# 실시간 데이터 이중화를 위한 고속 블록 전송기술 설계

한재승[O], 안재훈(교신저자)[*], 김영환[*], 박창원[*]
[O*]전자부품연구원 휴먼케어시스템연구센터
e-mail: wotmd127@keti.re.kr[O], corehun@keti.re.kr[*], {yhkim93, parkcw}@keti.re.kr[*]

# Design of high-speed block transmission technology for real-time data duplication

JaeSeung Han[O], Jae-Hoon An[*], Young-Hwan Kim[*], Chang-Won Park[*]
[O*]Human Care System Research Center, Korea Electronics Technology Institute.

● 요 약 ●

본 논문에서는 데이터 이중화 저장시스템의 장애발생으로 인한 백업서버 데이터 손실을 보호하기 위해 무 손실 실시간 데이터 이중화 시스템 설계방안을 제안한다. 이는 원본서버의 데이터와 백업서버의 데이터가 특정 시점 T에서 100% 일치하지 않는 비동기 방식을 동기방식으로 해결하기 위한 시스템 설계 제안으로, 원본서버의 데이터 생성과 동시에 실시간 데이터 백업을 목적으로 한다. 이를 위해 전송단계에서 필요한 가장 빠른 압축인 LZ4 압축 알고리즘을 기반으로 Intel AVX 명령어를 사용하여 보다 압축속도를 증진시켜 실시간 시스템을 구축한다. 또한 전송 중 보안위협으로부터 보호하기 위해 Key 전달 기법과 AES 암호화 알고리즘에 대해 기술한다.

**키워드:** 데이터 보호(Data Protection), 실시간(Real Time), AVX(Advanced Vector Extensions)

## I. Introduction

인터넷과 스마트폰의 보급은 데이터 폭발현상을 일으켰고 이는 빅데이터, 클라우드, 인공지능, IoT, VR/AR, Digital Twin 등 주요 IT기술의 기반 데이터를 확보할 수 있게 되었다. 하지만 이러한 기술은 높은 데이터 의존도를 가지기 때문에 데이터 보호는 선택이 아닌 필수가 되었다. 많은 국가와 기업들은 데이터 보호를 위해 별도의 백업서버를 구축하고 데이터를 이중 저장 관리하여 데이터 소실을 예방 및 복구한다. 하지만 기존 이중화 시스템은 실시간 데이터 동기화 기능은 제공되지 않아 백업서버 장애 발생 시 100% 데이터 복구는 어려운 상황이다. 이에 본 논문에서는 무 손실 데이터 동기화를 목적으로 실시간 데이터 이중화 방법과 시스템을 제시한다.

## II. Preliminaries

### 1. Related works

#### 1.1 국내 및 해외 동향

많은 기업들은 정보보호의 중요성을 이미 인식하고 데이터 보호를 위해 별도의 백업서버를 두어 관리 하고는 있다. 하지만 기존 시스템은 분 단위의 데이터 저장 시스템 때문에 예기치 못한 오류나 자연재해 등 특정 상황에 장애 발생 시 100% 데이터를 복구하지 못하고 영구손실이 발생하여 개발자들의 개발기간 증가, 고객 데이터 유실 등의 문제가 발생하고 있다.

#### 1.2 LZ4 (Lempel Ziv 4)

LZ4는 무 손실 압축 알고리즘 중 하나로 압축률은 조금 떨어지지만 가장 빠른 압축속도를 가진다.

Table 1. 압축 알고리즘 비교[1]

| Compressor | Ratio | Compression | Decompression |
|---|---|---|---|
| memcpy | 1.000 | 13100 MB /s | 13100 MB /s |
| **LZ4 default (v1.8.2)** | **2.101** | **730 MB/s** | **3900 MB/s** |
| LZO 2.09 | 2.108 | 630 MB/s | 800 MB/s |
| QuickLZ 1.5.0 | 2.238 | 530 MB/s | 720 MB/s |
| Snappy 1.1.4 | 2.091 | 525 MB/s | 1750 MB/s |
| Zstandard 1.3.4 -1 | 2.877 | 470 MB/s | 1380 MB/s |
| LZF v3.6 | 2.073 | 380 MB/s | 840 MB/s |
| zlib deflate 1.2.11 -1 | 2.730 | 100 MB/s | 380 MB/s |
| LZ4 HC -9 (v1.8.2) | 2.721 | 40 MB/s | 3920 MB/s |
| zlib deflate 1.2.11 -6 | 3.099 | 34 MB/s | 410 MB/s |

### 1.3 SIMD[2] (Single Instruction Multiple Data)

SIMD는 하나의 명령어로 여러 개의 데이터를 벡터화 하여 한 번에 처리하는 기법으로 프로그램 수행 속도를 증진시킨다. SIMD의 종류로는 SSE(Streaming SIMD Extensions), AVX(Advanced Vector Extensions)가 있으며 처리할 수 있는 최대 bit수 제한이 있다.

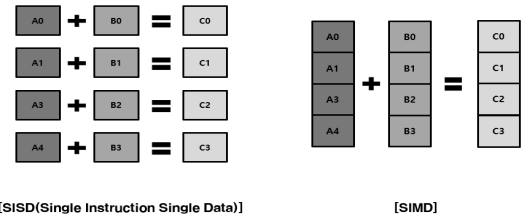


Fig. 1. SISD/SIMD 처리 비교

### 1.4 AES[3] (Advanced Encryption Standard)

AES는 고급암호화 표준으로, 암호화 할 때의 Key가 복호화 할 때의 Key와 같은 대칭형 알고리즘이다. 이는 비대칭형 암호화 알고리즘과 비교해서 암호화 속도가 빠르지만, 상대방과 안전한 Key 공유가 어렵고 통신 상대방이 많아질수록 Key 관리에 어려움이 있다. AES는 SubByte, ShiftRow, MixColumns, AddRoundKey 알고리즘을 사용한다[4].

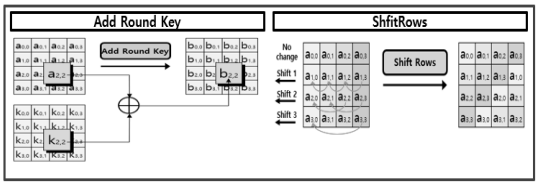


Fig. 2. AES 암호화 알고리즘 A (ref. Wiki)

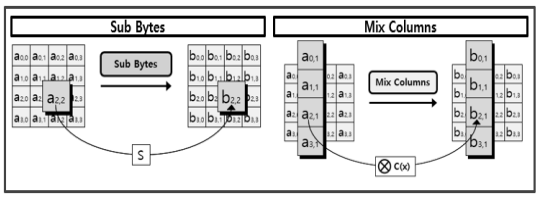


Fig. 2. AES 암호화 알고리즘 B (ref. Wiki)

### 1.5 RSA (Public Key Cryptosystem)

RSA는 비대칭 암호화 알고리즘으로 송신자가 암호화에 쓰는 Public Key와 수신자가 복호화에 쓰는 Private Key를 사용한다. 이는 Public Key로 암호화 했다면, 반드시 Private Key로만 복호화가 가능하다.

비대칭 암호화 알고리즘은 대칭형 암호화 알고리즘에 비해 암호화 속도가 느리지만 Key 공유가 용이하다.

## III. The Proposed Scheme

### 1. 시스템 구성 및 설계

백업 서버의 장애 발생 시 데이터 유실 원인은 원본 서버와의 데이터 동기화가 실시간으로 이루어지지 못하기 때문이다. 본 논문에서 제안하는 실시간 데이터 이중화 시스템은 백업서버 장애 발생 시 복구되지 않는 데이터를 100% 복구하기 위해 다음 Fig3과 같은 시스템 설계를 제안한다.

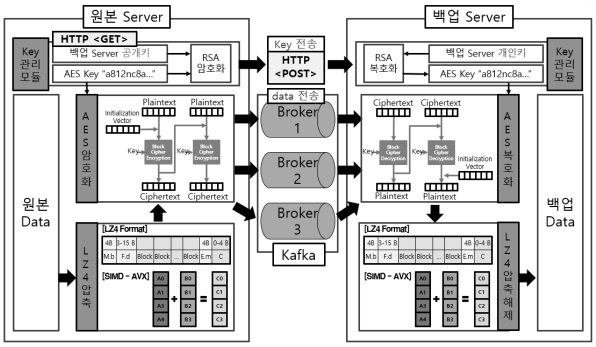


Fig. 3. System Architecture

Fig3의 모듈은 크게 세 부분으로 나뉘며, 각 모듈의 역할은 다음 Table2와 같다.

Table 2. System 구성 모듈

| 번 | 모듈명 | 설명 |
|---|---|---|
| 1 | Key 관리모듈 | AES 암호화에 필요한 Key를 안전하게 전달한다. |
| 2 | LZ4 압축모듈 | 원본 데이터 전송 전 Intel AVX 명령어를 사용해 빠른 속도로 압축한다. |
| 3 | AES 암호화모듈 | 원본 데이터 전송 전 데이터 암호화를 통해 보안성을 증가시킨다. |

1. Key 관리모듈은 데이터 암호화에 필요한 AES Key를 원본서버와 백업 서버에 안전하게 전달하는 역할을 한다. 2. LZ4 압축모듈은 LZ4 압축 알고리즘을 기반으로 Intel AVX 명령어를 적용하여 원본데이터 크기를 빠르게 압축하는 역할을 한다. 3. AES 암호화 모듈은 AES 암호화 알고리즘을 사용하여 데이터 전송 시 보안위협으로부터 정보보호의 목적을 갖는다.

2. 모듈 구성 및 설계

2.1 Key 관리 모듈[5]

데이터 이중화에 앞서 데이터를 안전하게 전달하기 위해 암호화는 필수적이다. 이 모듈은 암호화에 필요한 AES Key를 원본서버와 백업서버에 안전하게 제공하는 것을 목적으로 한다. 이는 RSA암호화 알고리즘을 통한 Key전달 기법을 사용하여 원본서버의 AES Key를 백업서버에 전달한다. 이는 다음 Fig4와 같은 구조를 가진다.

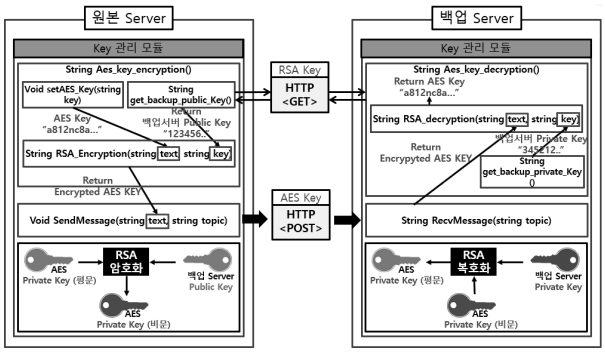


Fig. 4. Key 관리 모듈 Architecture

원본서버는 HTTP Client 역할을 하며, 백업서버는 HTTP Server 역할을 한다. Server 가동 시작 시 RSA Private Key를 생성하고 Client의 접속을 기다린다. 이 때 Client가 Get 요청을 보내면 Server는 RSA Private Key로부터 RSA Public Key를 생성하여 Client에게 Response 한다. Client는 RSA Public Key를 이용해 입력된 AES Key를 암호화 시킨 후 Post 요청으로 Server에 전달한다. 이를 받은 Server는 RSA Private Key를 사용하여 AES Key를 복호화 한다. 이러한 일련의 과정을 거쳐 동일한 AES Key를 Client와 Server가 가지게 된다. 이를 Key 전달과정이라 하며, 아래 Fig5는 이와 같은 과정을 보인다.

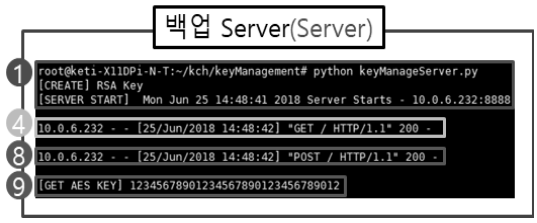


Fig. 5. Key 관리 모듈 구현 A


Fig. 5. Key 관리 모듈 구현 B

Fig 5의 흐름 및 설명은 아래 Fig 6에서 나타내며, S는 Server, C는 Client를 나타낸다

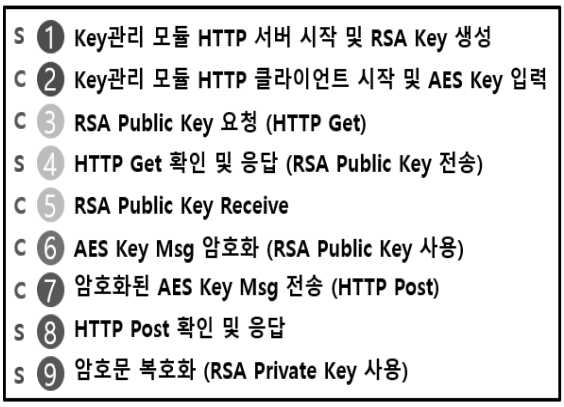


Fig. 6. Key 관리 모듈 흐름도

## 2.2 LZ4 압축 모듈

본 논문에서 제안하는 실시간 데이터 이중화 시스템의 핵심은 '압축이 얼마나 빠른 시간 내에 완료 되는가' 이다. 따라서 가장 빠른 압축 알고리즘 LZ4를 기반으로 하며, Intel AVX 명령어를 사용하여 압축속도를 향상시킨다. 다음 Fig 7은 Intel AVX 명령어의 성능 테스트를 위해 20억 개의 배열 a와 b를 할당하고, 모든 원소에 (a[i]*b[i])(1/8)수식을 SISD 명령어(Native)와 SIMD(SSE, AVX) 명령어를 적용하여 수행시간을 측정하였다.

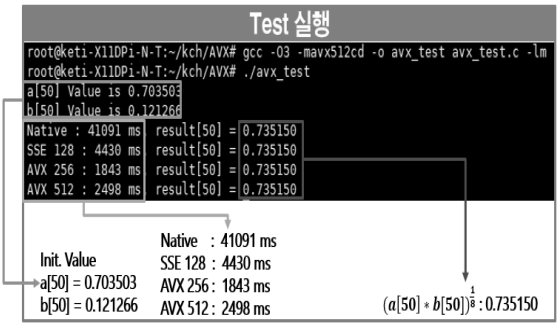


Fig. 7. SISD / SIMD 수행시간 테스트

위 결과에서 볼 수 있듯이 SISD와 SIMD는 같은 계산 결과를 보이지만 속도 면에서 큰 차이를 보인다. 따라서 Intel AVX 명령어를 LZ4 압축 알고리즘에 적용한다면 보다 빠른 속도로 압축이 가능할 것이다. 때문에 LZ4 압축 알고리즘의 적절한 함수에 Intel AVX 명령어를 삽입하는 것이 중요한데, LZ4 Format은 Block이라는 일정 크기의 데이터가 반복해서 압축되므로 반복되는 함수에 Intel AVX 명령어 적용 시 성능향상을 가질 수 있다. 다음 Fig 8은 압축 흐름도를 나타낸다.

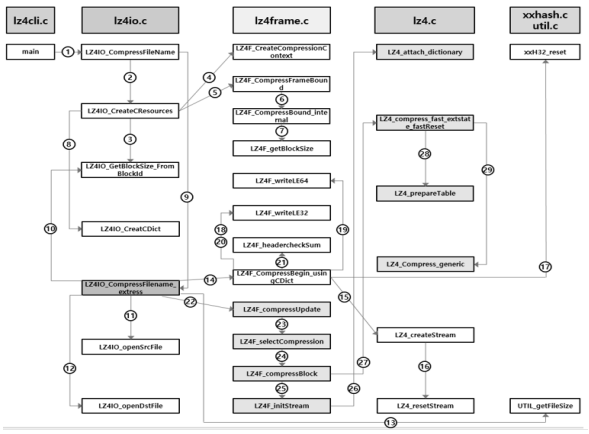


Fig. 8. LZ4 압축 흐름도

Fig 8의 LZ4IO_CompressFilename_extress함수는 하나의 데이터 Block에 대한 반복문이 실행되는 함수이며, 이 함수에 Intel AVX 명령어 적용을 통해 압축 속도를 향상 시킬 수 있는 설계를 추가하였다.

### 2.3 AES 암호화 모듈

데이터 이중화를 위해 원본서버의 데이터를 백업서버에 전달해야 한다. 이 때 실시간 데이터 이중화 시스템의 전송속도를 보장하며, 보안성이 뛰어난 블록단위 AES 암호화 알고리즘을 사용한다. AES 암호화에 필요한 Key는 Key모듈로부터 이미 할당 받았으므로, 데이터를 해당 Key로 암호화만 하면 된다. 아래 Fig 9는 AES의 암/복호화 과정을 도식화한 그림이다.

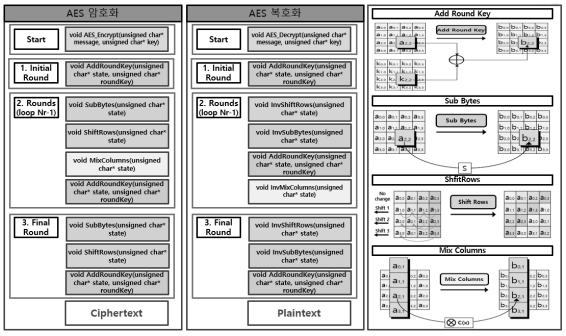


Fig. 9. AES Architecture

AES는 Start, Initial Round, Rounds, Final Rounds의 처리 과정을 거치며, 각 과정은 Add Round Key, Sub Byte, Shift Row, Mix Column 알고리즘을 사용한다. Rounds에서는 AES 128bit, 192bit, 256bit에 따라 지정된 횟수만큼 Loop 처리된다.

## IV. Conclusions

본 논문에서 제안하는 실시간 데이터 이중화는 원본서버와 백업서버의 실시간 데이터 동기화가 핵심요소이다. 이를 위해 가장 빠른 압축 알고리즘 LZ4에 Intel AVX 명령어를 적용하고, 빠른 블록 암호화 기법인 AES 알고리즘의 사용을 제안하는 바이다. 향후 연구에서는 LZ4 알고리즘에 Intel AVX 명령어를 적용하여 속도향상방안을 연구하고자 한다.

## ACKNOWLEDGEMENT

이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (NO.2018-0-00395, 데이터 실시간 이중화 기반 데이터 통합관리 스토리지 소프트웨어 기술 개발)

## REFERENCES

[1] LZ4 Project, https://github.com/lz4/lz4, accessed 2018-07-02

[2] Go-Eun Park, "Implementation of Dropout in Machine Learning algorithm using AVX512 Vector Instruction," The Journal of Korean Institute of Communications and Information Sciences, pp. 1711~1713, Dec, 2017.

[3] Sung-Gi Kim, Gil-Ho Kim and Gyeong-Yeon Cho, "Development of Stream Cipher using the AES," The Journal of Korean Institute of Communications and Information Sciences, Vol. 38, No. 11, pp. 972~981, Nov, 2013.

[4] https://en.wikipedia.org/wiki/Advanced_Encryption_Sta n dard, accessed 2018-07-02

[5] Slide Share, https://www.slideshare.net/ssuser800974/ss-76664853, accessed 2018-07-02