다수 클라우드 스토리지로의 데이터 분할 및 부분 중복을 통한 데이터 가용성 향상

종신회원 박 준 철*

Improving Data Availability by Data Partitioning and Partial Overlapping on Multiple Cloud Storages

Jun-Cheol Park* *Lifelong Member*

요 약

시스템의 고장, 크래킹, 오작동, 정전 등의 이유로 저장된 고객 데이터의 영구적 유실이나 일정 기간 동안의 접근 불가 상황이 발생할 때, 클라우드 스토리지 서비스 고객은 클라우드 서비스 제공자에 의한 데이터 복원이 가능하기를 기대할 수밖에 없다. 본 논문에서는 이 문제에 대해 클라우드 스토리지 시스템 내에서가 아니라 클라우드 고객의 영역에서 실현할 수 있는 솔루션을 고려한다. 본 논문은 고객이 다수의 클라우드 스토리지 제공자 들을 선택하여, 저장할 데이터 단위를 분할, 암호화 후 부분 중복 저장함으로써 일부 클라우드 스토리지에 접근이 불가능한 상황에서도 데이터 단위를 복원할 수 있는 기법 및 그 적용 구조를 제안한다. 제안 방식은 높은 데이터 가용성 보장과 더불어, 개별 사용자에 투명하게 데이터 암호화 키를 갱신할 수 있으며, 사용자별로 접근했던 데이터 단위의 목록들을 명시할 수 있어 필요 시 데이터 유출의 범위를 명확히 규정할 수 있도록 한다.

Key Words : 클라우드 스토리지, 데이터 가용성, 데이터 분할, 데이터 부분 중복, 보안

ABSTRACT

A cloud service customer has no other way but to wait for his lost data to be recovered by the cloud service provider when the data was lost or not accessible for a while due to the provider's system failure, cracking attempt, malfunction, or outage. We consider a solution to address this problem that can be implemented in the cloud client's domain, rather than in the cloud service provider's domain. We propose a high level architecture and scheme for successfully retrieving data units even when several cloud storages are not accessible at the same time. The scheme is based on a clever way of partitioning and partial overlapping of data for being stored on multiple cloud storages. In addition to providing a high level of data availability, the scheme makes it possible to re-encrypt data units with new keys in a user transparent way, and can produce the complete log of every user's data units accessed, for assessing data disclosure, if needed.

Ⅰ. 서 론

클라우드 컴퓨팅은 주로 비용 절감의 목적을 위해 서비스나 컴퓨팅 능력, 데이터 등을 외부 기관에 위탁하는 형태의 컴퓨팅 모델을 의미하는데, 최근 활발한 연구 개발을 통해 다양한 형태의 클라우드 서비스들이 제공되고 있다[1-3]. 클라우드 컴퓨팅이 제공하는 서비스로 IaaS(Infrastructure as a Service), PaaS

※ 이 논문은 2010학년도 홍익대학교 학술연구진흥비에 의해 지원되었음.
* 홍익대학교 컴퓨터공학과(jcpark@hongik.ac.kr)
논문번호 : KICS2011-06-239, 접수일자 : 2011년 6월 7일, 최종논문접수일자 : 2011년 11월 30일

(Platform as a Service) 및 SaaS(Software as a Service)가 있는데, IaaS는 고객이 서비스 제공자의 컴퓨팅, 스토리지 또는 네트워킹 인프라를 사용하는 것이고, PaaS는 고객이 자신의 응용 프로그램을 실행시키기 위해 서비스 제공자의 자원을 활용하는 것이며, SaaS는 고객이 서비스 제공자의 인프라 상에서 실행되는 소프트웨어를 사용하는 것을 말한다[2,3].

최근 여러 클라우드 스토리지 서비스(예: Amazon의 S3, Microsoft의 SkyDrive, Nirvanix의 CloudNAS 등)가 널리 보급되고 있다. 회사나 기관 같은 고객은 이러한 서비스를 이용함으로써 자체의 스토리지 인프라를 구축, 관리 및 개선하는 비용의 투자 없이 인터넷을 통해 실제 사용한 만큼의 비용만을 지불하면 된다. 하지만, 이런 장점들에도 불구하고, 고객의 입장에서는 데이터 보안성 보장의 우려 때문에 클라우드 스토리지 이용을 주저하는 것 역시 사실이다. 개인의 의료 정보 등의 민감한 데이터의 경우 데이터 기밀성과 무결성에 대한 보장이 필수적이며, 더불어, 언제 어디서나 데이터 접근이 가능토록 하는 가용성에 대한 보장 역시 매우 중요한 요구 사항이다. 따라서 이런 보안 성질에 대한 적절한 수준의 보장이 없다면, 많은 장점에도 불구하고 클라우드 스토리지의 사용은 제한적일 수밖에 없을 것이다. 이런 우려에 대응하여 클라우드 서비스 제공자들은 보안 수준을 높이기 위해 노력하고 있으나, 근래 발생한 여러 사례들(Google Docs 시스템의 소프트웨어 오작동으로 인한 프라이버시 침해 사례[11], Amazon S3에서 하드웨어 고장으로 사용자 데이터가 훼손된 사례[12], LinkUp이라는 스토리지 서비스 제공자가 시스템 관리자의 실수로 저장된 고객 데이터의 45%를 잃은 사례[13], Carbonite라는 스토리지 서비스 제공자가 고객의 데이터 유실에 책임을 물어 하드웨어 백업 장치 제조회사를 고소한 사례[14], 기타 Google Gmail, Amazon S3, FlexiScale의 수 시간 이상의 서비스 다운 사례[15-17] 등)에서 볼 수 있듯이 아직까지 높은 수준의 보안을 요구하는 고객을 만족시킬 수 있는 서비스는 제공하지 못하고 있다.

본 논문에서는 높은 수준의 보안성을 요구하는 고객을 위해 데이터 기밀성 및 무결성 보장과 더불어, 클라우드 스토리지를 통해 데이터 가용성을 향상시킬 수 있는 방법을 제안한다. 제안 방식은 일부 클라우드 스토리지의 고장이나 오작동 등이 발생하더라도 고객이 저장한 데이터를 복원할 수 있는 기능을 제공하며, 이 과정에서 기존 클라우드 스토리지에 새로운 기능이나 보안 수준 강화 등을 추가로 요구하지 않는다.

본 논문의 구성은 다음과 같다. 2장에서는 데이터 분할 및 부분 중복 후 저장 아이디어와 이를 통한 데이터 복원 가능성을 논한다. 3장에서는 제안하는 데이터의 클라우드 스토리지 업로드 프로토콜 및 데이터 검색 후 획득 과정을 설명하고, 일부 스토리지에 문제가 생긴 경우에도 데이터 단위의 복원이 가능함을 보인다. 4장에서는 가용성 제고를 비롯한 제안 방식의 보안성을 분석하고, 성능 분석과 함께 키 관리 및 갱신의 효율성을 설명한다. 5장에서는 기존의 관련 연구들과 제안 방식을 비교, 분석한다. 6장에서 결론 및 향후 과제를 제시하면서 맺는다.

Ⅱ. 데이터 단위의 분할 및 부분 중복 방법과 데이터 단위의 복원 가능성

데이터 단위를 분할 및 부분 중복한 후 다수 클라우드 스토리지 들에 분산 저장하는 방법을 소개하고, 분할 데이터들로부터 데이터 단위의 복원이 가능한 조건과 그 복원 방법을 설명한다.

2.1 구조 및 데이터 흐름

아래의 그림 1은 데이터 단위의 분할 및 부분 중복 후 저장을 주관하는 가상의 회사 TheCorp 소속의 데이터 관리(Data Management, 이하 DM) 서버와 이를 통해 TheCorp의 사용자 Alice가 키워드 검색으로 데이터 단위를 얻기 위해 여러 클라우드 스토리지 들과 통신하는 과정을 보여준다.

Alice는 먼저 회사의 인증 서버를 통해 자신이 적법한 사용자임을 인증 받고, 이를 증빙하는 자료(인증토큰 등)를 데이터 검색 키워드와 함께 DM 서버에 보낸다. DM 서버는 인증토큰을 검증하여 Alice의 검색 자격을 확인하고, 키워드에 해당하는 데이터 단위를 얻는데 필요한 데이터요청토큰(스토리지에 저장된 분할을 다운로드 받기 위해 스토리지에 제시하는 정보) 모음 및 관련 비밀 값들을 Alice에게 반환한다.

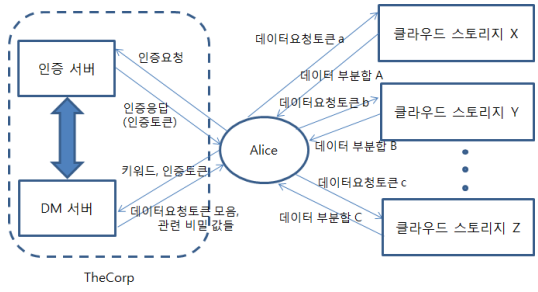

그림 1. 제안하는 시스템 구조 및 사용자의 키워드 검색 후 데이터 단위 획득 과정 흐름도

Alice는 데이터요청토큰의 목적지 정보에 따라 해당되는 데이터요청토큰을 지정된 클라우드 스토리지에게 각각 전송하고, 그 응답을 기다린다. Alice는 여러 클라우드 스토리지 들로부터 수신한 데이터 부분합(스토리지마다 저장된 데이터 단위의 일부분)들과 DM 서버에게 받은 비밀 값들을 이용하여 원하는 데이터 단위를 복원해내고, 그 무결성을 검증한다.

2.2 DM 서버

핵심 요소인 DM 서버는 입력 키워드에 매치되는 데이터 단위를 (1) 클라우드 스토리지 들에 업로드하고, 또한 (2) 사용자가 클라우드 스토리지 들로부터 획득할 수 있도록 적절한 정보를 사용자에게 제공하는 역할을 한다. 데이터 업로드를 위해서는 해당 데이터 단위를 적절히 나누고, 나눈 부분들을 조합하여 여러 부분합 들을 생성하며, 이 부분합 들이 한 클라우드 스토리지에 각각 하나씩 저장되도록 한다. 배치할 데이터는 기밀성을 보장하기 위해 AES 암호화한 상태로 스토리지에 저장하며, 복원한 데이터 단위의 무결성을 확인하기 위해 부분별 해쉬 값을 연접한 결과를 다시 해쉬한 값을 DM 서버에 별도 저장한다. 사용자의 다운로드 및 데이터 단위 복원을 위해서는 일정한 수의 부분합 들이 필요하며(아래 [정리 1] 참조), DM 서버는 검색한 데이터 단위를 저장하고 있는 클라우드 스토리지 들 중 일부를 선택하여 이들에게 제시할 데이터요청토큰 모음 및 관련된 비밀 값들을 사용자에게 제공한다. 사용자는 클라우드 스토리지 들에게서 수신한 데이터 부분합 들과 추후 설명할 데이터 복원 알고리즘을 이용하여 해당 데이터 단위를 복원할 수 있다.

사용자가 클라우드 스토리지에 제시하는 데이터요청토큰에는 사용자의 인증 정보나 비밀 키 값이 포함되어 있지 않으며, 따라서 클라우드 스토리지는 토큰을 통해 어떤 기관의 DM 서버를 통해 요청이 온 것인지만 알고 실제 데이터를 요청한 사용자에 대해서는 전혀 알지 못한다. 따라서 제안 방식을 적용하기 위해 클라우드 스토리지에 별도의 사용자별 인증 기법이 구현될 필요는 없다.

2.3 데이터 분할 및 원래 데이터 단위의 복원

어떤 회사에서 서로 다른 $t(t \geq 2)$ 개의 클라우드 스토리지를 사용할 수 있다고 가정한다. 데이터 단위 D_i 의 분할을 위해 우선 D_i의 크기를 고려한 분할 개수 $n(2 \leq n \leq t)$을 선택하고, D_i를 다음과 같이 n 개의 부분 D_i^j, $j=0,...,n-1$, 으로 나눈다.

$$D_i = D_i^0 \| D_i^1 \| \cdots \| D_i^{n-1}$$

(‖ 는 연접(concatenation) 연산자임)

이제 D_i의 부분들을 포함하는 n개의 q-부분합 $D_i^{[j,q]}$, $j=0,...,n-1$, 를 아래와 같이 정의한다. 각 q-부분합 $D_i^{[j,q]}$는 $q(>0)$ 개의 부분을 포함하는데, 이 때 각 부분 D_i^j 에서 위첨자 j는 $j \bmod n$을 의미한다(즉, 0에서 $(n-1)$ 사이의 값이 됨).

$$D_i^{[j,q]} = D_i^j \| D_i^{j+1} \| \cdots \| D_i^{j+q-1},\ j=0,...,n-1$$

DM 서버는 n 개의 클라우드 스토리지를 선택하고, D_i의 q-부분합 $D_i^{[j,q]}$,$j=0,...,n-1$, 이 선택된 각 클라우드 스토리지에 하나씩 저장되도록 한다.

그림 2는 $t=n=5$ 일 때, 여러 q 값에 대하여 각 클라우드 스토리지 c_j에 q-부분합 $D^{[j,q]}$가 저장된 것을 보여준다. 다음의 [정리 1]에서 $r=n-q+1$ 개의 q-부분합 만을 가지고 원래의 데이터 단위 D를 복원해낼 수 있음을 보인다. 예를 들어, $q=3$일 때 그림 2의 (b)에서 $r=n-q+1=3$ 개의 스토리지 c_0, c_1, c_3 에 저장된 3-부분합 $(D^0\|D^1\|D^2)$, $(D^1\|D^2\|D^3)$ 및 $(D^3\|D^4\|D^0)$ 로부터 $D=(D^0\|D^1\|D^2\|D^3\|D^4)$를 얻을 수 있음은 자명하다. 한편, $r=3$ 보다 작은 수만큼의 스토리지를 선택하게 되면 D를 복원하지 못할 수도 있는데, 예를 들어 c_0 과 c_1 두 개를 선택하면, 3-부분합 $(D^0\|D^1\|D^2)$과 $(D^1\|D^2\|D^3)$으로부터 D^4를 얻을 수 없으므로 D를 복원할 수 없다.

[정리 1] D_i의 n개의 q-부분합 $D_i^{[j,q]}$, $j=0,...,n-1$, 중 임의로 선택한 $r=n-q+1$ 개의 q-부분합 들로부터 D_i를 복원할 수 있다. 즉, 모든 $j=0,...,n-1$에 대하여 D_i^j를 구할 수 있다.

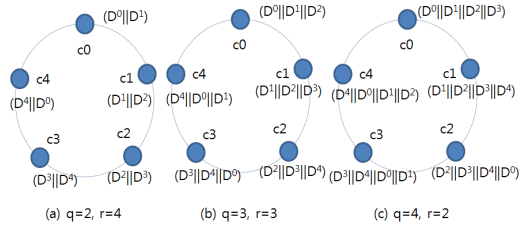

그림 2. $t=n=5$일 때 여러 q 값에 대한 q-부분합 저장 내역 및 r 값

[증명] r 개의 q-부분합 $D_i^{[m(j),q]}$, $j=0,...,r-1$, $(0 \le m(0) < m(1) < \cdots < m(r-1) \le n-1$)이 선택되었다고 가정하자. $D_i^{[m(j),q]}$, $j=0,...,r-1$,으로부터 구할 수 없는 데이터 부분 D_i^x 가 존재함(즉, 어떤 $D_i^{[m(j),q]}$도 D_i^x를 포함하지 않음)을 가정하여 모순을 유도하고자 한다. (경우1) $x=0$: D_i^0 를 포함하지 않아야 하므로, $m(0) \ge 1$ 이어야 한다. 따라서 $m(r-1) \ge m(0)+(r-1) \ge 1+(r-1)=r$ 라는 (부등식 1.1)이 성립한다. 또한, $D_i^{[m(r-1),q]}$ 이 D_i^0을 포함하지 않아야 하므로, $D_i^{[m(r-1),q]}$의 마지막 부분인 $D_i^{m(r-1)+q-1}$의 위첨자 $m(r-1)+q-1$ 는 n보다 작아야 한다(n과 같아지면, $\bmod n$ 연산에 의해 0이 되므로). 즉, $m(r-1)+q-1 < n$ 이 성립해야 하고, 이를 정리하면 $m(r-1)+q-1 \le n-1$ $\Rightarrow m(r-1) \le n-q$ 라는 (부등식 1.2)를 얻는다. (부등식 1.1)과 (부등식 1.2)에 의해 $r \le m(r-1) \le n-q$ 가 성립하고, 이로부터 $r \le n-q \Rightarrow n-q+1 \le n-q \Rightarrow 1 \le 0$ 이 유도된다. 이는 모순이다. (경우2) $x=n-1$: D_i^{n-1}을 포함하지 않아야 하므로 $D_i^{[m(r-1),q]}$의 마지막 부분인 $D_i^{m(r-1)+q-1}$의 위첨자 $m(r-1)+q-1$ 은 $n-1$보다 작아야 한다. 즉, $m(r-1)+q-1 < n-1$ 이 만족되어야 하므로 $m(r-1)+q-1 \le n-2 \Rightarrow m(r-1) \le n-q-1$ 라는 (부등식 2.1)이 성립한다. 또한 $m(0) \ge 0$ 이므로, $m(r-1) \ge m(0)+(r-1) \ge 0+(r-1)=r-1$라는 (부등식 2.2)도 성립한다. (부등식 2.1)과 (부등식 2.2)에 의해 $r-1 \le m(r-1) \le n-q-1$ 이므로, $r-1 \le n-q-1 \Rightarrow n-q+1-1 \le n-q-1 \Rightarrow 0 \le -1$ 에 의해 모순이 유도된다. (경우3) $0 < x < n-1$: 위첨자를 기준으로 전체구간 $[0:n-1]$을 x 중심의 좌측 구간 $[0:x-1]$과 우측 구간 $[x+1:n-1]$으로 구분한다. q-부분합의 최초 부분(즉, $D_i^{[j,q]}$에서 D_i^j)의 포함 여부를 기준으로 판단할 때, 좌측 구간 $[0:x-1]$에는 최대 $x-q+1$ 개의 q-부분합이 존재할 수 있다. 왜냐하면 $x-q$ 보다 큰 j에 대해 $D_i^{[j,q]}$는 선택된 q-부분합일 수 없으므로(D_i^x를 포함하지 않아야 하므로), $[0:x-q]$ 구간이 가질 수 있는 최대 q-부분합의 수인 $(x-q)-0+1=x-q+1$가 좌측 구간에 존재할 수 있는 q-부분합의 최대 개수와 같기 때문이다. 한편, 우측 구간 $[x+1:n-1]$에서는 최초 부분의 포함 여부를 기준으로 판단할 때 최대 $(n-1)-(x+1)+1=n-x-1$ 개의 q-부분합이 존재할 수 있다. 따라서 좌측 및 우측 구간에는 최대 $(x-q+1)+(n-x-1)=n-q=r-1(\because r=n-q+1)$ 개의 q-부분합이 포함될 수 있어, 그 수가 r 개보다 작으므로 모순이다. ■

[따름정리 1] 제안 방식에서 D_i의 q-부분합을 저장하고 있는 클라우드 스토리지 n 개 중에서 최대 $(q-1)$ 개의 클라우드 스토리지가 q-부분합을 제공하지 못하더라도, D_i의 복원이 가능하다.

[증명] D_i의 q-부분합을 제공하는 클라우드 스토리지가 최소 $n-(q-1)=n-q+1=r$ 개 존재하므로 [정리 1]에 의해 D_i를 복원할 수 있다. ■

Ⅲ. 데이터 업로드 및 검색 후 획득과 복원

3.1 데이터 업로드 프로토콜

데이터 단위 D_i를 DM 서버를 통해 여러 클라우드 스토리지에 분할 후 부분 중복 저장하는 프로토콜로서, D_i의 검색 및 다운로드를 위해 선행되어야 한다. 이하, 데이터 단위 D_i는 신규 작성 또는 수정된 자료로서 회사 TheCorp의 소유물이며, 이 회사 소속의 사용자들은 키워드(또는 고유 ID) Q_{D_i}를 통해 D_i에 접근할 수 있다고 가정한다. DM 서버는 D_i 및 접근 허용 범위를 가지고 다음과 같이 각 클라우드 스토리지에 분산 저장할 내용을 생성하고, 관련된 비밀 값들을 자체 저장한다.

(단계 1) D_i의 크기 및 사용 가능한 클라우드 스토리지의 수를 고려하여 n을 정하고, $q+r=n+1$ 을 만족하는 $q(q>0)$와 $r(r>0)$을 정한다.

(단계 2) D_i를 n 개의 부분 D_i^j, $j=0,...,n-1$, 로 분할한다. 즉, $D_i = D_i^0 \| D_i^1 \| \cdots \| D_i^{n-1}$ 이 된다. 각 부분 D_i^j 의 길이는 $l(D_i^j)$로 나타낸다.

(단계 3) 각 $j=0,...,n-1$에 대해 D_i의 q-부분합 $D_i^{[j,q]} = D_i^j \| D_i^{j+1} \| \cdots \| D_i^{j+q-1}$ 를 저장할 클라우드 스토리지 $c(D_i^{[j,q]})$를 선택한다.

(단계 4) 각 $j=0,...,n-1$에 대해 $D_i^{[j,q]}$ 를 압축(compress)하고, 그 결과를 $\widetilde{D_i^{[j,q]}}$ 라고 한다.

(단계 5) 각 $j=0,...,n-1$에 대해 $\widetilde{D_i^{[j,q]}}$를 암호화할 키 $K_{D_i^{[j,q]}}$ 를 선정하고, $K_{D_i^{[j,q]}}$ 로 $\widetilde{D_i^{[j,q]}}$를 암호화

한 결과 $E_{K_{D_i}^{[j,q]}}(\widetilde{D_i^{[j,q]}})$를 얻는다.

(단계 6) 회사 TheCorp의 사용자 및 소유 데이터를 위해 다음과 같은 데이터를 저장한다.

(a) (사용자 : 사용자의 비밀 키) 쌍
(예: Alice : K_{Alice})

(b) 각 데이터 단위 D_i 에 대해
키워드(또는 데이터 고유 ID), n, q, r, $h(h(D_i^0)\|\cdots\|h(D_i^{n-1}))$, $l(D_i^j)$, $K_{D_i^{[j,q]}}$, $c(D_i^{[j,q]})$, 단, $j=0,...,n-1$, 이고 $h(\cdot)$는 안전한 해쉬 함수(예: SHA-256)를 의미함.

DM 서버와 각 클라우드 스토리지 $c(D_i^{[j,q]})$, $j=0,...,n-1$, 사이에 다음 데이터 업로드 프로토콜이 실행된다.

DM 서버 → 클라우드 스토리지 $c(D_i^{[j,q]})$:

(a) 데이터 키워드: Q_{D_i} , (b) 데이터 소유자(공유 범위): TheCorp, (c) 목적지 클라우드 스토리지: $c(D_i^{[j,q]})$, (d) $E_{K_{D_i}^{[j,q]}}(\widetilde{D_i^{[j,q]}})$, (e) DM 서버의 ID, (f) DM 서버의 서명: [(a)‖(b)‖(c)‖(d)‖(e)]에 해쉬 함수 적용한 값을 DM 서버의 개인키로 암호화한 값

클라우드 스토리지 $c(D_i^{[j,q]})$:

- 수신 데이터로부터 DM 서버의 ID를 확인하고, 저장된 해당 DM 서버의 공개키로 수신 데이터의 서명을 검증한다.
- 수신 데이터에 명시된 목적지 클라우드 스토리지가 $c(D_i^{[j,q]})$ 인지를 확인하고, 데이터 소유자가 이 클라우드 스토리지에 자료를 저장할 수 있는 자격이 있는지 확인한다.
- (Q_{D_i}, TheCorp, $E_{K_{D_i}^{[j,q]}}(\widetilde{D_i^{[j,q]}}), T$) 를 클라우드 스토리지의 적절한 위치에 저장한다(T는 저장한 시각).
- 저장 확인((수신 데이터 전체 ‖T)에 해쉬 함수 적용한 값을 $c(D_i^{[j,q]})$의 개인키로 암호화한 것)을 생성한다.

클라우드 스토리지 $c(D_i^{[j,q]})$ → DM 서버 :

(a) 저장 확인, (b) T

DM 서버 :

- 수신된 T 값이 최근의 시각인지 확인한다.
- 수신된 저장 확인을 $c(D_i^{[j,q]})$의 공개키로 복호화한 결과, 그 내용이 자신이 보낸 수신 데이터 및 T의 연결 값을 해쉬 함수 적용한 결과와 같은지 확인한다.

3.2 데이터 검색 후 획득 과정과 데이터 복원

회사 TheCorp 소속의 사용자 Alice는 TheCorp 내의 인증 서버에게 자신이 해당 키워드 검색을 통해 데이터를 얻을 자격이 있음을 인증 받는다. 이후 Alice는 인증 서버에게 인증을 받았음을 증명할 수 있는 데이터를 검색 키워드와 함께 DM 서버에 전달한다. 이 과정에서 적절한 방법(인증 서버와 DM 서버 사이의 안전한 통신, 또는 인증 서버에서 발급하는 인증토큰)을 통해서 DM 서버가 Alice가 인증 서버로부터 인증 받았음을 알 수 있다고 가정한다.

Alice가 TheCorp의 DM 서버에게 데이터요청토큰 모음을 요청 및 수신하는 과정은 아래와 같다.

Alice → DM 서버 :

(a) Alice가 키워드 Q_{D_i}에 대해 인증 서버를 통해 데이터 접근을 허락받은 사실 증명(인증토큰 등), (b) Q_{D_i}

DM 서버 :

- Q_{D_i}를 키로 검색하여, 저장된 n, q, r, $h(h(D_i^0)\|\cdots\|h(D_i^{n-1}))$, $l(D_i^j)$, $K_{D_i^{[j,q]}}$, $c(D_i^{[j,q]})$, $j=0,...,n-1$ 를 얻는다.
- n 개의 $c(D_i^{[j,q]})$ 중 임의로 r개 $c(D_i^{[m(j),q]})$, $0\le m(j)\le n-1$, $j=0,...,r-1$를 택한다.
- 각 $j=0,...,r-1$ 별로, 아래와 같이 데이터요청토큰 $DToken(Q_{D_i}, c(D_i^{[m(j),q]}))$을 생성한다.

(a) 클라우드 스토리지: $c(D_i^{[m(j),q]})$, (b) 키워드(또는 데이터 고유 ID): Q_{D_i}, (c) 데이터 소유자(공유 범위): TheCorp, (d) DM 서버의 ID, (e) 유효 기간: V (시각 V까지만 이 데이터요청토큰이 유효함), (f) DM 서버의 서명: [(a)‖(b)‖(c)‖(d)‖(e)]에 해쉬 함수를 적용한 값을 DM 서버의 개인키로 암호화한 값

- 다음과 같이 암호화된 키 값들을 생성한다. (이하 $\oplus$는 bitwise XOR 연산을 의미함)

(a) $K_{Alice}^{D_i,Cipher}=K_{Alice}(H_{D_i^{[m(0),q]}}\|H_{D_i^{[m(1),q]}}\|\cdots\|H_{D_i^{[m(r-1),q]}})$, 단, $H_{D_i^{[m(j),q]}}=K_{D_i^{[m(j),q]}}\oplus K_{Alice}$, $j=0,...,r-1$

(b) $K_{Alice}^{D_i Partition} = K_{Alice}(\ n \| l(D_i^0) \| l(D_i^1) \| \cdots \| l(D_i^{n-1}) \| \ h(h(D_i^0) \| \cdots \| h(D_i^{n-1})) \ )$

(c) $K_{Alice}^{D_i Storages} = K_{Alice}(q \| r \| m(0) \| m(1) \| \cdots \| m(r-1))$

DM 서버 → Alice :

(a) $DToken(Q_{D_i}, c(D_i^{[m(j),q]}))$, $j=0,...,r-1$ (데이터요청토큰 모음이라 칭함)

(b) $K_{Alice}^{D_i Cipher}$, $K_{Alice}^{D_i Partition}$, $K_{Alice}^{D_i Storages}$

Alice는 이제 각 클라우드 스토리지에게 데이터 부분합을 요청하고, 수신된 데이터 부분합 들로부터 데이터 단위 D_i를 복원한다. 각 $j=0,...,r-1$ 에 대해, 아래 과정을 수행한다.

Alice → 클라우드 스토리지 $c(D_i^{[m(j),q]})$:

$DToken(Q_{D_i}, c(D_i^{[m(j),q]}))$

클라우드 스토리지 $c(D_i^{[m(j),q]})$:

- 수신한 $DToken(Q_{D_i}, c(D_i^{[m(j),q]}))$의 DM 서버 ID를 확인하고, 저장된 해당 DM 서버의 공개키로 수신 데이터의 서명을 검증한다.
- 수신한 $DToken(Q_{D_i}, c(D_i^{[m(j),q]}))$의 클라우드 스토리지가 $c(D_i^{[m(j),q]})$인지 확인한다.
- 수신한 $DToken(Q_{D_i}, c(D_i^{[m(j),q]}))$의 유효 기간 V가 지나지 않았음을 확인한다.
- 수신한 $DToken(Q_{D_i}, c(D_i^{[m(j),q]}))$의 키워드(또는 데이터 고유 ID) Q_{D_i} 값으로 저장된 자료 (Q_{D_i}, TheCorp, $E_{K_{D_i}^{[j,q]}}(\widetilde{D_i^{[j,q]}}), T$)를 검색하고, 수신한 데이터 소유자(공유 범위)가 저장된 값 TheCorp와 일치하는지 확인한다.
- 위의 검증/확인 과정 중 하나라도 실패하면 Alice의 요청에 대한 처리를 즉시 중지한다.

클라우드 스토리지 $c(D_i^{[m(j),q]})$ → Alice :

$E_{K_{D_i}^{[j,q]}}(\widetilde{D_i^{[j,q]}})$

수신한 $E_{K_{D_i}^{[j,q]}}(\widetilde{D_i^{[j,q]}})$, $j=0,...,r-1$, 로부터 Alice는 다음과 같이 데이터 단위 D_i를 얻는다.

Alice :

- $K_{Alice}^{D_i Cipher}$ 를 키 K_{Alice}로 복호화하여 $H_{D_i^{[m(j),q]}}$, $j=0,...,r-1$,를 얻고, $H_{D_i^{[m(j),q]}} \oplus K_{Alice}$ 를 계산하여 $K_{D_i^{[m(j),q]}}$, $j=0,...,r-1$,를 얻는다.
- 각 $j=0,...,r-1$에 대해 $E_{K_{D_i}^{[m(j),q]}}(\widetilde{D_i^{[m(j),q]}})$를 $K_{D_i^{[m(j),q]}}$로 복호화하여 $\widetilde{D_i^{[m(j),q]}}$를 얻는다.
- 각 $j=0,...,r-1$에 대해 $\widetilde{D_i^{[m(j),q]}}$를 압축해제(uncompress)하여 $D_i^{[m(j),q]}$를 얻는다.
- $K_{Alice}^{D_i Partition}$ 및 $K_{Alice}^{D_i Storages}$를 키 K_{Alice}로 복호화하여 D_i의 분할 및 수신한 자료의 q-부분합 관련 정보를 얻는다. 이들과 $D_i^{[m(j),q]}$, $j=0,...,r-1$,로부터 $D_i = D_i^0 \| D_i^1 \| \cdots \| D_i^{n-1}$를 얻는다. (방법은 아래 알고리즘에서 설명)
- 구한 D_i^j 로부터 $h(D_i^j)$, $j=0,...,n-1$, 및 계속해서 $h(h(D_i^0) \| \cdots \| h(D_i^{n-1}))$를 계산한다. 이 값이 $K_{Alice}^{D_i Partition}$을 복호화해서 얻은 $h(h(D_i^0) \| \cdots \| h(D_i^{n-1}))$ 와 같다면 $D_i = D_i^0 \| D_i^1 \| \cdots \| D_i^{n-1}$를 D_i로 확정한다.

수신한 q-부분합 $D_i^{[m(j),q]}$, $j=0,...,r-1$,를 $m(j)$ 값의 오름차순으로 정렬하여 $D_i^{[s(j),q]}$, $j=0,...,r-1$, 를 얻는다(즉, $\{m(0),...,m(r-1)\}=\{s(0),...,s(r-1)\}$이고, $0 \le s(0) < s(1) < ... < s(r-1) \le n-1$). D_i를 복원하기 위해서는 우선 인접한 q-부분합 들로부터 연속된 D_i의 부분들을 순서대로 연결하면서 중복된 부분은 한번 씩만 반영하는 과정을 반복한다. 그 결과 얻은 값을 필요 시 순환 쉬프트(circular shift) 시켜 D_i^0에서 시작하는 D_i 값을 얻는다(아래 알고리즘 참조).

Algorithm q-부분합 $D_i^{[m(j),q]}$, $j=0,...,r-1$, 으로부터 $D_i = D_i^0 \| D_i^1 \| \cdots \| D_i^{n-1}$ 를 구함

$R = \varnothing$; // R의 초기값. D_i의 부분 D_i^j 들이 // 연속적으로 R의 뒷부분에 연접됨

$pnum = 0$; // R에 포함된 D_i^j 의 개수

$k = 0$; // q-부분합 $D_i^{[s(0),q]}$ 에서 시작

$p = 0$; // 각 q-부분합 내에서 $p=0,...,q-1$ 의 // 순서로 진행

$offset[0..r-1] = 0$; // 각 q-부분합 $D_i^{[s(k),q]}$내의 // 데이터 부분을 추출할 때 사용할 시작 위치

while (true)
  for $p=0$ to $q-1$ do
    if $s(k)+p \neq s(k+1)$ then
  // 현재 q-부분합 내의 부분 중 다음
  // q-부분합에는 포함되지 않는 것
      $D_i^{[s(k),q]}$의 $offset[s(k)]$ 위치에서
      $l(D_i^{s(k)+p})$만큼 취해서 $D_i^{s(k)+p}$ 얻음 ;
      $R=R \| D_i^{s(k)+p}$; $pnum=pnum+1$;
      $offset[s(k)]=offset[s(k)]+l(D_i^{s(k)+p})$;
    else break // for loop에서 탈출하여,
      //다음 q-부분합을 기준으로 다시 반복
  end for
    if $pnum=n$ then break // R에 모든
    // 데이터 부분이 포함되면 while loop 탈출
    $k=(k+1) \bmod r$;
end while
if $s(0)>0$ then
  $R=$ R을 $\sum_{j=0}^{s(0)-1} l(D_i^j)$ 만큼 오른쪽 순환 쉬프트;
    // 그 결과, $R=D_i$가 됨

End of Algorithm

제안 방식이 실제 어떤 기관의 시스템에 적용되어 사용될 때, DM 서버는 클라우드 스토리지 들 중 고장이나 오동작으로 데이터 다운로드가 불가능한 스토리지가 발생하는지 스토리지 서비스 회사의 통보 등을 통해 계속 점검한다. 만약 서비스 제공 불능 상태의 스토리지가 통보되면, DM 서버는 사용자의 요구에 의해 데이터요청토큰 모음을 구성할 때 문제가 발생한 스토리지를 제외시킨다. 만약 사용자가 어떤 클라우드 스토리지에 데이터요청토큰을 전송한 상태에서 해당 클라우드 스토리지의 고장 또는 오동작을 감지한 경우, 사용자는 즉각 DM 서버에 해당 스토리지로부터 응답을 받지 못했음을 알린다. 이후 DM 서버는 문제가 된 스토리지 대신 새로운 스토리지를 포함하는 새 데이터요청토큰을 사용자에게 즉시 발급하고, 사용자는 데이터 검색 과정을 재차 시도한다. 이 때, 동시에 $(n-r+1)$ 개 이상의 서비스 제공 불능 스토리지가 생기지 않는 한(즉, n 개 중 최소 r 개의 스토리지가 정상인 한), 검색한 데이터 단위 D_i는 문제되는 스토리지(들)의 복구를 기다릴 필요 없이 즉시 사용자에 의해 검색 및 복원이 가능해진다. 결국, 데이터 복원은 어떤 경로로 문제가 생긴 스토리지(들)이 파악되었는지에 상관없이 가능하다.

Ⅳ. 보안성 및 효율성 분석

4.1 보안성

제안 방식은 데이터 기밀성과 무결성의 보장은 물론, 일부 클라우드 스토리지 들의 고장 등의 경우에도 데이터의 가용성을 보장함을 보인다.

4.1.1 기밀성

(1) *클라우드 스토리지 데이터의 기밀성 보장*

클라우드 스토리지에 저장되는 데이터는 우선 압축되어 원본 데이터의 크기가 줄어든 후, AES 암호화가 적용된다. 압축된 데이터는 원본 데이터보다 중복성이 적기 때문에 압축 후 암호화를 적용함으로써 암호 해독을 더 어렵게 만드는 효과가 있다. 각 클라우드 스토리지에 일부분만이 저장되는 데이터 단위의 전체 내용을 알기 위해서는 여러 클라우드 스토리지에 서로 다른 키로 AES 암호화된 채 저장된 데이터 일부분들에 대한 동시 해독이 필요하다. 여러 클라우드 스토리지 들이 서로 연합하더라도, 해독하려는 데이터 단위가 몇 부분으로, 어떤 순서로 분할되어 있는지 모르기 때문에 전체 데이터 단위를 복원하는 것은 사실상 불가능하다.

(2) *사용자 비밀 키의 기밀성 보장*

사용자 Bob이 Alice의 비밀 키를 알아내고자, Alice가 요청했던 것과 같은 데이터 단위를 요청한다고 하자. Bob은 자신의 비밀 키를 이용하여 이 데이터 단위를 복원하면서, 수신한 각 데이터 부분을 AES 암호화하는데 사용했던 키 값들을 얻을 수 있다. Alice와 Bob의 요청에 대해 DM 서버가 응답을 할 때 공교롭게 동일한 r 개의 클라우드 스토리지 들이 선택된 경우에도, Alice의 비밀 키는 Bob에게 노출되지 않는다. DM 서버가 Alice에게 제공한 $K_{Alice}^{D_i,Cipher}$와 Bob에게 제공한 $K_{Bob}^{D_i,Cipher}$는 원문이 다르며, $K_{Alice}^{D_i,Partition}$, $K_{Alice}^{D_i,Storages}$의 경우 같은 원문을 다른 키(Bob의 비밀키)로 AES 암호화 한 것이 Bob에게 $K_{Bob}^{D_i,Partition}$, $K_{Bob}^{D_i,Storages}$로 주어지기 때문이다. 따라서 Alice의 비밀 키를 알기 위해 Bob은 알려진(known) 원문 및 이의 암호문을 가지고 암호화에 사용한 키를 찾아야 한다. 그런데 AES-256 의 14 개의 전체 라운드(round)를 모두 쓰는 경우 알려진 원문 공격을 포함한 어떤 공

격도 실질적 위협이 되지 못하기에, 이는 불가능하다.

4.1.2 무결성

사용자는 복원 시 데이터 단위를 구성하는 부분 각각의 해쉬 값을 계산하고, 이들을 연접한 후 다시 해쉬 값을 계산한 결과가 DM 서버로부터 받은 해쉬 값과 같은지 확인함으로써 데이터 무결성을 확인한다. 각 클라우드 스토리지가 저장하고 있는 것은 해당 데이터 단위의 하나의 부분합으로 데이터 부분들 전체를 포함하지 않는다. 또한 이 데이터 단위의 부분합 들을 저장할 때 각 클라우드 스토리지는 서로 다른 암호화 키를 사용해서 부분합 전체에 대해 암호화(즉, 부분 각각에 대해 암호화한 것이 아니고 여러 연속된 부분들에 대해 암호화)한 결과만을 저장하고 있다. 따라서 다수의 클라우드 스토리지에 대해 각각 AES 암호화를 모두 풀지 않는 한 데이터 부분 각각에 대해 해쉬 값을 계산하고 이들을 연접한 결과에 해쉬 함수를 적용한 결과는 계산해낼 수 없다. 그 결과 사고에 의한 데이터 변경은 물론, 클라우드 스토리지에 의한 고의적 데이터 변경이 발각되지 않을 가능성은 사실상 없다.

4.1.3 데이터 접근 즉시 거부 및 유출 범위 평가

클라우드 스토리지를 사용하는 기관은 해고된 사용자에 대해 해고 시점부터 어떤 데이터요청토큰도 발급하지 않는다. 그 결과로 이 해고자는 이전에 획득했던 데이터 단위들 이외에 추가로 새로운 데이터 단위에 접근하는 것이 즉시 불가능해진다. 왜냐하면 모든 데이터요청토큰에 각각 요청하는 데이터 단위가 명시되어 있고, DM 서버를 통하지 않고는 클라우드 스토리지를 속일 수 있는 데이터요청토큰을 만들 수 없기 때문이다.

제안 방식은 사용자별로 요청한 데이터 키워드(또는 데이터 고유 ID)에 대해 고유한 토큰을 발급하기 때문에 DM 서버에 사용자별, 키워드별 토큰 발급 내역 로그를 남길 수 있고, 이 로그로 부터 특정 사용자가 현재까지 접근한 모든 데이터 단위를 규명할 수 있다. 사용자별 비밀 키가 다르므로 어떤 사용자도 다른 사용자에게 전송된 메시지를 통해 요청된 데이터 단위를 복원할 수 없기 때문에 이와 같은 확정이 가능하다. 물론, Bob이 자신이 적법하게 다운로드 받은 데이터 단위를 Alice에게 직접 제공하는 개인적 데이터 공유는 막지 못한다.

4.1.4 가용성 향상 분석

데이터 단위 D_i 전체를 하나의 클라우드 스토리지에만 저장하는 경우와 비교하여 제안 방식이 어느 정도의 가용성 향상을 가져오는지를 데이터 단위를 복원할 수 있는 확률 계산을 통해 살펴본다. 모든 클라우드는 서로 독립적으로 동작하며, 비교 편의상 각 클라우드 스토리지의 고장 확률은 동일하게 f라고 가정한다. 단일 클라우드 스토리지에 전체 D_i를 저장하는 방식에서 D_i를 복원(획득)할 수 있는 확률은 $1-f$ 가 된다. 이제, 제안 방식에 따라 D_i의 q-부분합 하나씩을 n 개의 클라우드 스토리지에 각각 저장했다면, D_i를 복원할 수 있는 확률은 P(n 개 중 임의의 r 개 이상의 스토리지가 정상적으로 동작)

$$=\sum_{i=r}^{n}\binom{n}{i}(1-f)^i f^{n-i} = \quad 1-\sum_{i=0}^{r-1}\binom{n}{i}(1-f)^i f^{n-i}$$

이 된다. 예를 들어, $f=0.01$, $n=5$, $q=3$, $r=3$ 일 때, D_i 복원 확률은 0.99 대 0.999990149(= $1-\sum_{i=0}^{2}\binom{5}{i}(1-0.01)^i \cdot 0.01^{5-i}$)로 제안 방식에서 단일 스토리지에 저장하는 방식보다 훨씬 복원 확률이 높아짐을 확인할 수 있다.

4.2 효율성

제안 방식에서 키 관리가 효율적이고, 사용자에 투명한 암호화 키의 변경이 가능함을 보이고, 데이터의 부분적 중복으로 인한 스토리지 저장 데이터양 및 통신 데이터양의 증가 정도를 분석한다.

4.2.1 DM 서버에서의 키 관리

(1) *데이터 단위별 저장되는 키 값들*

DM 서버는 각 데이터 저장 단위 D_i에 대하여, 다음과 같은 정보를 저장한다.

키워드(또는 데이터 고유 ID), n, q, r, $h(h(D_i^0)\|\cdots\|h(D_i^{n-1}))$, $l(D_i^j)$, $K_{D_i^{[j,q]}}$, $c(D_i^{[j,q]})$, $j=0,...,n-1$

여기서, AES 키 $K_{D_i^{[j,q]}}$의 크기 및 해쉬 함수 적용 결과 $h(D_i^j)$ 는 각각 256 비트, 키워드(또는 데이터 고유 ID)의 크기는 128 비트로 간주하고, 나머지 값들 중 $l(D_i^j)$ 는 32 비트, n, q, r, $c(D_i^{[j,q]})$는 각각 16 비트면 충분히 표현 가능하다(사실 n, q, r 중 하나의 값은 나머지 두 값으로부터 계산할 수 있으므로 생략해도 무관함). 어떤 기관이 신뢰성 및 가격 경쟁력 등을 고려하여 10개의 클라우드 스토리지 서비스 업체와 계약을 맺었다고 가정하자(현실적으로 계약 대상

서비스 제공자들의 수가 이보다 크게 많지는 않을 것으로 판단됨). 이제 $t=10$이 됨에 따라, 각 데이터 단위의 분할된 부분의 수 n은 10 이하가 된다. 결국, 각 D_i에 대하여 DM 서버는 $128+(16\times3)+256+(32+256+16)\times n=432+304n\le3472$ 비트=434 바이트$\approx0.42\,KB$의 정보를 저장한다. 저장 장소의 크기는 데이터 단위의 수에 선형 비례하기 때문에 제안 방식은 확장성을 가진다고 볼 수 있다. 예를 들어, 어떤 대형 기관에서 1,000,000 개 정도의 데이터 단위를 관리한다 하더라도, DM 서버는 단지 $410\,MB$ 정도의 저장 공간만을 필요로 한다.

또한 DM 서버는 기관의 모든 사용자에 대해 각각 고유한 비밀 키를 저장하여, 동일한 데이터를 요청한 경우라도 사용자별로 다른 $K_{user}^{D_i,Cipher}$ 값을 제공한다. 사용자 비밀 키의 크기는 256 비트이고 기관의 전체 사용자수를 N이라 하면, 전체 사용자에 대해 (사용자, 사용자 비밀 키) 쌍을 저장하는데 필요한 공간은 $(k+256)N$ 비트(단, $2^{k-1}<N\le2^k$)가 된다. 사용자 수 $N=10{,}000$ 정도의 대형 기관에 대해서도 $k=14$이므로 필요 공간은 $(14+256)\cdot10{,}000=2{,}700{,}000$ 비트(=337,500 바이트 $\approx330\,KB$)에 불과하다.

(2) *사용자에 투명한 데이터 단위 암호 키 변경*

데이터 단위의 소유주는 데이터 단위의 내용 변경, 오래된 키의 주기적 폐기, 혹은 크래킹 등의 이유로 어떤 데이터에 대해 새로운 암호 키를 이용해 암호화를 다시 적용하고자 할 수 있다. 새로운 키로 암호화된 데이터가 생길 때마다 이 데이터를 다운로드받을 가능성이 있는 모든 사용자에게 새로운 키 정보를 알리는 것은 매우 비효율적이다. 데이터 단위 D_i의 q-부분합 $D_i^{[x,q]}$이 기존에는 키 $K_{D_i^{[x,q]}}^{\text{old}}$ 로 암호화되었으나, 이제 새로운 키 $K_{D_i^{[x,q]}}^{\text{new}}$ 로 암호화한다고 가정하자. 그러면, 이 q-부분합 $D_i^{[x,q]}$에 대한 Alice의 요청은 새로운 암호화 이전과 이후로 구분하여 각각 다르게 처리(다른 $K_{Alice}^{D_i,Cipher}$를 제공)된다. DM 서버가 $K_{Alice}^{D_i,CipherOld}=K_{Alice}(\cdots\|H_{D_i^{[x,q]}}^{\text{old}}\|\cdots)$, 단, $H_{D_i^{[x,q]}}^{\text{old}}=K_{D_i^{[x,q]}}^{\text{old}}\oplus K_{Alice}$,로 응답했을 때 Alice는 수신된 q-부분합에 대해 $H_{D_i^{[x,q]}}^{\text{old}}\oplus K_{Alice}$, 즉 $K_{D_i^{[x,q]}}^{\text{old}}$ (왜냐하면 $H_{D_i^{[x,q]}}^{\text{old}}\oplus K_{Alice}=(K_{D_i^{[x,q]}}^{\text{old}}\oplus K_{Alice})\oplus K_{Alice}=K_{D_i^{[x,q]}}^{\text{old}}$)를 이용하여 복호화를 시도한다. 한편 DM 서버로부터 $K_{Alice}^{D_i,CipherNew}=K_{Alice}(\cdots\|H_{D_i^{[x,q]}}^{\text{new}}\|\cdots)$, 단, $H_{D_i^{[x,q]}}^{\text{new}}=K_{D_i^{[x,q]}}^{\text{new}}\oplus K_{Alice}$,를 받으면, Alice는 수신된 q-부분합에 대해 $H_{D_i^{[x,q]}}^{\text{new}}\oplus K_{Alice}$, 즉 $K_{D_i^{[x,q]}}^{\text{new}}$ 를 이용하여 복호화하려 할 것이다. 결국 Alice는 수신한 q-부분합이 새로운 키로 다시 암호화된 점을 알 필요가 없이(투명하게), 자신의 비밀 키 K_{Alice}만을 알고 있으면 데이터를 복호화할 수 있다. 즉, DM 서버는 언제든 새로운 키를 사용하여 데이터를 암호화할 수 있고, 어떤 키로 암호화된 데이터를 사용자에게 제공할지를 사용자와 무관하게 결정할 수 있다.

4.2.2 저장 데이터양 및 성능 상 부담 증가 정도

데이터 단위의 분할된 부분들의 크기가 서로 같다고 가정하자. 제안 방식은 기존 방식에 비해 데이터 단위 당 저장량에서 $\alpha\cdot q$ 배의 차이를 보이는데, 만약 기존 방식에서 압축 후 저장하는 방법을 택한다면 그 배수가 q 로 약간 더 늘어난다(표 1 참조). [따름정리 1]에 의해 $(q-1)$ 값이 동시 고장을 감내할 수 있는 클라우드 스토리지의 개수이기 때문에, 만약 어떤 데이터 단위가 기관의 입장에서 매우 중요하다면 q 값을 크게 잡아(이 경우 전체 데이터의 양은 더 늘어나지만) 더 적은 수의 클라우드 스토리지 들을 통해서도 이 데이터 단위를 복원하도록 조정할 수 있다. 제안 방식에서는 이와 같이 데이터 단위별로 n 및 (q,r) 값을 적절히 선택함으로써 가용성과 저장 데이터양

표 1. 데이터 단위 당 스토리지에 저장되는 전체 데이터양 및 다운로드 받는 통신 데이터양 측면에서의 비교

	제안 방식	기존 방식	비고
클라우드 스토리지에 저장되는 전체 데이터양	$(\alpha\cdot(q/n)\cdot l(D_i))\cdot n$ $=\alpha\cdot q\cdot l(D_i)$ $(<q\cdot l(D_i),\ \because\alpha<1)$	$l(D_i)$	α: 평균 압축률($\alpha<1$) n: 데이터 단위 D_i를 분할한 결과 부분의 수 q: 클라우드 스토리지 당 저장하는 D_i의 부분의 수 r: 검색(복원) 프로토콜에서 다운로드 받는 부분합의 수 $l(D_i)$: 데이터 단위 D_i의 크기(길이)
클라우드 스토리지로부터 다운로드 받는 통신 데이터양	$(q\cdot l(D_i)/n)\cdot r$	$l(D_i)$	

사이의 비중을 데이터 단위 수준에서 세부 조절하는 것이 가능하다.

한편, 제안 방식을 적용하면 데이터 단위 당 기존 방식에 비해 더 많은 r 개의 클라우드 스토리지에 접근하며, 중복으로 인해 프로토콜 상에서 더 많은 양의 데이터를 수신해야 하므로 성능에 영향을 미친다. 사용자가 느끼는 응답 시간에 가장 큰 비중을 차지하는 프로토콜 상의 데이터 수신 시간을 기준으로 볼 때, 제안 방식은 기존 방식에 비해 데이터 단위 당 평균적으로 $q \cdot r/n$ 배 더 많이 걸린다(표 1 참조). 이 값은, 예를 들어, $n=5,\ q=3, r=3$이라면, 감내할 정도인 1.8 배 수준에 해당한다.

Ⅴ. 관련 연구

최근 암호화 기법을 활용하여 클라우드 스토리지의 데이터 기밀성과 무결성을 보장하려는 Cryptographic Cloud Storage 연구 결과[4]가 발표되었는데, 이 연구에서는 모든 데이터가 클라우드 스토리지의 사용자에 의해 제어되도록 하기 위해 데이터 암호화, 데이터 무결성 검증, 클라우드에 제시할 토큰 생성, 사용자 인증 같은 처리 모듈을 클라우드 클라이언트 쪽에 도입한 구조를 제안하였다. 이 구조에 따르면, 개인 또는 기업 사용자의 별도 서버를 통해 암호화된 데이터만이 클라우드 스토리지에 저장되므로, 암호화 키가 보호되는 한 스토리지 내의 데이터는 어느 위치에, 어떻게 저장되더라도 유출에 안전하다. 본 논문의 제안 방식은 클라우드 스토리지 제공자가 데이터를 제어하지 않는다는 측면에서 [4]와 같은 아이디어를 채택하고 있지만, 암호화된 후 저장되는 데이터에 대해 분할 및 부분 중복을 제안하여 기밀성과 무결성의 보장 이외에 데이터 가용성을 크게 높였다는 특징을 가진다. 사용자(클라이언트) 측에서 데이터 제어를 담당한다는 개념은 사용자가 클라우드에 입력하는 데이터를 난독화(obfuscation)시켜 보냄으로써 사용자의 프라이버시를 보호하는 클라우드 계산 서비스에 관한 연구[5]에서도 활용하고 있지만, 이 연구는 스토리지 관련 응용 부분은 다루지 않았다.

클라우드 클라이언트가 아니라 클라우드 시스템 내부에서 보안 문제를 다루고자 하는 여러 시도들이 발표되었다[6,7,9,10]. 클라우드 스토리지의 보안 관련하여, [8]에서는 해쉬 함수를 이용한 무결성 검증 기법, 클라우드 스토리지가 데이터를 저장하고 있음을 확인하기 위한 암호화 프로토콜, 다수에 의한 동시 데이터 접근 시 쓰기와 읽기 사이의 일관성 보장 기법 등과 더불어, 클라우드 스토리지에 문제가 발생하는 경우 Byzantine 고장 감내 프로토콜의 사용이 가능하다고 소개하고 있다. 하지만, Byzantine 고장 감내 프로토콜의 접근 방식은 클라우드 서비스 제공자의 시스템 내의 중복화 및 제어를 통한 것으로 본 논문의 제안 방식과 차이가 있다. 한편, [7]은 신뢰 컴퓨팅(Trusted Computing) 기술의 사용을 제안하는데, 이는 클라우드 서버에 장착한 신뢰 컴퓨팅 모듈을 통해 클라우드 시스템의 모니터 결과를 고객에게 신뢰성 있게 전달하자는 것이다. 이러한 연구 결과들은 클라우드 사용자 측면에서 채택할 수 있는 본 논문의 연구 결과와 함께 서로 병행해서 활용하는 것이 가능하다.

데이터의 중복 저장을 통해 가용성을 높이는 것은 잘 알려진 기법으로, 클라우드 컴퓨팅 분야에서는 데이터를 지역적으로 분산된 여러 클라우드 데이터 센터들에 중복시키는 방식[9,10] 등이 제안되었다. 하지만, 이런 기법들은 기본적으로 서비스 제공자의 시스템 차원에서의 중복을 논하고 있기 때문에, 클라우드 서비스의 사용자 입장에서는 자신의 데이터에 대한 통제 권한을 보장받지 못한다. 이에 비해, 본 논문의 제안 방법은 사용자 주도의 가용성 향상을 도모하기 위해 데이터의 분할, 부분 중복 후 저장에 대한 제어를 전적으로 클라이언트 측에서 담당하면서, 데이터 단위별로 세밀한 가용성 수준의 조절이 가능하다는 특징을 가진다.

Ⅵ. 결 론

높은 보안성을 필요로 하는 클라우드 스토리지 서비스 고객은 신뢰도가 가장 높다고 평가되는 업체 하나 또는 둘 정도만을 선택하는 것이 일반적이었다. 이 경우 고객인 회사나 기관의 데이터를 저장하는 서비스 제공자는 해당 고객의 데이터를 독점에 가까운 지위에서 저장하게 되며, 서비스 수준이나 비용 등을 협의할 때 고객에 비해 우월적인 입장에 서게 될 가능성이 크다. 물론 클라우드 스토리지 제공자들은 자체적으로 여러 데이터 센터에 중복 저장 및 백업 등 다양한 방법으로 고객의 데이터를 보호하려 애쓰고 있으나, 실제 발생한 여러 사례들에서 보듯이 어떤 서비스 제공자도 완벽한 고객 데이터 보호 및 가용성 제공을 보장하지는 못한다. 본 논문의 제안 방식은 고객이 다수의 클라우드 스토리지 제공자들을 선택하며, 검색 단위인 각 데이터 단위에 대해 분할된 일부분만을 각 스토리지가 저장하도록 제어함으로써, 현 수준의 클라우드 스토리지 서비스를 사용하면서도 데이터 가용성

을 크게 향상시킬 수 있다. 즉, 어떤 데이터 단위에 대해 다수(q 개 이상)의 클라우드 스토리지에 저장된 해당 데이터 부분합 들이 동시에 모두 접근 불가가 되는 극히 예외적인 경우를 제외하고는 항상 데이터 단위의 복원이 가능해진다. 또한, 클라우드 스토리지 제공자 누구도 특정 데이터 단위의 배타적 저장에 따른 지위를 보유하지 못한다.

제안 방식은 기관 소속의 사용자들을 또 다른 보안 취약성으로 간주하여, 데이터 단위에 접근할 때 사용자별로 각각 DM 서버에게 받은 고유한 키 값을 사용하도록 함으로써 타 사용자가 요청한 데이터에 대한 접근을 원천적으로 봉쇄하였다. 또한, 이런 구분을 통해 DM 서버는 사용자별 접근 데이터 단위 목록로그를 유지할 수 있어서 필요 시 사용자별로 유출된 데이터 단위들을 파악할 수 있다.

향후 데이터 단위 분할 시 각 클라우드 스토리지의 신뢰성 및 저장 비용을 연계시켜 최적화된 분할을 적용하는 기법을 제안 방식에 추가할 예정이다. 그 외, 자주 갱신되는 데이터 단위들을 주로 다룰 때 읽기와 쓰기 트랜잭션 사이에서 데이터 일관성을 유지하는 방식 및 검색 시 여러 검색어의 다양한 조합을 허용하면서 대량 데이터의 효율적 검색을 가능케 하는 기법의 연구가 필요하다

참 고 문 헌

〔1〕 D. Terry(Ed), ACM Tech Pack on Cloud Computing, http://techpack.acm.org/cloud/, 2010.

〔2〕 B. Hayes, "Cloud Computing", *Communications of the ACM*, 51(7), pp. 9-11, July 2008.

〔3〕 M. Creeger(Moderator), "Cloud Computing: An Overview", *ACM Queue*, 7(5), pp. 1-5, June 2009.

〔4〕 S. Kamara, K. Lauter, "Cryptographic Cloud Storage", *in Proc. of the 14th Int'l Conf. on Financial Cryptography and Data Security (FC '10)*, 2010.

〔5〕 M. Mowbray, S. Pearson, "A Client-Based Privacy Manager for Cloud Computing", *in Proc. of the 4th Int'l ICST Conf. on Communication System Software and Middleware(COMSWARE '09)*, 2009.

〔6〕 J. Bardin et al., "Security Guidance for Critical Areas of Focus in Cloud Computing", *Technical Report, Cloud Security Alliance*, April 2009.

〔7〕 R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina, "Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control", *in Proc. of 2009 ACM Workshop on Cloud Computing Security(CCSW '09)*, 2009.

〔8〕 C. Cachin, I. Keidar, A. Shraer, "Trusting the Cloud", *ACM SIGACT News*, 40(2), pp. 81-86, June 2009.

〔9〕 D.J. Abadi, "Data Management in the Cloud: Limitations and Opportunities", *IEEE DE Bulletin*, 32(1), pp. 3-12, 2009.

〔10〕 L.C. Voicu, H. Schuldt, "How Replicated Data Management in the Cloud can benefit from a Data Grid Protocol – The Re:GRIDiT Approach", *in Proc. of the 1st Int'l Workshop on Cloud Data Management(CloudDB '09)*, 2009.

〔11〕 http://blogs.wsj.com/digits/2009/03/08/1214/

〔12〕 http://blogs.sun.com/gbrunett/entry/amazon_s3_silent_data_corruption

〔13〕 http://blogs.zdnet.com/projectfailures/?p=999

〔14〕 http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9130682&source=NLT_PM

〔15〕 http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9117322

〔16〕 http://status.aws.amazon.com/s3-20080720.html

〔17〕 http://www.thewhir.com/web-hosting-news/103108_FlexiScale_Suffers_18_Hour_Outage

박 준 철 (Jun-Cheol Park) 종신회원

한국통신학회 논문지 제36권 제9호 참조

현재 홍익대학교 컴퓨터공학과 교수