Tech Notes (vol.11)

Informix Dynamic Server 7.2x, 7.3x 또는 9.1x에서 Informix Dynamic Server 9.2X로 성공적인 업그레이드

소개

이 글은 버전의 마이그레이션 과정에 대한 기술적인 개요입니다. 버전을 마이그레이션하기 전에 수행해야 할 작업도 소개하고, 중요한 설명서에 대한 참조할 만한 설명서도 소개하고 있으며 업그레이드 권장 사항도 설명합니다.

이 글은 버전 9.21로 마이그레이션을 계획하는 데 초점을 두고 있지만, 9.20으로 마이그레이션하는 데에도 적용할 수 있습니다.

Informix Dynamic Server 9.2x으로 업그레이드하는 과정은 복잡하지 않지만 데이터 무결성 및 원활한 역변환(reversion) 과정을 보장하려면 반드시 단계에 따라 작업해야 합니다. 업그레이드할 때 필요한 환경 변화 및 구성 파일뿐만 아니라 Informix Dynamic Server 9.2x에서 제공하는 새로운 기능에 친숙해져야 합니다.

Informix Dynamic Server 9.2x의 배경

Informix Dynamic Server 9.2x은 Informix Dynamic Server 7.3의 주요 다음 버전이며, Informix Dynamic Server 7.3*의 모든 기능을 지원합니다. 또한 Informix Dynamic Server 9.2x은 Informix Dynamic Server/Universal Data Option 9.14의 주요 다음 버전이며 9.14의 확장성 기능을 모두 포함합니다. Informix Dynamic Server 9.2x의 완성으로 Informix에서는 Informix Dynamic Server 7.3과 Informix Dynamic Server/Universal Data Option 9.1x의 코드 병합을 완료했습니다. 다시 말해서, Informix Dynamic Server 9.2x 이 Informix Dynamic Server의 새로운 코드 기반입니다. 7.3에서 9.2로 업그레이드하는 고객은 이전에 9.1x에서만 찾을 수 있었던 확장성 기능의 이점을 이용할 수 있으며, 9.2로 업그레이드하는 9.1x 고객은 9.1x가 분할된 후 7.x 코드 스트림에 추가된 기능들의 이점을 이용할 수

있을 것입니다.

* 9.21 이상 설치되어야 합니다.

Informix Dynamic Server 9.2x에는 Informix Dynamic Server 9.2x과 Informix Internet Foundation 등의 두 가지 다른 패키지가 있습니다. Informix Dynamic Server 9.2x에는 데이터베이스 서버(7.3 패리티), 확장성(9.1 패리티), Informix Connect(런타임 연결) 및 DataBlade Developers Kit (DBDK)이 들어 있습니다. Informix Internet Foundation에는 Informix Dynamic Server 9.2x(위 참조), Informix J/Foundation™, Web DataBlade, Excalibur Text DataBlade 및 Informix Office Connect™(Excel 프런트 엔드)가 들어 있습니다. Informix J/Foundation 기능은 Informix Dynamic Server 내에서 Java Virtual Machine(JVM)을 호스트하는 기능을 제공하여 고객이 데이터베이스 서버에서 Java를 실행할 수 있습니다. 이 기능으로 Java 내장 프로시저 및 Informix DataBladeR 모듈에 대한 지원이 강화되어 고객이 Informix 데이터베이스에서 Java "구성 요소"를 실행하고 저장할 수 있습니다.

Informix Dynamic Server 9.2x의 향상된 기능

업그레이드하기 전에 Informix Dynamic Server 9.2x의 새로운 기능 및 향상된 기능에 친숙해지는 것이 좋습니다. 이러한 기능 중 일부는 사용자가 제어할 수 없지만 다른 기능, 특히 SQL과 관련되어 향상된 기능은 사용자가 직접 구현할 수 있습니다. 역변환이 필요하게 되었을 때 Informix Dynamic Server 을 제대로 분석하기 위해서 이러한 새로운 기능을 이해하는 것이 좋습니다. 향상된 기능의 전체적인 목록 및 설명은 릴리즈 정보에서 보실 수 있습니다. Informix Dynamic Server 9.2x에는 다음과 같은 부분의 새로운 기능이 포함됩니다.

9.20에서 소개된 기능

패리티 기능

1. **7.3 기능 중 세 가지를 제외한 모든 기능** - 로그되지 않은 테이블, 인덱스 이름 변경 및 ANSI 외부 조인을 제외한 Informix Dynamic Server 7.3의 모든 기능이 9.20에 포함되어 있습니다. 제외된 세 가지 기능은 9.21에서 구현됩니다.

2. **9.1의 모든 기능** - Informix Dynamic Server/Universal Data Option 9.1의 모든 기능이 9.20에 포함되어 있습니다.

SQL 기능

1. **긴 식별자** - 9.2 이전의 Informix 릴리즈는 식별자 크기를 18자까지 지원하며, 사용자 이름은 8자까지 지원합니다. Informix Dynamic Server 9.2에서 식별자의 길이는 18자에서 128자로 확장되고, 사용자 이름은 8자에서 32자로 확장됩니다. 또한, Informix Dynamic Server 9.2 서버에서는 '$'도 식별자의 첫 글자가 아닌 경우에, 유효한 식별자 기호(문자나 숫자 아님)가 됩니다.

2. **업데이트 lock 유지** - 현재 사용자의 격리 수준이 RR(Repeatable Read)보다 낮게 설정되어 있으면 행에 대한 업데이트 lock은 커서에서 다음 행을 가져 오는 순간 릴리즈됩니다. 이 기능과 함께, 트랜잭션이 끝날 때까지 이러한 업데이트 lock을 유지할 수 있는 스위치가 제공됩니다. 이 기능은 dummy 업데이트나 RR 격리 수준 같은 작업을 피할 수 있는 응용 프로그램의 성능을 향상시킵니다.

3. **On-Select 트리거** - Select 트리거는 Informix Dynamic Server 및 Informix Dynamic Server 서버들의 현재 트리거 메커니즘을 확장한 것입니다. Select 트리거는 테이블이나 열에서 정의되고 테이블이나 열이 특정 형식의 SELECT 문에 사용될 때 시작됩니다. 응용 프로그램 개발자는 select 트리거를 사용하여 응용 프로그램에 특정한 감사나 계정 규칙을 강화할 수 있습니다. 예를 들어, 사용자는 특정 테이블이 선택될 때마다 감사 테이블에 감사 레코드를 삽입하는 select 트리거를 정의할 수 있습니다. select 트리거의 경우 정식 명령문 집합에는 SELECT 문과 하위 질의가 포함됩니다. 트리거가 시작되면 트리거는 최대 세 가지 유형의 동작을 실행할 수 있는데, 그 동작은 before action, after action, for-each-row action 등입니다. "before-action"은 문을 트리거하는 것이 적용되기 전에 실행합니다. "after-action"은 문을 트리거하는 것이 완료된 후 실행합니다. 이 두 가지 유형의 동작은 트리거에 필요한 초기화 및 정리 작업을 수행하는 데 사용할 수 있습니다. "for-each-row action"은 트리거 테이블에서 검색된 각각의 정식 행에 대해 한 번 실행하며, 응용

프로그램에 필요한 특별 감사 및 계정을 수행하는 데 사용할 수 있습니다.

4. **인용 문자열에서 포함된 새라인(newline)** - 이 기능을 사용하면 onconfig 매개 변수를 통해 현재의 ifx_allow_newline() 기능을 사용할 수 있습니다.

5. **중첩된 도트(dot; .) 표현식** - 9.2 이전의 Informix 서버에서 필드 참조는 열, SPL 변수 및 행 형식의 명령문 로컬 변수로 제한됩니다. 이 기능은 행 형식의 필드 참조를 일반화하여 임의 표현식을 허용합니다. 행 형식의 모든 필드에 액세스하는 데 단일 필드 참조 뿐만 아니라 별표(asterisk) 표시법을 사용합니다. 행 형식의 모든 표현식은 필드 프로젝션에 사용할 수 있습니다.

6. **Row 형식 안에서의 Serial 형식** - 이 기능은 테이블 상속에서 SERIAL 및 SERIAL8 카운터에 대한 지원을 추가합니다. serial 칼럼을 상속하는 테이블로 삽입하면 상속 테이블이 아닌 일반 테이블이 가지고 있는 고유 식별자를 유지합니다. 이렇게 변경하지 않으면 serial은 테이블 계층에서 허용되지 않는데, 이것은 하위 테이블로 삽입하면 개별 카운터를 사용하고 중복되어 삽입되기 때문입니다.

7. **Collection 하위 질의(subquery)** - Informix Dynamic Server 버전 9.1x는 Collection 데이터를 SPL 및 ESQL/C의 테이블로 취급하는 기능을 이미 제공하고 있습니다. Collection 하위 질의는 관계형 데이터를 Collection으로 취급하는 기능을 제공하여 일반적인 생각을 다른 방향으로 확장합니다. 이 기능으로 사용자는 하위 질의의 결과를 사용하여 Collection을 만들 수 있습니다. 이 Collection은 기존 Collection이 사용된 모든 방식으로 사용할 수 있습니다. 이 Collection을 수정해도 원본으로 사용하는 테이블의 행은 변경되지 않는 대신 테이블 행의 복사본이 수정됩니다.

8. **유도 Collection 테이블(Collection Derived Tables)** - 이 기능으로 사용자는 Collection 표현식의 결과를 질의의 FROM 절에서 사용될 수 있는 가상 테이블로 취급할 수 있습니다. Collection의 요소는 가상 테이블의 행으로 취급됩니다. 유도 Collection 테이블을 사용하면 사용자는 'SELECT ... FROM' 구문을 사용하여 Collection 요소를 질의할 수 있습니다. 새로운 확장으로 사용자는 일부 시나리오에서 더 적은 호스트 변수로 ESQL 프로그램을 더 간단하게 작성할 수 있습니다. 그런 응용 프로그램 중 하나는 중첩된 Collection에서 데이터를 검색하는 프로그램입니다. 이러한 응용 프로그램이 없으면 사용자는 여러 커서와 임시 테이블의 조합을 사용해야 하는데, 이렇게 하면 더 복잡하고 효율성은 떨어집니다.

9. **Collection 구성자** - 현재 Informix Dynamic Server 9.1x 제품은 Collection 구성을 지원하지만 ANSI 표준이 아닌 Collection 문자열 서식을 사용하고 Collection 요소는 리터럴만 허용됩니다. 9.2의 Collection 구성자로 ANSI 표준 구문을 사용하게 됩니다. Collection 구성자를 사용하면 사용자는 모든 표현식 요소의 형식이 같은 임의 표현식을 사용하여 Collection(SET, MULTISET, LIST)을 구성할 수 있습니다. Collection의 요소가 NULL이 될 수는 없습니다. Collection의 요소 형식은 내장, 사용자 정의 형식 또는 복합 형식이 될 수 있습니다.

DBA 기능

1. **동적 lock 할당** - 이 기능으로 서버는 LOCK TABLE이 오버플로될 때 응용 프로그램에 오류를 반환하는 대신 더 많은 lock을 할당할 수 있습니다. Informix Dynamic Server는 새로운 lock 집합을 15번까지 할당하는데, 할당되는 새로운 lock의 수는 MIN(100K, LOCKS onconfig 매개 변수)에 설정됩니다. DBA는 이 정보를 사용하여 다음에 서버를 종료하는 lock의 기본 수를 늘릴 수 있습니다.

2. **보관 검사기(Archive Checker)** - ON-BarTM 9.2 릴리즈에는 Level 0 보관을 복원하는 데 필요한 모든 페이지를 확인하는 특수한 "-v" 옵션이 들어 있습니다. ON-Bar의 -v 명령으로 사용자는 서버에서 실제로 복구를 실행하지 않고 Level 0 백업 중에 만들어진 테이프나 다른 저장 매체를 검사할 수 있습니다. 확인 중에 발견된 모든 불일치는 특별 오류 파일에 기록됩니다. 원칙적으로 보관 검사 옵션은 수준 보관의 실제 복구가 실행되기 전에 사용됩니다.

3. **계층적 복제(Hierarchical Replication; HR)** - HR 기능은 복제 메시지에 대한 네트워크 토폴로지의 명시적인 명세를 제공합니다. 그렇기 때문에 사용자가 가능한 서버간 연결의 하위 집합만 지정하는 것이 허용됩니다. 그러면 구현은 한 서버에서 다른 서버로 데이터 복제가 필요할 때 메시지를 라우팅할 수 있습니다. ER(Enterprise Replication)의 이전 릴리즈에서는 네트워크의 각 서버에서 네트워크에 있는 다른 모든 서버에 연결할 수 있었습니다. 버전 7.30에서 상시 연결에 대한 요구 사항은 완화되었지만 전체 연결 가능에 대한 요구 사항은 완화되지 않았습니다. 이 기능으로 연결 요구 사항은 더 완화되었습니다. Enterprise Replication은 스마트 BLOB이나Informix DataBlade 모듈을 지원하지 않는다는 것에 주의하십시오.

4. **R-트리 용 OnCheck/OnLog** - R-트리 인덱스에는 특수한 검사 논리가 필요합니다. 이 기능으로

R-트리에 고유한 행동을 고려하여 인덱스 내의 내부 및 외부 구조를 모두 검사하고 표시할 수 있습니다. 또한 이 기능은 다른 형식의 인덱스에도 적용할 수 있도록 디자인되었습니다. 이 기능은 R-트리 인덱스가 유효한지 확인하려는 DBA와 R-트리 인덱스를 디버깅 및/또는 튜닝하려는 인덱스 디자이너와 사용자 모두에게 꼭 필요합니다. oncheck 및 onlog 유틸리티는 이 기능을 지원하도록 확장되었습니다.

5. **ON-Bar 진행률 피드백** - 긴 작업 중에 ON-Bar는 ON-Bar 활동 로그로 출력된 비율을 저장합니다. 출력 빈도는 사용자가 설정한 onconfig 매개 변수 값으로 결정됩니다. 이 매개 변수가 설정되지 않거나 없으면 진행률은 출력되지 않습니다.

6. **ON-Bar 무시(-O)** - 이 기능으로 사용자는 내부 오류 검사를 무시하고 이전에는 허용되지 않은 작업을 수행할 수 있습니다. 명령줄 옵션 "-O"는 무시 기능을 활성화합니다. 이제 사용자는 DB영역이 다운될 때 로그 백업을 수행하고, 공간이 존재하지 않는 경우 복구하는 동안 공간을 만들고 온라인 공간의 복구를 수행할 수 있습니다.

7. **ON-Bar 객체 만료 및 동기화 도구** - 이 기능은 서로가 저장하고 있는 객체와 관련하여 동기화 시간 초과에서 발생되는 스토리지 관리자와 ON-Bar의 문제를 해결합니다. 또한 더 많은 백업 기록 정보가 sysutils 데이터베이스 및 응급 시동 파일에 저장될 때 발생할 수 있는 ON-Bar 성능 저하 문제 및 공간 문제도 해결합니다. onsmsync 유틸리티는 다음과 같은 작업을 지원합니다.

   a. 이전의 ON-Bar 백업 객체가 만료되게 합니다.

ii. 백업 기간을 기준으로 이전 백업 객체 제거

iii. 백업된 횟수를 기준으로 이전 백업 객체 제거

또한 onsmsync를 사용하여 손상된 응급 시동 파일을 다시 만들거나 sysutils 데이터베이스를 동기화할 수 있습니다. 온라인이나 대기 모드에 있는 데이터베이스 서버와 함께 onsmsync를 사용하여 sysutils 데이터베이스와 응급 시동 파일을 모두 동기화합니다. 데이터베이스 서버가 오프라인이면 onsmsync는 스토리지 관리자를 응급 시동 파일과 동기화합니다. sysutils 데이터베이스를 동기화하려면 데이터베이스 서버를 온라인으로 가져와서 onsmsync를 실행합니다. onsmsync 명령은 다음 작업을 수행하여 sysutils 데이터베이스, 스토리지 관리자 및 응급 시동 파일을 동기화합니다.

f. 응급 시동 파일에는 있지만 sysutils 데이터베이스에는 없는 백업 기록을 sysutils에 추가합니다.

g. sysutils 데이터베이스에서 전체 시스템 복원, 모조 백업(fake backup), 외부 백업 및 실패한 백업 또는 복원 레코드를 제거합니다.

h. 작성 횟수, 보존 날짜 또는 기간을 기준으로 전체 시스템 백업에 관한 레코드를 제거합니다.

i. 더 이상 필요하지 않은 이전 논리 로그가 만료되게 합니다.

j. sysutils 데이터베이스에서 응급 시동 파일을 다시 만듭니다.

스마트 대형 객체(SLOB) 기능

1. **스마트 대형 객체용 라이트 I/O** - 이 기능으로 고객들은 버퍼 풀을 무시하고 디스크에서 스마트 대형 객체 I/O를 직접 수행할 수 있습니다. 그렇기 때문에 버퍼링에서 이점을 얻지 못하는 스마트 대형 객체 I/O 작업의 성능이 크게 향상됩니다. 대용량으로 읽기가 수행되므로 일반적인 I/O 읽기보다 훨씬 더 빨라질 수 있지만 독자가 대용량으로 데이터를 요청하는 경우에만 해당됩니다.

2. **스마트 대형 객체용 메타데이터 압축** - 이 기능으로 이진 대형 객체(BLOB) 헤더 정보가 한 페이지의 메타데이터에서 스마트 대형 객체 당 450바이트로 압축됩니다.

3. **스마트 대형 객체용 점증적 보관(incremental backup) 지원** - 이전 릴리즈에서의 스마트 대형 객

체의 Level-1이나 Level-2 보관은 Level 0 보관으로 되돌려졌습니다. Level-0은 청크 프리 목록을 사용하여 사용 중인 익스텐트를 결정합니다. Level-1과 Level-2는 페이지 타임스탬프를 사용하여 보관할 내용을 결정하고, 더 낮은 숫자로 마지막 보관 이후에 수정된 페이지만 저장됩니다. 이제 Level-1이나 Level-2 보관은 지난 더 높은 수준의 보관 이후에 만들어지거나 수정된 스마트 대형 객체만 검토합니다. 따라서 보관 테이프는 더 적게 필요하고 보관은 더 빨라집니다.

4. **스마트 대형 객체용 Alter Table** - ALTER TABLE 문의 새로운 PUT 절을 사용하여 사용자는 새 칼럼의 스마트 대형 객체 스토리지 특성을 지정하거나 기존 칼럼의 스토리지 특성을 변경할 수 있습니다.

5. **스마트 대형 객체 라운드 로빈 분할** - CREATE/ALTER TABLE 문의 PUT 절에 있는 칼럼에 여러 개의 스마트 BLOB 영역(sbspaces)을 지정할 수 있습니다. 이 기능은 여러 개의 스토리지 장치 사이에 스마트 대형 객체를 배포하여 많은 부하가 걸릴 때 성능을 향상시킵니다. 이 기능으로 분할 테이블에 있는 모든 대형 객체의 방향을 하나의 SB영역으로 지정했을 때 발생하는 잠재적인 병목 현상을 완화하며, 동시에 특정 칼럼에 대한 대형 객체의 방향을 잘 정의된 SB영역의 집합으로 지정하는 융통성을 제공합니다.

6. **Byte/Text를 BLOB/CLOB으로 변환** - Byte/Text는 이진 대형 객체 및 문자 대형 객체(CLOB)로 변환할 수 있습니다.

7. **스마트 대형 객체용 바이트 범위 LOCK(Byte Range Locking)** - 이 기능은 여러 명의 작성자와 독자가 같은 대형 객체의 다른 바이트 범위에 동시에 액세스할 수 있게 하여 동시성을 향상시킵니다. 이 기능으로 트랜잭션은 현재처럼 전체 대량 객체에 lock을 걸지 않고 대량 객체에서 필요한 바이트 범위에만 lock을 걸 수 있습니다. 바이트 범위 LOCK 기능은 UNIX나 NT 파일 시스템에서 지원하는 것과 유사한 LOCK 기능을 제공합니다. 이 기능을 활용하는 응용 프로그램은 동시성이 현저하게 개선될 것입니다.

확장성 기능

1. **병렬 사용자 정의 루틴** - 현재의 Informix Dynamic Server 9.1x 릴리즈는 질의에서 사용자 정의 루틴(UDR)을 호출하는 경우 질의의 병렬 실행을 사용할 수 없습니다. Informix Dynamic Server 9.2에서는 이런 제한 사항이 제거되었고 병렬 데이터 질의(PDQ) 문에 대해 UDR 병렬화를 사용할 수 있습니다. 이러한 병렬화는 병렬화를 이용하도록 디자인된 질의의 성능을 향상시킵니다.

2. **병렬 사용자용 Pload 지원** - 정의된 루틴-병렬 로더(Pload)는 UDR의 병렬 실행을 이용하도록 확

9.21에서 소개된 기능

1. **Informix Dynamic Server 7.3 기능 패리티** - 이 기능은 9.20 릴리즈에서 구현되지 않은 7.3의 세 가지 기능을 9.2 코드 스트림으로 가져옵니다. 버전 9.21은 로그되지 않은 테이블, 인덱스 이름 변경 및 ANSI 외부 조인을 지원합니다.

2. **MaxConnectTM 지원** - MaxConnect는 Informix Server 환경을 위한 새로운 네트워킹 제품입니다. MaxConnect는 시스템 확장성을 높여 수천 개를 연결할 수 있습니다. MaxConnect를 배포하면 시스템 리소스가 절약되고 반응 시간과 DB Server CPU 소모가 줄어듭니다. 클라이언트/서버 환경에서 클라이언트 연결은 DB 서버의 시스템 리소스를 사용합니다. 수백 개의 클라이언트를 연결하면 서버 CPU의 매우 많은 부분을 사용하여, 사용자 반응 시간도 늘어나고 CPU 요구 사항도 많아집니다. MaxConnect는 DB 서버로의 훨씬 더 적은 수의 네트워크 연결 사이에 여러 클라이언트 세션을 멀티플렉싱하여, DB 서버 시스템에서 사용자 연결 관리를 이동시킵니다. DB 연결(멀티플렉싱 비율)로의 클라이언트 연결 비율은 100:1이 될 수 있으며, DB 서버의 운영 체제 요구 사항을 현저하게 줄이고 시스템 확장성은 수 만 개의 연결까지 늘릴 수 있습니다.

3. **SQL 문 캐시 확장** - 이 기능에는 고객 벤치마크 실습 중에 식별된 많은 확장이 포함되어 있습니다. 지정한 수만큼의 히트 후 캐시 삽입, 가변 기본 LRU의 수, 최대 캐시 크기에 도달한 후 삽입 사용 안 함, 더 효율적인 캐시 정리 알고리즘 등이 들어 있습니다.

4. **Direct 함수 R-트리 호출** - 세 개의 새로운 API가 액세스 방법 API에 추가되어 행 ID가 있는 행을 검색합니다. 이러한 API는 전체 다각형이 아닌 테두리가 있는 상자에 R-트리만을 구축할 때 필요합니다. R-트리 인덱스 스캔 도중에 R-트리는 이 API를 사용하여 자격을 갖춘 테두리가 있는 상자에 추가적인 필터링을 적용할 수 있도록 실제 데이터를 검색합니다.

5. **JDBC 2.0** - 이 기능은 JDBC의 클라이언트 버전으로 버전 패리티를 제공합니다.

6. **JVM 1.2** - 이 기능은 Informix Internet Foundation 패키지를 JVM 다음 버전으로 업그레이드합니다.

업그레이드 계획

1. **Migration Guide 읽기**

   먼저, Informix Dynamic Server 9.2x Migration Guide(Informix Migration Guide-December 1999, Part Number 000-6531, (http://www.Informix.com/answers/english/docs/gn8392/6531.pdf)를 읽어 보십시오. 성공적인 마이그레이션에 필요한 중요한 정보가 들어 있습니다. Migration Guide에 있는 단계를 순서대로 정확하게 수행해야 합니다.

2. **릴리즈 정보 읽기**

   Informix Dynamic Server 제품과 함께 제공되며, 설치 디렉터리의 "release"라는 하위 디렉터리에 있는 릴리즈 정보 또한 매우 중요합니다.

   특히 다음 릴리즈 정보는 중요한 핵심 정보를 제공합니다

릴리즈 정보	핵심 정보
IDS_9.2	각 시스템에 따라 - 설치해야 하는 OS 패치 - 권장하는 커널 매개 변수 튜닝 - 지원되는 기능과 지원되지 않는 기능 - 알려진 문제들과 그 해결책
SERVERS_9.2	9.21 제품용 릴리즈 정보 - 새로운 기능과 릴리즈 정보 또는 문서 참조 목록 - 9.21의 제한 사항 및 문제 해결책 - 역변환 문제 - Enterprise Replication으로 마이그레이션 및 역변환하기 - Enterprise Replication 정보
MIGRATEDOC_9.2	Migration Guide가 다루지 않거나 guide가 발행된 이후에 변경된 마이그레이션 문제 - 해결되지 않은 적절한 교체 테이블로 역변환 - Enterprise Replication으로 역변환 - 데이터를 로드하기 위해 로그되지 않는 원시 테이블 사용 - 역변환 후의 프로시저를 위해 통계 업데이트
ONBARDOC_9.2	HDR을 ISM과 ON-Bar와 함께 사용하는 프로시저

3.

4. **환경 및 구성 파일의 변경 사항 검토**

Informix Dynamic Server 9.2x에서 새로 도입되는 몇 가지 새로운 환경 변수가 있습니다. 또한 업그레이드 후 설정해야 하는 새로운 구성 파일 매개 변수도 몇 가지 있습니다. 역변환이 필요한 경우, 환경 및 구성 파일을 검토하여 새로운 매개 변수들이 설정되었는지 확인해야 합니다. 구성 파일 복사본은 Informix Dynamic Server 9.2x용과 이전 버전용으로 두 개를 갖고 있는 것이 좋습니다.

5. **분리된 새로운 인덱스 배치 기본값 검토**

Informix Dynamic Server 7.x에서 데이터 및 인덱스 페이지는 기본적으로 같은 테이블 익스텐트에 배치됩니다. Informix Dynamic Server 7.x에서는 CREATE INDEX 문의 "IN dbspaces" 절을 사용하여 인덱스를 명시적으로 분리할 수 있습니다. Informix Dynamic Server 9.2x에서 기본 행동은 인덱스 페이지를 테이블에서 분리된 익스텐트에 배치하는 것입니다. 이렇게 변경하면 계산에 인덱스 공간이 포함되는 경우 테이블 익스텐트 계산에 영향을 줄 수 있습니다. 테이블 익스텐트에서 인덱스용으로 할당된 공간이 인덱스에 사용되지 않고 인덱스 익스텐트를 위해 청크에서 추가 공간을 사용합니다. 따라서 상대적으로 가득 찬 청크에 테이블을 만들면 공간이 부족할 수 있습니

Informix Dynamic Server 9.2x에서 이전 버전으로 역변환하기 위한 단계

Informix Dynamic Server 9.2x 데이터베이스 서버에서 이전 버전으로 성공적으로 역변환하려면 다음의 이벤트 순서에 따라야 합니다. 자세한 내용은 Migration Guide를 참조하십시오.

- 역변환 가능 여부 확인(위의 역변환 제한 사항 참조)
- 데이터 무결성 확인
- Informix Dynamic Server 9.2x 기능 제거
- Informix Dynamic Server 9.2x 백업
- 역변환 명령(onmode -b) 실행
- 환경 조정

관리자(Premier Services Account Manager)가 업그레이드 계획 프로세스에 참여해야 합니다. 관리자는 값진 정보를 능동적으로 검색하고 제공해서 업그레이드 작업이 부드럽게 진행되도록 합니다.

- 7개 내지 9개의 시험용 프로그램

  사내에 Enterprise 지원 엔지니어나 Informix 컨설턴트가 있는 고객의 경우 Informix Dynamic Server 9.2x으로 업그레이드하는 복잡한 설치 과정을 도와드리기 위한 프로그램이 있습니다. 시험용 프로그램의 목적은 7.x에서 Informix Dynamic Server 9.2x으로 업그레이드하는 것의 용이성을 보여 주고 업그레이드 중에 발생하는 장애물을 제거하는 것입니다.

업그레이드 준비

1. **소프트웨어 수준 확인**

   OnLine 4.1x나 5.x 또는 Informix Dynamic Server 6.x나 7.1x에서 Informix Dynamic Server 9.2x로 업그레이드하기 전에 Informix Dynamic Server 7.2x나 7.3x로 업그레이드해야 합니다. 9.14 이전의 Informix Dynamic Server/Universal Data Option 버전이 있으면 Informix Dynamic Server 9.2x으로 업그레이드하기 전에 9.14로 업그레이드해야 합니다.

2. **충분한 논리 로그 및 DB영역 공간이 있는지 확인**

   성공적으로 업그레이드하기 위해 고려해야 할 디스크 공간 요구 사항이 있습니다. 필요한 디스크 공간은 모두 데이터베이스 서버를 종료하기 전에 추가해야 합니다. 디스크 공간 요구 사항은 다음과 같습니다.

   - 사용 가능한 논리 로그 공간 3000페이지
   - 해당 DB영역에 포함된 각 데이터베이스의 DB영역 당 2MB. 예를 들어 세 개의 데이터베이스를 포함하는 DB영역에는 6MB의 사용 가능한 공간이 필요합니다.
   - Informix Dynamic Server 7.2에서 업그레이드하는 경우-DB 영역 페이지의 DB영역 설명자 같은 루트 예약 페이지 구조의 일부는 버전 7.2에서 보다 Informix Dynamic Server

9.2x에서 더 큽니다. 따라서 마이그레이션 과정의 일부로서 이 구조를 최신 형식으로 변환하는 내부적인 변환 과정이 있으며, 확장된 예약 페이지를 만들거나 더 많은 확장된 페이지가 필요합니다. 확장된 예약 페이지가 루트 청크(청크 번호 1)에만 할당될 수 있는 경우라면 루트 청크에 새 구조를 수용할 수 있는 공간이 있는지 확인하십시오.

3. **Informix Dynamic Server 7.2에서 업그레이드하는 경우 - 충분한 InformixDIR 파일 시스템 공간이 있는지 확인**

   Informix Dynamic Server 7.2x에서 업그레이드하는 경우에는 Informix Dynamic Server 7.2x 설치 보다 Informix Dynamic Server 9.2x 설치에 더 많은 공간이 필요하다는 사실을 알아야 합니다.

   Informix Dynamic Server 9.2x을 설치할 파일 시스템에 디스크 공간이 충분한지 확인하십시오.

   Sun Solaris용 Informix Dynamic Server 7.24를 설치하려면 약 120MB의 파일 시스템 공간이 필요하지만 Informix Dynamic Server 9.2x의 경우에는 약 180MB의 파일 시스템 공간이 필요합니다.

4. **Enterprise Replication 지원에 필요한 클라이언트 소프트웨어 개발자 키트(InformixClient SDKTM) 또는 Informix Connect**

   ER (CDR) 명령줄 도구는 ESQL/C를 사용하여 만들어졌으므로 사용하려면 InformixClient SDK 또는 Informix Connect를 설치해야 합니다. 따라서 Enterprise Replication 및 관련 명령줄 기능을 사용하려는 경우에는 InformixClient SDK™ 또는 Informix Connect가 Informix Dynamic Server 9.2x 제품과 함께 설치되어 있는지 확인하십시오.

소개

이 글은 Tech Notes 지난 호(Vol10)에 실린 "객체 관계형 데이터베이스 응용 프로그램 개발" 내용의 2부이며, 응용 프로그램 구현을 중점적으로 다룹니다. Tech Notes의 지난 호(Vol10)에 이 글의 1부가 실려 있습니다.

초기 구현

1부의 끝 부분에서는 객체 관계형 데이터베이스 응용 프로그램을 구축하여 지원해야 하는 문제 영역(problem domain)의 개념적 모델을 구축하는 방법에 대해 살펴보았습니다. 이 개념적 모델은 의미상(semantic)의 데이터 모델(확장된 ER 또는 UML 다이어그램)과 해당 스키마 내에서 사용되는 영역(또는 객체 클래스) 각각의 구조 및 행동을 설명하는 정의 모음으로 구성됩니다. 2부에서는 이 분석의 결과를 작업 소프트웨어로 변경하는 단계에 대해 설명합니다.

간단하게 말하면 개념적 구상을 효율적인 코드로 변환하는 문제입니다. 그러기 위해서 두 단계의 과정이 필요합니다. 먼저 정확하고 완벽한 코드 본문을 디자인하려면 이전 분석의 결과를 사용해야 합니다. 다음으로 이 코드를 분석하여 기술적인 디자인과 하드웨어가 시스템의 기능, 성능 및 안정성을 충족시키는지 확인해야 합니다. 이러한 것을 충족시키지 못한다면 디자인을 조정하거나 아키텍처를 조정하여 목표를 달성해야 합니다.

분석 마비(Analysis Paralysis) 현상에 대한 설명

더 진행하기 전에 주의할 점이 있습니다. 소프트웨어 개발자에게 중요한 규칙은 분석 작업을 적게 하면 일이 어려워지지만 너무 많이 하면 아무 것도 이루지 못한다는 것입니다. 너무 포괄적인 개념적 모델을 구축하려고 하면 대부분은 쓸모 없는 연습이 되어 버립니다.

첫 번째로, 어떤 주제를 오랫동안 연구하면 거의 예외 없이 그 주제에 대해 배울 것이 무한하다는 것을 발견하게 됩니다. 지식 자체가 짐이 될 수는 없지만 대부분의 정보 시스템 구축은 시간과 자료 면에서 제한을 받게 되므로 충분히 연구할 시간적 여유가 없습니다. 코딩을 시작하기 전에 모든 내용에 대해 이해하려고 한다면 프로젝트는 아마 이미 끝나 있을 것입니다.

두 번째로, 문제 영역이 변한다는 것입니다. 얼마 전에는 참이었던 내용이 일주일 후에는 거짓이 될 수 있습니

다. 사용자는 결정을 내리지 못하고, 생각을 바꾸며, 사용자들끼리 논쟁을 하고, 심지어 노골적인 거짓말까지 하게 됩니다. 현실에 맞춰서 여러분의 계획을 평가한다는 것은 매우 어려운 일입니다. 계속 변하는 요구 사항을 모두 만족시키기 위해 노력하면 할수록 결과는 공상이 되어 버리고 계속 수정만 하게 됩니다.

감당할 수 없을 만큼 빠르게 변하는 문제에 대해 분석가들이 끝없이 연구하는 현상을 뜻하는 "분석 마비" 현상은 몇몇 프로젝트에서 직면하는 상황입니다.

Don Knuth의 유명한 말처럼, 컴퓨터 프로그램을 만드는 일은 컴퓨터 내부에 현실 세계의 모델을 구축하는 것과 같습니다. 모델은 추상적인 개념으로, 중요하거나 관계된 측면에 초점을 맞추어 복잡한 현상을 단순화하는 한편 그 나머지 부분은 무시합니다. 중요한 것은 바로 집중하고 있는 문제의 부분이 올바른 부분이며 여러분의 모델이 적절하게 문제를 설명하고 있는가 하는 것입니다.

따라서 분석은 최소화하고 즉각적으로 시스템의 초기 버전을 구축하는 것이 좋습니다. 그런 다음 소프트웨어를 실제와 비교합니다. 컴퓨터 프로그램과 현실 세계가 작동하는 방식 사이의 차이점을 찾아내기는 쉬우며, 이를 통해 모델을 정교하게 수정할 수 있습니다. 그 밖에, 성공적인 소프트웨어를 만드는 궁극적인 수단은 유틸리티입니다. 결함을 갖고 운영되는 시스템은 수정할 수 있고 개선할 수 있습니다.

개념적 구상을 수정하는 일은 꼭 필요하며 유용합니다. 그러나 궁극적으로 최종 사용자에게 도움이 되지는 않습니다.

디자인 방법론 개요

개념적 모델을 데이터베이스 구현으로 변환하는 단계는 다음과 같습니다.

1. 영역/객체 분석을 기반으로 형식 시스템(type system)을 구현합니다. 사용자 정의 형식 및 사용자 정의 함수 메커니즘을 사용합니다.
2. 형식 시스템을 사용하여 문제 영역의 상태를 설명하는 사실(facts)을 기록하는 초기 관계형 디자인을 만듭니다. 여기에는 "create table" 문을 작성하는 일이 포함됩니다.
3. 스키마를 일반화하여 모호한 부분이 없도록 합니다. 이렇게 하면 테이블 디자인을 조정할 수 있습니다.
4. 생산 스케일(production scale) 데이터로 스키마를 채우고 해당 데이터베이스에 대해 일련의 워크로드 질의를 실시하여 성능을 측정합니다. 이것은 데이터베이스 스키마를 기반으로 작동하는 DML 질의 작성과 관련이

유기적 정보 시스템에 대한 설명

이 글의 1부에서는 개발자가 정보 시스템을 좀 더 전체적으로 보면 객체 관계형 DBMS 제품을 최대한 활용할 수 있다는 것을 설명했습니다. 객체 관계형 또는 확장 가능한 데이터베이스 관리 시스템을 사용하면 개발자는 정보 시스템 논리의 상당 부분을 데이터 관리 프레임워크 내에 포함시킬 수 있으며 선언 질의어를 사용하여 논리와 데이터를 조합할 수 있습니다.

그러나 ORDBMS에는 또 다른 장점이 있습니다. 개발자는 새로운 기능을 추가할 수 있으며 기존 논리를 변경하거나 운영 중인 시스템을 중단시키지 않고 업그레이드할 수 있습니다. 이는 정보 시스템이 관리되는 방식을 바꿉니다. 일반적으로 소프트웨어 업그레이드는 정기적으로 릴리스됩니다. 이것은 컴파일된 모듈이 실행 파일에 연결될 때 그 사이의 참조를 해결하는 전통적인 프로그래밍 언어 기술의 방식 때문입니다. 그러나 ORDBMS에서는 연결이 점진적으로 이루어지므로 훨씬 더 융통성이 있습니다.

이러한 작업의 효과는 ORDBMS를 플랫폼으로 사용하여 구축한 정보 시스템이 그 개발 관점에서 더 유기적이라는 점입니다. 오류 및 확장은 실행 중인 시스템 내에 개별적인 모듈을 추가하거나 변경하여 해결할 수 있습니다. ORDBMS를 소프트웨어 스택의 맨 아래로 격하된 데이터 저장소 이상의 것으로 본다면 사실상 더 건설적인 작업을 할 수 있습니다.

오히려, 단일 프레임워크 내에 논리와 데이터를 융통성있게 조합하는 ORDBMS의 기능은 전통적인 DBMS나 미들웨어 소프트웨어와 구분됩니다. 유기적이고 진화적인 원리를 사용하는 정보 시스템을 배포하는 것의 장점은 매일 계속 변경되는 최종 사용자의 요구에 더 잘 대응할 수 있다는 것입니다.

맺음말

엔티티-관계형 다이어그램, 일반화 이론 및 물리적 튜닝 같이 관계형 DBMS에서 사용되는 방법론 및 기법은 모두 객체 관계형 DBMS에 똑같이 적용됩니다. 그러나 RDBMS 제품과 달리 ORDBMS는 개발자들이 사용자 정의 형식 및 사용자 정의 함수를 사용하여 객체를 모델링할 수 있으므로 더 좋은 데이터 모델링을 할 수 있습니다. 개념적 분석의 결과물은 문제 영역의 개념적 보기를 나타내는 일련의 다이어그램과 다이어그램 내부에 있는 객체의 구조 및 행동입니다.

마지막인 2부에서는 이런 다이어그램을 작업 코드로 변환하는 기법을 살펴보았습니다. 먼저, 개념적 분석 단계에서 식별된 영역이나 객체의 집합에 상응하는 사용자 정의 형식과 함수를 개발합니다. 그런 다음, 새로운 형식이 테이블의 칼럼으로 사용되는 제한 조건과 테이블의 집합인 데이터베이스 스키마로 이러한 객체를 조합합니다. 데이터베이스 스키마에 오류의 가능성이 포함되지 않게 하는 데 필요한 일반화 이론을 객체 관계형 데이터베이스에 적용할 수 있습니다.

마지막으로 좀 더 새로운 객체 관계형 DBMS 기술을 설명했습니다. 소프트웨어 개발에 있어 가장 큰 문제는 최종 사용자, 고객 또는 소비자의 요구에 빠르게 응답하는 것입니다. 웹 속도로 이런 요구에 적응하려면 성능, 확장 가능성 및 안정성을 융통성 및 현실 세계를 모델링하는 향상된 기능과 조합하는 방식처럼 서버 소프트웨어에 대한 새로운 접근 방식이 필요합니다. 이런 종류의 유기적인 정보 시스템을 지원하는 것이 바로 객체 관계형 기술의 핵심입니다.

Checkpoint 튜닝: 사례 연구

소개

온라인 트랜잭션 프로세스(OLTP) 시스템에서 checkpoint는 시스템 오류가 발생했을 경우 복구 지점을 제공합니다. 따라서 checkpoint 튜닝은 중요한 작업입니다.

Checkpoint의 목적은 실제적으로 복구를 위한 일관성 있는 지점을 제공하기 위해 수정된 모든 페이지를 공유 메모리 버퍼 캐시에서 디스크로 쓰는 것입니다. Checkpoint가 발생하는 방법을 결정할 때는 두 가지 시간 값을 고려해야 합니다.

첫 번째는 checkpoint 간격으로, checkpoint 사이의 시간을 나타냅니다. 이 시간 값은 오류가 발생했을 경우 복구에 필요한 시간과 관련되어 있다는 점에서 중요합니다. Checkpoint 사이의 간격이 길수록 복구 시간도 길어집니다.

고려해야 할 두 번째 시간 값은 checkpoint 지속 시간입니다. Checkpoint 중에는 수정된 모든 페이지를 버퍼 캐시에서 디스크로 씁니다. Checkpoint 시간의 대부분은 이 작업을 수행하는 데 사용됩니다. Checkpoint 도중에는 코드의 critical section(디스크에 쓰려고 하는 공유 메모리 버퍼 캐시의 수정이 필요한)을 실행할 수 없습니다. 따라서 checkpoint 도중에는 공유 메모리 버퍼 캐시를 수정할 수 없습니다. 데이터베이스 사용자는 이런 현상을 일시 중지 또는 중단으로 생각하므로 checkpoint 지속 기간을 최소로 유지해야 합니다.

이 글에서는 checkpoint 간격과 checkpoint 지속 시간을 결정하는 매개변수를 검토하고 checkpoint 튜닝 관련 예와 실습 내용을 제공합니다.

Checkpoint 간격 결정

Checkpoint 간격은 다음과 같은 매개변수로 결정됩니다.

- 실제 checkpoint 간격
- 물리 로그의 크기

Checkpoint 간격은 급속 복구(fast recovery)의 복구 시간과 관계가 있음을 기억하십시오. 피크 작업 중에는 오류가 발생할 경우 더 많은 복구 작업을 수행해야 하므로 간격이 더 작아도 문제가 없습니다.

물리 로그 파일이 사용되는 양을 모니터하려면 onstat -l을 사용하고 물리 로그 파일 라인을 참조합니다. Checkpoint에 근접한 시간에 사용된 페이지 수가 물리 로그 파일에 있는 페이지의 75퍼센트에 도달하지 못하면 물리 로그 파일 크기를 줄이십시오. 불필요한 디스크 공간을 소모하고 있는 것입니다.

Checkpoint 지속 시간 결정

Checkpoint 지속 시간은 다음과 같은 매개변수로 결정됩니다.

■ 디스크에 써야 하는 수정된 페이지 수

- Checkpoint 사이의 LRU writes 수
- 응용 프로그램의 특성

■ 디스크에 해당 페이지를 쓰는 데 필요한 입출력양

- 사용 중인 드라이브 수
- 쓰기의 임의 특성

■ 하드웨어에 대한 데이터베이스 엔진의 요청 속도

- 페이지 클리너 수
- AIO VP 수

모든 checkpoint의 지속 시간은 메시지 로그 파일에 기록됩니다. 메시지 로그 파일에서 각 checkpoint의 지속 시간을 찾을 수 있습니다. 로그 파일을 모니터하여 대부분의 checkpoint가 적절한 시간(일반적으로 5초 이내)에 완료되는지 확인하십시오. 적절한 시간에 완료되지 않으면 읽기를 계속합니다.

Checkpoint 지속 시간 병목 튜닝

디스크 드라이브에서 데이터베이스 엔진으로 역방향 작업을 하면 예상되는 checkpoint 지속 시간 병목은 다음과 같습니다.

■ 디스크 장치

■ 디스크 컨트롤러

■ CPU

■ AIO VP

■ CPU VP

■ 페이지 클리너

■ LRU 큐 수

■ 버퍼 캐시의 수정된 페이지 수

디스크 드라이브는 **sar, iostat** 및 **glance**를 비롯한 여러 가지 UNIX 유틸리티로 모니터할 수 있습니다. 데이터를 쓰려고 하는 디스크 드라이브가 이미 100퍼센트 사용 중이면 다음과 같은 두 가지 작업을 수행할 수 있습니다.

■ 데이터베이스 재구성의 경우처럼 사용할 수 있는 디스크에 데이터를 더 균등하게 분산시킵니다.

■ 더 많은 디스크를 구입하고 디스크 입출력을 재구성합니다.

위 두 경우가 모두 여의치 않으면, 성능이 좋지 않은 상태로 그대로 사용할 수도 있습니다.

디스크 컨트롤러는 디스크처럼 작동합니다. 컨트롤러 사용이 최대가 되면 더 이상 컨트롤러를 통해 데이터를 얻을 수 없습니다. 입출력을 여러 컨트롤러에 분산시키거나, 필요할 경우 추가로 구입하십시오.
물리적인 CPU는 sar, vmstat 또는 glance를 사용하여 모니터할 수 있습니다. CPU가 항상 100퍼센트 사용 중이면 디스크에서 데이터를 더 이상 빠르게 얻을 수 없습니다. 그러므로, 디스크에서 데이터를 더 빨리 가져오기 위해서는 클라이언트 시스템의 응용 프로그램 로드를 해제하여 경쟁을 줄이거나 CPU 수를 늘려야 합니다. 아니면 더 빠른 CPU를 추가해야 합니다. 하드웨어가 100퍼센트 사용 중이 아니라면 Informix 엔진을 튜닝하여 성능을 더 좋게 만들 수 있습니다.

다음으로 고려할 부분은 AIO VP입니다. 각 AIO VP가 현재 엔진 내에서의 요청을 파악하여 실제로 디스크 쓰기를 수행하고 있는지 확인해야 합니다. onstat -g ioq를 모니터하고 큐 길이 및 AIO VP 및 GFD (GFD는 청크에 대한 open file handler입니다.)의 최대 큐 길이를 살핍니다. 큐 중에 최대 길이(>25)나 일관된 길이

(>10)가 있으면 입출력 요청은 필요한 만큼 빠르게 발생하지 않습니다. 하드웨어가 포화 상태가 아니라면 AIO VP(커널 AIO가 사용 중인 경우는 CPU VP)를 더 추가할 경우 한 번에 처리할 수 있는 요청 수는 늘어나고 큐 길이는 줄어듭니다.

AIO 큐가 계속 늘어나고 성능이 여전히 좋지 않으면 요청이 필요한 만큼 빨리 큐에 도달하지 못합니다. 큐에 대한 요청을 담당하는 페이지 클리너를 더 많이 추가하면 더 빠르게 요청을 전달할 수 있습니다. 페이지 클리너의 상태는 onstat -F로 확인할 수 있습니다. 페이지 클리너는 활성화되어 있는데 요청과 더불어 큐가 활성화되지 않으면 페이지 클리너를 더 추가하십시오.

LRU 큐의 수도 고려해야 합니다. 하나의 CPU VP 당 최소 하나의 LRU 큐가 있는지 확인하십시오. 버퍼 캐시를 변경하기 위해 CPU VP는 현재 액세스하고 있는 LRU 큐에 뮤텍스를 넣습니다. CPU VP보다 큐의 수가 더 적으면 경쟁 문제가 발생할 수 있습니다. 버퍼 캐시가 커서 LRU 큐가 더 많아지면 각 큐의 길이는 더 짧아집니다. 각 큐의 길이가 짧아지게 하는 것도 성능을 향상시킬 수 있습니다.

결국 checkpoint의 지속 기간은 checkpoint 동안 디스크에 써야 할 수정된 페이지 수를 기반으로 합니다. Checkpoint 지속 시간을 줄이는 가장 좋은 방법은 디스크에 쓸 수정된 페이지 수를 줄이는 것입니다. 그러나, checkpoint 지속 시간을 줄이기 위해 최종 사용자에게 현재 변경하는 내용의 10퍼센트 정도만 수정하도록 하는 방법으로 수정 페이지를 줄이는 것은 해결책이 될 수 없습니다. 또한 버퍼 양을 제한하는 것은 읽기 캐시율에도 심한 영향을 끼칠 수가 있습니다. 결과적으로 시스템 성능의 다른 부분을 저하시킬 것입니다. 그러므로, 이 문제를 처리하기 위해서는 다른 방법을 간구해야 합니다.

임의 checkpoint 쓰기

OLTP 시스템의 일부 매개변수를 살펴 보도록 하겠습니다.

- 응답 시간은 2초 이하로 짧아야 합니다.
- 색인을 통해 데이터에 임의로 액세스합니다.
- 사용자의 요청에 리턴되는 행의 수가 적습니다.
- 데이터는 수정(삽입, 업데이트, 삭제)됩니다.
- Checkpoint 지속 시간은 5초 이하로 짧아야 합니다.

■ 급속 복구 시간은 10분에서 20분 정도로 짧아야 합니다.

■ 버퍼 캐쉬율은 95퍼센트 이상이어야 합니다.

위의 매개변수가 허용된다면, 현저하게 성능을 저하시키지 않고도 삽입, 업데이트 및 삭제를 위해 페이지를 메모리로 가져오기 위해 필요한 임의 읽기(random reads) 사이에서 임의 쓰기(random writes)를 실행할 수 있을 것입니다. 사용자가 임의 읽기를 실행 중이면 몇몇 임의 쓰기는 응답 시간의 속도를 현저하게 저하시키지 않습니다. 전체적으로는 checkpoint 시간에 쓰기를 하는 것이 시스템에는 더 효율적이지만 긴 checkpoint 지속 시간은 최종 사용자에게는 심각한 문제가 됩니다.

Checkpoint 튜닝은 사용자가 인식하는 시스템 성능과 전체적인 시스템 성능의 차이를 보여 주는 좋은 예입니다.

Checkpoint 사이에 디스크에 수정된 버퍼의 쓰기를 수행할 수 있도록 설정하는 두 가지 매개변수가 있습니다. LRU_MIN_DIRTY와 LRU_MAX_DIRTY가 그것입니다. 이 매개변수들은 다음과 같은 비유를 들어 설명할 수 있습니다.

우리가 어렸을 때 우리 방은 항상 어질러져 있었다는 사실을 기억할 수 있을 것입니다. 방이 어느 정도 어지러워지면 "어떻게 이런 돼지 우리 같은 데서 살 수가 있니? 이 방을 치울 때까지는 아무 데도 갈 수 없어!"라고 말씀하시던 어머니의 모습 또한 기억하실 겁니다. 그러면 여러분들은 "그렇게 지저분하지도 않은데"라고 중얼거리면서 청소를 했을 것입니다.

하지만 우리가 청소를 하는 데는 일정한 한계가 있었습니다. 우리는 방을 어느 정도까지만 청소하면 되는지를 알고 있었습니다. 아이들은 바로 그 시점에서 청소를 멈추었습니다. 그래서 옷들은 서랍과 벽장으로 치워 버리지만 바닥에 흩어져 있는 몇 가지의 장난감들은 내버려 둡니다. 그런 다음, 다시 잠들어 버리고 경우에 따라 재미있는 놀이를 찾아 보았을 것입니다.

청소해야 하는 어질러진 것들의 양(수정된 버퍼 풀 페이지)은 LRU_MAX_ DIRTY 매개변수로 제어됩니다.

LRU_MIN_DIRTY 매개변수는 버퍼를 어느 정도까지 깨끗하게 청소해야 하는 지를 제어합니다. 페이지 클리너(아이들)는 쓰기 요청을 담당하는 스레드이며, LRU_MAX_DIRTY 매개변수에 도달해서 CPU VP(어머니)가 깨울 때까지 잠을 잡니다.

그런 다음 페이지 클리너는 LRU_MIN_DIRTY에 도달할 때까지 수정된 페이지를 디스크에 쓰면서(옷과 장난감을 치우는 과정) 버퍼(방)를 청소하고 다시 잠을 잡니다.

OLTP 시스템에 대한 checkpoint 튜닝 예

Checkpoint 문제의 예를 살펴 보고 위의 정보를 이용하여 문제를 해결할 수 있는 방법을 확인해 봅시다. 100명의 사용자가 동시에 사용하는 OLTP 시스템에서 checkpoint 지속 시간 문제가 발생했습니다. 현재 5분마다 checkpoint를 실행하고 있으며 완료하는 데 30초가 걸립니다. 메시지 로그 파일을 확인하여 이 정보를 얻을 수 있습니다.

먼저 다음과 같은 계산을 합니다.

(60분/시간) / (1 checkpoint/5분) = 12 checkpoint/시간

12 checkpoint/시간 * 30초 지속 시간 = 6분/checkpoint 지속 시간

이 시간은 데이터베이스 엔진이 10%의 유휴시간을 가지는 것과 동일합니다. 즉, 매 시간 페이지를 버퍼 캐시에서 디스크로 쓰는 데 6분을 낭비하는 것입니다. 이 시간 동안, 사용자는 많은 작업을 할 수 없습니다. 8시간의 근무 시간 동안 각 사용자가 시스템에서 대기하는 시간은 48분이며, 이는 100명이 일하는 이 시스템에서 하루에 80시간을 손실하는 것과 같습니다. 인사부에 이 사실을 설명해 보십시오.

Informix의 입장에서 이 문제를 분석해 보도록 하겠습니다. 먼저 onstat -F를 사용하여 수행되는 쓰기 유형을 결정합니다. 청크, LRU 또는 포그라운드 쓰기가 가장 많이 발생하는 것처럼 표시될 것입니다. 활성 OLTP 시스템에서는 청크 쓰기보다 LRU 쓰기를 더 많이 수행하는 것이 가장 좋습니다. 다음과 같이 출력될 것입니다.

Fg Writes LRU Writes Chunk Writes

0 0 2189

address flusher state data

c0408444 0 I 0 = 0X0

states: Exit Idle Chunk Lru

다음은 LRU_MIN_DIRTY 및 LRU_MAX_DIRTY 매개변수의 설정 값을 결정하는 단계입니다. onconfig 파일을 검토하거나 onstat -c를 사용하여 그 값을 찾아 봅시다.

LRUS 8 # Number of LRU queues

LRU_MAX_DIRTY 60 # LRU percent dirty begin cleaning limit

LRU_MIN_DIRTY 50 # LRU percent dirty end cleaning limit

위의 매개변수는 OLTP 시스템의 경우라면 높아 보입니다(아래의 "DSS 시스템에 대한 checkpoint 튜닝" 절 참고). onstat -b 또는 onstat -R 옵션으로 checkpoint 시간이나 근접한 시간에 시스템을 관찰하십시오. 두 출력은 모두 버퍼 캐시에 있는 수정된 페이지와 사용하지 않은 페이지 수를 비교해서 나타냅니다. Checkpoint가 발생하면 헤더 정보에는 CKPT REQ가 표시됩니다. 시스템에서 수정된 페이지의 평균 수를 퍼센트 단위로 구하려면 일정 시간 이상 이 정보를 관찰해야 합니다.

수정된 페이지의 비율이 LRU_MAX_DIRTY 비율보다 낮으면 엔진은 LRU 쓰기를 수행하고 있지 않은 것입니다. 모든 쓰기가 checkpoint 시간에 발생하고 있는 것입니다.

onstat -R 출력:

8 buffer LRU queue pairs

f/m length % of pair total

0 F 153 83.2% 184

1 m 31 16.8%

2 f 148 81.3% 182

3 m 34 18.7%

4 f 152 81.3% 187

5 m 35 18.7%

6 f 146 79.3% 184

7 m 38 20.7%

8 f 151 82.5% 183

9 m 32 17.5%

10 f 144 79.1% 182

11 m 38 20.9%

12 f 145 78.8% 184

13 m 39 21.2%

14 f 150 80.6% 186

15 m 36 19.4%

283 dirty, 1472 queued, 1500 total, 2048 hash buckets,

2048 buffer size start clean at 60% (of pair total) dirty,

or 112 buffs dirty, stop at 50%

Checkpoint 지속 시간을 줄이려면 checkpoint 시간에 발생하는 쓰기 수를 줄여야 합니다. Checkpoint에 근접한 시간에 LRU 큐가 20퍼센트 사용되고 있다는 것을 알 수 있습니다. LRU_MAX_DIRTY 매개변수를 10퍼센트로 변경하고 LRU_MIN_DIRTY 매개변수를 5퍼센트로 변경하면 checkpoint 시간에 쓸 페이지 수의 절반이 checkpoint 사이에서 발생하는 쓰기를 확인할 수 있습니다. onstat -b 또는 onstat -R을 사용하여 매개변수의 값을 변경하고 checkpoint 시간의 버퍼 캐시를 다시 관찰하십시오.

수정된 페이지 수가 LRU_MAX_DIRTY 매개변수보다 크면 다른 영역을 확인해야 합니다. 다음과 같은 사항을 확인해야 합니다.

■ sar 또는 iostat 으로 디스크 드라이브가 포화 상태가 아닌지 여부

■ sar 로 CPU 가 포화 상태가 아닌지 여부

■ onstat -g ioq 로 AIO VP 가 포화 상태가 아닌지 여부

■ 페이지 클리너가 충분한지 여부

■ 다른 영역에 문제가 없으면 페이지 클리너를 더 추가합니다.

수정된 버퍼 수가 LRU_MAX_DIRTY 보다 작거나 비슷하면 checkpoint 의 지속 시간을 다시 확인합니다. 현재 변경하기 전 페이지 수의 절반 정도를 쓰고 있으면 checkpoint 지속 시간도 거의 그 길이의 절반이

되어야 합니다. 이 예에서는 30 초에서 15 초로 되어야 합니다.

이제 checkpoint 간격을 살펴 봅시다. checkpoint 사이의 간격은 복구 시간 계산에 사용됩니다. 간격이 길수록 정전 등으로 인한 복구에 걸리는 시간이 길어질 수 있습니다. 5 분도 너무 깁니다. 그 시간을 15 분까지 늘려 봅시다. 이 시간이 급속 복구에는 적합한 시간이지만 checkpoint 시간에도 적합한지 확인해 봅시다.

Checkpoint 간격: 15분(시간 당 네 번)

Checkpoint 지속 시간: 15초

경과된 전체 시간: 1분/시간(원래 10퍼센트 기준으로 1.6퍼센트 유휴 시간)

LRU_MAX_DIRTY 및 LRU_MIN_DIRTY 매개변수를 더 낮게 하여 checkpoint에 지속 시간을 훨씬 더 적게 할당할 수 있습니다. LRU_MAX_DIRTY를 2퍼센트로 낮추고 LRU_MIN_DIRTY를 1퍼센트로 낮추면 checkpoint 지속 시간이 최소값이 됩니다. 그러나 checkpoint 지속 시간을 2초보다 작게 하려면 페이지 클리너, LRU 큐, AIO VP, 드라이브 컨트롤러 및 물리 디스크 드라이브가 충분하게 있어야 합니다.

Checkpoint 간격: 15분(시간 당 네 번)

Checkpoint 지속 시간: 2초

경과된 전체 시간: 8초/시간(원래 10퍼센트 기준 0.2퍼센트 유휴 시간)

차이점은 LRU_MAX_DIRTY 및 LRU_MIN_DIRTY 매개변수가 줄어들면 전체적인 입출력 효율성이 감소한다는 점입니다. 그러나 사용자의 특정 응용 프로그램 및 구성을 기반으로 한 성능의 향상만큼 checkpoint 지속 시간도 계속 향상되어야 합니다. 결과적으로 줄어든 입출력 비율과 checkpoint 사이의 사용자 가동 시간 사이에 정확하게 균형이 맞도록 보완해야 합니다.

사용자의 느낌도 중요한 요인입니다. 전체적으로 시스템에서는 같은 양의 작업을 수행하거나 약간 더 많은 작업을 수행하지만 사용자가 느끼는 동결된 시간이 더 적으면 사용자들은 만족합니다. 사용자가 느끼는 것이 좋다면 시스템은 튜닝한 것이나 다름없습니다.

DSS 시스템에 대한 checkpoint 튜닝

의사 결정 지원 시스템(DSS) checkpoint 튜팅은 OLTP checkpoint 튜닝보다 좀 더 간단합니다. DSS 시스템은 다음과 같은 특징이 있습니다.

■ 데이터에 대한 액세스가 순차적입니다.

■ 일반적으로 데이터는 읽기 전용입니다.

■ 일반적으로 테이블의 모든 행이나 대부분의 행에 대해 읽기를 수행합니다.

■ 데이터를 거의 수정하지 않으므로 급속 복구 시간은 그리 중요하지 않습니다.

■ 데이터에 대한 수정은 거의 없고, 대부분의 시간동안 질의가 실행되므로 일반적으로 checkpoint 지속 시간은 중요한 요인이 아닙니다.

DSS 시스템으로 디스크 드라이브 헤드는 읽기를 계속하고 읽기에 방해가 되는 쓰기가 발생하지 않게 하려고 합니다. 따라서 청크 쓰기를 최대화하고 checkpoint 횟수를 최소화해야 합니다. 매 시간마다 checkpoint 가 발생하게 하는 것이 좋은 간격이 될 수 있습니다. 모든 쓰기가 checkpoint 시간에 발생하게 하려면 LRU_MIN_DIRTY 와 LRU_MAX_DIRTY 매개변수를 높게(각각 50 과 60 정도) 설정해야 합니다.

색인 작성, 데이터 로드 또는 다량 업데이트나 삭제 같은 중요한 시스템 이벤트 후에는 반드시 수동으로 checkpoint 를 실행해야 합니다. 이렇게 하면 이러한 이벤트 후 오류가 발생했을 때 확장되는 복구 시간을 방지할 수 있습니다.

checkpoint 튜닝 실습

다음은 Wisconsin 벤치마크 데이터베이스 및 데이터를 사용하여 성능 튜닝을 하기 위한 실습 내용입니다.

요구 사항

■ Informix Dynamic Server™ 7. x

■ dbspaces 에 필요한 디스크 공간 50MB

■ 공유 메모리 20MB

■ 공유 메모리를 통한 다섯 명의 사용자 연결

■ 실습을 실행하는 프로그램 및 스크립트:

wisc.sql - 색인으로 데이터베이스 만드는 데 사용

load_data - 적절한 데이터 양을 로드하는 데 사용(약 16MB)

start_all - 시뮬레이트된 다섯 명의 OLTP 사용자를 시작하는 데 사용

kill_all - 모든 사용자를 중지시키고 실행 통계를 제공하는 데 사용

oltp.ec - OLTP 사용자를 시뮬레이트하는 ESQL/C 프로그램

load_tab.ec - 데이터를 발생시키고 로드하는 ESQL/C 프로그램

compile - ESQL/C 프로그램을 컴파일하는 데 사용

*) 위의 파일들이 필요하신 분은 Informix 고객지원부(02-3488-2580)로 연락을 주십시오.

기본 매개변수를 사용하여 온라인 엔진을 만듭니다(onconfig.std 복사). rootdbs 경로를 변경하고 시스템을 초기화합니다. dbaccess를 통해 wisc.sql을 실행하여 Wisconsin 데이터베이스를 만듭니다.

컴파일을 실행하여 oltp.ec와 load_tab.ec 프로그램을 컴파일합니다. load_data를 실행하여 시뮬레이션할 모든 데이터를 로드합니다.

start_all을 실행합니다. 이렇게 하면 시뮬레이션이 시작됩니다. 한 번에 10분 간(checkpoint 두 개) 시스템을 모니터한 다음 kill_all로 중지시킵니다. kill_all의 통계를 기록합니다. 이것은 실행 중 경과된 시간과 oltp.ec 프로그램에서 사용자가 실행할 수 있었던 루프 수가 됩니다.

시뮬레이션 중에는 다음 사항을 관찰합니다.

■ 읽기 캐시 퍼센트

■ 사용된 버퍼 퍼센트

■ 쓰기 유형

■ 페이지 클리너 활동

■ 물리 로그 이용

■ Checkpoint 간격

■ Checkpoint 지속 시간

그리고 다음과 같은 매개변수를 튜닝합니다.

■ 버퍼

■ 물리 로그 크기

■ Checkpoint 간격

■ AIO VP

■ 페이지 클리너

■ LRU_MIN_DIRTY 및 LRU_MAX_DIRTY

■ LRU 큐

다음과 같은 최종 사용자 요구 사항이 충족될 때까지 위의 매개변수를 튜닝합니다.

■ 98퍼센트 이상의 캐시 읽기

■ 5분의 checkpoint 간격

■ 2초 이하의 checkpoint 지속 시간

예제 결과

다음 결과는 HP9000 G30 1 CPU를 사용한 경우입니다.

첫 번째 실행

초기 매개변수로 얻은 결과의 기준선:

버퍼: 1500

LRU_MIN_DIRTY: 50

LRU_MAX_DIRTY: 60

Ckpt 간격: 300

물리 로그 크기: 1000

페이지 클리너: 1

AIO VP: 6

LRU 큐: 8

결과:

읽기 캐시: 96.94 퍼센트

checkpoint 간격: 45초

checkpoint 지속 시간: 4초

경과 시간: 8:59

루프: 4664

두 번째 실행

98퍼센트 이상의 읽기 캐시를 얻을 때까지 버퍼를 늘립니다. onstat -p로 버퍼 읽기 캐시를 관찰합니다.

버퍼: 5000

LRU_MIN_DIRTY: 50

LRU_MAX_DIRTY: 60

Ckpt 간격: 300

물리 로그 크기: 1000

페이지 클리너: 1

AIO VP: 6

LRU 큐: 8

결과:

읽기 캐시: 99.20 퍼센트

Checkpoint 간격: 45초

Checkpoint 지속 시간: 4초

경과 시간: 10:18

루프: 5942

세 번째 실행

Checkpoint 간격이 5분이 될 때까지 물리 로그 파일의 크기를 늘립니다. 5분 간격이 다 되기 전에 물리 로그는 75퍼센트까지 채워집니다. onstat -l로 관찰합니다.

버퍼: 5000

LRU_MIN_DIRTY: 50

LRU_MAX_DIRTY: 60

Ckpt 간격: 300

물리 로그 크기: 10000

페이지 클리너: 1

AIO VP: 6

LRU 큐: 8

결과:

읽기 캐시: 99.35 퍼센트

Checkpoint 간격: 5분

Checkpoint 지속 시간: 16초

경과 시간: 9:45

루프: 6382

네 번째 실행

LRU_MIN_DIRTY 및 LRU_MAX_DIRTY를 줄여 checkpoint 사이에서 LRU 쓰기를 얻습니다. onstat -F로 LRU 쓰기와 청크 쓰기를 비교 관찰합니다.

버퍼: 5000

LRU_MIN_DIRTY: 5

LRU_MAX_DIRTY: 10

Ckpt 간격: 300

물리 로그 크기: 1 0000

페이지 클리너: 1

AIO VP: 6

LRU 큐: 8

결과:

읽기 캐시: 99.23 퍼센트

Checkpoint 간격: 5분

Checkpoint 지속 시간: 14초

경과 시간: 10:08

루프: 6104

다섯 번째 실행

페이지 클리너 수를 늘려 LRU 쓰기에 대해 쓰기 요청을 합니다. 지속 시간은 많이 변경되지 않았으며, 페이지 클리너는 충분한 작업을 수행할 수 없었습니다. 페이지 클리너를 더 추가합니다. onstat -F로 페이지 클리너 동작을 살펴 보고 onstat -R로 효율성을 관찰합니다. 사용된 페이지의 비율이 LRU_MAX_DIRTY보다 작아야만 합니다.

버퍼: 5000

LRU_MIN_DIRTY: 5

LRU_MAX_DIRTY: 10

Ckpt 간격: 300

물리 로그 크기: 10000

페이지 클리너: 6

AIO VP: 6

LRU 큐: 8

결과:

읽기 캐시: 98.88 퍼센트

Checkpoint 간격: 5분

Checkpoint 지속 시간: 4초

경과 시간: 10:08

루프: 5786

여섯 번째 실행

LRU_MIN_DIRTY와 LRU_MAX_DIRTY를 더 줄여 checkpoint 지속 시간이 2초가 되게 합니다.

버퍼: 5000

LRU_MIN_DIRTY: 1

LRU_MAX_DIRTY: 3

Ckpt 간격: 300

물리 로그 크기: 10000

페이지 클리너: 6

AIO VP: 6

RU 큐: 8

결과:

읽기 캐시: 98.73 퍼센트

Checkpoint 간격: 5분

Checkpoint 지속 시간: 2초

경과 시간: 9:59

루프: 5570

실 행	1	2	3	4	5	6
→버 퍼	1500	5000	5000	5000	5000	5000
→LRU Min	50	50	50	5	1	1
→LRU Max	60	60	60	10	3	3
→Ckor 간격	300	300	300	300	300	300
→Plog 크기	1000	1000	10000	10000	10000	10000
→페이지 클리닉	1	1	1	1	6	6
→읽기 캐시	96.94%	99.20%	99.37%	99.23%	99.88%	98.73%
→Ckot 간격	0.45	0:45	5:00	5:00	5:00	5:00
→Ckot 지속 시간	0.04	0:04	0:16	0:13	0:04	0:02
→루프/Sec	8.6531	9.6149	10.9094	10.0395	9.3625	9.2988
→시간종료/Hour	5:20	5:20	3:12	2.36	0:48	0:24

표 요약

시간종료/Hour는 checkpoint가 발생하는 한 시간 동안 사용자 스레드가 경험한 일시 중단된 시간을 나타냅니다.

루프/Sec은 시뮬레이션을 실행하는 동안의 시스템 성능이나 처리량을 나타냅니다. 이것은 oltp.ec 프로그램에서 1초 간 실행된 루프 수입니다.

이 결과로부터 전체 처리량에서 일부 시스템 성능의 손실이 있음을 확인할 수 있습니다. 그러나 사용자가 느끼는 중단 시간은 상당히 줄어들었습니다. 사용자는 초 당 루프의 1/8이 줄어든다는 사실은 인식하지 못하지만 RETURN 키를 누를 때 16초 간 일시 중지한다는 사실은 인식할 수 있습니다. 시스템 수준에서 튜닝이 더 잘 된 엔진의 경우는 페이지 클리너는 더 적고 좀 더 긴 지속 시간과 더 긴 간격으로 튜닝하는 것이 더 좋을 수도 있습니다(세 개의 페이지 클리너와 15분 간격을 사용하는 위의 네 번째 실행 같은 경우). 그러나 이 실습의 목표를 기준으로 하면 여섯 번째 실행에서 우리가 얻으려는 결과를 볼 수 있습니다.

맺음말

각 시스템은 특정한 목적과 사용자에 맞게 튜닝되어야 합니다. Checkpoint 튜닝 프로세스를 이해하였으므로 이제 각자 필요한 목표에 맞게 시스템을 적절히 튜닝할 수 있을 것입니다.