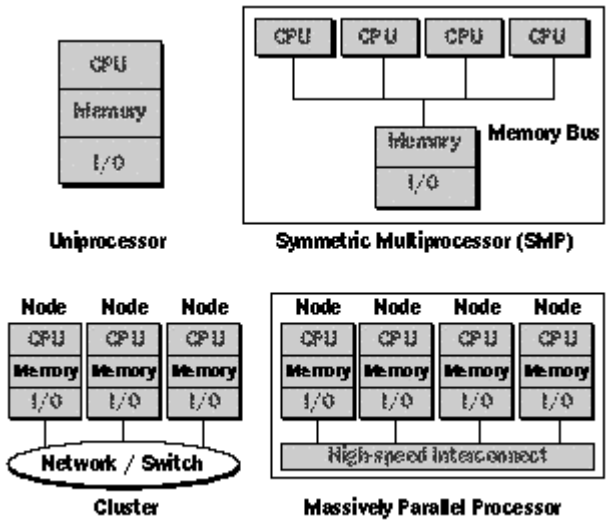


그림 1: 컴퓨터 구조를 단순화한 그림

단일프로세서 시스템의 성능은 해당 시기에 가장 빠른 프로세서의 성능으로 제한됩니다. 이러한 한계점을 극복하기 위하여 균형 잡힌 다중처리(SMP)가 소개되었습니다. 메모리 버스의 캐시와 결합된 여러 개의 프로세서를 제외하면, SMP 구조는 단일 프로세서 시스템의 구조와 유사합니다. 한 개의 운영 체제가 병렬 처리될 수 있는 CPU 중심적 작업들과 시스템을 관리합니다. 소프트웨어의 SMP 버전(Informix OnLine Dynamic Server, 버전 7에서의 예처럼)들은 하드웨어 구조로 인한 이익을 봅니다. 이러한 시스템들은 단일프로세서

에서보다 높은 성능을 발휘합니다. 주기억장치의 한계와 운영 체제가 갖는 부하 그리고 응용 프로그램이나 응용 프로그램 실행자의 제한된 시스템 이용 능력이 성능을 제한할 것입니다. 이러한 한계는 하드웨어 업체들이 프로세서(PowerPC 601을 604로 업그레이드하는데 사용된 IBM's RISC System/6000 SMP)와 메모리 기술(RS/6000 SMP 버스를 제공하는 크로스바 스위치) 및 (효율적 멀티스레드 커널 등의) 운영 체제 등이 발전함으로써 지속적으로 대두되고 있습니다. Informix와 같은 소프트웨어 업체들도 SMP에 대하여 가장 적절한 제품 구현을 계속하고 있습니다.

현재 SMP의 시스템 규모는 어떻게 되는가? 이것은 이미 언급된 모든 요소들 뿐 아니라 시스템 부하의 종류에 따라 달라집니다. 가장 큰 UNIX 기반의 관계형 데이터베이스 SMP 시스템이 이제 막 500 GB를 넘어섰습니다. 덧붙여서, 성능은 질의의 복잡도에 따라 달라질 수 있습니다. 보다 나은 성능을 얻기 위한 소프트웨어 방법론은 denormalized table의 사용과 같은 작업을 포함하고 있습니다. 그러나, 이러한 것들은 제한적인 효율성을 갖습니다.

기업에서의 거대한 데이터베이스와 복잡한 질의

최근 15년간 새로운 컴퓨터 기술의 발전은 새로운 데이터의 산업적 사용을 가능하게 했습니다. 1970년대의 컴퓨터가 트랜잭션 연산에 치중된 트랜잭션 기반(OLTP)의 시스템인 반면, 1980년대의 기술적 워크스테이션들은 업무 영역에서의 보다 복잡한 데이터 분석을 가능하게 하였습니다. 1990대에 와서 회사들은 의사 결정 시스템(DSS)과 데이터 저장소를 구현하고 있습니다.

된 테이블을 사용하는 것은 트랜잭션 행위의 원인을 놓칠 수도 있습니다. 최종적인 저장소의 크기는 많은 회사들이 이미 갖고 있는 온라인 저장소의 몇 배가 될 수 있습니다.

앞에서 다루었듯이, 이러한 복잡한 질의와 거대한 데이터베이스는 단일프로세서와 SMP 시스템의 경우 받아들일 수 없을 정도의 나쁜 성능을 갖게 할 수도 있습니다. 고객 데이터들을 보다 잘 파악하는 데에 발생하는 이러한 장애물들을 극복하기 위해서 많은 회사들이 다른 컴퓨터 구조를 시험해 보고 있습니다.

클러스터와 Massively Parallel Processing

프로세서와 I/O의 성능을 향상시키고 주기억장치의 성능을 높일 수 있는 길은 단일프로세서나 SMP 시스템들을 하나로 결합시키는 것입니다. 하나로 결합시키는 것은 또한 단일 운영 체제에서의 병목현상을 해소해 줄 수 있으며, 한 곳의 고장은 보다 쉽게 없앨 수 있다는 보다 큰 가능성을 제공해 줍니다.

이러한 해결책에 문제가 없는 것은 아닙니다. 넓게 배포된 운영체제를 관리한다는 것은 개방된 시스템에서 잘 알려진 문제입니다. 성능의 관점에서 보면, 네트워크 연결이 병목현상이 될 수 있습니다. FDDI, ATM 및 Fibre Channel Standard가 주파수 대역폭을 크게 향상시킨 반면, 의사 결정 지원에서 사용자들은 거대하고 예측할 수 없는 테이블 조인을 새로운 데이터 관계를 밝히기 위하여 종종 수행할 수도 있습니다. 데이터베이스 관리자는 기업 분석가나 회사의 최고 경영자가 데이터를 어떻게 검색할 것인가를 예측하여 네트워크를 통한 조인을 최소화함으로써 데이터베이스를 설계하는 것이 임무가 될 것입니다.

Massively Parallel Processor(MPP) 시스템들은 유연성과, 관리성 그리고 발전성을 고려하여 설계됩니다. IBM, NCR(이전의 ATGIS), Siemens/Pyramid 및 Unisys 등의 많은 컴퓨터 하드웨어 업체들이, 이러한 시스템에 대한 커지는 요구를 인식하고 있습니다.

이러한 구조의 한 예가 IBM의 RISC System/6000 Scalable POWERparallel (SP) System 입니다. 이것은(노드라고 하는) 각각이 CPU, 메모리, I/O 시스템을 갖고 있는 RS/6000들로 이루어져 있습니다. 이러한 시스템들은 상업적으로 128노드까지 확장할 수 있습니다(기술적으로는 이미 512노드까지 가능). SP 노드들은 Omega 스위치(혹은 High Performance Switch, or HPS)라고 불리는 고속의 무저항 스위치를 통하여 연결됩니다. 네트워크 연결과는 달리, 이 스위치는 새로운 노드의 추가에 따라 확장할 수 있는 다중 지점간 연결을

제공합니다. 각각의 연결은 초당 40 BM의 양방향 전송 능력을 갖습니다. 이러한 설계는 노드 간의 다중 경로를 통하여 항상 일정한 길이의 경로를 제공합니다.

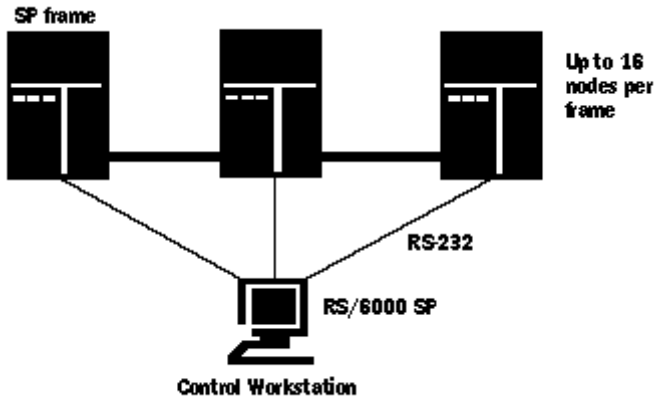


그림 2: 최대 512 노드의 SP에 대한 당일 지점 관리를 제공하는 제어 워크스테이션

관리의 관점에서 각 노드에 하나의 AIX가 작동되고 있다고 할지라도(여러 개의 RISC System/6000에 같은 AIX가 실행될 수 있음), RS/6000 제어 워크스테이션은 복합된 시스템을 단일 시스템으로 관리합니다. AIX에 포함되어 있는 Sp에 대한 병렬 시스템 지원 프로그램(PSSP)은 이 필수적 부가 함수를 제공합니다.

이 컴퓨터 구조의 이점을 얼마나 잘 살리는가 하는 것은 MPP를 사용한 새로운 응용 프로그램과 지원 기능을 개발하는 소프트웨어 업체들의 능력입니다. Informix는 1995년에 SP를 지원하는 Informix OnLine Extended Parallel Server(OnLine XPS)를 발표하였으며, 이것을 여러 고객의 사업장에 설치하였습니다. OnLine XPS는 아무것도 공유하지 않는 메시지 전달 구조를 사용하며, 거대한 데이터베이스와 복잡한 질의를 사용하는 고객의 요구를 만족시켰습니다. 예를 들어, MCI Communications Corporation에서 Informix는 처음에는 48 노드의 RS/6000 SP를 구현하였고, 이 데이터베이스를 90 노드까지 확장하여 왔습니다. OnLine XPS는 Informix가 이전 제품(Version 6과 7)에 추가한 병렬 처리 기술을 특징으로 하고 있으며, 질의에 대한 병렬 처리(PDQ)를 포함하고 있습니다. OnLine XPS만의 향상된 연산 기능은 데이터의 삽입,수정, 삭제, 로드, 언로드의 병렬 처리를 포함하고 있습니다.

RS/6000 SP가 병렬 처리 프로그램에서 뿐만 아니라 직렬 처리 프로그램에서도 작동할 수 있기 때문에 회사들은 하나의 노드에 직렬 분석을 설치하고, 그 기계에 많은 양의 벌크 데이터를 전달할 수 있는 고성능 스위치의 이점을 사용할 수도 있습니다. 현재 많은 OLTP 응용 프로그램 역시 SP와 유사한 구조를 사용하고 있습니다. OLTP 응용 프로그램 업체들도 병렬 처리 프로그램들을 개발하려는 계획을 갖고 있으며, 하부 구조에서의 보다 나은 병렬 처리의 활용을 생각하고 있습니다. 사용자가 구축한 응용 프로그램에 대해서는 CICS/6000, Encina, Tuxedo 등의 OLT 환경에서 사용되는 다양한 트랜잭션 모니터가 있습니다.

수 없이 구축된 MPP 시스템은 이러한 구조가 기업 전산화의 주류가 되었음을 보여 줄 것입니다. IBM의 첫 번째 상업적 구현인 SP2가 1994년 중반에 발표되었고, 1995년 말에는 500개가 넘는(대부분 데이터베이스 솔루션인) SP 시스템이 고객에게 구축되었습니다.

성공적인 구축의 열쇠

시키고, 이 기술의 도입을 단순화시키며, 실제적인 점검 시점을 제공합니다.

미래

모든 컴퓨터 구조는 진화를 계속합니다. 메모리와 I/O의 전송폭이 넓어지는 것 같이 CPU의 성능은 향상되고 있습니다. (현재 12-18 개월의 간격을 두고 2배로 향상되고 있음). 그래서, 단일프로세서와 SMP는 그들이 다룰 수 있는 데이터베이스의 크기와 질의의 복잡도를 향상시키고 있습니다. 네트워크 전송 속도도 증가하고 있으며 클러스터의 성능을 향상시키고 있을 것입니다. 그러나, MPP는 지금처럼 최고의 성능을 유지할 것입니다. 단일프로세서와 SMP에서의 같은 종류의 발전이 MPP 시스템에 사용될 것이며, 기술의 상호작용에 의한 더 나은 진보가 이루어질 것입니다. MPP 시스템은 하루 하루의 사업 방향 결정의 바탕이 되는 결정 제공 능력과 기존의 메인 프레임에서 이식된 OLTP 프로그램에서, 최고로 강력한 성능을 발휘할 것입니다.

참고사항

RS/6000, SP, RISC System/6000, Scalable POWERparallel은 IBM Corporation의 상표입니다. 다른 제품명들도 각 해당 회사의 제품명입니다.

객체 관계형 DBMS 차세대 경향

개요

이 글은 DBMS 기술을 필요로 하는 응용 프로그램에 대한 분류를 다루고 있으며, 관계형 DBMS와 객체지향적 DBMS 객체 관계형 DBMS가 각각 어디에 적합한지를 나타내고 있습니다. 이 글의 목적은 각 종류의 DBMS가 해결할 수 있는 문제들의 유형을 다루고 있습니다. 아래 내용에서 보여 주듯이, 모든 응용 프로그램의 요구를 만족시켜 줄 수 있는 DBMS는 없습니다. 마지막으로, 이 글은 객체 관계형 DBMS들에 의해서 다루어지는 문제들이 1990년대의 남은 기간 동안 점점 더 중요해질 것이라고 생각되는 이유를 지적할 것입니다. 그러한 이유에서, 객체 관계형 DBMS(Ojbect-Relationl DBMS; ORDBMS)는 "차세대 경향"을 나타냅니다.

분류 계획은,표 1에 나타나 있듯이 2x2 행렬을 이용하는 것입니다.

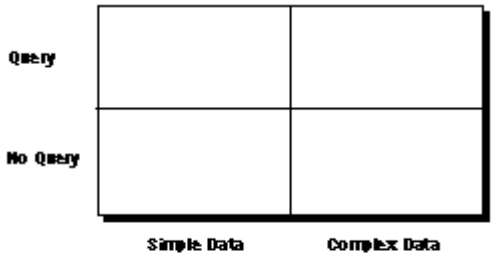


표 1: DBMS 응용 프로그램의 분류

표 1에서 수평축은 왼쪽 부분에 매우 간단한 데이터를 갖고 있고 오른쪽 부분에는 매우 복잡한 데이터를 갖고 있습니다. 일반적으로, 응용 프로그램이 다루게 될 데이터들은 이 두 극단적인 값들 사이에 연속적으로 분포하게 될 것입니다. 여기서는 단순히 별개의 두 가능성, 즉 단순한 데이터와 복잡한 데이터에 대해서만 다루고 있습니다. 유사한 방법으로, 수직축은 응용 프로그램에서 질의가 필요한지의 여부를 분별하는 것을 도와 줍니다. 이러한 결정은 "결코"와 "항상" 사이에서 변할 수 있으며, 이 두 극단적인 값들 사이에 연속된 분포가 존재합니다. 그러나, 여기서는 단순히 "질의"와 "무질의"의 두 가지 가능성만을 가정합니다.

사용자는 응용 프로그램을 시험해 본 후, 응용 프로그램의 특징에 따라 그것을 표 1에 있는 4개의 사각형 중 하나에다가 갖다 놓을 수 있습니다. 이 분류 작업을 묘사하기 위해서는, 4 개의 사각형 각각에 있는 응용 프로그램을 사용하여 보고, 계속해서 각 응용 프로그램이 필요로 하는 DBMS를 결정해 보는 것이 도움이 될 수 있습니다. 덧붙여서, 여기서는 각 4개의 응용 프로그램에 대하여 데이터 관리자를 사용해 볼 것을 추천하고 있습니다.

왼쪽아래부분

Microsoft Word for Windows, FrameMaker, Word Perfect, vi, 또는 emacs 등과 같은 표준 문서 편집 패

키지가 있다고 가정해 봅시다. 모든 문서 편집기들은 사용자가 파일명에 따라 열게끔 허용할 것이고, 이에 따라 파일의 내용은 가상 메모리로 올라올 것입니다. 그 다음 사용자는 이 가상 메모리 객체를 갱신할 것이고, 중간 중간에 이 객체는 디스크에 저장될 것입니다. 마지막으로, 사용자는 파일을 닫게 될 것이고, 이것은 가상 메모리에 있던 복사본이 파일 시스템으로 다시 저장되게끔 합니다.

문서 편집기가 수행할 유일한 질의 작업과 갱신 작업은 "파일 불러오기 와 "파일 저장 "입니다. 이처럼, 문서 편집기는 SQL의 사용을 필요로 하지 않는 "무질의" 응용 프로그램으로 분류할 수 있습니다. 덧붙여서, 문서 편집기는 파일 시스템이 제공하는 데이터 모델, 곧 문자열의 길이와 순서의 변화 모델 만으로 충분히 사용할 수 있는 응용 프로그램입니다. 이와 같이, 이것은 2x2 행렬의 왼쪽 아래에 적합한 "무질의 단순 데이터"의 경우입니다.

다른 내용을 다루기 전에 두 가지의 요점을 언급하겠습니다. 첫째로, 여기에서는 문서 편집기에 대해서만 다루고 Lotus Notes와 같은 복잡한 그룹웨어 상품들은 다루지 않습니다. 다음으로, 여기에서는 Documentum이 제공하는 것과 같은 문서 관리 시스템에 대해서 다루지 않습니다. Documentum에 대한 보다 상세한 설명은, 이 글에 있는 "Raising the Standard for Enterprise Document Management in Informix Environments"를 참고하십시오.

이 행렬의 왼쪽 아래 부분의 응용 프로그램에 대한 명확한 DBMS는 주어진 하드웨어 플랫폼에 따라 운영체제 공급자가 제공하는 파일 시스템입니다. 사실, 대부분의 문서 편집기들은 이러한 DBMS 서비스의 초기 단계를 사용하고, 보다 복잡한 어떤 것에 대한 어떠한 대비책도 마련되어 있지 않습니다. 그 이유는 간단합니다. 복잡한 데이터에 대하여 질의할 필요가 없다면, 파일 시스템에 의한 서비스가 적절할 것이기 때문입니다. 더구나, 파일 시스템은 보다 복잡한 시스템에 비하여 보다 높은 성능을 일정하게 제공하기 때문입니다.

아래 부분은 간단합니다. 표 1의 왼쪽 아래 부분에 있는 응용 프로그램의 경우 컴퓨터가 제공하는 파일 시스템의 상에 배치시키면 됩니다. 이와 같이, 왼쪽 아래 부분은 "파일 시스템"이라고 명명할 수 있습니다. 이제부터는 왼쪽 위에 있는 다른 사각형에 대해서 다루게 됩니다.

왼쪽윗부분

예를 들어, 각 사원들의 이름, 나이, 급여, 소속 부서와 같은 가상의 회사 정보를 저장해야 할 경우를 생각해 봅시다. 추가로, 부서의 이름이나 예산 위치와 같은, 이 회사의 부서들에 대한 정보도 기록해야 한다고 생각해 봅시다. 이러한 정보들에 대한 스키마는 다음의 표준 SQL 문들을 사용하여 얻을 수 있을 것입니다.

```
create table emp (

        name varchar(30),

        age int,

        salary float,

        dept varchar(20));

        create table dept (

        dname varchar(20),

        budget float,

        floor int);
```

사용자들은 정수, 부동 소수점, 문자열의 속성을 가진 각각의 구조화된 레코드들의 집합을 저장하기를 원한다는 것을 생각하시기 바랍니다. 그렇기 때문에, 레코드들은 SQL-92의 표준 데이터 형식을 사용합니다. 이 데이터들은 "단순형"으로 분류할 수 있습니다. 사용자들은 다음과 같은 질의들의 집합을 사용할 수 있습니다.

- 40세 미만이고 $40,000 이상의 급여를 받는 사원들의 이름

```
select name

    from emp

    where age <40 and salary> 40000;
```

- 건물 2층에서 일하는 사원들의 이름

```
select name

    from emp

    where dept in

    select dname

    from dept

    where floor = 1;
```

- 총무부에서 일하는 사원들의 평균 급여

```
select avg(salary)

   from emp
```

```
where dept = *Shoe*;
```

사용자가 바라는 질문들이 이러한 것들이고, 표준 SQL-92 문으로 표현될 수 있습니다. 이처럼, 이 응용 프로그램은 "단순한 데이터 질의" 사각형에 해당되는 경우입니다. 이러한 특징을 갖고 있는 왼쪽 윗 부분의 응용 프로그램들은 "상업용 데이터 처리"프로그램에 해당되며 다음의 사항들을 필요로 합니다.

질의어

질의어는 SQL-89에 따르며,새로운 SQL-92 표준에서도 적합합니다.

클라이언트 도구

프로그래머들이 데이터의 입력과 표현에 대한 형태를 정하는데 사용하는 툴킷은 클라이언트 도구를 필요로 합니다. 이 툴킷은 제어 흐름 논리를 통하여 서로 다른 형식간의 순서를 정하는 것도 할 수 있어야 합니다. 이러한 툴킷을 4세대 언어(4GL)라고 하며, 그 예로 Informix 4GL이 있습니다. 덧붙여서, 클라이언트 도구는 보고서 작성기와, 데이터베이스 디자인 도구, 성능 감시기, 그리고 C, COBOL, FORTRAN등의 다양한 3세대 언어로부터 DBMS 서비스를 호출할 수 있는 능력을 갖고 있어야 합니다.

성능

대부분의 상업용 데이터 처리 시장은 거래 처리라고 하는 것을 필요로 하고 있습니다. 많은 사용자들이 PC나 클라이언트 단말기로부터 동시에 DBMS 서비스를 요청할 것입니다. 이러한 사용자 트랜잭션은 아주 간단한 SQL 문으로 구현됩니다. 이중 많은 것은 갱신 요구입니다. DBMS에 의해 복잡한 병렬처리 갱신 작업이 처리될 경우에도, 사용자들은 예측 가능한 갱신 결과를 요구합니다. 이러한 곤란한 경우로 인하여, 직렬화라는 것을 확립 시킨 2 단계 로킹이라는 개념이 만들어졌습니다. 이 개념에 익숙하지 않은 사용자들은 Date, Bradley 또는 Ulman이 쓴, DBMS 교과서를 참고하십시오.

더구나, 어떤 오류 상황에서도 사용자의 데이터를 잃어버리지 말아야 한다는 것이 절대적인 요구 사항입니다. 이러한 상황들은 운영 체제의 고장 뿐 아니라 디스크 손상일 수도 있습니다. 손상의 복구는 전통적으로 write-ahead log (WAL) 기술을 사용하여 이루어집니다. 2단계 로크와 write-ahead log는 함께 트랜잭션 관리(사용자들이 요청한 질의 작업과 갱신 작업을 묶어서 하나의 작업 단위로 만드는 일)를 합니다. 각 트랜잭션은 끝까지 모두 실행되거나 또는 전혀 실행되지 않아야 하며(atomic; all or nothing), 다른 모든 병렬 트랜잭션이 실행되기 이전이나 실행이 끝난 뒤에 처리되어야 하고(seriablizable), 실행 결과를 절대로 잃어서는 안될(durable) 것입니다.

보안성

프로그래머들이 데이터의 입력과 표현에 대한 형태를 정하는데 사용하는 툴킷은 클라이언트 도구를 필요로 합니다. 이 툴킷은 제어 흐름 논리를 통하여 서로 다른 형식간의 순서를 정하는 것도 할 수 있어야 합니다. 이러한 툴킷을 4세대 언어(4GL)라고 하며, 그 예로 Informix 4GL이 있습니다. 덧붙여서, 클라이언트 도구는 보고서 작성기와, 데이터베이스 디자인 도구, 성능 감시기, 그리고 C, COBOL, FORTRAN등의 다양한 3세대 언어로부터 DBMS 서비스를 호출할 수 있는 능력을 갖고 있어야 합니다.

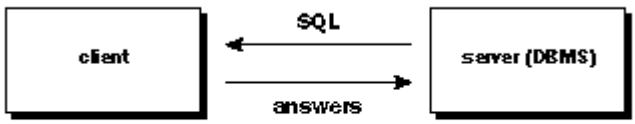


그림 1: 표준 클라이언트-서버 구조

어떤 경우에도, 사용자들은 트랜잭션 처리와 다수의 동시 사용자 환경에 가장 적합한 SQL DBMS를 요구할 것입니다. 이러한 종류의 트랜잭션에 대한 전형적인 표준 성능측정이 Transaction Processing Council의 TPC-A입니다. TPC-A는 Jim Gray가 저술한 "The Benchmark Handbook"에서 다루고 있습니다.

트랜잭션 처리에 가장 적합하게 만들어진 4GL 클라이언트 툴을 갖추고 있는 SQL DBMS가 갖춰야 하는 이러한 사항들은 관계형 DBMS 모델에서 얻어질 수 있습니다. 이 글의 목적에 따라, 결정은 명백합니다. 만약 표 1의 왼쪽 윗 부분에 해당되는 응용 프로그램일 경우, 관계형 DBMS (Relational DBMS; RDBMS)를 사용하십시오.

오른쪽 아랫부분

한 사용자가 모든 사원들을 칸막이로 나누어진 사각형 공간 안에 있게 하는 계획을 갖고 있는 회사의 편의 시설 설계자라고 생각해 봅시다. 경우에 따라, 부서들은 커지거나 작아지고, 건물에서의 각 구성원들의 자리 배치는 최적의 상태가 아닐 수 있습니다. 그러므로, 공간의 전체적인 재조정이 필요하게 됩니다. 빈 공간과 지속적으로 발생하는 사원들의 자리 조정에 대한 정보 수집이 해당 응용 프로그램에서 할 일입니다. 이러한 응용 프로그램을 위한 데이터베이스는 다음의 SQL 명령어로 나타낼 수 있습니다.

```
create table employee (

        name varchar(30),

        space polygon,

        adjacency set-of(employee));

        create table floors (

        number int,

        asf swiss-cheese-polygon);
```

같은 칸막이를 사이에 두고 인접해 있는 사원들의 정보를 수집하는 것 뿐 아니라, 각 사원들의 이름과 현재의 자리 배치를 기록하는 것이 필수적입니다. 각 층에 대하여 층의 숫자와 할당받을 수 있는 공간(asf:assignable square feet)을 기록하는 것도 필수적입니다. 이 값은 건물 전체에서 휴게실과, 엘리베이터 통로, 비상 계단의 값을 뺀 값입니다. 이처럼, 공간은 입체이며 인접도는 집합입니다. 이러한 데이터는 앞에서 살펴본 emp 데이터와 dept 데이터보다 복잡합니다. 따라서 이러한 응용 프로그램은 표 1의 오른쪽 부분에 해당합니다.

이 정보 수집 프로그램은 다음과 같은 의사 코드를 사용하여 나타낼 수 있습니다.

```
main()

        {

        read all employees;

        read all floors;

        compact();

        write all employees;

        }
```

여기서, 프로그램은 asf 정보를 갖고 있는 층들에 관한 정보 뿐 아니라, 사원들에게 할당된 공간 정보를 얻기 위하여 모든 사원 레코드를 읽어야 할 것입니다. 프로그램은 다음 단계를 위한 가상 메모리 구조를 구축할 것입니다. 다음에 여기에 있는 공간 조정 루틴이 각 층의 asf에 사원들의 새로운 위치를 정하기 위하여 이 구조를 측량합니다. 공간 조정 작업이 끝나면, 각 사원 레코드들은 새로운 자리배치 정보로 갱신될 것입

니다.

이 프로그램은 수집된 두 가지 정보 모두를 읽고, 처리하여, 기록합니다. 그래서, 하나의 파일에서 읽고, 처리하고, 기록하는 문서 편집기와 유사합니다. 문서 편집기에서와 같이, 질의는 사용되지 않습니다. 문서 편집기와 다른 점은 이 응용 프로그램에서 사용되는 데이터가 복잡하다는 점입니다. 이러한 것이 오른쪽 아래 부분에 위치하는 응용 프로그램입니다.

이러한 응용 프로그램들은 일부러 만들어 낸 경우라고 생각하시는 분들도 있을 것입니다, 그러나, 대부분의 전자 CAD(ECAD) 프로그램들이 이 경우에 해당합니다. 예를 들면, 복잡한 객체의 형태로 디스크에 저장되어 있는 컴퓨터 칩의 설계는 메인 메모리로 읽혀져서, 최적화 프로그램에 의한 공간 압축이 이루어진 후, 다시 디스크에 저장됩니다. 이처럼, ECAD 프로그램은 비록 단순하기는 하지만, 입방 정보 수집 프로그램과 유사합니다.

이러한 고유 식별자들은 읽기 작업이 진행되는 동안 가상 메모리의 포인터로 변환되어야 합니다. 가상 메모리 포인터는 가변적이고 데이터가 실제 읽혀지는 메모리의 주소에 따라 바뀌기 때문에, 이후 계속되는 프로그램의 실행에서 다시 사용될 수 없습니다. 이처럼, 디스크의 포인터는 메모리의 포인터와 본질적으로 다르며, 읽기 작업은 디스크 포인터를 메모리 포인터로 바꾸는 작업입니다. 마찬가지로, 데이터가 디스크에 다시 기록될 경우 공간 압축 루틴에 의하여 바뀌게 된 인접 자리 배치 정보는, 반대로 메모리에서 디스크로의 변환을 필요로 하게 될 것입니다.

데이터의 읽기와 변환 그리고 기록과 재변환(응용 프로그램의 데이터 저장 엔진에 파일 시스템이 사용될 경우)은 공간 압축 루틴을 작성하는 개발자에게는 엄청난 수고를 요할 것입니다. 보다 나은 방법은 프로그래밍 언어에서 디스크 저장을 지원하는 것입니다. 예를 들어, C++의 경우를 생각해 봅시다. 공간 압축 루틴은 그것의 처리 작업을 위해 정의된 데이터 구조의 집합을 가질 것입니다. 다음의 예를 보면, 그러한 선언 중의 하나가 있습니다.

```
integer I;
```

보통의 프로그래밍 언어에서, "I"는 가변적인 변수입니다. 즉 그것은 프로그램에 의해서 초기화되기 전에는 값을 갖지 않습니다. 프로그램이 종료되면 그 변수가 갖고 있던 값은 없어집니다. 다음과 같이, 영구적인 변수가 선언되었다고 가정해 봅시다.

persistent integer J;

정수형 변수 "J"는 지속됩니다. 그러므로, 공간 압축 프로그램이 종료하게 되면 그 값은 자동적으로 저장되게 됩니다. 프로그램이 다음 번에 시작하게 되면 이 값이 자동적으로 다시 사용됩니다. persistent 변수를 사용하면, 언어를 지원하는 시스템은 데이터를 디스크로부터 메인 메모리로 읽어 오고 다시 메인 메모리로부터 디스크로 저장해야 하는 것과 마찬가지로, 데이터를 읽어오고 저장해야 합니다. 공간 압축 루틴을 작성하는 사람은 알고리즘만 작성하면 되고 다른 세부 사항은 생각하지 않아도 됩니다.

DBMS를 가장 잘 지원하는 persistent 프로그래밍 언어가 이 공간 압축 응용 프로그램을 지원합니다. 이러한 편리한 상황 하에서, 코드는 다음과 같이 바뀔 수 있습니다.

```
main()
        {
    read all employees;
    read all floors;
        compact();
    write all employees;
    }
```

를 단순히 다음의 코드로 작성할 수 있습니다.

```
main()
        {
        compact();
        }
```

근본적으로, persistent 프로그래밍 언어는 특정 언어와 밀접하게 관련되어 있습니다. 확실히 persistence 시스템은 목표하는 언어의 특정 데이터 선언을 이해해야 합니다. 그래서, 만약에 공간 압축 루틴을 COBOL로 작성한다면, persistent COBOL이 필요할 것이며, persistent C++은 필요 없을 것입니다. 이처럼 각 언어마다 하나의 persistence 시스템이 필요합니다.

응용 프로그램이 다음의 DBMS 요구사항을 갖고 있다고 생각해 봅시다.

질의어

필요하지 않습니다.

클라이언트 도구

공간 압축 루틴 작성자는 Parc Place나 Next의 것과 같은 어떤 종류의 프로그래밍 언어 툴킷을 사용할 수 있습니다. 그렇기 때문에, 사용자는 프로그램 언어 제작 회사에서 클라이언트 툴을 얻을 수 있으리라 생각합니다. persistent 데이터 저장 회사가 만든 툴이 그렇게 중요한 것은 아닙니다.

성능

이 응용 프로그램이 해결해야 하는 근본적인 성능 문제는 다음의 문제입니다. 만약 공간 압축 루틴이 "평범한" C++에서 수행되고 자체의 데이터 저장 관리 기능을 갖고 있다면, 사용자는 확실한 성능을 보장 받을 것입니다. 만약 사용자가 공간 압축 루틴을 영구적 언어의 지원하에서 실행시킨다면, 사용자들은 공간 압축 루틴이 10 퍼센트 보다 더 느려지기 않기를 바랄 것입니다. 그러므로, 영구적 언어가 갖추어야 할 주된 사항은, non-persistent 언어에 결코 "뒤떨어지지 않는 성능" 입니다.

보안

위의 성능 조건은 persistent 언어의 구조에 근원을 두고 있습니다. 특정한 예로, 사용자가 다음의 명령을 실행한다고 가정해 봅시다.

```
J = J + 1;
```

위의 명령은 J를 1씩 증가시키는 것입니다. 만약 J가 지속되는 값이 아니라면, 이 문장은 100만분에 일초나 이보다 짧은 시간동안에 실행될 것입니다. 다시 말해, 만약 J가 지속적인 값이라면 이 문장은 갱신 작업이 됩니다. 만약 데이터 저장 시스템이 사용자의 프로그램과 다른 메모리 주소 공간에 존재한다면 주소 공간의 변환이 이 명령을 처리하기 위해 발생할 것입니다. 그 결과, 이 명령은 지속적이지 않은 변수의 경우보다 두

세배 정도 느려질 것입니다. 이러한 성능 저하는 사용자들이 받아들일 수 없는 것이고, 그림 2와 같이, 지속적 데이터 저장 시스템 설계자들이 명령어를 사용자 프로그램과 동일한 주소 공간에서 사용하도록 만들 것입니다.

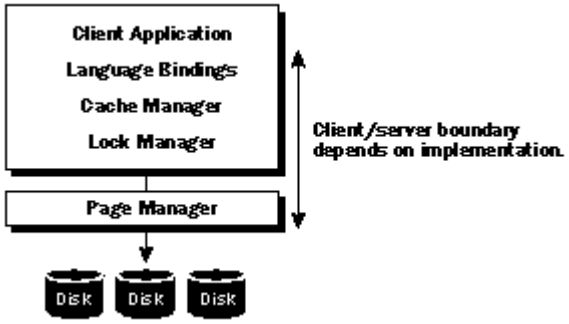


그림 2: ODBMS의 구조

주소 공간의 변환을 방지함으로써 훨씬 높은 성능을 얻을 수 있습니다. 그러나, 이것은 아주 치명적인 부작용을 갖고 있습니다. 데이터 저장 시스템이 데이터를 읽거나 쓰기 위해서 사용하는 운영 체제의 입출력 호출을 나쁜 목적을 가진 프로그램이 사용할 수도 있다는 것입니다. 둘 다 같은 주소 공간을 사용하기 때문에, 운영 체제는 각각의 호출을 보안상의 관점에서는 분별하지 못합니다. 그 결과, 어떤 사용자 프로그램이건 간에 지속적으로 유지되어야 하는 데이터 저장 시스템에 액세스하거나 운영 체제와 직접 접촉함으로써 데이터베이스에 대한 입출력 작업을 할 수 있습니다.

대부분의 회사의 경우, 이러한 시스템에 사원의 급여 데이터를 저장할 수는 없을 것입니다. 어떠한 데이터베이스 관리자도 보안이 심각하게 문제되는 시스템에 중요한 데이터를 보관하지 않을 것입니다.

persistent 언어의 설계자는 보안성을 고려하면서 높은 성능을 얻어내야 합니다. 그러나, 오른쪽 아래 부분에 해당하는 응용 프로그램의 경우에는 종종 보안성을 고려하지 않아도 됩니다.

이 글을 읽는 분들은 왜 관계형 DBMS에서 이러한 사항이 적용되지 않는가 의심할 수도 있을 것입니다. 이 두 영역간에는 중대한 차이가 있습니다. 영구적 언어 영역에서, 갱신은 "가벼운" 작업입니다. 즉, 이것은 아

주 작은 시간만을 소모합니다. 다음 갱신 작업 예를 보십시오.

J = J + 1;

이 갱신 작업은 기껏해야 백만분의 일초 정도 걸릴 뿐입니다. 이러한 이유로, 주소 공간 때문에 발생하는 부하는 심각해지는 것입니다. 관계형 연산의 세계에서 갱신 작업은 더 많은 부하가 걸립니다. 즉, 이것은 B++ 트리를 통한 하나 이상의 레코드에 대한 설정을 필요로 합니다. 이 작업은 갱신 작업을 실행하기 위하여 실제적인 경로 길이를 필요로 하므로 persistent 언어에서의 작업보다 2,3 배의 시간이 소요됩니다. 예를 들어, 다음의 SQL 질의를 보십시오.

```
select name
    from emp
    where age > 40;
```

이러한 점이 SQL과 같은 고급 언어와 C++과 같은 저급 언어의 차이점입니다. SQL에서의 갱신 작업이 상대적으로 많은 시간을 소요하기 때문에, 주소 영역을 이동하는데 걸리는 시간은 전체 질의 비용 중 작은 비중을 차지합니다. 지속적인 데이터의 사소한 갱신 작업의 경우에는 주소 공간 비용이 많은 비중을 차지합니다. 프로그래밍 언어와의 밀접한 결합과 영구적 변수에 대한 갱신, 그리고(가능하다면) 약간의 SQL에 대한 고려에 중점을 둔 시스템은 C++과 객체 지향적인 DBMS 공급자들에게 유용할 것입니다.

DBMS에 대한 선택은 매우 쉽습니다. 표 1에서 오른쪽 아래 부분에 해당하는 응용 프로그램의 경우, 해당 프로그래밍 언어에 대한 persistent 언어를 제공하는 회사를 선택하십시오. 그 회사가 표 1에서의 이 부분에 필요한 성능과 특성들을 담당할 것입니다. "만약 내가 오른쪽 아래에 해당하는 응용 프로그램을 관계형 DBMS 환경에서 실행시킨다면 어떤 일이 발생할 것인가?" 관찰력이 있는 분은 이러한 질문을 할 수도 있을 것입니다. 대답은 간단합니다. 다음의 명령어를 주의해 보십시오.

J = J + 1;

데이터베이스를 액세스하기 위해서는, 이 명령어가 반드시 SQL로 표현되어야 합니다. C++의 데이터 구조 체계가 SQL보다 훨씬 더 정교하기 때문에, 사용자는 SQL에서 C++ 데이터 구조 체계를 가상적으로 구현해야 합니다.

이러한 가상적 구현은 시간이 소요되며, 사용자들이 C++의 변수들을 SQL의 입력과 출력에다 짜맞추는 것을 필요로 합니다. 더구나, 사용자는 대부분의 명령어에서 많은 부하가 걸리는 클라이언트-서버 주소 영역의 변환을 하여야 합니다. 그 결과, 응용 프로그램은 많은 시간을 소모하게 되고 느리게 작동될 것입니다. 또 다른 경우, 한 종류의 응용 프로그램을 위하여 고안된 DBMS를 매우 다른 환경 하에서 사용하는 것은 큰 문제를 초래할 수도 있습니다. 결론은 관계형 시스템이 오른쪽 아래부분의 문제에 있어서는 이상적이 아니라는 것입니다.

"만약 내가 표 1에서의 왼쪽 윗부분에 해당하는 문제를 갖고 있고, 이것을 객체 지향적 DBMS 환경에서 실행시킨다면 어떤 일이 일어날 것인가?"라고 반대로 질문할 수도 있을 것입니다. 다시 말하지만, 그 결과는 만족스럽지 못합니다. 특히, 대부분의 객체 지향적 언어 회사들은 매우 제한된 SQL을 갖고 있으며, SQL을 사용한 데이터 갱신을 전혀 지원하지 않는 것들도 있습니다. 사용자는 자신의 트랜잭션을 C++로 표현해야 하며 훨씬 많은 양의 코드를 작성해야 할 것입니다. 덧붙여서, 이러한 제품들은 50회나 100회의 연속되는 데이터 갱신에 적합하도록 만들어진 것이 아니며, 다수의 사용자 환경에서는 아주 낮은 성능을 나타내는 경향이 있습니다.

바꾸어 말하면, 한 종류의 응용 프로그램에 적합하도록 만들어진 DBMS를 매우 상이한 환경에서 사용하는 것은 문제를 야기시킬 수도 있습니다. 결론은 객체지향적 DBMS(Object-Oriented DBMS; OODBMS)는 왼쪽 위의 경우에 해당하는 문제에는 적합하지 않다는 것입니다.

오른쪽 윗 부분

이제 우리는 질의 중심적이고 복잡한 데이터를 요구하는 응용 프로그램을 다루겠습니다. 이것은 오른쪽 윗

부분의 대표적인 것입니다. 캘리포니아주 수자원국은 캘리포니아주의 대규모 수자원 프로젝트를 포함한 수도 사업 뿐 아니라, 캘리포니아의 거의 모든 강과 관개 수로를 관리하는 곳입니다. 설비를 문서화하기 위하여, 캘리포니아주 수자원국은 35밀리 슬라이드 데이터들을 유지하고 있습니다. 시간이 지남에 따라, 이 데이터들은 500,000개의 슬라이드로 불어났으며, 캘리포니아주 수자원국의 직원들과 그 외 관계자들에 의해서 매일 다루어지고 있습니다.

클라이언트는 대부분 그림의 내용을 요청할 것입니다. 예를 들어, 한 사원은 북부 캘리포니아의 물을 끌어오고 있는 거대한 펌프 장비의 그림을 포함한 프리젠테이션을 할 수 있습니다. 또 다른 사용자는 샌프란시스코 만의 일몰 장면을 요청할지도 모르며, 동시에 수위가 매우 낮은 저수지의 사진을 원할 수도 있을 것입니다.

캘리포니아주 수자원국은 슬라이드를 내용에 따라 찾는 작업이 어렵다고 판단했습니다. 미리 정해진 개념의 분류에 따라 모든 슬라이드에 일일이 손으로 색인을 넣는 작업은 매우 비용이 많이 드는 일입니다. 더구나, 클라이언트에서 관심을 갖고 있는 내용들은 시간이 감에 따라 바뀝니다. 예를 들어, 저수지의 수위가 낮다는 것은 가뭄의 경우가 아니면 관심사가 되지 않을 것입니다. 또한, 멸종 위기에 처한 생물이 현재의 쟁점이라면, 이것은 단지 몇 년간의 쟁점이 될 것일 뿐 몇 십년간 지속적인 관심사가 되지는 않을 것입니다.

현재, 각 슬라이드마다 제목이 있습니다. 그 예는 다음과 같습니다.

비계 건축 중의 어번 댐 사진.

매우 원시적인 시스템도 특정한 검색어에 의한 슬라이드 검색을 할 수 있습니다. 그러나, 많은 관심 대상 개념이 제목에 잘 나타나 있지 않기 때문에 이러한 검색어 시스템은 잘 작동하지 않을 것이며, 그래서 슬라이드는 검색어만으로 표현될 수 없습니다.

결과적으로, DWR은 모든 슬라이드에 대한 스캔 작업을 하여 디지탈 형식으로 만들어 다음의 데이터베이스를 구축하였습니다.

```
create table slides (

        id int,

        date date,

        caption document,

        picture photo_CD_image);

        create table landmarks (

        name varchar(30),

        locationpoint);
```

각 슬라이드는 식별자를 갖고 있습니다. 곧, 슬라이드가 찍힌 날짜와 특수한 제목, 그리고 Kodak Photo-CD 형식의 디지탈화 된 비트들이 식별자입니다. 사실상, 이 Photo-CD 형식은 128 x 192의 "썸네일"로부터 2K x 3K의 컬러 이미지까지의 다섯 종류의 이미지의 모음입니다. 현재, 캘리포니아주 수자원국은 20,000개의 이미지를 디지탈화하여 3 Tbyte의 데이터베이스를 구축하고 있습니다.

캘리포니아주 수자원국은 이미지를 전자적으로 분류하는 것에 중점을 두고 있습니다. 위에서 언급한 대로, 이미지들을 손으로 분류하는 것은 받아들이기 어려운 일입니다. 캘리포니아주 수자원국은 원하는 사항 중의 하나는 각 슬라이드의 지리적 위치입니다. 이 지리 등록 기술에는 미국의 공용 공간 데이터베이스가 포함됩니다. 특별히, 캘리포니아주 수자원국은 캘리포니아의 모든 지형 지도상에 나타나는 경계 표시의 명칭들을 경계 표시의 위치와 함께 갖고 있습니다. 이 경계 표시들은 앞의 코드 예에서 table landmarks에 반영되었습니다. 다음에, 캘리포니아주 수자원국은 각 슬라이드의 제목이 경계 표시 이름을 포함하고 있는지 알아볼

것을 제안했습니다. 만약 그렇다면 경계 표시의 위치는 슬라이드의 지리적 위치를 가리키는 좋은 데이터가 되기 때문입니다.

추가로, 캘리포니아주 수자원국은 이미지를 검사하고 이미지에 속성을 부여하는 기록 프로그램에 중점을 두었습니다. 사실, 이 특수한 슬라이드 데이터 중에서 그림의 윗 부분의 색깔이 오렌지 색인 데이터를 찾음으로써, "일몰 장면"을 찾을 수 있습니다. 저수지의 수위가 낮은 그림은 갈색에 둘러 쌓인 파란 물체를 찾음으로써 얻을 수 있습니다. 캘리포니아주 수자원국이 관심을 갖고 있는 그림에 대한 많은 속성들은 매우 평범한 패턴 매칭 기술입니다. 물론, 멸종 위기에 처한 생물들과 같이, 어떠한 속성들은 분별하기가 약간 까다롭습니다. 이러한 속성들은 미래의 패턴 인식 기술의 발전을 기다려야 할 것입니다.

결과적으로, 앞에서 다룬 스키마는 caption 필드에 간략한 제목을 포함하고 있고 picture 필드에는 Photo-CD 이미지를 담고 있습니다. 이러한 응용 프로그램은 명백히 표 1에서 행렬의 오른쪽 부분에 해당합니다.

더구나, 이 데이터베이스의 모든 클라이언트는 임시 변통적인 질의를 요청합니다. 이러한 질의 중 하나는 20 마일의 Sacramento 일몰 장면을 찾는 것이 될 수도 있습니다. 고객들은 그들이 사용하는 환경이 다음의 SQL 질의를 할 수 있는 친숙한 환경이기를 바랍니다.

```
select id

        from slides P, landmarks L S

        where sunset (P.picture) and

        contains (P.caption, L.name) and

        L.location |20| S.location and

        S.name = "Sacramento';
```

이 질의는 다음과 같이 설명될 수 있습니다. 첫째, landmarks 테이블에서 세크라멘토를 찾는 것이 필수적입니다. 이 테이블(여러 개의 지형 지도에 있는)은 Sacramento의 지리적 위치를 찾습니다. 다음엔, S.location에서 20마일 내에 있는 다른 경계 표시(L.location)들이 검색될 것입니다. |20|은 2개의 피연산자를 위해 사용자가 정의한 연산자이고, 각 지점에 대하여 두 지점 사이의 거리가 20마일 이내이면 true를 리턴할 것입니다. 경계 표지가 그림의 제목에 나타날 경우, 이러한 경계 표지들의 집합이 확실히 해당될 것입니다. "contains"는 제목과 검색어라는 두 개의 인수를 받아들이는 사용자 정의 함수입니다. 이 함수는 질의 결과를 만족할 수 있는 몇 개의 그림을 찾아 줄 것입니다. 마지막으로 sunset이 그림이 윗 부분에 오렌지색을 갖고 있는지를 알아보기 위하여 비트를 검사하는 두 번째의 사용자 정의 함수가 됩니다. 질의의 실제 결과는 고객이 바라는 내용이 됩니다.

명백하게 이 응용 프로그램은 복잡한 데이터에 대한 "질의 위주의 작업"을 합니다. 이러한 작업이 표 1의 오른쪽 윗 부분에 해당하는 응용 프로그램의 예입니다. 이러한 종류의 응용 프로그램이 필요로 하는 사항이 이제부터 다루어질 것입니다.

질의어

질의어의 질의 예가 Sacramento의 일몰 장면에 관한 것임을 주목하십시오. 네 개의 절들은 질의의 속성에 관한 것입니다. 첫번째는 사용자가 정의한 함수인 sunset이며, 그러므로 SQL-92로는 불가능합니다. 두 번째 절은 사용자가 정의한 연산자인 |20|을 포함하고 있으며, 이것도 역시 SQL-92로는 안됩니다. 오직 마지막 절만이 SQl-92에서 표현할 수 있습니다. 이처럼, 오른쪽 윗 부분의 응용 프로그램은 최소한 사용자가 정의한 함수나 연산자를 사용할 수 있는 질의어를 필요로 합니다. 이러한 기능을 갖고 있는 SQL 중 최초의 표준 버전은, 현재 기본 형식으로 되어 있는 SQL-3입니다. 그러므로, 사용자는 SQL-3 DBMS를 필요로 합니다. SQL-2로 표현할 수 없는 4개의 절이 있기 때문에, 어떠한 SQL-2 DBMS도 본질적으로 이 응용 프로그램에는 쓸 수 없습니다.

도구

캘리포니아주 수자원국은 사용자가 쓰는 응용 프로그램이 캘리포니아 주의 지도를 화면에 보여주는 것을 원

할 것입니다. 그 다음, 주의 영역 중 사용자가 관심을 갖고 있는 곳에 동그라미를 그릴 수 있는 지시 장치가 사용될 수도 있습니다. 예를 들어, 사용자들은 화면으로 Sacramento 지방의 지도를 보면서 지도 각 부분에 있는 각 이미지의 "대략적인 스케치"를 볼 수 있을 것입니다. 각 지방을 "지적하는" 기능을 사용하여 사용자는 관심 지역의 대략적인 그림을 볼 수 있습니다. 추가로, 사용자들은 Photo-CD 객체에 저장되어 있는 고해상도 이미지로 특정 지역을 "확대하여" 볼 수 있을 것입니다. 이렇게 "지적하고 확대하는" 사용자 환경은 Khoros, Explorer 및 AVS와 같은 과학 분야의 가시화 제품에 많이 사용되는 방법입니다. 이처럼 사용자는 DBMS와 세밀하게 결합되어 있는 투시 시스템의 효과를 볼 수 있습니다. 업무적인 형태가 여기엔 없기 때문에, 이 응용 프로그램에서 4GL은 거의 쓸모 없다는 것을 주시하기 바랍니다.

성능

사용자는 Sacramento의 일몰 장면과 같은 질의의 실행이 매우 빨리 이루어지기를 바랍니다. 이러한 질의들은 서술어의 의미를 사용하여 결정을 하는 질의입니다. 이러한 환경에서 성능을 발휘하기 위해서는 최적화된 설계가 필요합니다. 예를 들면, sunset 함수는 10억개 이상의 명령어를 소모합니다. 이러한 경우, 만약 질의 최적화기가 다음 형식의 절을 처리한다면

WHERE sunset (image) AND date <"jan 1, 1985"

두 번째 절은 가장 처음 실행되어야 할 것이며, 그렇게 함으로써 여러 장의 이미지를 검색하지 않을 수 있습니다. 그 다음 sunset 절이 처리될 것입니다. 행렬의 오른쪽 윗 부분에 해당하는 응용 프로그램에서는 어떠한 함수가 많은 계산을 필요로 하는가를 결정하는 것이 중요합니다. 덧붙여서 말하면, 많은 질의가 sunset 함수를 사용한다면 각 이미지에 대한 이 함수의 값을 미리 구하여 놓는 것이 바람직할 것입니다. 그러면, 함수는 일몰 장면에 관련된 질의 당 한 번씩 실행되지 않고, 이미지 당 한 번씩만 실행될 것입니다.

이미지가 추가되거나 갱신될 때마다 미리 계산 작업을 하는 것은 유용한 최적화 전략입니다. 마지막으로, Sacramento에서 20 마일 이내에 있는 경계 표시를 찾기 위해서는, 효율적으로 "동그라미" 질의를 사용하는 것이 필요합니다. 이러한 2차원적인 질의들은 1차원적인 검색 방법을 갖고 있는 B+ 트리 색인에 의해서는 빨리 실행될 수 없습니다. 이처럼, 전통적인 검색 방법(B+트리와 해싱)은 이러한 경우에는 적합하지 않습니

다.

이러한 공간적인 절들을 빨리 처리하려면 사각형 격자 파일이나, R 트리 혹은 K-D-B 트리와 같은 공간적 검색 방법이 필요합니다. DBMS는 이러한 "객체 관련 검색 방법(object-specific access methods)"을 갖고 있거나 사용자나 시스템 전문가가 검색 방법을 추가하는 것을 허용하여야 합니다.

최적화 TPC-A가 이 응용 프로그램에는 적합하지 않다는 것을 주지하십시오. 어떠한 트랜잭션 처리 프로그램도 오른쪽 위 부분에는 적합하지 않습니다.

보안

표 1에서 오른쪽 윗 부분에 속하는 어떤 응용 프로그램은 보안을 필요로 합니다. 그렇기 때문에, DBMS는 그림 1에 나타나 있는 클라이언트-서버 구조하에서 수행되어야 합니다. 이러한 환경에서, 보안성은 성능을 높이기 위한 고려 대상이 거의 될 수 없습니다. 더구나, 이것이 질의의 세계이기 때문에 속도면에서 명령어가 많은 시간을 소비하게 되며, 보안성에 대한 비중을 감소시키는 것은 오른쪽 아래 부분에서보다 성능 향상 효과가 적을 것입니다.

변형된 SQL-3 언어를 지원하는 DBMS들과 복잡한 의사 결정 지원 SQL-3 질의들을 최적화시켜 주는 DBMS들은, 객체 관계형 DBMS라 할 수 있습니다. 이러한 DBMS들은 본질적으로, SQL을 지원하기 때문에 관계형입니다. 본질적으로, 이들은 복잡한 데이터를 지원하기 위하여 객체 지향적입니다. 요점을 말하면, 이러한 DBMS들은 관계적 영역의 SQL과 객체 영역의 모델링 원리가 결합한 것입니다.

이 글에서는, SQL-92 시스템은 사용자 정의 질의를 표현하지 못하기 때문에 오른쪽 윗 부분의 경우 문제를 발생시킬 수 있다고 이미 언급하였습니다. 결과적으로, 사용자 질의는 사용자 프로그램에서 실행되어야 하며, 이것은 사용자들에게 무거운 짐이 될 것입니다. 덧붙여 말하면, 만약 "일몰 장면"이 사용자 공간에서 실행된다면, 매우 큰 이미지가 서버로부터 클라이언트-서버의 연결을 통하여 클라이언트로 전송되어야 할 것입니다. 이러한 전송은 심각한 성능 문제를 유발할 수 있습니다. 이처럼, 관계형 DBMS는 오른쪽 윗 부분의 문제를 해결하기에는 너무 많은 시간을 소비합니다.

이와 마찬가지로, persistent 언어도 사용자 정의 질의를 표현할 수 있는 질의어를 제공하지 못합니다. 그래서, 질의는 C++ 프로그램으로 표현되어야 합니다.

앞서 언급한 바와 같이 오른쪽 아래 부분 문제의 해결에는 관계형 DBMS를 이용할 수는 없습니다. SQL은 오른편 유형의 인터페이스가 아닙니다. 같은 이유로 SQL-3 시스템은 올바른 선택이 될 수 없습니다. 그렇기 때문에 오른쪽 아래 부분 문제 해결에는 객체 관계형 DBMS가 적합하지 않습니다.

결론적으로, 오른쪽 윗 부분의 문제는 일반적으로 객체 관계형 DBMS가 해결할 수 있습니다. 유사한 이유로, SQL-3는 SQL-92보다 고급이며, 객체 관계형 DBMS를 사용하여 SQL-92의 문제를 해결할 수 있습니다.

요약

이미 설명한 바와 같이, 각각의 특수한 시장을 갖고 있는 세 종류의 다른 DBMS들이 있습니다. 이러한 시장 분야들은 매우 다른 여러 질의 언어와 툴, 그리고 엔진의 성능 향상에 따른 최적화를 필요로 합니다. 사실, 각 종류의 엔진은 세밀하게 시장의 분야를 "선별하며" 각 분야에 가장 적합한 형태로 바뀝니다.

덧붙여 말하면, 한 분야에서 비롯된 DBMS는 일반적으로 다른 분야에서 문제를 발생시킵니다. 이제, 여기서 다루어진 모든 내용을 요약하면 간단합니다. 문제를 네 가지 영역 중의 하나로 분류하고, 각 영역에 가장 적합한 DBMS를 사용하라는 것입니다. 이러한 사항은 표 2에 요약되어 있습니다.

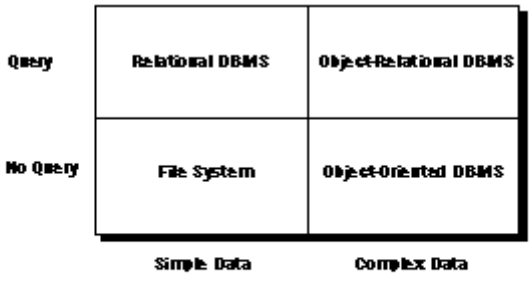


표 2: DBMS 응용 프로그램의 분류

우선, 상당한 수의 응용 프로그램이 이 네 가지 영역에 부합됩니다. 이 중에는 의학적 영상 문제와 디지탈 데이터 도서관 문제, 오락 산업에서의 자산 관리 문제 그리고 대부분의 과학용 데이터베이스 응용 프로그램들이 포함됩니다. 다른 대안이 없었기 때문에 과거로부터 계속 파일 시스템에 의존하여 온 이러한 시장을 기반으로 객체 관계형 DBMS(ORDBMS) 시장은 관계형 DBMS(RDBMS) 시장과 객체 지향형 DBMS(OODBMS) 시장의 중간 정도의 크기가 될 것으로 예상됩니다.

그러나 시간이 지남에 따라, 왼쪽 윗 부분의 많은 응용 프로그램들이 오른쪽 윗 부분으로 옮겨 가고 있습니다. 예를 들어, 관계형 DBMS에서 작동하는 일반적인 상업용 데이터 처리 프로그램에 의해서 구현되고 있는 고객 데이터베이스와 소송 데이터베이스를 갖고 있는 보험 회사의 경우를 생각해 봅시다. 또한, 이 회사는 모든 사고 장소에 대한 간략한 그림과 경찰 보고서를 스캔한 이미지, 파손된 차의 사진, 사고 장소의 위도와 경도, 그리고 고객 집의 위도와 경도 정보들을 추가하려는 계획을 갖고 있습니다. 그 다음 이 회사는 각 지방에서 가장 위험한 지역 10곳을 찾아낸 다음, 그 지역에서 1마일 이내에 거주하는 고객에게 위험 부담금을 부과할 계획입니다. 위험도가 높은 고객을 찾는 일 이외에도, 이 보험 회사는 과도하게 차를 고치는 부정한 정비소들도 찾아내기를 원합니다. 결국, 이 회사 역시 다른 종류의 결정 제공 함수를 추가할 계획을 세운 것입니다.

이처럼, 일반적인 보험 프로그램들은 왼쪽 윗 부분에서 옮겨가서 왼쪽 윗 부분과 오른쪽 윗 부분의 요소를 모두 갖고 있는 프로그램이 될 것입니다. 동시에, 왼쪽 윗 부분의 요소는 "기술적 가치 저하"를 겪게 될 것입니다.

특히, 일반적인 왼쪽 윗 부분의 응용 프로그램의 경우 상업용 데이터 처리 부담이 완만하게 증가할 것이며, 이것은 아마 1년에 10퍼센트 정도가 될 것입니다. 고객의 수가 천천히 증가하기 때문에, 사고의 수도 천천히 증가할 것입니다. 이처럼 처리 부담은 천천히 증가합니다. 동시에, 이러한 처리 부담을 실행하는 컴퓨터의 가격은 감소합니다. 일반적으로 CPU의 개발 주기와, 디스크, 주기억장치의 가격은 1년에 약 2배 정도의

크기로 감소합니다. 이러한 경향으로 상업용 데이터 처리의 부담을 1년에 2배 정도 낮출 수 있습니다. 이처럼 응용 프로그램이 오른쪽 위로 옮겨갈 뿐 아니라, 왼쪽 윗 부분의 응용 프로그램 가격도 싸질 것입니다.

이러한 경향은 응용 프로그램을 왼쪽 윗 부분으로부터 오른쪽 윗 부분으로 이끌어갈 것이며, 그래서 시간이 지남에 따라 객체 관계형 DBMS의 시장이 커질 것이고, "차세대 DBMS"를 선도해 갈 것입니다.