VOD 스트리밍 데이터를 위한 Consistency 알고리즘 설계

장승주*

Design of a Consistency Algorithm for VOD Streaming Data

Jang, Seung Ju*

이 논문은 2005학년도 동의대학교 교내연구비에 의하여 연구되었음. (2005AA166)

요 약

본 논문에서는 VOD에서 스트리밍 데이터를 효율적으로 서비스할 수 있는 Consistency 알고리즘을 제시한다. 미디어 데이터 서비스를 위해 우선 하나의 미디어 데이터를 라운드 로빈 방식으로 스트라이핑하여 각 호스트의 저장 노드에 저장한다. 본 논문에서는 일반적인 계산 용도로 사용되는 barrier 메커니즘을 미디어 플레이 최소 단위인 (SH, GOP) 미디어 데이터에 서비스할 수 있도록 제시하였다. 데이타 처리 방식은 하나의 호스트에 한 프래그먼트 크기의 공유메모리를 할당한다. 이 공유메모리는 barrier 메커니즘을 통하여 RTP패킷 전송 단위로 결정될 프래그먼트 데이터의 조합을 형성한다. 본 논문에서 제시한 알고리즘에 대해서 실험을 수행하기 위해서 VOD 시스템에서 프로그램을 작성하였다.

ABSTRACT

This paper proposes a consistency algorithm that is able to serve streaming data efficiently in VOD system. The media data is stripping into several pieces of data by the Round Robin method in order to media data service. The barrier mechanism is changed into the minimum data factor(SH. GOP) in this paper. The shared memory is allocated at one host with one fragment size. Data is combined with RTP packet transmission data format using barrier mechanism. I experiment and program the suggested algorithm on the VOD system.

키워드

VOD, Consistency Algorithm, DSM(Distributed Shared Memory)

Ⅰ. 서 론

초고속 통신망을 통한 인터넷의 발달은 많은 사회적 변화를 일으켰지만, 그 중에서도 제일 큰 변화를 가져온 것은 영상 산업 분야일 것이다. 현재 VOD(Video On Demand)에 대한 수요가 급증하여 다양한 VOD 서비스가 도입되고 있는 상황에서 데이터 및 사용자의 증가에 따른 문제점을 해결하는 것이 필수적이다. VOD 서버의 문제점을 해결하기 위해 병렬 VOD 서버에 대한 여러 연구가 진행되고 있다. 병렬 VOD 서버는 저가의 일반 시스템을 통해 여러 대 클러스터링으로 연결하여 높은 가용성을 유지하면서 사용자들의 요구를 보다 많이 수용하면서 추후

* 동의대학교 컴퓨터공학과

접수일자 : 2006. 3. 16

클라이언트 및 데이터의 사용의 증가에 따라 유용하게 대비할 수 있도록 하고 있다.

본 논문은 분산 공유 메모리 방식의 프로그래밍을 이용하여 VOD 스트리밍에서 제공되는 MPEG(Moving Picture Exports Group) 미디어 데이터를 분산 처리할 수 있는 Consistency 알고리즘을 설계한다. 이 알고리즘은 barrier 메커니즘을 이용하여 미디어 데이터에 대한 동기화 메커니즘을 적용함으로써 효율적인 데이터 일관성을 유지한다. 일반적으로 구성되어 있는 스트리밍 서버의 RAID(Redundant Array Inexpensive Disk) 구조와는 달리 Consistency 모델 기반 스트리밍 서버는 여러 대의 미디어 서버 간에 공유메모리를 형성함으로써 미디어 데이터 처리 속도를 향상시킨다. 미디어 파일에 대한 스트라이핑 정책은 라운드 로빈 방식을 통해 미디어 데이터를 분산시키며, barrier 메커니즘을 통해 RTP(Real-time Transport Protocol) 패킷으로 전송하게 될 미디어 데이터가 결정된다. 공유메모리에 존재하는 미디어 데이터는 시간 동기화와 예상 페이징 기법을 적용함에 따라 연속된 공유메모리의 미디어 데이터에 대한 순서화된 서비스 제공과 동시에 좀 더 빠른 데이터 로딩을 보장할 수 있다. 본 논문에서 제시하는 분산 consistency 알고리즘은 VOD 시스템 데이터 처리의 무결성과 성능을 보장한다.

본 논문의 구성은 2장에서 관련 연구를 살펴보고, 3장에서는 Scope Consistency[1]의 Barrier 메커니즘을 통한 미디어 데이터의 분산 처리 알고리즘을 제시한다. 마지막으로 4장에서 실험, 5장에서 결론을 맺는다.

Ⅱ. 관련 연구

2.1. VOD 스트리밍

VOD와 AOD 같은 주문형 멀티미디어 서비스를 지원하는 대표적인 프로토콜로는 H.323, SIP, RTSP 등이 있는데, 이들 프로토콜들은 RTP(Real-time Transport Protocol)[4]를 멀티미디어 데이터 전송 프로토콜로 채택하고 있다. RTP에 의해서 제공되는 서비스들은 time reconstruction, loss detection 그리고 content identification을 포함한다. 멀티미디어 컨텐츠를 수신하여 재생하기 위해서는 적절한 타이밍을 요구하는데, RTP는 이를 위해서 time stamping, sequence numbering과 같은 메커니즘을 제공한다. 이 메커니즘을 이용하여 RTP는 종단간 실시간으로 데이터를 전송하여 어플리케이션 측에서 timestamp, sequence number를 이용하여 서로 다른 스트림 데이타를 동기화시킨다.

2.2. JIAJIA

JIAJIA는 scope consistency를 이용하여 분산 공유메모리 시스템에서 데이타 처리를 보장한다. JIAJIA는 두 가지 주요 특징을 가지고 있다. 첫째로, NUMA와 같은 구조를 가지고 있으며, 더욱 큰 공유 메모리 공간을 형성하기 위하여 다중 컴퓨터의 물리적인 메모리 공간을 조합한다. 둘째로, lock 기반 캐쉬 일관성 규약을 구현하며 lock에서 가지게 되는 write notice에 접근함으로써 일관성을 유지한다. JIAJIA에서는 세 가지 동기화 기법을 제공하는데, lock, barrier와 조건 변수가 있다. 그 중 barrier는 모든 프로세스들의 처리가 완료될 때까지 어떠한 프로세스로부터의 처리를 금지함으로써 전역적인 동기화 기법을 제공한다[1, 2].

2.3. Consistency Model

DSM(Distributed Shared Memory)은 병렬화(parallelism) 컴퓨팅을 지원한다. DSM 방식으로 구현하게 되는 스트리밍 서버는 한 대의 서버를 통해 제공되는 스트리밍 서비스를 여러 대의 서버로 병렬 처리하여 클라이언트에게 스트리밍 서비스를 제공해줌으로써 좀 더 향상된 성능의 서비스를 제공해 줄 수 있다.

분산 공유 메모리(DSM : Distributed Shared Memory)은 병렬 프로그래밍 기법 중에 하나로, 다른 머신 간에 서로 공유하는 추상적인 메모리를 프로그래머에게 제공한다. 물리적으로 다음 그림 1은 분산 공유 메모리의 구성을 나타낸다.

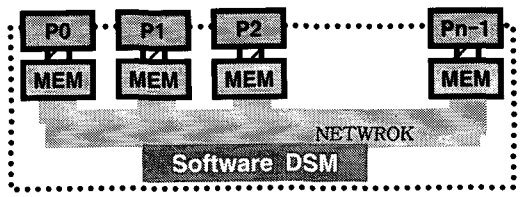

그림 1. 소프트웨어 분산 공유 메모리의 구성
Fig.1 Structure of Software Distributed Shared Memory

DSM은 요구되어지는 다른 프로세서로부터의 메모리 내용을 가져올 수 있어야 한다. 이것은 다른 물리적인 메모리와 같이 공유된 메모리의 다중 복사를 초래한다.

DSM은 이들 다른 복사들의 일관성(consistency)을 유지해야 하며, 공유 메모리에 접근한 프로세서는 올바른 결과를 반환해야 한다. 이러한 작업은 Memory Consistency Model이 담당한다[3, 8]. 다음 그림 2는 Memory Consistency Model 종류를 나타낸다.

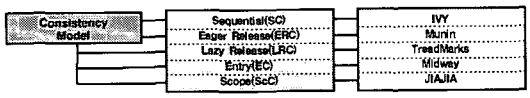

그림 2 Memory Consistency Model 종류 및 대표적인 사용 시스템
Fig. 2 Representative Using System or Memory Consistency Model

그림 2의 Memory Consistency Model들은 이후에 설명한다.

2.3.1 Sequential Consistency

공유 메모리 다중 프로세서를 위해 고안된 가장 일반적인 메모리 일관성 모델은 Sequential Consistency이다. 어떠한 실행 결과가 마치 모든 프로세서의 오퍼레이션이 임의의 순차적인 순서로 실행했던 것과 같고, 각 프로세서의 오퍼레이션들은 프로그램에 의해 명시된 명령의 순서로 나타난다 [4, 10]. 다음 [표 1]은 Sequential consistency에서 같은 네트워크를 구성하는 모든 프로세서 간에 데이터 처리 방법을 나타낸 것이다 [5, 8].

표 1. Sequential consistency 데이터 처리 방법 예제
Table 1. Example of Sequential Consistency Data Management

순서 / 프로세서	1	2	3	4	5
p1	W(x)1				
p2			W(y)2		
p3		R(y)2		R(x)0	R(x)1

[표 1]은 Sequential consistency 모델을 사용함으로써 각 프로세스에 대한 변수의 상태와 공유 메모리에 접근하는데 순서를 가지고 접근하는 것을 보여주고 있다. 프로그램을 수행하는데 있어 순서가 정해져 있다 [6, 9].

2.3.2 Eager Release Consistency(ERC)

Eager Release Consistency 모델은 write 접근 정보가 다음 접근 때의 복사로 인한 지연을 줄이기 위하여 release point에 모든 공유 복사가 제공되며, Release는 모든 cacher들로부터 인정을 받을 때까지 block된다. 접근 실패시, 메시지는 페이지를 위한 디렉토리 관리자로 보내진다. 다음 그림 3은 ERC 기능을 가진 DASH와 LRC 기능을 가진 Munin의 메시지 전송 방법을 나타낸 것이다.

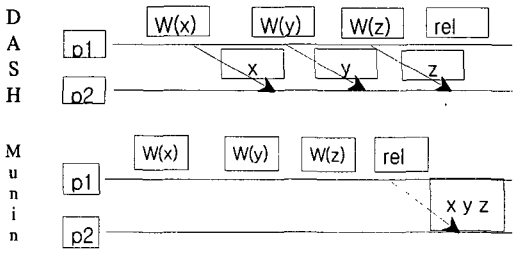

그림 3. DASH와 Munin의 메시지 전송 방법
Fig. 3. Message Passing in DASH and Munin

그림 3의 DASH는 remote memory writes pipeline처리와 release point에서의 동기화를 하며, Munin에서는 remote memory writes 기록과 하나의 메세지로 같은 목적지를 가지는 모든 write들의 병합 메세지 교환을 감소시킨다[5].

2.3.3 Lazy Release Consistency(LRC)

LRC(Lazy Release Consistency)모델의 consistency message는 단지 마지막 releaser와 새로운 acquirer 사이에서만 나타난다. Lazy release consistency 모델은 eager release consistency 모델보다 복잡하다. release 후에, Munin은 release이전에 만들어진 releasing processor의 모든 수정을 잃어버릴 수 있다. lazy consistency 모델에서는 이러한 경우는 없다 [7].

LRC는 다음과 같은 특징을 가지고 있다.: Write 접근 정보는 다음 acquire 지점에 다음 acquiring copy를 위해 제공된다. lock을 취득한 프로세서는 transitive sense에 lock acquire를 선행한 모든 변경을 확인한다. 교환되는 메시지 데이터의 양을 줄인다.

다음 그림 4는 LRC와 ERC의 차이점을 비교할 수 있다.

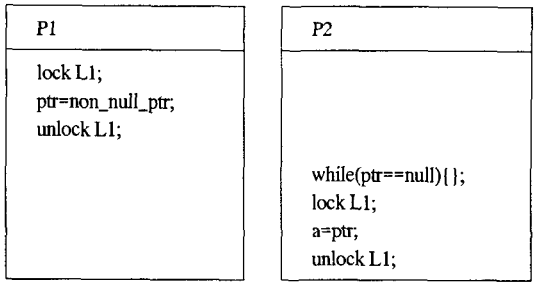

그림 4. LRC와 ERC의 알고리즘
Fig. 4 LRC and ERC Algorithm

그림 4은 ERC와 LRC에서 볼 수 있는 동작으로 새로운 non-null pointer가 P1에 의해 unlock(release)전에 P2로 전달되는 것을 보장한다. LRC일 경우, P2가 lock을 실행할 때까지 P1은 쓰기를 하지 않는다. while loop에서 read가 있기 전에나 synchronization으로 분류하기 전에, 적절한 acquire synchronization이 위치해야 한다[11].

2.3.4 Entry Consistency

Entry consistency 모델은 lock에서 exclusive와 non-exclusive 접근으로 구별된다. 주어진 lock과 함께 연관된 변수들을 write하기 위해서 프로세스는 lock을 소유해야 하고 exclusive 모드에서 대응하는 lock를 얻어야 한다. non-exclusive 모드에서 다중 프로세스들은 같은 lock을 얻을 수는 있지만 연관된 변수들에 대해 단지 read만 가능하다 [12]. 다음 그림 5는 Entry Consistency 알고리즘을 나타낸다.

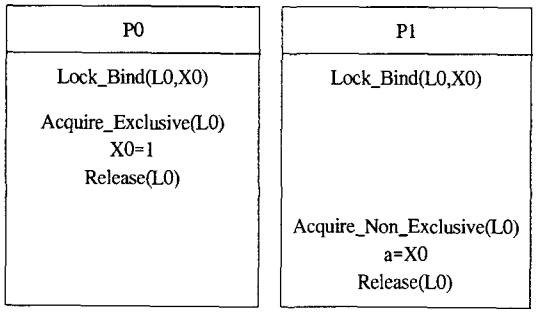

그림 5. Entry Consistency 알고리즘
Fig. 5. Entry Consistency Algorithm

entry consistency를 위한 일관성 규칙들은 다음과 같다. acquire가 실행되는 것을 허용하기 이전에, 보호된 공유 데이터에 대한 모든 업데이트는 프로세스에 대하여 실행되어져야 한다. lock이 exclusive mode에서 취득될 때, 다른 프로세스는 lock을 취득할 수 있는 것은 없고 non-exclusive 모드에서도 마찬가지다. lock이 exclusive 모드에서 취득된 후, 다른 프로세스에 의해 실행된 lock의 다음 non-exclusive acquire는 lock의 소유자에 대하여 실행되고 난 후에만 실행되는 것을 허용한다.

2.3.5 Scope Consistency(SCC)

ScC 모델[1]에서는 scope 개념을 같은 lock을 사용하는 모든 임계영역들로 정의된다. 이것은 lock이 scope를 확실히 정의한다라는 의미이다. scope는 acquire 에서 opened 되고 release에서 closed된다. 다음 그림 6은 Scope Consistency의 동작을 설명한 것이다[6].

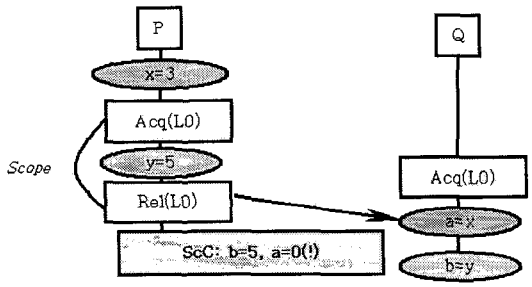

그림 6. Scope Consistency 동작 흐름
Fig. 6. Control Flow of Scope Consistency

그림 5에서는 프로세서 Q가 L0를 취득함에 따라 scope를 열 때, P에서 같은 scopes 내에 업데이트는 Q로 전달된다. 그러므로, b는 5를 읽는 것이 보장되고, P에 의해 업데이트된 y값을 읽을 수 있다. 그러나, a는 3을 읽는 것을 보장하지 못한다. 왜냐하면, P는 L0에 의해 보호되는 scope 밖에서 x를 업데이트하기 때문이다[3].

Ⅲ. 제안하는 Consistency 알고리즘

3.1. 공유 메모리를 이용한 VOD 스트리밍 데이터 처리 방식

논문에서 제안하는 알고리즘을 구현하기 위한 데이타 처리 방식은 하나의 호스트에 한 프래그먼트 크기의 공유 메모리를 할당한다. 이 공유메모리는 barrier 메커니즘을 통하여 RTP패킷 전송 단위로 결정될 프래그먼트 데이터의 조합을 형성한다. 다음 그림 6은 미디어 파일의 프래그

먼트들에 의해 설정된 공유메모리 데이터를 처리하는 방법을 보여준다.

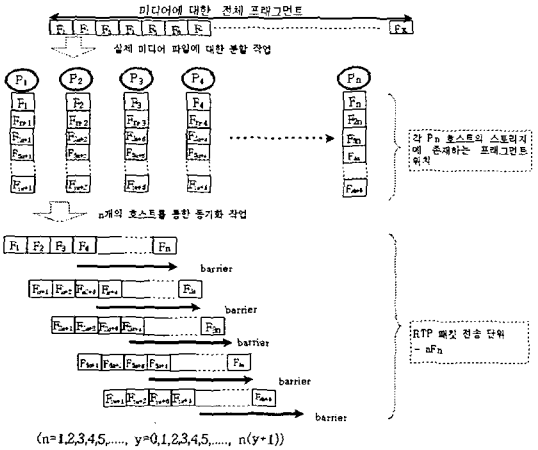

그림 7. Barrier 메커니즘을 통한 미디어 파일의 데이타 처리 과정

Fig. 7. Data Management Flow of Media File in Barrier Mechanism

그림 7에서는 공유메모리의 미디어 데이터에 barrier 메커니즘을 적용하여 RTP 패킷으로 전송하게 되는 과정을 보여준다. 미디어 파일을 구성하는 프래그먼트들은 F1부터 Fy로 구성한다. 이들 프래그먼트들은 라운드로빈 방식으로 각 호스트의 스토리지에 저장된다. 실제 미디어 파일에 대한 분할 작업은 P1에 Fyn+1, P2에는 Fyn+2, Pn에는 Fn(y+1)(y=0,1,2,3,4,…, n=호스트개수) 프래그먼트가 배치된다. 공유 메모리의 프래그먼트들은 barrier 메커니즘을 통하여 제어한다. 이것은 같은 호스트에 배치되어 있는 프래그먼트들간에 동기화를 적용함으로써, 같은 호스트의 다음 위치에 존재하는 프래그먼트들은 n(호스트 개수)만큼 만족하게 될 때까지 읽어들일 수 없도록 하고 현재 공유 메모리에 존재하는 데이터는 barrier를 적용한다. barrier 메커니즘은 공유 메모리에 대한 사용이 모든 호스트에서 완료될 때까지 대기하게 되고, 공유 메모리의 사용이 완료되면 공유메모리 데이터 값을 서비스 할 수 있다.

3.2. 미디어 파일 처리를 위한 동기화 알고리즘

미디어 데이타를 서비스하기 위한 프래그먼트에 대한 동기화 알고리즘은 다음 알고리즘 1과 같다.

```
비디오 데이터(MPEG)에 대한 (SH_GOP)를 기본 단위로 하여 각 노드에 분산하여 저장
(SH_GOP)의 크기 → SH_GOP
각 호스트의 분산 처리를 위해 비디오 서버들( N 대 )을 연결
각 호스트의 프래그먼트 처리를 위한 공유 메모리 설정
if( 클라이언트로부터 미디어 파일에 대한 요청이 발생하면 )
{
  while( 미디어 파일에 대한 프래그먼트를 다 읽을 때까지 반복 )
  {
    if( 각 호스트의 pid가 n 이면 )
    {
      n 번째 호스트의 저장 노드에 위치한 프래그먼트 읽기
      공유 메모리에 읽어 온 프래그먼트를 저장 → 프래그먼트가 저장되는 위치 결정
        → 공유메모리의 주소((SH_GOP×n)의 주소)에 저장
      if(공유 메모리가 full이면)
      {
        RTP패킷 전송 단위로 결정
      }
      공유 메모리에 N개의 프래그먼트 저장을 위한 동기화(barrier)
      if ( pid가 0 이면 )
        공유 메모리의 저장된 데이터 RTP 패킷으로 전송
    }
  }
}
```

알고리즘 1. 미디어 파일 처리를 위한 동기화 알고리즘

Algorithm 1. Synchronization Algorithm for Media File

알고리즘 1은 제안한 Consistency 알고리즘에 대한 의사코드이다. 이 알고리즘은, 분산 공유 메모리 방식의 프로그래밍을 지원하는 JIAJIA 라이브러리를 이용하여 고안된 알고리즘이다. 각 호스트의 저장노드에는 (SH, GOP)를 묶어 프래그먼트 단위로 결정하고, 공유메모리의 사이즈를 결정하는 문제에 있어서 사용될 값으로 SH_GOP로 정의하였다. 프래그먼트의 각 호스트 간의 분산처리를 위한 서버 연결은 JIAJIA 라이브러리를 통해 초기화된다. 공유 메모리를 설정하기 위해서는 *jia_alloc3()*을 이용하여 각 호스트에 프래그먼트 크기 만큼의 공유메모리를 설정하고 전체 공유 메모리 크기를 설정한다. 각 호스트의 스토리지에 저장되어 있는 미디어 파일의 프래그먼트들은 N대 간격으로 프래그먼트들이 배치되어 있다. n(pid) 호스트에 위치한 프래그먼트를 읽기 위해서는 jiapid(프로세스 id)에 따라 구분지을 수 있다. jiapid가 0이면, master 호스트를 가리키게 되고 master 호스트 위치에 첫번째 프래그먼트를 읽어서 공유메모리 처음 위치에 배치한다. 즉, jiapid가 x이면, x번째 호스트의 프래그먼트를 읽어들여 x번째의 공유메모리에 프래그먼트를 위치시키도록 한다. 프래그먼트들이 공유메모리에 배치되는 위치는 현재 해당하는 호스트 영역에 할당된 공유메모리에 프래그먼트를 배치시킬 수 있도록 하기 위해 (SH_GOP×n)의 주소에 저장될 수 있도록 하였는데, 다음 그림 7은 공유메모리 특정 위치에 미디어 파일이 저장되는 것을 나타낸 그림이다.

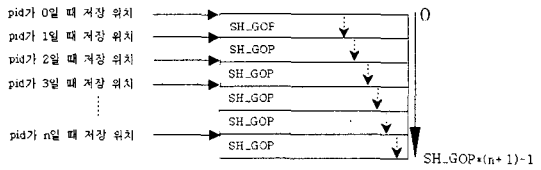

그림 8. pid에 따른 공유 메모리에서의 저장 위치
Fig. 8. Storage Position of Shared Memory in the PID

그림 8은 pid에 따른 공유메모리에 프래그먼트(SH,GOP)의 정보가 동시에 저장이 되면서 전체 공유 메모리 크기인 SH_GOP*(n+1)-1을 채울 때까지 수행한다. N개 프래그먼트 읽기를 전체 호스트로부터 완료하여 공유메모리의 데이터를 생성할 때까지 jia_barrier()를 사용하여 한 호스트의 스토리지에 저장된 프래그먼트에서 다음 프래그먼트를 읽어들이는 작업을 대기할 수 있도록 jia_barrier()함수를 사용하여 동기화한다.

결국, n개의 프래그먼트를 이루어 생성되는 공유메모리의 미디어 패킷을 RTP 패킷 전송단위를 결정하여 사용하며, 공유메모리에 저장된 미디어 데이터를 전송하는 호스트는 pid가 0인 즉 master 호스트에서 전송업무를 담당한다. 이와 같이, 전체 미디어 파일을 모두 읽어들일 때까지 공유메모리에 대한 프래그먼트 저장과 전송 작업을 반복한다.

Ⅳ. 실 험

4.1 실험 환경

실험은 두 대의 리눅스 서버를 사용하였다. 해당 서버는 RedHat Linux 8.0 버전을 이용하였고 gcc 버전은 3.2.2이다. 각 호스트의 CPU와 RAM 은 Pentium III 700Mhz/512M와 Celeron 434Mhz/385M이다. 이 두 대의 시스템에 DSM 환경을 구축하여 본 논문에서 지시한 알고리즘을 적용하였다.

4.2 실험

본 실험은 MPEG 프래그먼트 처리 단위를 GOP로 가정하고, 이 GOP 데이터의 평균적인 크기는 28kbyte 정도이다. 따라서 이 크기의 데이터를 이용하여 본 논문에서 제시한 알고리즘에 대해서 실험을 수행하기 위한 프로그램을 작성하였다. 프로그램은 28kbyte크기의 배열을 만든 후에 여기에 데이터를 읽어들여서 수행하였다.

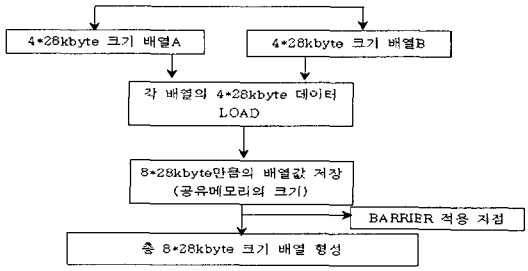

그림 9. 공유메모리를 8*28kbyte로 설정하였을 경우 배열을 합치는 프로그램 구조
Fig. 9 Program Structure of Array Merge in 8*28kbyte Shared Memory

그림 9는 공유메모리의 크기를 배열A와 배열B의 크기를 모두 합친 8*28kbyte로 설정하였다. 이것은 일반적으로 병렬 프로그래밍을 할 때 작성할 수 있는 프로그램 형태로 barrier를 적용하는 시점이 공유메모리에 존재하게 될 모든 데이터의 업데이트가 완료되고 난 다음 마지막 지점에 한번만 사용한다.

프래그먼트 크기를 unsigned int형으로7*1000크기(28kbyte)로 설정하여 각각 4개의 프래그먼트를 가지고 있는 두 배열(local1[ ], local2[ ])을 형성하였다. 이 두 배열을 읽어들이기 위한 공유메모리로 설정된 배열(total[])의 크기는 2*7*1000(unsigned int)를 가지는 배열 크기로 설정하여 barrier를 적용하였다. 적용되는 barrier의 횟수는 총 4회이다. local1배열과 local2 배열이 가지는 값은 다음 [표 2]와 같다.

표 2. local1배열과 local2배열이 가지는 값
Table 2. Local1 and Local2 Array Value

배열 종류 / 배열 위치	local1[ ]	local2[ ]
0~6999	0~6999	7000~13999
7000~13999	14000~20999	21000~27999
14000~20999	28000~34999	35000~41999
21000~27999	42000~48999	49000~55999

[표 2]의 local1과 local2 배열의 데이터 값은 7000 단위로 데이터 값을 읽는다. 즉, 2*7000 크기를 가지는 공유메모리의 total 배열에 배열 local1, local2순으로 7000개의 데이터를 순차적으로 읽은 다음 0~13999 크기를 가지는 total 배열에 데이터 값을 전달하고, 총 4회 반복하여

local1과 local2의 전체 데이터 값을 전달한다. 공유메모리로 전달된 데이터 값은 전체 배열을 형성하는 8*7*1000크기의 main[]배열에 최종적으로 전달되어 local1과 local2 배열을 합친 데이터 값으로 확인할 수 있다. local1배열에 대한 데이터 값 처리는 첫 번째 호스트에서 작업을 담당하며, local2에 대한 데이터 처리는 두 번째 호스트에서 작업을 담당한다.

두 배열을 합치는데 소요되는 시간을 측정한 결과 0.059(∓0.001) 초 정도 소요됨을 확인할 수 있었다. 또한 일반적인 VOD 시스템 형태를 그림 9, 표 2와 같이 구성하여 실험한 결과 배열의 값을 합치는데 소요되는 시간을 측정한 실험에서 소요되는 시간은 0.078(∓0.001) 초 정도 소요됨을 확인할 수 있었다.

따라서 본 논문에서 제시한 VOD 시스템을 위한 Consistency 알고리즘은 VOD 시스템 구조에서 기존의 VOD 시스템 구조에서의 데이터 병합 과정보다 성능이 우수함을 알 수 있다.

V. 결 론

본 논문에서는 VOD 스트리밍 데이터를 처리하기 위한 방안으로 멀티미디어 데이터를 분산 공유 메모리 방식으로 처리할 수 있는 Consistency 알고리즘을 설계하였다. 멀티미디어 데이터는 MPEG-2 미디어 포맷을 기본으로 하며, 미디어 플레이 가능한 최소단위로 SH(Sequence Header)와 GOP(Group Of Picture) 그룹을 공유 메모리에서 처리할 수 있는 기본 단위로 한다. 미디어 데이터 서비스를 위해 우선 하나의 미디어 데이터를 라운드 로빈 방식으로 스트라이핑하여 각 호스트의 저장 노드에 저장한다. 각 호스트의 저장 노드에 스트라이핑된 미디어 파일의 프래그먼트는 분산 공유 메모리 방식을 통하여 RTP 패킷 전송 데이터로 결정될 공유메모리의 조합된 미디어 데이터로 구성된다. 각 노드로부터 미디어 데이터를 합치는 과정에서 barrier 메커니즘을 이용한다. barrier 메커니즘은 분산 공유 메모리 방식에서 사용하는 동기화 기법이다. 본 논문에서는 일반적인 계산 용도로 사용되는 barrier 메커니즘을 미디어 플레이 최소 단위인 (SH. GOP) 미디어 데이터에 서비스할 수 있도록 제시하였다. 미디어 데이터를 처리하기 위해 적용된 barrier 메커니즘은 각 호스트 프래그먼트들을 합치는 과정이 완료될 때까지 대기상태로 있다가 미디어 데이터 병합이 완료되면, RTP 프로토콜을 통하여 클라이언트에 서비스한다.

RTP 패킷 데이터로 결정된 데이터는 빠른 미디어 서비스 제공을 위하여 본 논문에서 제시한 consistency 알고리즘에 추가 기능이 설계되었다. 동기화에 따라 공유메모리에 존재하는 멀티미디어 데이터의 순서화된 서비스를 보장하는 것이다. RTP 프로토콜을 통해 제공할 공유메모리의 미디어 데이터 간에 올바른 데이터 서비스를 위하여 이 기능이 필요하다.

또한 본 논문에서 제시한 VOD 시스템을 위한 Consistency 알고리즘은 VOD 시스템 구조에서 기존의 VOD 시스템 구조에서의 데이터 병합 과정보다 성능이 우수함을 알 수 있다.

참고문헌

[1] L.lftode, J.P. Singh and K.Li. "Scope Consistency: A Bridge between Release Consistency and Entry Consistency". In Proc. of the 8th Annual ACM Sym. on Parallel Algorithms and Architectures, June 1996.

[2] W. Hu, W. Shi, Z. Tang, M.Rasit. Eskicioglu "JIAJIA User's Manual", June 3, 1998.

[3] The JUMP Software DSM Software, http://www.srg.csis.hku.hk /srg/html/jump.htm

[4] H. Schulzrinne, et al., "RTP: A Transport Protocol for Real-Time Application", RFC 1889, Jan. 1996

[5] J. Cater, J. Bennet, and W. Zwaenepoel, "Implementation and Performance of Munin", in Proc. of the 13th ACM Sym. on Operating Systems Principles, pp.152~164, Oct.1991.

[6] W. Hu, W. Shi, Z. Tang, and M. Li, "A Lock-based Cache Coherence Protocol for Scope Consistency", Journal of Computer Science and Technology, Vol. 13, No. 2, pp. 97~109, March 1998.

[7] P. Keleher, A. Cox, and W. Zwaenepoel, "Lazy Release Consistency for Software Distributed Shared Memory", in Proc. of ISCA '92, pp. 13~21, 1992

[8] Sarita V. Adve, Kourosh Gharachorloo, "Shared Memory Consistency Models: A Tutorial", WRL Reasearch Report, July 1995

[9] David Mosberger, "Memory Consistency Model[Mos93]", Dept. of Computer Science, The Univ. of Arizona, Tucson, AZ85721, September 1993

[10] Leslie Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs", IEEE Transactions on Computer, C-28(9): 690~691, September 1979.

[11] S.V. Adve, A.L. Cox, S. Dwarkadas, R. Rajamony, and W. Zwaenepoel. "A Comparison of Entry Consistency and Lazy Release Consistency Implementation." In The 2nd IEEE Symposium on High-Performance Computer Architecture, February 1996.

[12] B.N. Bershad and M.J. Zekauskas. "Midway: Shared Memory Parallel Programming with Entry Consistency for Distributed Memory Multiprocessors". Technical Report CMU-CS-91-170, Carnegie Mellon University, September 1991.

[13] Yang, Z., Sun, C. Sattar, A., Yang, Y., "A new look at multimedia synchronization in distributed environments", Parallel Architectures, Algorithms, and Networks, 1999.(I-SPAN'99) Proceedings. Fourth International Symposium on 23~25, June 1999.

[14] William I. Grosky, Ramesh Jain, Rajiv FMehrotra, "The Handbook of multimedia information management", Prentice-Hall, Inc. 1997.

저자소개

장 승 주(Jang, Seung Ju)

1985년 부산대학교 계산통계학(전산학) 학사
1991년 부산대학교 계산통계학과(전산학) 석사
1996년 부산대학교 컴퓨터공학과 박사
1987년~1996년 한국전자통신연구원 시스템 S/W연구실
1993년~1996년 부산대학교 시간강사
2000년~2002년 University of Missouri at Kansas City, visiting professor
1996년~현재 동의대학교 컴퓨터공학과 부교수
※관심분야 : 운영체제, 임베디드 운영체제, 분산시스템, 시스템 보안