Journal of Korea Multimedia Society Vol. 21, No. 2, February 2018(pp. 232-241)
https://doi.org/10.9717/kmms.2018.21.2.232

유효시간 운영변환을 이용한 메모리 절약형 실시간 협업 편집 시스템

권오석[†], 김영봉[††], 권오준[†††], 이석환[††††], 권기룡[†††††]

Memory-saving Real-time Collaborative Editing System using Valid-Time Operational Transformation

Oh-Seok Kwon[†], Young-Bong Kim[††], Oh-Jun Kwon[†††],
Suk-Hwan Lee[††††], Ki-Ryong Kwon[†††††]

ABSTRACT

Operational Transformation (OT) algorithms for real-time collaborative editing systems are becoming increasingly important due to the increased demand for collaborative data processing. The operational transformation algorithm is a technique for real-time concurrency control and consistency maintenance with non-locking technique, and many studies have been conducted to overcome three issues of convergence, causality-prevention, and intention-prevention. However, previous work has the disadvantage of wasting memory by storing all operations that occurred during an edit operation in the history buffer to solve this problem. Therefore, we propose a memory-saving real-time collaborative editing system that maintains a constant memory space and concurrency control through a method of applying the valid-time to each user-generated operation in order to reduce memory waste. This system prevents long-term memory occupation of client-generated operations, thus it reduces the space and time complexity even with low-rate of collaboration work, so that the performance degradation avoids.

Key words: Operational Transformation, Collaborative Editing System, Real-Time Concurrency Control, Non-Locking, Valid-Time

1. 서 론

오늘날 정보통신 기술의 발전으로, 폭발적으로 데이터가 증가하는 이른바 빅 데이터(Big Data)시대가 도래하였다. 특히 최근에는 Facebook, Instagram등 소셜 네트워크 서비스 (SNS)의 확산으로 사진 및 동영상 등 큰 사이즈의 멀티미디어 콘텐츠 또한 폭발적으로 증가되고 있는 추세이다. 그에 따라 방대한 데

※ Corresponding Author : Ki-Ryong Kwon, Address: Dept. of IT Convergence and Application Eng., Pukyong Nat'l Univ., 45 Yongso-ro, Nam-gu, Busan, Korea, TEL : +82-51-629-6257, FAX : +82-51-629-6230, E-mail : krkwon@pknu.ac.kr
Receipt date : Oct. 16, 2017, Revision date : Jan. 13, 2018
Approval date : Jan. 26, 2018
[†] Dept. of IT Convergence and Application Eng., Pukyong National University (E-mail: inartistical@gmail.com)
[††] Dept. of IT Convergence and Application Eng., Pukyong National University (E-mail: ybkim@pknu.ac.kr)
[†††] Dept. of Computer Software Eng., Dongeui University (E-mail: ojkwon@deu.ac.kr)
[††††] Dept. of Information Security, Tongmyong University (E-mail: skylee@tu.ac.kr)
[†††††] Dept. of IT Convergence and Application Eng., Pukyong National University
※ This research was supported by Institute for Information & communications Technology Promion(IITP) grant funded by the Korea government(MSIT) (No.2015-0-00225, Development of Media Application Framework based on Multi-modality which enables Personal Media Reconstruction) and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT(No. 2016R1D1A3B03931003, No. 2017R1A2B2012456).

이터를 창의적으로 재활용하기 위해 많은 연구가 진행되고 있으며, 오늘날 발전된 인공지능 기술을 활용하기까지 그 연구 분야가 확장되고 있다[1]. 그러나 인공지능 기술과는 반대로 인간의 협력적 작업을 통해 콘텐츠 재생성 및 재활용 등의 창의성을 필요로 하는 업무를 빠르게 처리하기 위한 기술 또한 많이 연구되고 있다[2,3].

대표적으로 고도의 창의성을 필요로 하는 에세이 및 신문기사 작성과 같은 문서 편집 작업은 다수의 전문가들의 공동 작업을 통해 빠르게 달성할 수 있다. 그러나 물리적으로 분산된 다수의 전문가 간의 실시간 협업 시스템을 위해서는 동시성, 일관성 유지 측면에서 많은 어려움을 가지고 있다. 일반적으로 지역적으로 분산된 사용자를 위한 실시간 협업 시스템은 웹 기반으로 구현된다. 그러나 웹 서비스 환경 특성상 높은 지연율과 낮은 응답률로 인해 사용자 간의 동시성 및 일관성 유지가 어려워 협업 기능을 구현하기 힘들다. 일반적으로 동시성 제어는 락킹 기법을 통해 이루어지는데, 한 사용자가 락을 설정한 동안 다른 사용자는 대기하여 협업이 원활하지 못하게 되며, 또한 무한정 대기해야 하는 상황이 발생한다. 따라서 락킹 기법 없이 동시성 제어의 어려움을 극복하기 위해 많은 연구가 진행되었으며, 대표적인 알고리즘으로 Ellis와 Gibbs가 제안한 운영변환 알고리즘이 있다[4].

운영변환 알고리즘은 협력적 작업환경에서 동시에 발생한 입력에 의한 문제점을 극복하기 위해 각 사용자가 생성한 입력 또는 삭제 오퍼레이션과 서버로부터 받은 오퍼레이션의 변환함수(T)를 통해 변환하여 적용함으로써, 데이터 일관성을 유지하는 방법이다. 이러한 운영변환 알고리즘은 동시성 및 일관성 유지뿐만 아니라, 동시에 발생한 오퍼레이션의 인과관계 및 사용자의 의도까지 보존하는 방향으로 그 연구 분야가 확장되었다. 그러나 기존의 방법은 편집 작업 시 발생한 모든 오퍼레이션을 서버로부터 수신된 새로운 오퍼레이션과의 변환연산을 위해 히스토리 버퍼에 저장하는 구조를 갖고 있다. 따라서 협업 작업율이 낮은 일방적인 편집작업을 수행하는 클라이언트에서는 히스토리 버퍼가 매우 커져 메모리 낭비 및 시간 복잡도가 증가하게 되는 문제가 발생한다[5].

따라서 본 논문에서는 유효시간 운영변환을 이용한 메모리 절약형 실시간 협업 편집 시스템을 제안한다. 이는 사용자가 생성한 오퍼레이션에 유효시간을 적용하여 오퍼레이션의 장기간의 메모리 점유를 방지하는 방법을 통해 공간 및 시간 복잡도를 감소시킨다. 또한 유효시간에 의해 오퍼레이션이 소멸하여도 소멸 동기화 프로토콜을 통해 모든 클라이언트-서버의 일관성 유지 및 인과관계, 의도를 유지시키는 방법을 제안한다. 제안한 방법을 통해 공간 및 시간 복잡도를 감소시켜, 협업작업율이 낮은 상황에서도 시스템의 성능 유지가 가능하도록 한다.

다음 2장에서는 관련연구에 대해 설명하고, 3장에서는 본 논문에서 제안한 방법에 대해 자세히 알아본다. 본 논문에서 제시한 시스템에 대한 실험결과 및 결론은 각각 4장과 5장에서 기술한다.

2. 관련연구

2.1 운영변환

실시간을 보장하기 위해서 기본적으로 전체문서를 서버로 전송하는 것이 아니라, 새롭게 입력/삭제된 텍스트만을 전송하여, 전송에 걸리는 오버헤드를 최소화하며 동시성 및 일관성 유지를 위한 방안으로 많이 연구되었다. 이러한 대부분의 운영변환 알고리즘은 다음과 같은 세 가지의 이슈를 다루고 있다.

이슈 1. 수렴(Convergence)
이슈 2. 인과성 보존 (Causality preservation)
이슈 3. 의도 보존 (Intention preservation)

기본적으로 공유 데이터의 공동 편집은 Fig. 1의 (a)와 같이 데이터 분기, 인과관계 위배, 의도 위배의 세 가지의 불일치성 문제를 가진다. 따라서 운영변환 알고리즘은 Fig. 1의 (b)와 같이 변환 함수 T를 통해 서버의 오퍼레이션을 $O \Rightarrow O'$로 변환함으로써 해결한다. 이러한 변환함수는 기본적으로 포함 변환(IT: inclusion transformation)과 제외 변환(ET: exclusion transformation)으로 구분된다. 먼저 포함 변환의 경우 $IT(O_a, O_b)$ 또는 $T(op1, op2)$로 표기하며, O_b의 영향이 포함되도록 O_a를 변환하여 반환하는 반면 제외 변환의 경우 $ET(O_a, O_b)$ 또는 $T^{-1}(op1, op2)$로 표기하며, O_b의 영향이 배제되도록 O_a를 변환하여 반환한다[5].

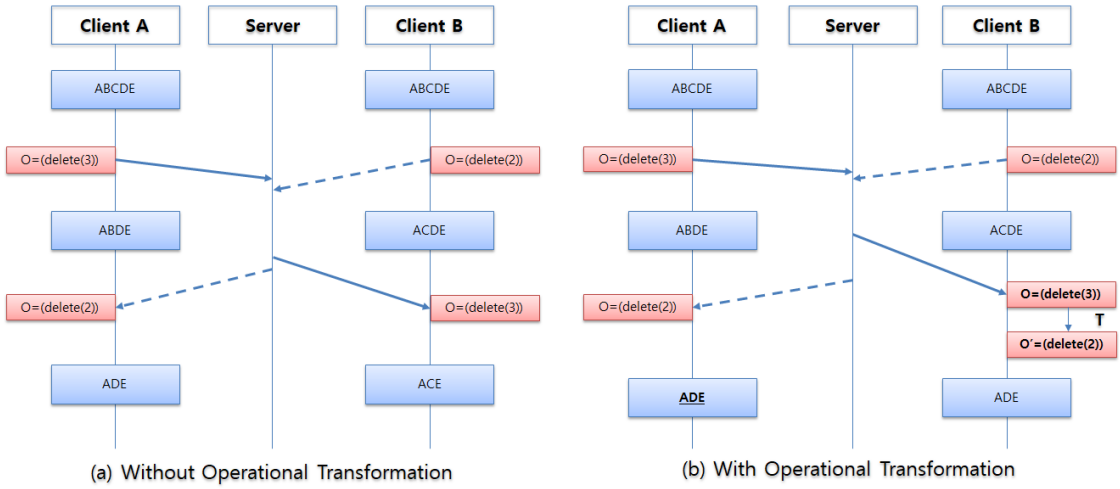


Fig. 1. The processes of two clients A, B by concurrent operations.

이러한 IT와 ET함수는 히스토리 버퍼에 저장되어 있는 클라이언트의 오퍼레이션과의 변환 시 수행되며, Fig. 2와 같이 선형적으로 수행되는 구조를 가진다. 이러한 방법을 통해 모든 클라이언트에서의 작업이 실행 된 후 데이터들은 결과적으로 수렴된 상태, 즉 일관성을 유지하게 되는 구조로 설계된다.

2.2 운영변환 기반 협업 서비스

대표적인 운영변환기반 협업 시스템으로는 아파치 웨이브와 구글 Docs와 같은 웹 서비스 기반의 협업 텍스트 시스템이 있다. 먼저 Apache Wave는 2009년에 발표된 실시간 협업 편집을 위한 프레임워크로 다양한 서식을 가지는 텍스트 문서의 실시간 동시 편집 기능을 제공하기 위해 대표적인 운영변환 알고리즘 중 하나인 Nichols의 주피터(Jupiter) 알고리즘을 개선하여 사용되었다[5]. 여기서 기존의 주피터 알고리즘의 경우 클라이언트가 생성한 오퍼레이션을 서버로부터 새로운 오퍼레이션이 들어오기 전까지는 동시성 제어를 위해 히스토리 버퍼에 저장하는 구조를 갖고 있다. 따라서 일방적인 편집작업을 수행하는 클라이언트에서는 히스토리 버퍼가 매우 증가하게 되며 이는 성능저하를 가져오는 문제가 발생한다. 따라서 Apache Wave의 운영변환 알고리즘인 Wave OT는 서버의 승인을 기다리는 방법을 통해 이러한 문제를 개선하였다. 일방적인 편집작업이 발생할 경우 서버의 승인 전 하나의 오퍼레이션에 문자열로 편집내역을 저장함으로써 $O(c^*s)$의 시간 복잡도를 가지는 변환연산의 성능을 $O(c\log c + s\log s)$으로 향상시켰다[6].

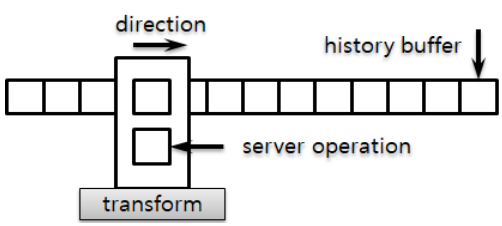


Fig. 2. The process of transformation function by server operation.

두 번째로 Google Docs는 웹 기반의 워드 프로세서로 온라인에서 문서를 만들고 편집하며, 사용자와 공동으로 작업할 수 있는 기능을 제공한다[5]. 특징으로는 수정사항을 실시간으로 처리하며, 다른 공동작업자가 변경한 내용을 시각적으로 볼 수 있다. 또한 중요 기능중 하나는 오프라인에서도 작동된다는 점이다. 네트워크가 소실되어도 모든 변경사항을 로컬 캐시에 저장하여 다시 온라인 상태가 될 경우 Google Docs에 동기화하여 협력적 작업을 제공한다[7].

3. 제안한 메모리 절약형 실시간 협업 시스템

본 논문에서 제안한 메모리 절약형 실시간 협업 시스템을 위한 클라이언트-서버의 데이터 처리 흐름도는 다음 Fig. 3와 같다. 제안한 실시간 협업 시스템의 서버와 클라이언트는 웹 소켓으로 통신하며, 서버

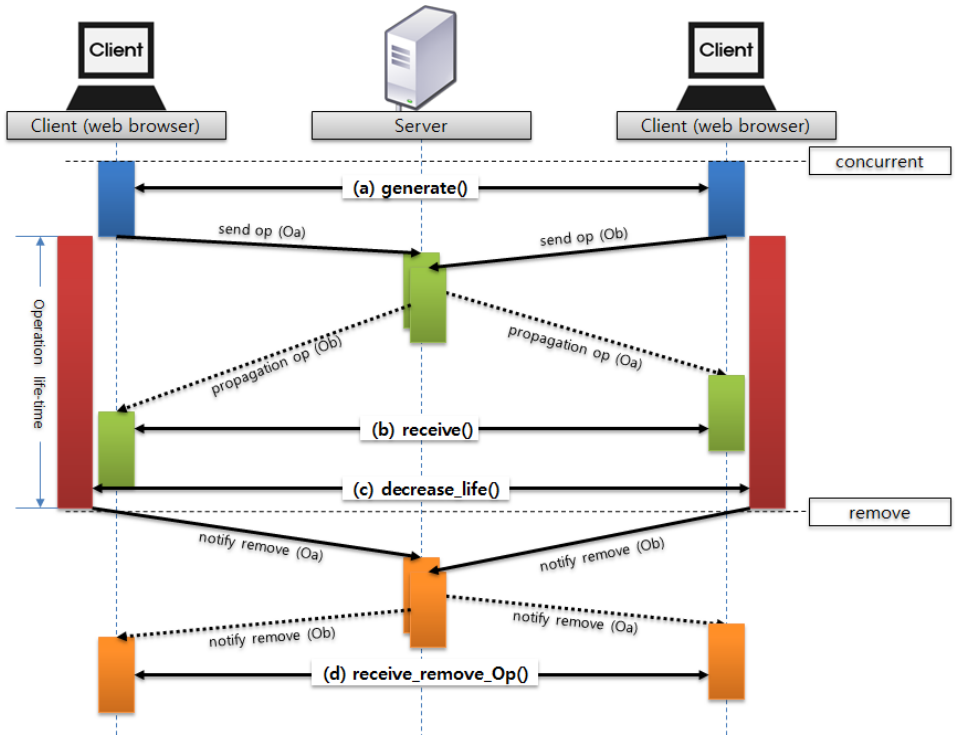


Fig. 3. Proposed flowchart of data-processing architecture.

는 각 클라이언트에서 발생한 오퍼레이션을 시간 순으로 직렬화한 후 다른 클라이언트에게 전파하는 역할을 한다. 반면에 클라이언트는 텍스트 편집의 입력·삭제 등의 오퍼레이션을 생성하고 서버로 전송하며, 다른 사용자의 오퍼레이션을 서버로부터 수신하는 기능을 수행한다. 이때 생성된 모든 오퍼레이션들은 생성과 동시에 유효시간을 가지며, 유효시간 후 히스토리 버퍼로부터 소멸된다. 또한 소멸 시 동기화 프로토콜을 통해 다른 클라이언트들과의 일관성을 유지하며 협업 작업을 수행한다.

3.1 클라이언트-서버 인터페이스

제안한 협업 시스템은 웹 기반의 텍스트 편집을 위해 HTML5의 textarea를 사용하며, 키보드 이벤트를 통해 입력·삭제 오퍼레이션을 생성하였다. 제안한 시스템에서의 서버와 클라이언트는 Fig. 3와 같이 generate(), receive(), decrease_life(), removal_sync()의 함수를 가진 동일한 아키텍처로 구성된다. 각 클라이언트는 Fig. 3의 (a) generate() 태스크를 통해 키보드 이벤트로부터 텍스트의 입력·삭제 오퍼레이션을 생성하고, 서버로 전송한다. 또한 generate() 태스크에서는 오퍼레이션을 히스토리 버퍼에 저장하고, 자신의 상태벡터 x를 증가시키는 작업을 수행한다. 여기서 서버는 송·수신 역할만을 수행하므로 generate() 태스크는 수행되지 않는다. 두 번째로 서버와 클라이언트는 Fig. 3의 (b)의 receive() 태스크를 통해 오퍼레이션 수신을 수행하는데, receive() 태스크에서는 서버/클라이언트로부터 받은 오퍼레이션과 히스토리 버퍼에 저장된 오퍼레이션과의 $transform(c, s)$ 연산을 통해 일관성 유지를 수행하며, 최종적으로 상태벡터 y를 증가시킨다. 그리고 Fig. 3의 (c) decrease_life() 태스크는 생성된 오퍼레이션 O의 유효시간(valid-time)을 시간의 경과에 따라 감소시키며, 소멸된 오퍼레이션이 발생하면 서버로 알리는 역할을 수행한다. 마지막으로 Fig. 3의 (d) removal_sync() 태스크는 오퍼레이션의 소멸을 수신하여 동기화 프로토콜을 수행한다. 여기서 유효시간을 가지는 오퍼레이션 O의 정의와 동기화 프로토콜은 다음 3.2절과 3.3절에서 자세히 기술한다.

```
{
    "Operation" : {
        "cid" : (string)client's identification,
        "pos" : (int)position,
        "char" : (char)character,
        "state" : {
            "x" : (int)client state,
            "y" : (int)server state
        },
        "opType" : (string)operation type,
        "timeStamp" : (string)create time,
        "valid-time" : (double)valid-time
    }
}
```

Fig. 4. The definition of operation.

3.2 유효시간 오퍼레이션 정의

유효시간을 가지는 오퍼레이션의 정의는 다음 Fig. 4와 같다. 오퍼레이션은 생성될 시점의 상태벡터 (x,y), 텍스트에 적용될 위치 (pos), 오퍼레이션 타입 (opType), 생성 시점 (timeStamp), 유효시간 (valid-time)의 속성을 가진다.

여기서 유효시간은 생성시에 초기값으로 부여하여 시간의 경과에 따라서 Fig. 3의 decrease_life()함수를 통해 감소하여, 0 이하의 오퍼레이션을 제거한다. 생성시의 오퍼레이션은 queue구조의 히스토리 버퍼에 저장한다. 이러한 히스토리 버퍼는 Fig. 5와 같이 서버로부터 수신된 오퍼레이션과의 변환연산을 수행하기 위해 사용된다. 여기서 서버 오퍼레이션 S_2는 $(0,1)$의 상태에서 생성되었기 때문에 히스토리 버퍼에 저장된 C_1의 생성상태와는 달라 변환함수 수행 시 올바르지 않은 결과를 반환하게 된다. 따라서 이러한 문제를 해결하기 위해 히스토리 버퍼에 저장된 모든 오퍼레이션과의 변환연산을 수행하여 히스토리 버퍼 및 서버의 오퍼레이션을 업데이트하여 수행되도록 한다. 또한 유효시간 후 히스토리 버퍼에서 소멸된 오퍼레이션은 서버로 소멸됨을 알린다.

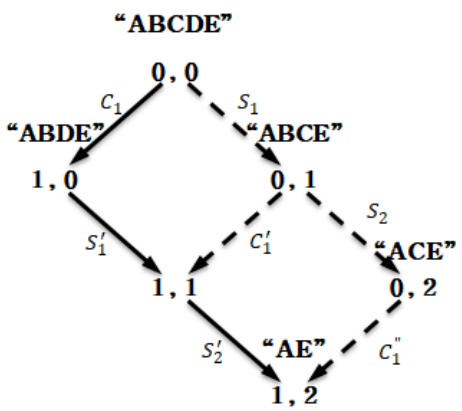


Fig. 5. The process of $H=\{C_1\}$ in history buffer of client by server operations S_1, S_2.

3.3 소멸 동기화 프로토콜

오퍼레이션은 시간의 지남에 따라 히스토리 버퍼에서 소멸된다. 이때 소멸 동기화 프로토콜이 작동되게 되는데, 이는 Fig. 3의 decrease_life()와 removal_sync() 태스크를 통해 동작하며, 의사코드는 그림 Fig. 6과 같다.

decrease_life()에서는 항상 히스토리 버퍼의 가장 첫 오퍼레이션의 유효시간만 감소시키도록 동작한다. 그 이유는 모든 오퍼레이션들을 참조하여 시간을 감소시키는 연산비용이 크므로 생성 시에 이전 오퍼레이션과의 시간차이를 바탕으로 유효시간을 부여하여 모든 오퍼레이션의 유효시간을 감소하는 연산비용을 줄 일수 있도록 하였다. 또한 제거된 오퍼레이션들로 인해 발생할 일관성 문제를 해결하기 위해 남아있는 다른 오퍼레이션의 상태 값 x를 감소하였다. 그 이유는 Fig. 7을 통해 알 수 있다. Fig. 7의 (a)의 상황과 같이 오퍼레이션 C_2가 소멸 될 경우 서

```
int myMsgCnt = 0;  /* number of messages generated */
int otherMsgCnt = 0; /* number of messages received */
int removalCnt = 0; /* number of remove operaions */
int mySid; /* identification of server or client */

func decrease_life() := {
    /* first operation of history buffer */
    var op = history_buffer.head;

    if(op != null) {
    /* removal conditions */
        if(op.lifeTime < 0) {
            removalCnt = removalCnt + 1;
            myMsgCnt = myMsgCnt - 1;
            outgoing.deQueue();

            send(mySid);
        }

    /* dt is in millisecends */
    op.lifeTime = op.lifeTime - dt;
    } else {
        removalCnt = 0;
    }
}

func removal_sync(sid) := {
    if(sid == "server") {
        otherMsgCnt = otherMsgCnt - 1;
    } else {
        for(c in client list) {
            if(c != sid)
                send(mySid);
        }
    }
}
```

Fig. 6. The pseudocode of decrease_life() and removal sync() tasks for removal synchronization.

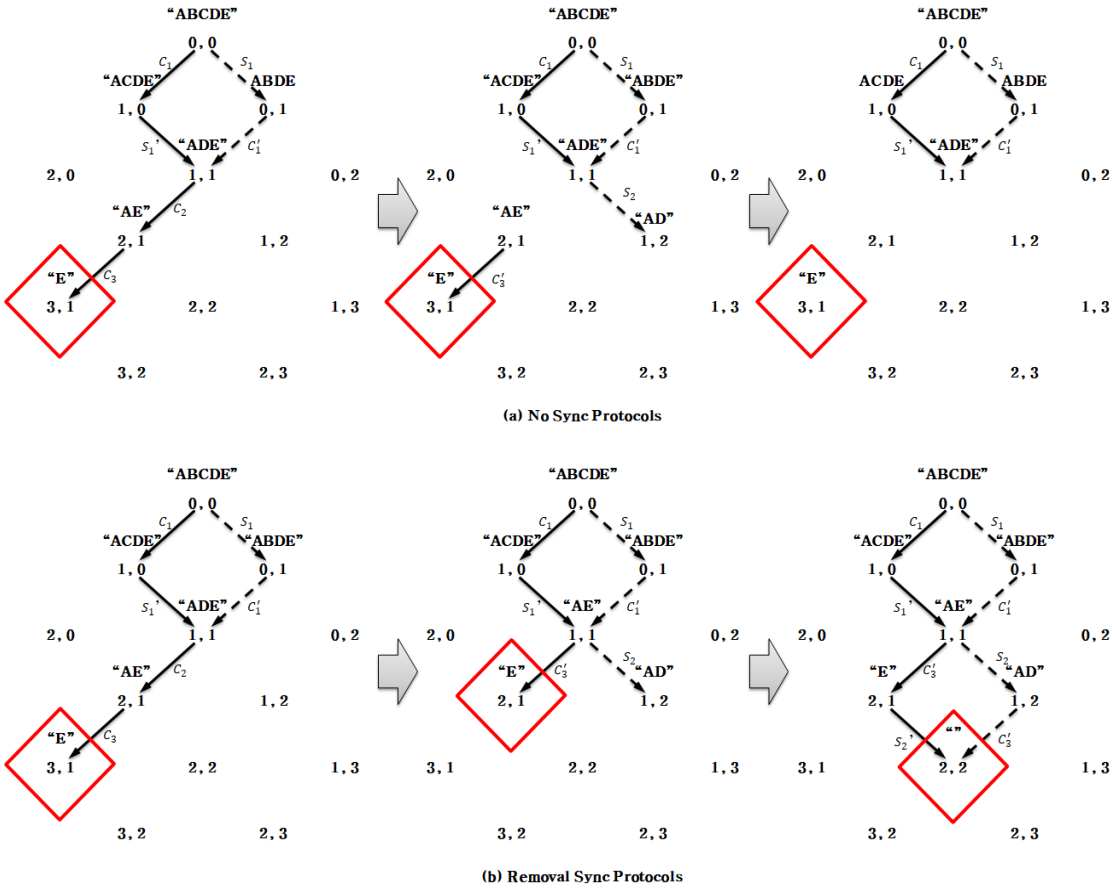


Fig. 7. The process of the operation removal. (a) without removal synchronization protocol, (b) with our removal synchronization protocol.

버의 오퍼레이션 S_2로 인한 변환함수는 $\{C_3, S_2\}$으로 변환연산을 수행하게 된다. 그러나 두 오퍼레이션의 상태벡터가 일치하지 않는 문제로 인해서 서버의 오퍼레이션 S_2는 정상적으로 수행되지 못한다. 따라서 이와 같은 문제를 해결하기 위해 Fig. 7의 (b)와 같이 오퍼레이션 C_i가 삭제 될 경우 이후의 모든 오퍼레이션 $C_{i+1}...C_n$의 상태벡터 x를 $x-1$으로 조절하여, 서버로부터 입력된 오퍼레이션 S_2와의 상태벡터 생성위치를 일치시킨다. 이때 매번 오퍼레이션이 소멸 될 때마다 감소시키는 것이 아닌 소멸횟수를 저장하여, 차후 $transform$연산이 발생할 때 생성위치를 보정하여 상태벡터를 조절하는데 드는 연산시간을 고려하였다. 또한 오퍼레이션이 소멸되었음을 서버로 통지하여 다른 클라이언트의 서버 상태벡터 y또한 같은 방법으로 조절한다. 제안한 소멸에 의한 동기화 프로토콜은 Fig. 7의 (b)의 오른쪽과 같이 히스토리 버퍼에 남아있는 오퍼레이션들의 상태벡터를 조절함으로써 서버의 오퍼레이션과의 변환연산이 정상적으로 수행되도록 한다.

4. 실험결과

본 논문에서 제안한 방법의 성능을 확인하기 위해, 대표적인 운영변환 알고리즘인 주피터[6]와 Apache Wave에서 적용한 Wave OT와의 비교실험을 수행하였다. 두 알고리즘과의 성능을 비교하기 위해서 다양한 조건에서 실험하였다. 먼저 Fig. 8의 (a)는 클라이언트의 타자속도에 따른 오퍼레이션의 증가율을 비교한 것으로 기존의 방법의 경우 선형적으로 증가하는 반면, 제안한 방법은 유효시간에 따라 오퍼레이션이 제거됨으로 오퍼레이션의 증가율이 완만함을

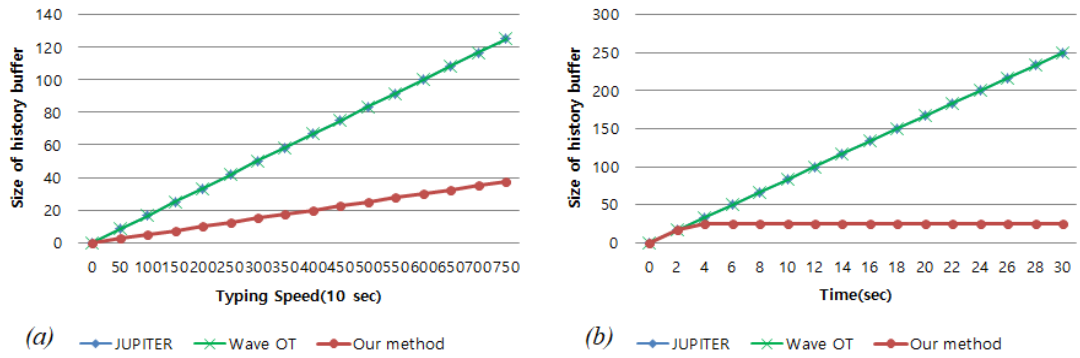


Fig. 8. Compare the size of history buffer in the other two conditions. (a) The rate of increase of operations by typing speed. (valid-time: 3 sec, typing-time : 10 sec), (b) The rate of increase of operations over time. (typing speed: 300/min, valid-time: 3 sec)

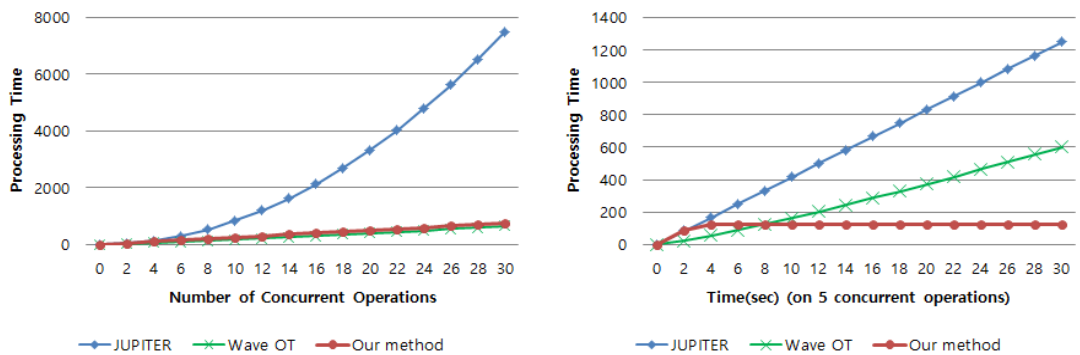
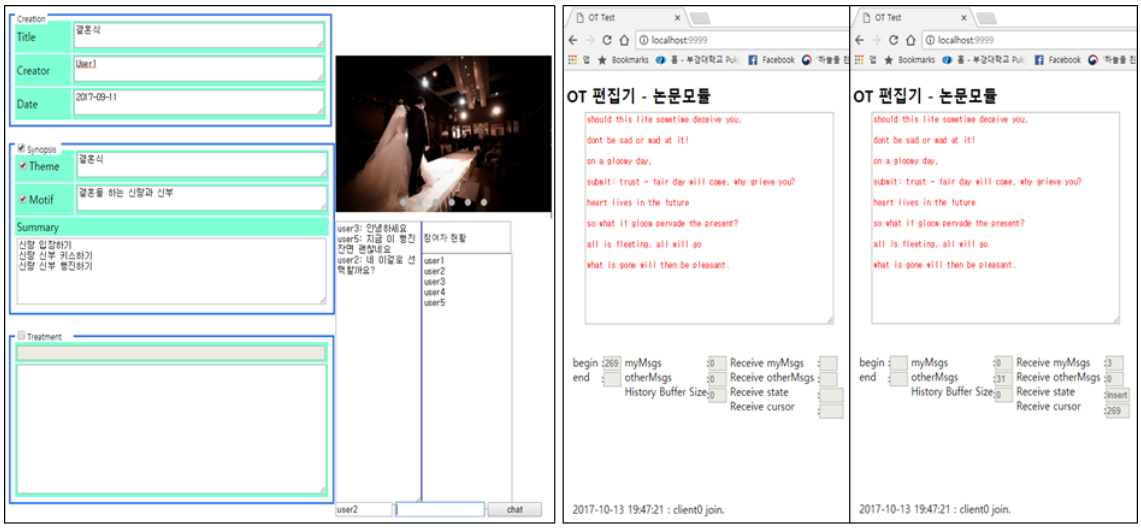


Fig. 8. Comparison of processing performance in two other conditions. (c) Processing performance by rate of the number of concurrent operations. (typing speed: 300/min, valid-time: 3 sec), (d) Processing performance over time. (typing speed: 300/min, valid-time: 3 sec, concurrent operation: 5)

알 수 있다. 두 번째로 Fig. 8의 (b)는 시간에 따른 오퍼레이션의 증가율을 비교한 것이다. 제안한 방법의 경우 오퍼레이션이 유효시간에 의해 소멸됨으로써 4초 이후로는 더 이상 증가하지 않는다는 것을 알 수 있다. 그리고 Fig. 8의 (c)는 동시에 생성된 오퍼레이션의 증가량에 따른 처리 성능을 비교한 그림이다. Wave OT는 $O(c\log c + s\log s)$의 성능을 보이는 반면 제안한 방법은 $O(c+s)$으로 동시에 생성된 오퍼레이션이 증가하여도 안정적인 성능을 확인할 수 있다. 마지막으로 Fig. 8의 (d)는 최종적으로 알고리즘간의 성능을 분석하기 위해 타이핑 속도는 분당 300타와 오퍼레이션의 유효시간은 3초, 동시 오퍼레이션은 5개가 발생하는 환경에서 비교한 결과이다. Wave OT와 주피터의 경우 시간이 지날수록 클라이언트의 히스토리 버퍼의 사이즈 증가하여 결과적으로 증가된 공간 복잡도로 인해 처리시간이 많이 걸리는 것을 알 수 있다. 두 알고리즘과의 비교는 Table 1과 같이 정리하여 알 수 있다. 따라서 제안한 방법의

Table 1. Comparation of Jupiter and Our method

OT algorithms (systems)	Central transformation server	Operation auto removal	stop and wait	Time complexity	Space complexity
Jupiter	Yes	No	No	$O(c*s)$	$O(T_s*t)$
Wave OT	Yes	No	Yes	$O(c\log c + s\log s)$	$O(T_s*t)$
Our method	Yes	Yes	No	$O(c+s)$	$O(T_s)$



(a) OpenScenario Editor: Collaborative editing system with proposed method.

(b) Visualize the operation state of the proposed method.

Fig. 9. A result of applying the proposed collaborative system.

경우 오퍼레이션이 일정시간 이후 소멸하게 됨으로써, 공간복잡도가 시간에 독립적인 결과를 나타낸다. 그로인해 낮은 협업작업률(8초 이상 타인의 작업이 발생하지 않는 상황)과 같은 환경에서 안정적인 성능 결과를 보이는 것을 확인하였다.

Fig. 9는 제안한 방법이 적용된 협업 편집 시스템의 결과이다. Fig. 9의 (a)는 5명의 편집 참여자가 공동으로 텍스트 입력을 수행하는 화면으로 제안한 방법을 통해 실시간으로 동기화된 정보를 5명의 사용자가 공유하며 편집작업을 수행할 수 있다. 그리고 Fig. 9의 (b)는 제안한 시스템의 성능 측정을 위해 오퍼레이션의 송수신에 따른 상태변화 값을 시각적으로 볼 수 있도록 한 결과이다. (b)의 좌우는 서로 다른 클라이언트로서 왼쪽 클라이언트가 일방적으로 편집작업을 수행하였으며, 아래 History Buffer Size의 값을 통해 유효시간에 따라 히스토리 버퍼가 비워져 메모리 관리가 되고 있음을 알 수 있다.

4.1 한계점

제안한 유효시간 방법은 실험결과를 바탕으로 메모리를 절약할 수 있음을 확인하였다. 기존의 운영변환 알고리즘은 웹 지연 및 응답률에 강인하도록 모든 작업내역 히스토리를 저장하여 일관성을 유지할 수 있도록 하는 반면, 제안한 방법은 히스토리를 주기적으로 비움으로써 메모리를 절약할 수 있도록 하였다. 그러나 유효시간보다 지연시간이 더 길 경우 일관성 유지가 되지 않는 문제가 발생할 수 있습니다. 이러한 문제를 해결하기 위해서는 유효시간을 조절함으로써 방지할 수 있으나, 메모리 절약의 효과가 늦게 나타날 수 있다.

5. 결론 및 향후연구

본 논문에서는 유효시간 운영변환을 이용한 메모리 절약형 실시간 협업 편집 시스템을 제안하였다. 제안한 방법은 각 클라이언트에서 발생한 오퍼레이션에 유효시간을 적용하여, 시간 및 공간 복잡도를 감소시키며 오퍼레이션 소멸에 의한 일관성을 유지를 위한 동기화 방법을 제안하였다. 일반적인 운영변환 알고리즘은 모든 오퍼레이션을 히스토리 버퍼에 저장한 후 선형적 변환연산을 통해 동기화를 수행한다. 그런데 협업작업률이 낮을 경우 히스토리 버퍼의 크기가 커져 메모리 낭비 및 시간 복잡도만 높아지게 되는 문제가 발생하였다. 따라서 본 논문에서는 오퍼레이션에 유효시간을 적용하여 일정시간 후 오퍼레이션이 소멸하는 방법을 통해 이러한 문제를 해결하였다. 또한 소멸에 의한 동기화 프로토콜을 수행함으로써 일관성을 유지하였다. 제안한 방법을 통해 편집자의 오퍼레이션 히스토리 버퍼의 공간을 시간과는

독립적으로 일정하게 유지함으로써, 이전 방법에 비해 공간적, 시간적 성능향상을 이룰 수 있다.

REFERENCE

[1] M. Chen, G. Eason, B. Noble, and I. Sneddon, "Big data: A Survey," *Mobile Networks and Applications*, Vol. 99, Issue 2, pp. 171-209, 2014.

[2] O. Kwon, Y. Kim, H. Lee, O. Kwon, S. Lee, K. Moon, and K. Kwon, "Collaborative Open Sinario Editing System Using Hierarchical Operational Transformation," *Proceeding of the Conference of Korean Institute of Communications and Information Sceience*, pp. 627-628, 2017.

[3] H. Lee, K. Kwon, S. Lee, Y. Park, and K. Moon, "Design of OpenScenario Structure for Content Creation Service Based on User Defined Story," *Journal of Korea Multimedia Society*, Vol. 19, No. 2, pp. 170-179, 2016.

[4] C. Ellis and S. Gibbs, "Concurrency Control in Groupware Systems," *ACM Sigmod Record*, Vol. 18, No. 2, pp. 399-407, 1989.

[5] M. Kaur, M. Singh, H. Kaur, and S. Kaur, "Operational Transformation In Co-Operative Editing," *International Journal of Scientific and Technology Research*, Vol. 5, Issue 2, pp. 16-20, 2016.

[6] Apache Wave Protocol Documentation, phttps://people.apache.org/~al/wave_docs/ApacheWaveProtocol-0.4.pdf, (Accessed Aug., 24, 2015).

[7] T. Weis and A Wacker, "Federating Websites with the Google Wave Protocol," *IEEE Internet Computing*, Vol. 15, Issue 2, pp. 51-58, 2011.

[8] D. Nichols, P Curtis, M. Dixon, and J. Lamping, "High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System," *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology, ACM*, pp. 111-120, 1995.

[9] T. Jungnickel and T. Herb, "Simultaneous Editing of JSON Objects via Operational Transformation," *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp. 812-815, 2016.

[10] A. Randolph, H. Boucheneb, A. Imine, and A. Quintero, "On Consistency of Operational Transformation Approach," *Proceeding of International Workshop on Verification of Infinite-State Systems*, pp. 45-59, 2013.

[11] D. Sun and C. Sun, "Context-based Operational Transformation in Distributed Collaborative Editing Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 20, Issue 10, pp. 1454-1470, 2009.

권 오 석

2012년 부경대학교 컴퓨터멀티미디어공학(공학사)
2014년 부경대학교 IT융합응용공학과(공학석사)
2015년~현재 부경대학교 IT융합응용공학과 박사과정

관심분야 : 컴퓨터 그래픽스, 3D의료영상 시각화, HCI

김 영 봉

1987년 서울대학교 계산통계학과(공학사)
1989년 한국과학기술원 전산학과(공학석사)
1994년 한국과학기술원 전산학과(공학박사)

1994년~1995년 삼성전자 정보기술연구소 선임연구원
1995년~현재 부경대학교 IT융합응용공학과 정교수
관심분야 : 컴퓨터 그래픽스, 3D 컴퓨터 시뮬레이션

권 오 준

1986년 경북대학교 전자공학과(공학사)
1992년 충남대학교 전산학과(이학석사)
1998년 포항공과대학교 전자계산학과(공학박사)

1986년~2002년 한국전자통신연구원 선임연구원
2000년~현재 동의대학교 컴퓨터소프트웨어공학과 부교수
관심분야 : 컴퓨터네트워크, 정보보호, 무선 인터넷, 패턴인식, 인공신경망

이 석 환

1999년 경북대학교 전자공학과(공학사)
2001년 경북대학교 전자공학과(공학석사)
2004년 경북대학교 전자공학과(공학박사)

2005년~현재 동명대학교 정보보호학과 부교수
2010년~현재 IEEE R10 창원섹션 임원
관심분야 : 워터마킹, DRM, 영상신호처리

권 기 룡

1986년 경북대학교 전자공학과(공학사)
1990년 경북대학교 전자공학과(공학석사)
1994년 경북대학교 전자공학과(공학박사)

2000년~2001년 Univ. of Minnesota, Post-Doc.
1996년~2006년 부산외국어대학교 디지털정보공학부 부교수
2011년~2012년 Colorado State Univ., 연구교수
2011년~2016년 IEEE R10 창원섹션 의장
2015년~2016년 한국멀티미디어학회 회장
2018년~현재 글로벌핀테크산업진흥센터 이사장
2006년~현재 부경대학교 IT융합응용공학과 교수
관심분야 : 멀티미디어정보보호, 영상처리, GIS 보안, 드론, 3D 프린팅