클라우드 저장소를 사용하는 고성능 백업 애플리케이션 설계

*양신형, 박민균, 이재유, 김수동
숭실대학교 컴퓨터학과
e-mail : {rememberthehill, parkmingyun1, jaeyoo1981, sdkim777}@gmail.com

Design of a High Performance Backup Application using Cloud Storage

*Shinhyung Yang, Min Gyun Park, Jae Yoo Lee and Soo Dong Kim
Dept. of Computer Science, Soongsil University

요 약

클라우드 저장소 서비스는 특정한 장비나 저장 공간의 제약 사항 없이, 언제 어디서나 신뢰성 높은 서버를 활용하여 사용자들에게 다양한 편의를 제공함으로써 사용량이 급증하고 있다. 더불어, 저장 데이터 요청 빈도, 저장 데이터의 크기, 파일 구조 복잡도의 증가로 인해 오버헤드의 발생에 따른 성능 하락에 관한 이슈가 제기된다. 본 논문에서는 클라우드 백업 애플리케이션의 성능 향상을 위해 컴포지트 패턴 기반의 백업 데이터 관리 기법과 동적 자원 할당 기법으로 구성된 설계 모델을 제안한다. 또한, 실사례의 적용을 통해 본 논문에서 제안하는 설계 모델의 실효성을 검증한다.

1. 서론

클라우드 저장소 서비스는 다수의 사용자가 네트워크 연결을 통해 다양한 디바이스의 접근 및 사용이 가능한 클라우드 서비스의 한 유형이다 [1]. 이 서비스에는 파일의 크기, 종류, 복잡도에 따라 다양한 데이터가 존재하며, 이러한 데이터는 여러 곳으로 복제 되기 때문에 데이터가 항상 유효한 값을 가지도록 하는 데이터 무결성을 유지하는 것이 중요하다. 이를 위해서 짧은 시간에 많은 양의 데이터를 동기화 할 수 있는 고성능 백업 저장소가 필요하다.

그러나, 클라우드 서비스는 원격지의 서비스에 네트워크를 통해 접근하는 특징에 따라 네트워크 오버헤드가 존재하고, 사용자 유형이 기업적 규모로 확대되면서 대용량의 복잡도가 높은 데이터 저장에 관한 성능 이슈가 제기된다.

본 논문은 3 장에서 고성능 클라우드 백업 애플리케이션에 관한 요구사항을 정의하고, 4 장에서 성능 향상을 위한 컴포지트 패턴 기반의 백업 데이터 변경사항추적 및 관리 기법과 동적 자원 할당 기법으로 구성된 설계 모델을 제안한다. 5 장에서는 제안된 설계 모델을 실사례에 적용하여 본 논문에서 제안하는 설계 모델의 실효성을 검증한다.

2. 관련연구

Wu, S.의 연구는 원격지의 저장소에 파일을 전송할 때 *iSCSI* 프로토콜 기반의 데이터 읽기 및 쓰기 속도 향상을 위한 블록 I/O 인터페이스를 제안한다 [2]. 블록 I/O 인터페이스는 데이터 및 상태 값, 파일 읽기, 쓰기를 위한 리소스를 직접 관리하여 파일 I/O 인터페이스 또는 디스크 I/O 인터페이스 만을 사용했을 때 발생하는 성능 문제를 해결한다. 그러나, 제안된 기법은 *iSCSI* 프로토콜을 사용하는 시스템에 대해서 한정되어 있다.

Wu, T.Y. 의 연구는 파일 전송 성능 향상을 위해 '인덱스 네임 서버'를 새로운 데이터 관리 구조로 제안한다 [3]. *INS* 는 동일한 데이터의 중첩을 없애고 전송기법을 최적화하여 서비스 성능을 향상시킨다. 그러나, 이 연구는 클라이언트 계층의 백업 알고리즘이나 성능 향상 기법은 미흡하다.

Di Francesco 의 연구는 백업 애플리케이션을 위한 효율적 스케줄링 기법을 제안한다 [4]. 스케줄링 알고리즘은 CPU 오버헤드를 기준으로 동적 작업 우선 순위 선정 기법을 통해 백업, 복구, 네트워크 이질성에 대한 제약사항을 해결했다. 그러나, 이 연구는 백업 수행을 위한 구체적인 설계 모델 및 성능 향상을 위한 기법은 다루어지지 않는다.

3. 고성능 클라우드 백업 애플리케이션 요구사항

본 장에서는 클라우드 저장소를 이용한 고성능 백업 시스템의 요구사항을 기술하며, 개인 및 기업 데이터의 다양한 복잡도의 백업을 대상으로 한다.

기능적 요구사항은 클라우드 저장소 서비스에 대한 사용자 계정 설정 및 백업 폴더 설정, 백업 주기 설정, 백업 주기 혹은 사용자 요청에 따른 백업 실행과 같은 기능성으로 구성된다. 사용자 계정 설정은 클라우드 저장소 서비스의 사용 권한을 획득하기 위한 기능을 제공하고, 백업 폴더 및 백업

주기 설정은 백업 기능 수행을 위한 환경 설정 기능이다. 설정된 주기을 기반으로 백업 기능이 수행되며, 사용자 요청에 따른 백업 수행도 가능하다.

비기능적 요구사항은 백업 성능의 향상을 목적으로 하며 다음의 세 가지로 정의된다.

백업 파일의 무결성: 신뢰성 품질에 대한 요구사항으로 클라우드 저장소의 백업 데이터와 로컬 시스템의 데이터는 정확히 일치해야 한다.

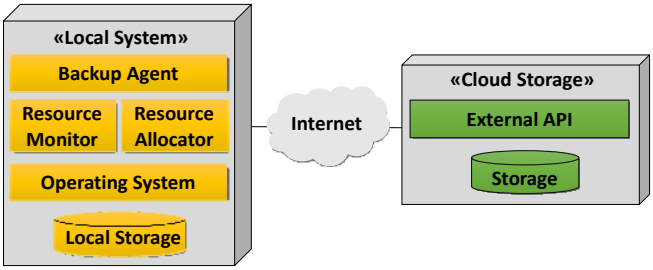
고성능 백업 알고리즘: 백업 성능 향상을 위해 이전 백업 상태에서 변경된 부분의 상태를 파악하고, 각 상태에 따른 백업 기능을 수행해야 한다. 예를 들어, 특정 폴더의 위치가 변경되었을 때, 삭제 후 업로드의 기능을 수행하는 것이 아닌 클라우드 저장소 내에서 해당 폴더의 정보를 수정하는 것이 보다 높은 성능을 보장한다. 따라서, 백업 데이터의 변경 사항 비교와 상태에 따른 백업 기능 선택을 지원해야 한다.

백업 자원의 동적 할당: 클라우드 백업 애플리케이션의 성능은 백업 기능을 수행하기 위해 할당되는 자원량에 비례한다. 따라서, 백업 순간의 가용 자원량을 측정하고 이를 기반으로 동적으로 백업 자원을 할당하여 성능을 향상시킬 수 있다.

4. 클라우드 백업 애플리케이션의 설계 모델

4.1. 클라우드 백업 애플리케이션 구성도

클라우드 백업 애플리케이션은 (그림 1)과 같이 로컬 시스템과 클라우드 저장소 서비스로 구성되며, 다양한 로컬 시스템을 대상으로 한다.



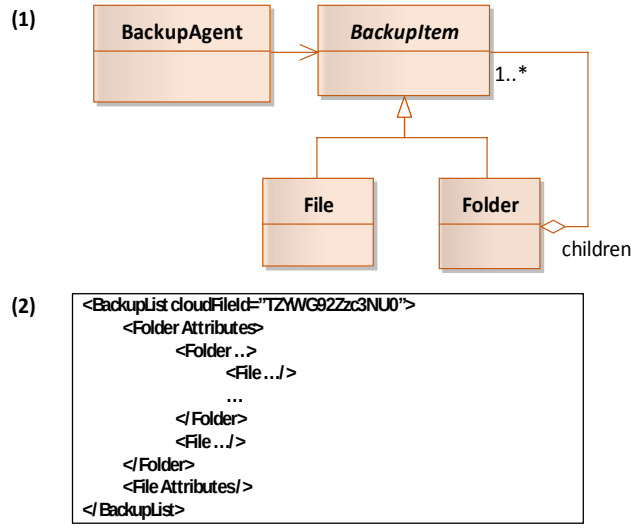
(그림 1) 클라우드 백업 애플리케이션 구성도

Backup Agent 는 백업 기능성을 제공하고, *Resource Monitor* 는 할당 가능한 자원 현황을 확인하는 기능을 제공한다. *Resource Allocator* 는 백업 자원을 동적으로 할당하거나 해제하는 기능을 제공한다. *Cloud Storage Service* 는 로컬 저장소의 폴더와 파일들에 대한 백업 공간과 서비스의 호출을 위한 외부 API 를 제공한다.

4.2. 컴포지트 패턴을 적용한 고성능 백업 설계

파일 시스템에서 폴더는 하위 폴더와 파일을 갖는 반복 구조이며, 백업 기능은 전체 폴더와 파일에 대해 일괄적으로 수행된다. 컴포지트 패턴을 이용하여 백업 목록에 효과적으로 접근하고 연관배열을 통한 로컬 데이터 관리 기법을 사용함으로써 파일 비교 연산에 따른 시간 복잡도를 줄일 수 있다.

Backup Agent 는 로컬 시스템의 백업 목록에 대해 변경 사항 업데이트 및 백업을 수행한다. 백업 목록은 컴포지트 패턴을 적용하여 각 개별 파일을 관리한다. (그림 2)는 컴포지트 패턴 기반의 백업 목록 관리 및 XML 형태의 저장 방식을 보여준다.



(그림 2) 컴포지트 패턴 적용 파일 구조와 XML 표현

백업 목록은 (그림 2)의 (1)처럼 컴포지트 패턴을 적용하여 폴더와 하위 파일 및 폴더의 반복 호출을 통해 로컬 파일 아이디, 클라우드 파일 아이디, 수정 날짜 등 백업 수행에 필요한 속성 정보를 통합 관리한다. 이를 통해 백업을 수행하면 매번 로컬 파일 시스템에 접근하여 속성정보를 가져오는 시간이 없어지므로 백업 성능이 향상된다. 백업목록이 최신 버전으로 갱신될 때 전체 내용이 파일로 저장되며 (그림 2)의 (2)는 이를 위해 정의한 XML 형식이다.

Backup Agent 는 컴포지트 백업을 적용한 백업 목록의 변경사항을 확인하는 기능(*updateState*)과 변경된 부분을 백업하는 기능(*runBackup*)을 제공한다.

updateState 기법은 백업 목록에 접근하여 각각의 파일에 대해 대응하는 로컬 저장소의 파일에 발생한 변경사항을 표시한다.

runBackup 기법은 변경사항이 표시되어 있는 백업 목록에 대하여 백업을 수행한다.

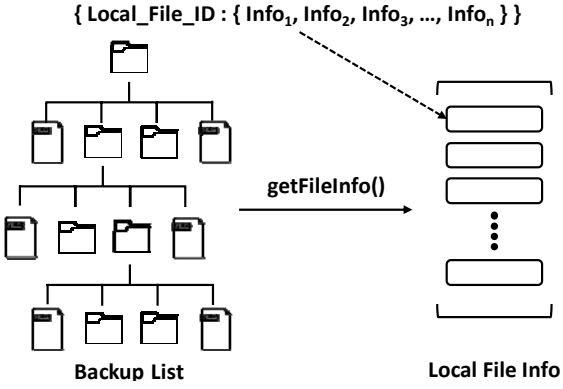
<표 1>은 *runBackup* 기법의 의사코드로 한 폴더와 그 하위 폴더에 대해 같은 메소드가 재귀적으로 호출됨을 보여준다.

<표 1> Folder 의 runBackup 메소드에 대한 의사코드

```
1.  def runBackup():
2.      if state == 'NoChanged':
3.          for backupItem in childrenList:
4.              backupItem.runBackup()
5.      elif state == 'Modified':
6.          update_file()
7.          for backupItem in childrenList:
```

```
8.              backupItem.runBackup()
9.      elif state == 'Deleted':
10.         delete_file()
11.     elif self.state == 'New':
12.         for f in currentPath:
13.             insert_file()
14.             backupItem.runBackup()
15.     self.add(backupItem)
```

백업 수행시 로컬 시스템과 클라우드 저장소의 파일 목록을 비교하여 <표 1>의 2, 5, 9, 11 번째 줄과 같이 변경사항을 확인하여 백업하며, 이는 $O(2^n)$의 시간 복잡도를 가진다. 여기서 n 은 전체폴더의 갯수와 전체파일의 갯수를 나타낸다. 클라우드 백업 애플리케이션은 백업목록의 비교연산 비용을 줄이기 위해 키와 값의 집합으로 이루어진 연관 배열을 사용한다. (그림 3)은 연관 배열로 관리하는 로컬 데이터의 정보와 백업 목록을 비교 연산하는 과정을 보여준다.



(그림 3) 연관 배열을 통한 로컬 데이터 관리

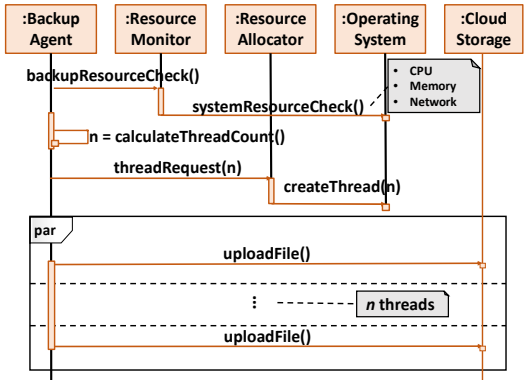
연관 배열을 사용하면 로컬 파일 아이디로 파일의 정보를 조회하므로 O(n)의 시간복잡도를 가지게 되어 앞에서 설명한 파일비교방식보다 효율적이다.

4.3. 백업 자원 동적 할당 기법

본 절에서는 클라우드 백업 애플리케이션의 백업 자원에 대한 동적 할당 기법을 제시한다. 조사한바에 따르면 구글 드라이브나 드롭박스에서 제공하는 백업 애플리케이션은 파일 백업 시 시스템 가용 자원을 최적으로 활용하지않는다. 이에 비해 본 기법은 다양한 로컬 시스템에서 실시간 가용 자원량을 고려하여 백업 시간을 단축시키고 자원 사용을 최적화하여 고성능 백업을 가능하게 한다.

본 논문에서는 백업 기능에 영향을 미치는 프로세스와 메모리, 네트워크 자원을 기준으로 자원 할당량을 제어한다.

(그림 4)는 백업 자원에 대한 동적 할당 기법의 수행 절차를 보여준다.



(그림 4) 동적할당기법 수행절차

자원 할당기(*Resource Allocator*)는 백업 기능 수행 중 (그림 4)의 *systemResourceCheck()* 기능을 통해 실시간으로 시스템의 가용 자원량을 확인하고 *calculateThreadCount(), threadRequest()*를 통해 지정받은 갯수만큼 쓰레드를 할당하여 시스템 가용 자원을 최적으로 활용하므로 고성능의 백업이 가능하다.

각 자원에 대한 할당 가능한 자원의 비율(*Available Resource Rate, ARR*)의 연산식은 (1)과 같다.

$$ARR_i = 1 - \frac{AllocatedResourceAmount_i}{TotalResourceAmount_i} \quad \text{.....(1)}$$

i 는 자원의 유형을 나타내며, 프로세스와 메모리, 네트워크 중 하나의 값을 나타낸다. 분모는 디바이스가 보유한 해당 자원의 총량을 나타내고, 분자는 이미 할당된 자원량을 나타낸다.

백업 기능은 이전 백업 목록과 비교 연산하는 부분과 네트워크를 통해 변경된 사항을 백업하는 부분으로 구성된다. 각 기능의 특징에 따라 영향을 받는 자원 유형이 다르다. 예를 들어, 비교 연산 기능은 프로세스 자원에 종속적이고, 네트워크 기능은 네트워크, 메모리, 프로세스 순으로 영향을 받는다.

$$NumOfThread = \frac{\lfloor ARR_{Process}, ARR_{Memory}, ARR_{Network} \rfloor}{AverageResourceRatePerThread} \quad \text{.....(2)}$$

분모는 하나의 쓰레드가 차지하는 로컬 시스템 자원의 평균 비율을 나타내고, 분자는 프로세스와 메모리, 네트워크 자원 중 최소값을 나타낸다. 따라서, 세 가지 자원 중에서 가장 작은 자원량을 가진 자원을 기반으로 백업 수행 쓰레드의 수가 정해진다.

5. 구현 및 평가

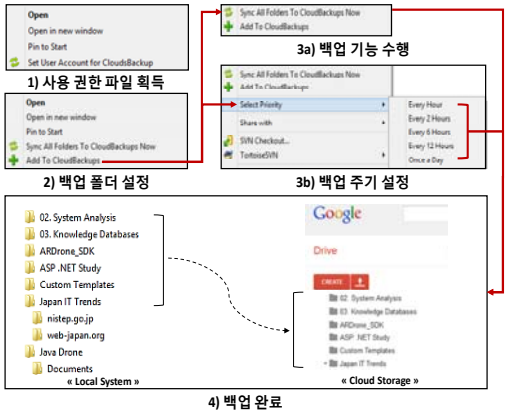
본 장은 4 장에서 제안한 설계 모델을 바탕으로 구현된 애플리케이션의 결과물을 평가한다.

5.1. 백업 애플리케이션 구현

애플리케이션은 윈도우 환경에서 구현하였고 제공되는 Context 메뉴를 활용하기 위하여 Visual C++ 라이브러리를 사용하였다. 클라우드 서비스는 구글 드라이브를 사용하였고 제공되는 API 를 윈도우 환경에서 호출하기 위해 Python 언어를 사용하였다.

(그림 5)는 윈도우 환경에서의 사용자 인터페이스와 클라우드 저장소에 백업된 결과를

보여주는 화면이다.

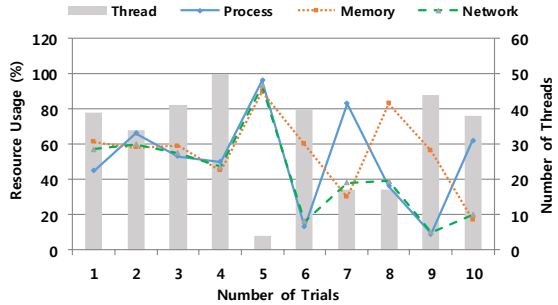


(그림 5) 고성능 백업 애플리케이션 구현 화면

5.2. 평가

본 절에서 수행되는 실험은 실행 시간을 기반으로 제시된 설계 모델의 적용성과 실용성 검증을 목적으로 한다. 실험을 위한 환경은 Windows7(64bit), CPU(Intel Core 3.20GHz), RAM(8G), Realtek RTL8168/100Mbps 을 사용하였다.

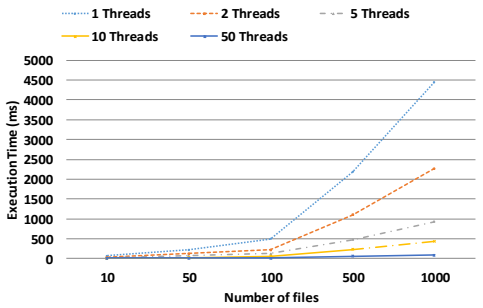
(그림 6)은 로컬 시스템의 프로세스, 메모리, 네트워크 세 가지 자원 상태에 따른 쓰레드의 수를 나타낸 그래프이다.



(그림 6) 자원 상태에 따른 쓰레드의 수

로컬 시스템의 자원량을 측정한 결과 첫 번째는 프로세스 45%, 메모리 61%, 네트워크 57%의 사용량이 측정됐으며, 4.3 절에서 제안한 연산식을 기반으로 수행한 결과 39 개의 쓰레드가 동적 할당되었다.

(그림 7)은 설계 모델에서 제안된 자원할당기를 사용하여 동적으로 할당된 쓰레드의 수와 파일의 수에 따라 백업 수행 시간을 비교한 그래프이다.



(그림 7) 쓰레드의 수에 따른 실행시간 비교

하드웨어 사양에 따라 자원의 임계치(Threshold)가 다르기 때문에 쓰레드의 수는 동적으로 변할 수 있으며 실험을 통하여 쓰레드 수에 따른 백업 수행 시간을 보여준다.

파일의 개수가 10 개 일 때 쓰레드 1 개를 사용하여 수행한 결과 66.5ms, 쓰레드 50 개는 12.3ms 가 소요되었다. 또한 파일의 개수가 1000 개일때 4453.1ms 와 878ms 이 소요되었다.

위의 결과를 분석하면 동적으로 할당된 쓰레드의 수가 많아 질수록 백업 수행 시간이 적게 걸리는 것을 확인할 수 있었으며 파일의 수가 많아 질수록 백업 수행 시간은 보다 차이를 보였다. 따라서, 동적 백업 자원 설계 모델을 기반으로 구현한 백업 애플리케이션은 성능 향상에 큰 영향을 미치는 것을 확인할 수 있다.

6. 결론

본 논문에서는 컴포지트 패턴 기반의 백업 데이터 변경사항 추적 및 관리 기법과 동적 자원 할당 기법을 정의하고, 이를 기반으로 고성능 클라우드 백업 애플리케이션의 설계 모델을 제안하였다. 또한, 실사례에 제안된 설계 모델을 적용하여 실시간으로 가용 시스템 자원이 변화하는 환경에서 쓰레드의 갯수를 조절했을 때 데이터의 양이 많아져도 일정비율로 업로드 시간이 단축되었음을 검증하였다.

본 논문에서 제안하는 고성능 클라우드 백업 애플리케이션을 기업단위의 환경에 적용하면 백업 성능의 향상으로 인해 데이터 관리 효율이 증가한다.

Acknowledgement

본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다(UD060048AD).

참고문헌

[1] Foster, I., Zhao, Y., Raicu, I., and Lu, S., "Cloud computing and grid computing 360-degree compared." In *Grid Computing Environments Workshop* (*GCE'08*), pp.1-10, 2008.

[2] Wu, S., Chen, J., Feng, D., and Mao, B., "Implementation and Evaluation of the Block I/O Interface between the iSCSI Target and the Storage Device." In *Convergence Information Technology*, pp.1451-1456, 2007.

[3] Wu, T.Y., Lee, W. T., and Lin, C. F., "Cloud storage performance enhancement by real-time feedback control and de-duplication." In *Wireless Telecommunications Symposium (WTS)*, pp.1-5, 2012.

[4] Di Francesco, P., and Sweden, V., "Design and implementation of a MLFQ scheduler for the Bacula backup software." Diss. Mälardalen University, 2012.