Journal of The Korea Society of Computer and Information
Vol. 18, No. 4, April 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.4.131

시스템수준 시뮬레이션과 디스크 I/O수준 시뮬레이션 연동을 위한 DEVSim++과 DiskSim 사이의 인터페이스 설계 및 구현

송 해 상*, 이 순 주**

Design and Implementation of DEVSim++ and DiskSim Interface for Interoperation of System-level Simulation and Disk I/O-level Simulation

Hae Sang Song*, Sun Ju Lee**

요 약

본 논문은 분산 스토리지 시스템과 같이 디스크를 내장하고 있는 다수의 컴퓨터 노드로 구성되어 있는 스토리지 시스템에서 개개의 디스크 스펙 변화에 따른 전체 성능을 분석하고자 할 때 잘 알려져 있는 블록 I/O 수준의 디스크 시뮬레이터인 DiskSim과 시스템수준의 시뮬레이터와의 연동을 위한 인터페이스의 설계 및 구현에 관한 기법을 제안하였다. 본 연구에서 시스템수준 시뮬레이션 엔진으로는 계층적이고 모듈러한 모델링 기법을 지원하는 DEVS 형식론을 지원하는 범용 이산사건시스템 시뮬레이션 엔진인 DEVSim++을 목표로 하였고 이식성을 위해 DiskSim과 DEVSim++ 시뮬레이션 엔진의 내부는 수정하지 않는 것을 가정하였다. 이를 만족하기 위해 I/O 수준의 DiskSim 시뮬레이터와 시스템 수준의 DEVSim++ 기반 시뮬레이터 사이의 연동 인터페이스 구조를 제안하였다. 이 구조에서는 여러 인스턴스의 DiskSim을 관리하는 DiskSimManager의 개념을 도입하여 I/O 수준 시뮬레이션과 시스템 수준 시뮬레이션 간의 시간과 사건(데이터) 동기화를 담당하도록 설계하였고, 시스템 수준의 DEVS 모델에서 간편하게 DiskSim을 접근할 수 있도록 감싸는 DEVS wrapper 모델을 제안하였다. 벤치마크 실험결과 설계 구현된 연동 인터페이스를 사용한 시뮬레이션 결과는 DiskSim만의 단독 시뮬레이션 결과와 정확히 일치함을 확인함으로써 설계 구현된 연동 인터페이스가 목적에 맞게 잘 동작함을 입증하였다.

▸ Keywords : DiskSim, DEVSim++, 시스템수준 시뮬레이션, 디스크 시뮬레이션, 인터페이스

•제1저자 : 송해상 •교신저자 : 송해상
•투고일 : 2013. 3. 11, 심사일 : 2013. 4. 16, 게재확정일 : 2013. 4. 22
* 서원대학교 컴퓨터공학과 교수(Dept. of Computer Engineering, Seowon University)
** 한국과학기술원 전기및전자공학과 석사과정 (Korea Advanced Institute of Science and Technology)

Abstract

This paper deals with the design and implementation of an interface for interoperation between DiskSim, a well-known disk simulator, and a system-level simulator based on DEVSim++. Such inter-operational simulation aims at evaluation of an overall performance of storage systems which consist of multiple computer nodes with a variety of I/O level specifications. A well-known system-level simulation framework, DEVSim++ environment is based on the DEVS formalism, which provides a sound semantics of modular and hierarchical modeling methodology at the discrete event systems level such as multi-node computer systems. For maintainability we assume that there is no change of the source codes for two heterogeneous simulation engines. Thus, we adopt a notion of simulators interoperation in which there should be a means to synchronize simulation times as well as to exchange messages between simulators. As an interface for such interoperation DiskSimManager is designed and implemented. Various experiments, comparing the results of the standalone DiskSim simulation and the interoperation simulation using the proposed interface of DiskSimManager, proved that DiskSimManager works correctly as an interface for interoperation between DEVSim++ and DiskSim.

▸ Keywords : DiskSim, DEVSim++, System-Level Simulation, Disk Simulation, Interface

I. 서 론

최근 빅데이터 처리에 대한 관심이 대두됨에 따라 하둡(Hadoop)과 같은 빅데이터 저장 및 처리 플랫폼에서 큰 데이터의 저장을 담당하는 저장장치가 전체 빅데이터 처리 성능에 미치는 연구가 중요해지고 있다. 특히 저장장치 중 반도체 저장장치인 SSD(Solid State Disk)는 기존의 HDD(Hard Disk)에 비해 성능이 월등히 빠르고 가격이 저렴해짐에 따라 HDD를 대체하려 하는 추세이이다. 여러 가지 SSD의 성능 스펙 중에서 빅데이터에 가장 적합한 스펙을 가진 디스크를 설계하기 위해서는 스펙의 변화가 전체 빅데이터 처리 성능에 미치는 영향을 분석해야 한다. 하지만 비용과 시간적 측면에서 실제 시스템으로 이를 분석하는 것이 불가능하기 때문에 시뮬레이션 기법을 사용하는 것이 필요하다.

이러한 분산저장장치 시뮬레이션은 적용하는 디스크에 대한 시뮬레이션과, 분산 저장을 위한 네트워크 프로토콜에 대한 시뮬레이션이 연동 또는 결합되어야 한다. 다행히도 디스크 성능 시뮬레이터로는 현재까지 DiskSim이 검증된 시뮬레이터로 알려져 있다[1]. 이는 HDD 뿐만 아니라 SSD까지 미세한 수준에서 시뮬레이션 할 수 있기 때문이다[4]. 이 때 디스크 스펙의 변경이 전체 빅데이터 처리 시스템에 미치는 영향을 분석하기 위해서는 DiskSim과 시스템 수준 시뮬레이터를 연동하는 것이 필요하다.

빅데이터가 저장되는 HDFS[7]와 같은 수많은 노드로 구성된 상위 수준의 복잡한 분산저장시스템의 거동을 모델링 시뮬레이션하기 위해서는 노드의 확장성과 복잡한 문제를 다룰 수 있는 이산사건 수준의 모델링 시뮬레이션 개발 환경이 필요하다. 다양한 개발환경이 있지만 그 중에서도 DEVS(Discrete Event Systems Specification)형식론은 계층적이고 모듈러하며 객체지향 표현기법과 개발환경인 DEVSim++를 지원하기 때문에 이러한 응용에 적합하다고 판단된다[2][3][9].

하지만 이러한 다중 노드로 구성된 디스크를 내장하고 있는 분산저장시스템 시뮬레이터 개발을 쉽게 지원하기 위해서는 I/O 수준의 시뮬레이터인 DiskSim과 상위 수준의 시스템 시뮬레이션 엔진인 DEVSim++를 연동하는 DiskSim-DEVSim++ 인터페이스를 제공하는 것이 요구된다. 따라서 본 논문에서는 다음과 같은 요구사항을 만족하는 인터페이스를 제안하고자 한다.

- 유지보수성 - 시뮬레이션 엔진 코드의 변경은 없어야 한다.
- 편의성 - 시스템수준 모델링 시 DiskSim과의 인터페이스는 단순해야 한다.
- 재사용성 - 다양한 유형의 모델을 교체 가능해야 한다.
- 확장성 - 멀티 노드의 확장을 지원해야 한다.

본 논문의 2장에서는 이론적 배경 및 연구목적을 다루고 3장에서는 제안된 인터페이스 설계 구현 기법을 제시하며, 4장에서는 다양한 벤치마크 실험 및 결과분석을 통해 설계 구현된 DiskSim-DEVSim++ 인터페이스가 정확히 동작함을 검증하고 5장에서 결론을 맺고자 한다.

II. 관련 연구

2.1 문제 정의

최근 MRSim[5], Mumak[6] 등 MapReduce 개념을 적용한 Hadoop[7] 등 빅데이터 처리 플랫폼에 대한 시뮬레이터 연구개발이 많이 이루어져 왔다. 하지만 이들은 모두 상위수준의 시스템 시뮬레이터로 디스크 등의 성능은 중요하게 모사하고 있지 않다. 한편 국내의 플래시 메모리 업계에서는 특히 SSD(Solid State Disk)[10]등의 새로운 디스크의 특성을 반영한 수천 대의 노드로 구성된 빅데이터 처리 클러스터의 성능에 관심을 가지게 되었고, 새로운 디스크 설계 스펙의 최적화 탐색을 위해 디스크를 자세하게 모사하는 빅데이터 처리 플랫폼 시뮬레이터가 필요하게 되었다. 그림 1은 최종 목표로 하는 간략한 빅데이터 플랫폼에 대한 시뮬레이터 구조이다.

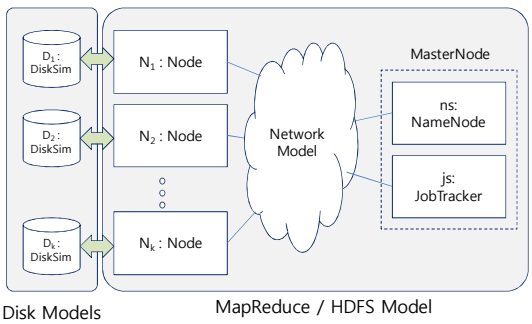


그림 1. 목표 시스템 및 인터페이스 문제 정의
Fig. 1.Target system with the interface required

여기서 디스크 스펙의 변화가 전체 처리 성능에 미치는 영향이 매우 중요하기 때문에 디스크 모델은 매우 상세해야 한다. 따라서 좌측은 상세한 디스크 스펙에 대한 성능을 평가하는 시뮬레이터인 DiskSim을 사용하는 것을 가정한다. 우측은 빅데이터 처리를 위한 컴퓨터 노드 모델들로 구성된 시뮬레이터를 도식화한 것으로 범용 이산사건시뮬레이션 엔진인 DEVSim++ 기반으로 설계되었음을 가정한다. 우측의 각 노드 모델은 실제와 유사하게 하나의 DiskSim 시뮬레이터와 독립적으로 연동하여 동작하도록 가정한다. 두 개는 서로 다른 형태의 시뮬레이터이기 때문에 연동을 위해서는 시간의 동기화와 데이터의 변환이 필요하다. 따라서 본 논문에서는 여러 개의 독립적인 DiskSim 시뮬레이터들과 DEVSim++시뮬레이션 엔진과의 이기종 시뮬레이터들에 대해 시간과 데이터를 교환하여 정확한 시뮬레이션을 가능하게 하는 인터페이스를 설계하고 구현하는 문제를 다루고자 한다.

2.2 DiskSim : 디스크전용 이산사건 시뮬레이터

DiskSim은 컴퓨터 기술의 발전에 따라 CPU, 기억장치 등의 속도가 빨라짐에 따라 디스크의 성능이 전체 컴퓨터 시스템의 성능에 미치는 영향이 커져서 다양한 워크로드에 대해 디스크의 성능을 실험하기 위해서 카네기멜론 대학에서 개발한 디스크 시뮬레이터이다[1]. 이는 I/O 버스, 디스크 컨트롤러, 그리고 컨트롤러에 부착된 여러 개의 디스크 구조에 대해 성능을 평가한다. DiskSim은 확장 가능한 구조를 제공하여 다양한 디스크 모델에 대해서 잘 검증된 시뮬레이터로서 알려져 왔으며 최근 SSD가 HDD의 대체 저장장치로 대두되면서 SSD에 대한 확장 모듈이 마이크로소프트웨어에 의해 개발되고 사용되어 왔다[4][8].

DiskSim은 컴퓨터 노드 1개 수준의 상위 시뮬레이터의 제어를 받는 슬레이브로 동작할 수 있도록 시스템 수준의 시뮬레이터와 결합할 수 있는 인터페이스를 제공하고 있다. 하지만 근본적으로 한 컴퓨터 노드에 부착된 형태로의 시뮬레이션을 가정하고 있고 C언어로 개발되어 객체지향 기술이 적용되지 않아 여러 노드로 이루어진 분산 저장시스템의 경우 여러 개의 DiskSim 인스턴스(프로세스)를 사용해야 하는 한계점을 가지고 있다.

2.3 DEVSim++ : 범용 이산사건 시뮬레이션 엔진

DEVS (Discrete Event System Specification) 형식론[3]은 이산사건시스템에 대한 모델링 형식론으로 이러한 응용시스템을 객체지향적으로 모델링하기에 적절한 모델링 방법론이다. DEVSim++ 환경은 DEVS 모델 개발자를 위한 시뮬레이터 구현환경으로, C++ 용 API 가 제공되며 모델 개발

자는 API 을 이용하여 DEVS 명세서를 C++ 로 구현하면 된다[2]. 구현된 DEVS 모델은 DEVSim++ 내부에 내장된 계층적 시뮬레이션 알고리즘에 의해 자동으로 실행된다.

2.4 이기종 시뮬레이터 사이의 연동 시 고려사항

일반적으로 이기종 이산사건시뮬레이터 간의 연동을 위해서는 먼저 시간이 정확히 동기화되어야 하며, 둘째 사건 또는 메시지가 서로 호환성을 가지고 교환되어야 한다. 이기종 시뮬레이터간의 시간 동기화는 1) 전역 시간관리 방법 2) 마스터-슬레이브 기법을 사용할 수 있다. 1)은 전역 시간관리 모듈을 두어 모든 시뮬레이터가 제어를 받는 방식이며, 2)는 시뮬레이터 중 어느 하나가 전체 시뮬레이터의 시간진행을 제어하는 방식이다. 본 논문에서는 성능과 기능을 고려하여 1)과 2)를 적절히 혼합한 하이브리드 기법을 제안한다. 데이터의 교환은 각 시뮬레이터에서 발생하는 사건에 대한 메시지를 다른 시뮬레이터에 변환하여 올바른 모델에 전달하는 것을 의미하는 것으로 이것은 구현에 의존한다.

디스크 시뮬레이터인 DiskSim은 디스크의 성능에 특화된 이산사건 시뮬레이터이며, DEVSim++는 범용 이산사건시뮬레이션 엔진이다. 하지만 모델링 형식론이 다르기 때문에 결합 또는 연동하여 시뮬레이션하기 위해서는 상호간의 시간과 사건의 동기화가 필요하다. 더구나 분산저장장치는 매우 많은 컴퓨터 노드들로 구성되어 있기 때문에 각 노드들은 하나의 DiskSim 인스턴스들과 DEVSim++기반 시뮬레이터의 연동이 요구된다.

III. DiskSim-DEVSim++ 인터페이스 설계

3.1 연동 구조 설계

이기종 시뮬레이터 간 시간 및 사건의 연동 인터페이스는 다음과 같은 원칙으로 설계하였다. 1) DiskSim은 한 컴퓨터 노드의 시뮬레이터이기 때문에 노드 모델당 1개의 프로세스를 생성한다. 그 이유는 DiskSim의 유지보수성 때문에 C로 된 소스코드를 수정하지 않는다는 전제를 가지기 때문이다. 2) DiskSim 인스턴스들간의 시간 동기화 및 관리를 위해 Manager 개념을 도입한다. 3) DEVSim++내에는 1개의 DiskSim 인스턴스에 대해 1개의 DEVS Wrapper 모델을 제공한다. 따라서 DEVSim++ 노드 모델은 직접 DiskSim과 연동하지 않고 제공된 Wrapper 모델에게만 이벤트를 전달하면 나머지는 자동으로 처리하도록 한다.

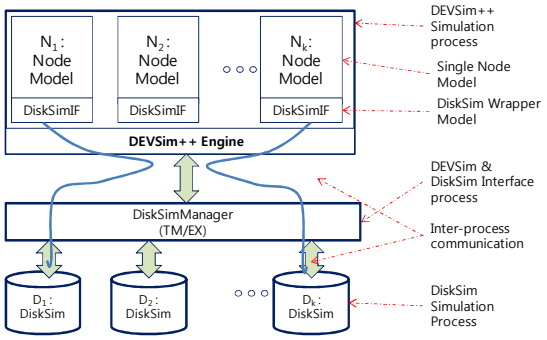


그림 2. 제안된 DiskSim-DEVSim++ 연동 아키텍춰
Fig. 2. Proposed interoperation architecture between DEVSim++ and DiskSim

그림 2는 위와 같은 설계 원칙에 의해서 설계된 실행구조이다. 굵은 실선으로 표기된 것은 프로세스로 크게 세 종류의 프로세스로 구분된다. 상단은 DEVSim++ 기반의 시스템 수준 시뮬레이션 프로세스이다. 하단은 다수의 DiskSim 시뮬레이션 프로세스들이다. 중단은 하단의 DiskSim 프로세스들을 관리하는 DiskSimManger (DSM) 프로세스이다.

이러한 프로세스들은 TCP/IP로 통신한다. 상단의 시스템 수준의 시뮬레이터 내에는 다수의 노드 모델로 구성되어 있으며, 그 내부에는 다시 여러 컴포넌트 모델들로 구성되지만, DiskSim으로 디스크블록에 접근 요구를 보내기 위해서는 DiskSimIF로 표기된 Wrapper 모델에게 요청 사건을 보내면, DiskSimIF는 이를 DEVSim++ 시뮬레이터 엔진으로 보내고, 엔진은 이를 DSM에게 통신 패킷으로 변환하여 보내며 최종적으로 해당 DiskSim 프로세스에게 전달된다. 처리가 완료된 결과는 역순으로 해당 노드의 DiskSimIF 모델로 보내지고 원래 요청한 모델에 결과 이벤트를 전달하는 방식으로 구조를 설계하였다.

이러한 구조의 장점은 사용자(모델러)는 DiskSim의 존재는 의식하지 않고 대표하는 DEVS 모델인 DiskSimIF 모델에게만 DEVS 이벤트를 보내면, 나머지는 설계된 인터페이스에 의해 자동으로 처리된다는 것이다. 부가하여, 추후 디스크의 상세한 시뮬레이션이 필요하지 않을 경우 DiskSim 시뮬레이터를 제거하고 간략한 메타모델로 대체할 수 있는 장점을 가진다.

DSM의 역할은 DiskSim 프로세스의 생성 및 제거 등의 하위 시뮬레이터 관리, DEVSim++ 엔진으로부터 요청된 디스크 요청 (디스크 블록 읽기, 쓰기 등)을 해당 DiskSim 프로세스에게 전달하는 사건전달(EX), 상위 시뮬레이션 시간과 하위 DiskSim 시뮬레이션 시간들과의 동기화 (TM)를 담당한다.

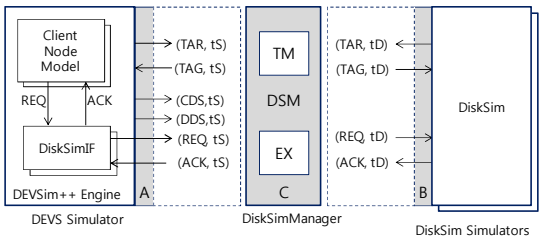


그림 3. 연동 메시지 및 인터페이스 정의
Fig. 3. Definition of interoperation messages

3.2 상호 연동을 위한 인터페이스 메시지 정의

그림 3은 그림2에서 제안된 구조의 연동 설계를 위해 필요한 DEVS Simulator 쪽의 입출력 메시지와 DiskSim 쪽의 입출력 메시지를 개념적으로 정의한 것이다. 모든 메시지는 항상 시간 스탬프를 지니며 DEVS 쪽 시간은 tS로, DiskSim쪽 시간은 tD로 구분하였다.

각 메시지의 의미는 다음과 같다. 먼저 시간 동기화와 관련하여 (TAR, tS)는 DEVSim++ 엔진이 어떤 입출력도 없을 경우 상위의 시간관리자에게 다음 시뮬레이션 시간을 tS로 스케쥴을 진행해도 좋은지 시간진행허가(Time Advance Request)을 요청하는 것이다. (TAG, tS)는 시간관리자가 tS까지 시간진행을 해도 좋다는 허락 (Time Advance Grant)메시지이다. DiskSim의 (TAR, tD) 및 (TAG, tD)도 같은 의미이다.

디스크 관련한 메시지 REQ는 DEVS 모델이 DiskSimIF 모델을 통해 DiskSim 쪽으로 보내는 디스크블록 I/O 요청으로 REQ는 (ID, N#, D#, B#, S) 형식을 가지며 여기서 ID는 요청식별자, N#은 컴퓨터 노드 번호, D#은 디스크 번호, B#은 블록 번호, S는 접근하고자 하는 인접된 블록의 개수를 의미한다. ACK는 DiskSim에서 DEVSim++쪽으로 보내는 것으로 (ID, RESULT)로 구성되며, ID는 요청했던 요청식별자이다. 마지막으로 CDS 메시지는 DEVS Simulator에서 DSM으로 보내는 메시지로 노드번호 N#을 파라미터로 보내어 해당 노드의 DiskSim 인스턴스를 생성하라는 명령이다. DDS 메시지는 노드번호 N#에 해당하는 DiskSim 인스턴스를 제거하라는 메시지이다.

이어지는 절에서는 그림 3에서 A, B, C로 표기된 인터페이스 및 DSM의 설계 및 구현기법에 대해 기술한다.

3.3 DiskSimManager 알고리즘 설계 구현

DSM은 DEVS Simulator 및 DiskSim 양쪽의 메시지를 받아서 시간관리, DiskSim 프로세스 관리, 디스크 I/O 메시지 순서 관리를 하여 양측의 서로 다른 프로토콜을 정합시켜

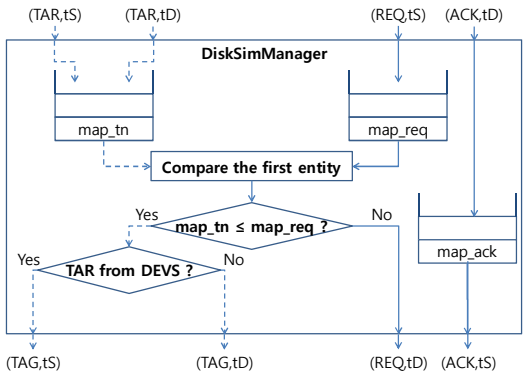


그림 4. DiskSimManager 알고리즘
Fig. 4. DiskSimManager algorithm

주는 역할을 한다. 이를 위하여 DSM의 자료구조는 그림 4와 같이 세 개의 큐를 사용한다. 먼저 큐 map_tn은 DEVS Simulator와 DiskSim이 DSM으로 전송하는 시간 진행 요청(TAR, tS), (TAR, tD)을 시간 순서대로 오름차순으로 저장하고, map_req는 DEVS Simulator에서 DSM으로 전송하는 (REQ,tS)를 시간 순서대로 오름차순으로 저장한다. 마지막으로 DiskSim의 시간진행결과 발생하는 I/O 요청에 대한 처리가 끝나면 (ACK,tD)가 생성되어 DSM의 map_ack에 저장된다.

DEVS Simulator와 DiskSim으로부터 시간 진행 요청과 TAR가 전송될 때마다 DSM은 map_tn의 첫 번째 인자와 map_req의 첫 번째 인자의 시간을 비교한다. map_tn의 첫 번째 인자의 시간이 더 작은 경우 해당 시간 진행 요청을 보낸 DEVS Simulator 혹은 DiskSim에게 시간 진행 허가 메시지 TAG를 전송한다. 반면 map_req의 첫 번째 인자의 시간이 TAR 시간보다 더 작은 경우 메시지(REQ,tD)를 DiskSim에게 전송한다. 이와는 별도로 DiskSim은 (TAG,tD)를 받았을 때 시간을 진행시키며, 그 결과로 I/O 요청이 완료되었음을 감지하였을 때 (ACK,tD) 메시지를 만들어 DSM으로 보낸다. DSM은 이를 해당시간에 (ACK,tS) 메시지로 DEVSim++ 쪽으로 보낸다. DSM은 이러한 과정을 메시지가 들어올 때마다 반복함으로써 시뮬레이션 시간을 관리하고, 각 시뮬레이터 간의 데이터를 교환하는 역할을 수행한다.

설계된 프로토콜 구현 시 프로세스 사이의 통신은 TCP/IP 통신을 사용하였으며 기본적으로 DSM이 서버, DEVS 시뮬레이터 및 DiskSim 프로세스들은 클라이언트로 동작하도록 구현하였다. DSM은 서버로서 대기하고 있다가 클라이언트들로부터 받은 메시지를 분류하여 설계한 바와 같

이 세 개의 큐 및 알고리즘을 C로 구현하였다.

3.4 DEVS-DSM 인터페이스 설계 구현

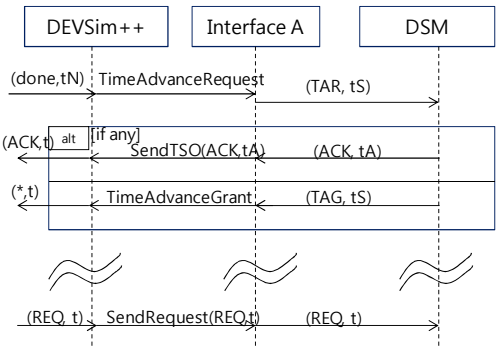


그림 5. DEVSim++과 DSM 인터페이스 설계 구현
Fig. 5. Interface A between DEVSim++ and DSM

그림 5는 그림 3에서 A로 표기된 DEVSim++ 인터페이스의 프로토콜이다. InterfaceA는 DEVS 시뮬레이터 측에서 DSM인터페이스를 위한 어댑터 코드부분을 의미한다. DEVSim++ 엔진에서 다음 스케쥴 시각 tN을 하기 전에 외부에 허락을 요청하는 함수인 TimeAdvanceRequest가 호출되는데, 이 때 InterfaceA 코드는 DSM으로 (TAR,tN)을 보낸다. 그리고 만약 tS 〈 tN인 완료 메시지 (ACK,tS)가 오면 그 후의 메시지를 무효화시키고 DEVSim++ 쪽에 ACK 사건을 TSO (Time Stamp Order) 큐에 삽입한다. 그렇지 않다면 TAG 메시지를 받고 다음 스케쥴을 진행한다. 만약 DEVS 모델에서 시뮬레이터 외부로 나가는 (REQ,t) 이벤트가 Wrapper에 의해 발생했을 때 InterfaceA는 이를 곧바로 DSM으로 메시지를 보낸다.

3.5 DiskSim-DSM 인터페이스 설계 구현

DiskSim은 상위 시스템수준 시뮬레이션과의 연동을 위해 외부 API 5개 및 특별한 사건이 발생할 때 호출하는 3가지 callback 함수를 등록할 수 있도록 하였다. 그 중 본 연구와 관련 있는 API는 다음과 같다.

- disksim_internal_event: 주어진 시각까지 DiskSim의 시뮬레이션을 진행하라는 함수이다.
- disksim_request_arrive: I/O request를 대기열에 삽입한다.

또한 DiskSim 내부에서 기 입력된 I/O 요청이 완료되었을 때 등록된 callback_complete 함수가 자동으로 호출된다.

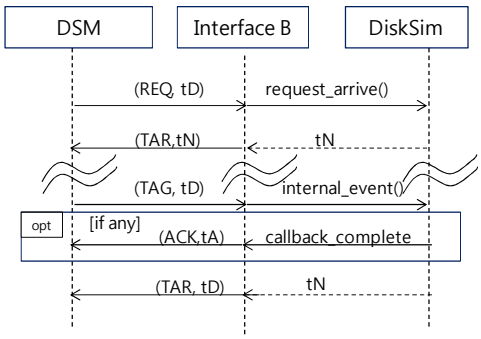


그림 6. DSM-DiskSim 인터페이스 설계 구현
Fig. 6. Interface B between DSM and DiskSim

그림 6은 그림 3에서 B로 표기된 DiskSim 및 DSM간의 인터페이스 프로토콜이다. DSM이 DEVS 모델에서 받은 REQ 메시지를 전달하면 DiskSim이 제공하는 API인 request_arrive함수를 호출한다. 이 때 다음 스케쥴 시각 tN이 결정되며, 이 때 (TAR, tN) 메시지를 보내 tN까지의 시간진행을 요청한다. 만약 DSM이 시간진행허가 메시지 (TAG, tD)를 보내면 DiskSim은 internal_event 함수를 불러 시간을 진행시킨다. 이 때 완료된 I/O 요청이 있으면 DSM을 보내고 다음 시각까지 TAR 메시지를 보내고 TAG를 기다리는 것을 반복적으로 수행하도록 구현하였다.

3.6 DiskSimIF (wrapper) DEVS 모델 설계 구현

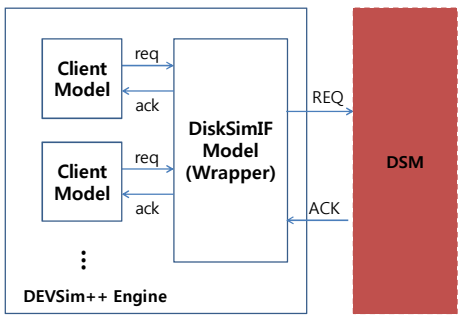


그림 7. DiskSimIF DEVS 모델
Fig. 7. DiskSim IF DEVS model

그림 7의 DiskSimIF (wrapper) 모델은 DEVS 모델로서 DEVS 클라이언트 모델들이 해당 DiskSim으로 I/O요청을 보낼 때 DiskSim을 대신하여 송수신하는 역할을 하며 DiskSim 인스턴스와 1:1의 관계를 가지고 존재한다. 즉 DEVS 시뮬레이터 내부의 클라이언트들은 직접 DSM을 통해 DiskSim으로 요청하지 않고 다만 DiskSimIF DEVS 모

델에게 요청하고 결과를 받아간다. 그림 7에서 좌측의 REQ 및 ACK의 송수신 주체는 바로 DiskSimIF이다. 이와 같은 설계는 모델링의 편이성을 제공하기 위한 것으로 추후 DiskSim이 불필요할 경우 추상화된 통계 모델로 대체할 수 있도록 한 것이다.

IV. 실험 결과

4.1 실험 설계

본 논문에서 제안하는 DEVS-DiskSim 연동 구조를 검증하기 위하여 다음과 같은 두 가지 실험을 설계하고, 그 결과를 비교하였다. 본 연구에서 제안 및 구현한 DiskSim과 DEVSim++ 연동 인터페이스 동작의 정확성을 비교 검증하기 위해 그림 8(a)와 같이 블록 I/O trace 파일을 입력하여 DiskSim을 동작시키는 단독 시뮬레이션 환경을 구축하였다. 그림 8(b)는 제안 및 구현한 연동 인터페이스의 정확성을 실험하기 위해 구축된 것으로 DEVSim++ 환경에서 두 개의 trace 실행 모델인 Generator #1 및 #2와 구현한 DiskSimManager를 비롯한 연동 인터페이스, 그리고 각 Generator 별로 각각 한 개씩 2개의 DiskSim 프로세스가 연동되는 실험환경을 나타낸 것이다. 그림 8(b)의 DiskSimWrapper은 DiskSimIF 모델을 나타낸다. 실험은

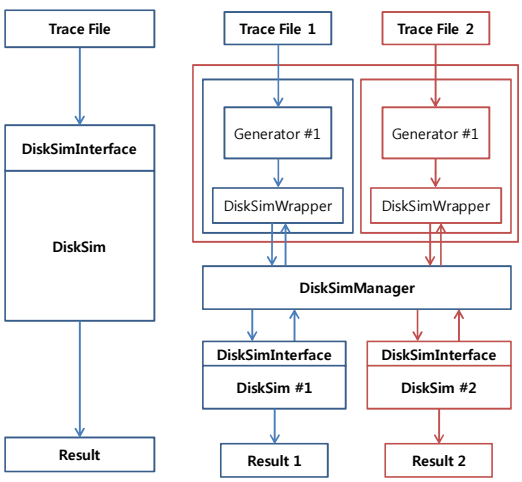


그림 8. (a) 단독 시뮬레이션 구조 (b) 연동시뮬레이션 구조
Fig. 8. (a) Standalone simulation (b) interoperated simulation architecture

2가지의 서로 다른 디스크 모델과 trace를 입력하여 단독 환경에서 얻은 각각의 결과들과 연동실험환경에서 얻은 결과들을 비교하여 서로 동일한 결과를 얻는지 확인하는 절차로 진행하였다. 즉, 단독 환경에서의 결과와 연동 환경에서의 결과가 같음을 보임으로써 제안 및 구현한 연동 구조를 검증한다. DiskSim은 CMU에서 제공하는 DiskSim 4.0을 사용하고, SSD를 시뮬레이션하기 위하여 Microsoft에서 개발한 SSD Extension 버전을 사용하였다.

표 4. 시뮬레이션 실험 환경 설정
Table 1. Experiments configuration

설정	실험 1	실험 2
입력 trace 파일	ssd-iozone-aligned2-100K.trace	ssd-postmark-aligned2.trace
디스크 모델	ssd-iozone.parv	ssd-postmark.parv
통계 설정	statdefs	

표 1은 두 가지 실험, 실험 1과 실험 2에 대한 입력 파라미터를 표시한 것이다. 입력 trace 파일은 DiskSim이 제공하는 2개의 표준 파일을 사용하였고, 디스크 모델도 서로 다른 두 개를 사용하였으며, 통계 설정 파일은 동일한 것을 사용하였다. 입력으로 사용하는 실험 1의 trace 파일은 10만 개, 실험 2의 trace파일은 약 6만 개의 블록 I/O requests로 구성되어 있고, 각각의 request는 request arrival time, device number, block number, request size, request flags로 구성되어 있다. 기타 파라미터는 bus, controller, driver, disk 등에 대한 특성들로 구성되어 있다. 통계 파라미터는 평균, 표준 편차 등 측정하고자 하는 통계출력결과 항목을 DiskSim에게 명세하기 위한 것이다.

4.2 단독 시뮬레이션 실험

그림 8(a)의 단독 시뮬레이션 환경은 그림 8(b)의 연동 시뮬레이션 환경의 동작을 검증하기 위하여 DiskSim에 단순하게 입력 trace 파일을 읽어 들여 전달하는 인터페이스만을 추가한 것이다. 표 1의 두 개의 실험 설정에 대해 그림 8(a)에 대해서 각각 단독 시뮬레이션을 행하며 실험1 및 실험2에서 각 1개씩 총 2개의 결과통계파일을 얻는다. 이 결과들은 4.3 연동 시뮬레이션 실험에서 얻는 결과의 대조군으로 사용된다.

4.3 연동 시뮬레이션 실험

연동 시뮬레이션은 그림 8(b)와 같이 구성하고, 실험의 편의를 위하여 DEVS-DiskSim 2:2 연동, 즉 두 개의

Generator들이 각각 실험1과 실험2의 설정으로 두 개의 DiskSim 프로세스들과 연동하는 실험을 실시하였다. DEVS Generator 모델 #1은 실험1의 설정에 따라 연동을 통해 실험 1의 trace 파일을 읽어 DiskSim #1에게 전송하고, DEVS Generator 모델 #2는 실험 2의 trace 파일을 읽어 DiskSim #2에게 전송한다. 이때, 두 trace 파일들은 request arrival time이 일부 중첩되기 때문에 연동 실험에서 시간 동기화와 데이터 교환의 정확성을 검증하는 데에 적합하다.

DiskSim #1은 실험 1의 설정에 따라 디스크모델로써 ssd-iozone.parv을, 통계 파라미터로 statdefs를 사용하고, DiskSim #2는 실험 2의 설정에 따라 디스크모델로써 ssd-postmark.parv, 통계 파라미터로 statdefs를 사용한다. 이 실험을 통해 DiskSim #1은 실험 1에 대한 결과통계파일을, DiskSim #2는 실험2에 대한 결과통계파일을 출력한다.

4.4 실험 결과 비교

결과분석은 표 1의 실험1에 대해서 4.2절의 단독 시뮬레이션 환경을 통해 얻은 결과통계파일과 4.3의 실험1 설정에 따른 DiskSim #1이 출력한 결과통계 파일을 비교하고, 또한 실험2에 대해서 4.2의 단독시뮬레이션 환경에 실험2의 설정으로 얻은 결과통계파일과 4.3에서 얻은 DiskSim#2가 생성한 결과통계파일을 비교하여 각각 일치성 여부를 확인하였다. 이론상 똑같은 입력 trace을 주었을 때, 두 환경 모두 동일한 출력을 낼 경우 제안 및 구현한 연동 구조의 정확성을 확인할 수 있다. 그림 9는 DiskSim이 출력하는 결과통계파일의 형식을 보여주는 것이다. 결과파일의 동일성 비교는 이러한 형식의 두 텍스트 파일을 비교해 주는 WinMerge라는 프로그램을 사용하였다. 그 결과 예상대로 각 실험별로 두 개의 통계결과파일들은 완벽히 동일함을 확인할 수 있었다.

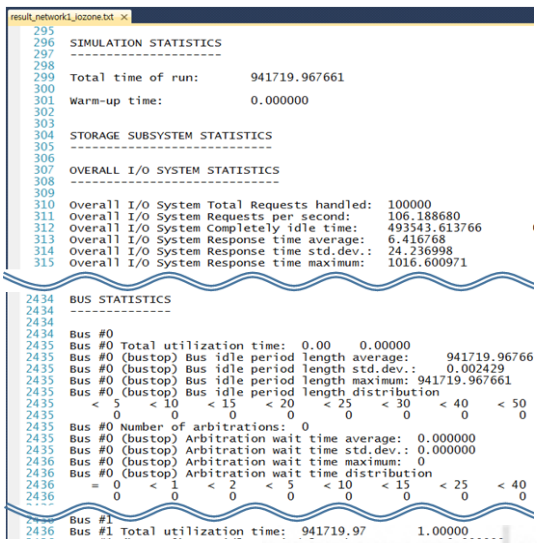


그림 9. DiskSim 의 출력통계결과 형식
Fig. 9. Result statistics file format of DiskSim

표2와 표3은 두 가지 구조에 대해 실험1과 실험2의 주요 통계 파라미터에 대한 결과를 발췌하여 비교 표기한 것이다. 주요 파라미터는 다음과 같다. total run time은 경과한 시뮬레이션 시간이다. total request는 블록 I/O 요청 패킷의 수이다. request per second는 평균 초당 I/O 요청 수이며, response time average는 한 요청 당 평균 처리 시간이다. 표2와 표 3과 같은 비교 결과에서 연동 구조를 사용한 실험결과들이 단독 구조의 실험결과와 정확히 일치하는 것을 알 수 있다. 결론적으로 이는 본 논문에서 제안하고 구현한 DEVS- DiskSim 연동 인터페이스를 통해 구현한 연동시뮬레이션 환경이 DiskSim과 정확히 동작함을 입증하였다.

표 5. 실험 1에 대한 단독 및 연동 결과 비교표
Table 2. Comparison of the results of experiment #1

결과 파라미터	연동 시뮬레이션 1	단독 시뮬레이션 1
Total rum time (ms)	94179.967661	94179.967661
Total Requests	100000	100000
Request per second (req/s)	106.188680	106.188680
Response time average (ms)	6.416768	6.416768
Response time std.dev (ms)	24.236998	24.236998

표 6. 실험 2에 대한 단독 및 연동 결과 비교표
Table 3. Comparison of the results of experiment #2

결과 파라미터	연동 시뮬레이션 2	단독 시뮬레이션 2
Total rum time (ms)	498398.728507	498398.728507
Total Requests	62257	62257
Request per second (req/s)	124.914043	124.914043
Response time average (ms)	4.140328	4.140328
Response time std.dev (ms)	10.240104	10.240104

V. 결론

본 논문은 여러 컴퓨터 노드들로 구성된 시스템수준 저장 시스템 시뮬레이션과 디스크 I/O수준 시뮬레이션 연동을 위

한 DEVSim++과 DiskSim 사이의 인터페이스를 제안하고 설계 구현한 것을 기술한 것이다. 이기종 시뮬레이터 간 연동을 위하여 시간 동기화와 데이터 교환을 담당하는 DiskSimManager의 개념을 도입하였으며, 두 시뮬레이션 엔진의 변경을 전혀 하지 않고 제공되는 외부 인터페이스 함수만을 이용하여 프로토콜을 설계 및 구현하였다. 또한, 시스템수준의 모델에서는 DEVS wrapper 모델 개념을 통해 모델링의 편의성을 제공하였다. 단독 시뮬레이션 및 설계 구현한 인터페이스를 통해 실험한 결과 동일한 입력과 설정에서 완벽히 동일한 결과를 얻어 설계와 구현이 정확함을 검증하였다. 향후 과제로서 현재 TCP/IP로 구현된 인터페이스의 속도를 향상시키기 위한 연구가 추가적으로 필요하다고 본다. 이 연구를 통해 획득된 인터페이스는 다중 노드로 구성된 여러 가지 분산저장시스템의 시뮬레이션 개발 시 유용하게 사용될 것으로 기대된다.

Acknowledgments

본 논문은 KAIST 김탁곤 교수의 SMS (Systems Modeling Simulation) 연구실에서 수행한 연구과제 (G01120261)에서 일부 지원을 받은 것입니다.

참고문헌

[1] John S. Bucy, Jiri Schindler, Steven W. Schlosser, Gregory R. Ganger, and Contributors, The DiskSim Simulation Environment Version 4.0 Reference Manual, Carnegie Mellon University, http://www.pdl.cmu.edu/DiskSim/, 2008.

[2] Tag Gon Kim, Chang Ho Sung, Su-Youn Hong, Jeong Hee Hong, Chang Beom Choi, Jeong Hoon Kim, Kyung Min Seo, and Jang Won Bae, "DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation," The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Vol. 8, No. 3, pp. 129 - 142, July, 2011.

[3] Bernard P. Zeigler, Herbert Praehofer and Tag Gon Kim, Theory of Modelling and Simulation (2nd Edition), Academic Press, 2000.

[4] V. Prabhakaran and T. Wobber, "SSD Extension for DiskSim Simulation Environment," http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4/, 2009.

[5] S. Hammoud, Maozhen Li, Yang Liu, N.K. Alham, Liu Zelong, "MRSim: A discrete event based MapReduce simulator," 2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), vol.6, pp.2993-2997, 10-12 Aug. 2010

[6] Apache JIRA, Mumak Hadoop MapReduce Simulator, https://issues.apache.org/jira/browse/MAPREDUCE-728, last retrieved Mar. 2013.

[7] Hadoop. http://hadoop.apache.org, last retrieved Mar. 2013.

[8] Jaeho Kim, Jongmin Lee, Jongmu Choi, Donghee Lee, Sam H. Noh, "An Efficient RAID Scheme for Reliable SSDs," Journal of KIISE : Computing Practices and Letters, vol.18, no.5, pp.359-368, 2012.

[9] Hae Sang Song, Jeong Man Seo, "BlockSim++: A Lightweight Block-oriented Hierarchical Modeling and Simulation Framework for Continuous Systems," Journal of the Korea society of computer and information, vol. 17, no. 12, pp. 11-22, 2012.

[10] Jeong Won Kim, "I/O Scheme of Hybrid Hard Disk Drive for Low Power Consumption and Effective Response Time," Journal of the Korea society of computer and information, vol. 16, no. 10, pp. 23-31, 2011.

저 자 소 개

송 해 상

2000년 한국과학기술원 전기및전자공학과 공학박사

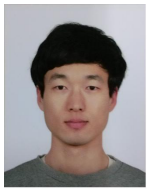
1999년~2000년 고등기술연구원

2001년~2002년 (주)스페이스네트

2002년~현재 서원대학교 컴퓨터공학과 교수

관심분야 : 시스템 모델링 시뮬레이션 이론 및 응용, 국방시스템 공학, 소프트웨어공학

Email: hssong@seowon.ac.kr



이 순 주

2012년 부산대학교 전자전기공학부 학사

2013년 현재 한국과학기술원 전기 및 전자공학과 석사과정 SMS 연구실

관심분야 : 시스템 모델링 시뮬레이션, 분산스토리지, 분산컴퓨팅

Email: sjlee@smslab.kaist.ac.kr