https://doi.org/10.7840/kics.2017.42.2.460

효율적인 빅 데이터 마이닝을 위한 iSSD 기반 협업 처리 방안

조 용 연◆, 김 상 욱°, 배 덕 호*

iSSD-Based Collaborative Processing for Big Data Mining

Yong-Yoen Jo◆, Sang-Wook Kim°, Duck-Ho Bae*

요 약

본 논문은 intelligent SSD (iSSD)를 통해 빅 데이터 마이닝을 효과적으로 처리하기 위한 방안에 대해서 소개한다. iSSD는 데이터 전송 비용을 줄이고 데이터가 저장된 장소와 가장 가까운 곳에서 데이터를 처리하기 위해, SSD 내부에 데이터 처리 능력을 부여한 장치이다. 본 논문에서는 먼저, iSSD의 등장 배경 및 효율적인 데이터 처리를 위한 iSSD 구조에 대해 소개한다. 더 나아가, iSSD를 이용하여 데이터 마이닝 알고리즘들을 빠르게 수행하는 방안을 소개한다. 끝으로, iSSD 뿐만 아니라 호스트 CPU, GPU 등 이질 (heterogeneous) 컴퓨팅 자원을 함께 활용하여 데이터 마이닝 알고리즘의 성능을 크게 향상시키는 협업 방안을 소개한다.

Key Words : Intelligent SSD, Simulator-based evaluation, Collaborative processing, Heterogeneous scheduling

ABSTRACT

We address how to handle big data mining effectively using the intelligent SSD (iSSD). ISSD is a storage device equipped with computing power inside SSD for reducing the transferring cost and for processing data nearby SSD where the data is stored. We first introduce the structural characteristics of iSSD for efficient data processing. Then, we present how to process data mining algorithms by using iSSD. Finally, we discuss how to improve the performance of data mining algorithms significantly by exploiting heterogeneous computing environment where host CPUs and GPU coexist for maximizing the performance.

Ⅰ. 서 론

빅 데이터 시대는 이미 도래되었다. 비정형 데이터 처리 시스템, 분산 시스템 등 규모가 큰 데이터를 다루는 시스템들이 점점 증가하고 있다[1-3]. 연구자들은 이미 빅 데이터를 효율적으로 처리하는 방법이 데이터 처리를 위한 데이터 전송량을 줄이는 것임을 인지하고 있었으며, 이를 위해 데이터가 저장된 근처에서 데이터를 처리 (near-data processing, NDP)하고자 시도하였다. 이러한 NDP 시스템의 대표적인 예로는 MapReduce[2] (혹은 Hadoop[3]), 오라클 Exadata[4], IBM Netezza[5] 등이 있다. 특히, 높은 I/O 대역폭, 짧은 access latency 등의 특성으로 인해 빅 데이터 저장 장치로 각광받는 solid-state drive (SSD) 대중화에 더

※ 본 연구는 한양대-삼성전자 반도체 산학협력 연구과제.의 지원을 받아 수행된 연구입니다.
※ 본 연구는 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구입니다 (NRF-2014R1A2A1A10054151).
◆ First Author : Hanyang University, Department of Computer and Software, jyy0430@hanyang.ac.kr, 정회원
° Corresponding Author : Hanyang University, Department of Computer and Software, wook@hanyang.ac.kr, 종신회원
* Samsung Electronics, Memory Business, duckho.bae@samsung.com, 정회원
논문번호 : KICS2017-01-006, Received January 9, 2017; Revised February 7, 2017; Accepted February 16, 2017

불어 SSD 내부에 데이터 처리 능력을 부여함으로써, 데이터가 저장된 가장 가까운 곳에서 데이터를 처리(in storage processing, ISP) 하고자 하는 intelligent SSD (iSSD)는 이러한 노력의 일환으로 볼 수 있다[1,6-8]. iSSD 내부에서 간단한 데이터 처리를 수행함으로써, 호스트 CPU로 전송되는 데이터의 양을 크게 줄일 수 있으며, 호스트 CPU의 부하를 줄이고 데이터 전송을 위한 에너지 사용량을 줄일 수 있다.

본 논문에서는 이러한 iSSD를 활용하여 데이터베이스에 초점을 맞추어 진행되었던 기존 연구들과 달리 데이터 마이닝에 초점을 맞추어 진행된 연구를 소개한다. 또한 iSSD 뿐만 아니라 다른 컴퓨팅 리소스를 함께 활용하는 방안에 대해서 논의하고자 한다.

본 논문에서는 다루는 내용은 다음과 같다. 먼저, 2장에서는 하드 디스크를 이용한 ISP의 등장 배경과 대중화되지 못한 이유에 대해 소개한다. 더 나아가, SSD에서 ISP를 재조명하는 이유에 대해 설명하며, 효율적인 내부 데이터 처리를 위한 iSSD의 구조에 대해 살펴본다. 3장에서는 iSSD의 킬러 응용으로 빅 데이터 마이닝 알고리즘들을 iSSD 내에서 효율적으로 수행 가능한 방안 및 이를 모델링한 수행비용 모델을 소개한다. 더 나아가, 수행비용 모델의 정확도 검증 방법에 대해 설명하며, 수행비용 모델을 기반으로 iSSD 내부 데이터 마이닝 알고리즘들의 수행 성능 향상 결과를 살펴본다. 4장에서는 iSSD 내부 데이터 처리의 제약 사항을 언급하고, 이를 극복하기 위해 호스트 CPU와 graphic processor unit (GPU) 등의 기존 컴퓨팅 자원과 협업할 때의 장점에 대해 살펴보며, iSSD를 고려한 이질 환경 (heterogeneous) 스케줄링 알고리즘에서 고려해야 할 사항들을 소개한다. 끝으로 협업을 통한 빅 데이터 마이닝 성능의 향상 결과를 살펴본다. 마지막으로 5장에서는 본 논문을 정리하고, 결론을 맺는다.

Ⅱ. Intelligent SSD

2.1 등장 배경

저장 장치 내에 데이터 처리 능력을 부여하여 데이터를 효율적으로 처리하고자 하는 시도들은 1990년 말부터 본격적으로 이어져왔다. 1990년 말, 프로세서와 메모리의 가격이 낮아짐에 따라 하드 디스크 내에 디스크 제어를 위한 범용 CPU와 메모리 등이 장착되었다. 이를 계기로 디스크 내부의 범용 CPU와 메모리를 이용하여 디스크 내에서 scan, selection, sorting, join 등 데이터베이스 연산을 수행하고자 하는 active disk가 등장하였다[9]. 그러나 active disk는 하드 디스크의 낮은 내부 I/O 대역폭, 대용량의 데이터 의존적인 킬러 어플리케이션 부재, 그리고 저장 장치 인터페이스의 제약으로 인해 널리 사용되지 못하였다[8]. 특히, 자기 디스크를 회전시키며 데이터를 읽는 하드 디스크의 물리적 한계로 인해, 하드 디스크 내부에서 데이터를 처리할 때, 자기 디스크에서 데이터를 읽어오는데 데이터 처리 시간의 대부분을 소요하게 되었다.

최근, 빠른 랜덤 성능과 높은 I/O 대역폭을 바탕으로 SSD는 빅 데이터 시대의 저장 장치로 자리 잡고 있다. 그림 1은 SSD 구조를 나타낸다. SSD의 내부에는 용량에 따라 차이는 보이지만 일반적으로 8 ~ 32개의 채널이 존재한다. SSD의 각 채널은 (1) 데이터 저장을 위한 8 ~ 16개의 NAND 플래시 메모리 셀, (2) 해당 채널의 플래시 메모리 셀들을 관리하기 위한 플래시 메모리 컨트롤러 (flash memory controller, FMC), 그리고 (3) 플래시 메모리 셀의 데이터 읽기/쓰기를 위한 한 페이지 크기의 DRAM (FMC 메모리)으로 구성된다[1]. SSD 내의 모든 채널들은 flash translation layer (FTL) 펌웨어에 의해 관리/제어된다[1]. 따라서 SSD 내에는 FTL 펌웨어를 수행하기 위한 embedded ARM CPU (SSD 코어)와 FTL 메타 데이터 저장을 위한 SRAM과 DRAM (SSD 메모리)이 존재한다. 이렇듯, SSD는 하나의 채널에 여러 개의 플래시 메모리 셀이 존재하는 멀티웨이 (multi-way), 하나의 SSD 내에는 여러 개의 채널이 존재하는 멀티채널 (multi-channel) 구조를 이루고 있다[10].

SSD의 내부 성능 발전은 호스트 인터페이스에서의 새로운 성능 병목점을 야기시켰다. 자기 디스크를 물리적으로 회전시키며 데이터를 읽는 하드 디스크를 저장 장치로 사용할 경우, 하드 디스크의 내부 대역폭이 데이터 접근 성능의 주된 병목점이었다. 그러나 하드 디스크와는 달리 전기적 신호로 데이터를 읽는 SSD의 내부 대역폭은 매우 높으며, 멀티웨이 및 멀티채널의 내부 병렬 구조로 인해 비교적 손쉽게 확장 가

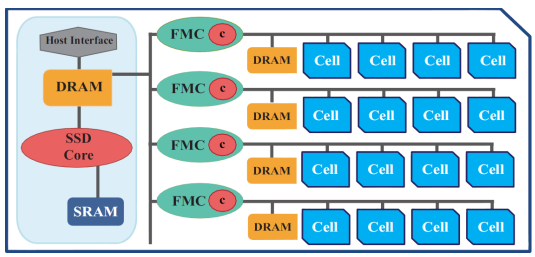


그림 1. iSSD 구조
Fig. 1. Architecture of iSSD

능하다. 그럼에도 불구하고 현재 SSD 내부 대역폭은 호스트 인터페이스의 대역폭 수준에 맞춰진 실정이다[7]. 이러한 호스트 인터페이스의 병목 현상을 극복하고, SSD의 빠른 내부 대역폭을 적극 활용하기 위해, SSD에 active disk 개념을 도입하고자 하는 연구들이 최근 진행되고 있다[1,6,11].

2.2 iSSD의 구조 및 특징

현재 제품화된 iSSD는 존재하지 않는다. 또한, iSSD를 위해서는 SSD 내부 FTL 펌웨어를 수정해야 하므로 제조사를 제외하고는 실제 제품수준의 iSSD 개발은 불가능하다. 그럼에도 불구하고 연구자들은 iSSD를 이용한 in storage processing (ISP)는 빅 데이터 시대의 효율적인 데이터 처리 방안으로 믿고 있으며, iSSD 내부 데이터 처리 능력 부여 방안, 처리 프로그래밍 모델, 킬러 응용 등 다양한 분야에서 활발히 연구를 진행하고 있다[1,6,7,11]. 본 절에서는 효율적인 내부 데이터 처리를 위한 iSSD의 구조에 대해 살펴본다. 한 가지 언급할 점은 iSSD 관련 논문들은 저마다의 iSSD 구조를 가정한다. 그러나 내부에서 데이터를 효율적으로 처리하기 위해서는 현재 SSD에 내장된 컴퓨팅 자원 이외의 추가적인 처리 성능이 더 필요하다는 것이 공통적인 견해이다.

iSSD의 구조는 다음과 같다. 첫째, iSSD는 현재의 SSD와 비교하여 더욱 높은 내부 총 대역폭을 가져야 한다. 호스트 인터페이스의 대역폭 수준으로 제한되었던 SSD의 내부 총 대역폭을 확장함으로써, 더 많은 데이터를 빠르게 접근하고 처리할 수 있게 된다. 이러한 내부 대역폭 확장은 iSSD 내의 채널 및 웨이를 추가함으로써 확장가능하다[9]. 둘째, iSSD의 확장된 높은 내부 대역폭을 충분히 활용하기 위해서는 일정 수준 이상의 내부 처리 능력을 갖추어야 한다. 그렇지 않다면, 오히려 iSSD 내부 데이터 처리에서 병목 현상이 발생하게 된다. 우리는 iSSD 내부에 데이터 처리 능력을 부여함에 있어 상대적으로 성능이 낮은 범용 CPU (FMC CPU)와 작은 크기의 메모리를 각 채널의 FMC에 장착하는 방안을 고려하고자 한다. 이러한 iSSD 구조는 성능이 우수한 하나의 (혹은 몇 개의) SSD CPU를 장착하는 방안과 비교하여 다음과 같은 장점을 갖는다[1]. 첫째, 채널별 데이터 병렬 처리를 가능하게 한다. 각 채널에서 처리할 데이터는 채널수에 비례하여 줄어들게 되어, 낮은 성능의 CPU로도 충분히 내부 대역폭을 활용할 수 있게 된다. 둘째, 채널을 추가함으로써 내부 대역폭의 확장과 내부 처리 능력의 향상을 동시에 이룰 수 있다. 셋째, 성능이 낮은 여러 개의 CPU를 이용함으로써, SSD 내의 thermal 이슈 해결에도 도움이 된다[12].

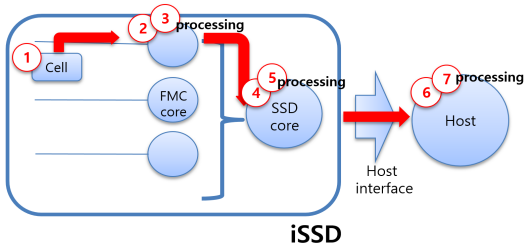
1) 각 채널별 컴퓨팅 자원을 추가하는 iSSD 구조는 기존 여러 논문들에서도 채택하고 있다[1,6,11].

Ⅲ. SSD 에서의 데이터 마이닝 알고리즘 수행

3.1 수행전략

본 절에서는 iSSD 내에서 데이터 마이닝 알고리즘들을 효율적으로 수행하기 위한 다음과 같은 세 가지 전략을 소개한다[1]. 첫째, iSSD 내의 다수 채널을 활용하여 데이터를 병렬적으로 처리한다. iSSD 내의 모든 FMC 코어들은 각자의 플래시 메모리 셀에 저장된 데이터에 대해 동일한 작업을 병렬적으로 수행한다. 이 때, SSD 코어는 모든 FMC 코어를 관리하고, 각 FMC 코어가 수행할 작업들을 배분하는 역할을 수행한다. 둘째, 플래시 메모리 셀을 가상 메모리로 사용한다. 몇몇 데이터 마이닝 알고리즘들은 수행 과정에서 중간 결과를 생성한다. 그러나 대용량의 데이터를 다루는 데이터 마이닝 알고리즘의 특성 상 수행에 필요한 중간 결과를 FMC 메모리에 모두 적재하지 못하고 overflow 될 수 있다. 이 경우, 각 FMC 코어는 자신들의 채널의 플래시 메모리 셀을 가상 메모리처럼 사용하여 데이터 마이닝 알고리즘들을 수행하는 전략을 사용한다. 셋째, SSD 코어를 이용한 전체 병합한다. 몇몇 데이터 마이닝 알고리즘들은 수행 과정에서 각 FMC 코어에서 처리한 local 결과에 대한 전체 병합을 필요로 한다. 이 때, 모든 FMC 코어들과 직접 통신이 가능하며, FMC 코어보다 성능이 더 우수한 SSD 코어를 이용하여 각 FMC 코어에서 처리한 local 결과를 전체 병합한다.

그림 2은 (1) 호스트 CPU에서 수행하는 기존 방안 (in host processing, IHP)과 (2) 위의 수행전략들을



- IHP: 1 → 2 → 4 → 6 → 7
- ISP: 1 → (2 → 3) ↔ (4 → 5) (global merge)

그림 2. 데이터 마이닝 알고리즘의 수행 과정
Fig. 2. Execution steps of data mining algorithms through iSSD.

바탕으로 iSSD에서 ISP의 두 경우에 대한 데이터 마이닝 알고리즘 수행 과정을 각각 나타낸다. iSSD를 통해 SSD 내부적으로 데이터를 처리하게 되면, (1) 더 이상 모든 데이터를 호스트 CPU로 전송하는 대신 상대적으로 크기가 매우 작은 처리 결과만을 전송하고[9]. (2) iSSD에서의 선처리를 통해 호스트 CPU의 데이터 처리 부하를 크게 줄일 수 있으며, (3) 데이터가 저장된 장소에서 데이터를 처리함에 따라 에너지 사용량을 줄일 수 있게 된다[8].

3.2 수행비용 모델

본 절에서는 데이터 마이닝 알고리즘들의 수행비용에 대해 분석하고 이에 대한 비용모델을 소개한다. 일반적인 응용 프로그램과 동일하게 데이터 마이닝 알고리즘들의 수행비용은 크게 (1) 수행에 필요한 데이터에 접근하는 데이터 접근 비용, (2) 접근한 데이터에 대한 데이터 처리 비용으로 구분 가능하다. 우리는 이러한 두 가지 비용 분석을 통해 IHP와 비교하여 ISP의 적용 가능성, 그리고 ISP에서의 새로운 병목점을 알 수 있다. 이 때, 데이터 마이닝 알고리즘에 대한 데이터 접근 비용 및 데이터 처리 비용을 함수 단위로 수립하여야 한다. 이는 하나의 데이터 마이닝 알고리즘에 속한 함수들이라 할지라도 각 함수의 특징에 따라 데이터 접근 및 처리 비용에 큰 차이를 보이기 때문이다[2]. 표 1은 수행비용 모델 수립에 사용한 기호들을 나타낸다.

표 1. 수행비용 모델을 위한 기호
Table 1. Notation

Notation	Description
N_{ch}	Number of channels (FMC cores)
N_{way}	Number of ways in a channel
S_{page}	Size of a single NAND flash memory page
M_{host}	Size of host memory
M_{core}	Size of SSD memory
M_{FMC}	Size of each FMC memory
$t_{cell-read}$	Time to read a page from flash memory cell
$t_{bus-load}$	Time to load a page from a flash memory bus after busy phase for a page read
$t_{core-read}$	Time to read d bytes from SSD memory
$t_{core-write}$	Time to write d bytes to SSD memory
$t_{host-write}$	Time to write d bytes to host memory through the host interface
$\|F\|$	Number of functions in the target data mining application
$t_{IHP(f_k)}$	Time for performing function f_k in the host CPU
$t_{ISP(f_k)}$	Time for performing function f_k in the FMC core
$N(f_k)$	Size of input data for function f_k
$M(f_k)$	Size of memory required for storing an immediate result for function f_k
$t_{merge}(f_k)$	Time for performing a global merge for function f_k in the SSD core
$M_{merge}(f_k)$	Size of memory required for performing global memory for function f_k
$R(f_k)$	Size of a local result from the FMC core for function f_k
N	Size of total data

3.2.1 데이터 접근 비용

본 항에서는 IHP와 ISP에서의 각 구간 별 데이터 전송 비용모델을 설명한다. d bytes 크기의 데이터에 대한 구간 별 전송 비용은 다음과 같다[8].

- **플래시 메모리 셀 → FMC 메모리**: 하나의 플래시 메모리 셀에서 FMC 메모리까지 d bytes의 데이터를 읽는데 드는 비용은 수식 1과 같다.

$$t_{cell \to FMC} = \left\lceil \frac{d}{S_{page} \times N_{way}} \right\rceil \times t_{cell-read} + \left\lceil \frac{d}{S_{page}} \right\rceil \times t_{bus-load} \quad (1)$$

- **FMC 메모리 → SSD 메모리**: FMC 메모리와 SSD 메모리 간의 데이터 전송은 전송의 효율을 위해 direct memory access (DMA) 기술과 파이프라인 전략을 사용한다[8]. FMC 메모리에서 SSD 메모리로 d bytes의 데이터를 읽는데 드는 비용은 수식 2와 같다.

$$t_{FMC \to core} = \max(t_{cell \to FMC}, t_{core-write}) \quad (2)$$

- **SSD 메모리 → 호스트 메모리**: FMC 메모리와 SSD 메모리 간의 데이터 전송과 동일하게 SSD 메모리와 호스트 메모리 간의 데이터 전송 역시 DMA 기술과 파이프라인 전략을 사용한다[8]. SSD 메모리에서 호스트 메모리로 d bytes의 데이터를

2) 각 함수는 프로그래머에 의해 적절한 granule로 잘 배분되어 있다고 가정한다.

읽는데 드는 비용은 수식 3과 같다.

$$t_{core \rightarrow host} = \max(t_{host-write}, t_{core-read}) \quad (3)$$

각 구간 별 전송 비용으로부터 도출한 IHP와 ISP에서의 d bytes의 데이터에 대한 접근 비용은 각각 수식 4 그리고 5와 같다. 그림 2에서 알 수 있듯이, ISP는 IHP에 비해 데이터 전송 경로가 짧다. 또한, ISP에서 데이터는 새로운 병목점인 호스트 인터페이스를 거치지 않는다. 따라서 ISP는 IHP와 비교하여 매우 작은 데이터 전송 비용을 갖는다.

- $data\_access\_\mathrm{cost}(IHP)$
$$= \sum_{k=1}^{|F|} (t_{FMC \rightarrow core}/N_{ch} + t_{core \rightarrow host})/d \times N(f_k) \quad (4)$$

- $data\_access\_\mathrm{cost}(ISP)$
$$= \sum_{k=1}^{|F|} (t_{cell \rightarrow FMC}/d/N_{ch}) \quad (5)$$

3.2.2 데이터 처리 비용

본 항에서는 IHP와 ISP 각각의 데이터 처리 비용 모델을 소개한다. 이 때, 각 처리비용은 수식 6 그리고 7과 같다. 데이터 처리 비용에는 각 코어의 제한된 메모리 크기로 인해 데이터 스와핑에 대한 데이터 접근 비용이 발생한다.

- $data\_processing\_\mathrm{cost}(IHP)$
$$= \sum_{k=1}^{|F|} (t_{IHP}(f_k) \times N(f_k)/d + \lceil M(f_k)/M_{host} \rceil \times data\_access\_\mathrm{cost}(IHP)) \quad (6)$$

- $data\_processing\_\mathrm{cost}(ISP)$
$$= \sum_{k=1}^{|F|} (t_{ISP}(f_k) \times \frac{1}{N_{ch}} \times N(f_k)/d + \lceil M(f_k)/M_{FMC} \rceil \times data\_access\_\mathrm{cost}(ISP)) \quad (7)$$

데이터 처리 비용은 데이터 마이닝 알고리즘의 각 함수를 수행하는 코어의 성능에 직접적인 영향을 받는다. 따라서 iSSD 내에서 여러 개의 FMC 코어들을 이용하여 함수를 병렬 수행할 수 있음에도 불구하고, ISP에서의 데이터 처리 비용은 IHP와 비교하여 매우 크다.

ISP에서 떳떳 데이터 마이닝 알고리즘들은 수행 과정에서 각 FMC 코어에서 처리한 local 결과에 대한 전체 병합을 필요로 한다. 이러한 전체 병합은 여러 FMC 코어에서 병렬적으로 처리할 수 없기 때문에 ISP의 성능에 악영향을 미칠 수 있다. ISP에서의 전체 병합 비용은 수식 8과 같다.

- $merge\_\mathrm{cost}(ISP)$
$$= \sum_{k=1}^{|F|} \left\{ \begin{array}{l} (t_{merge}(f_k) + (t_{channel \rightarrow core} \times N_{ch}/d)) \times R(f_k) \\ + \lceil M_{merge}(f_k)/M_{core} \rceil \times t_{FMC \rightarrow core} \end{array} \right\} \quad (8)$$

전체 병합 비용은 (1) 각 FMC 코어에서 처리된 local 결과의 크기, (2) iSSD의 FMC 코어의 수, 그리고 (3) 병합 작업의 복잡도에 영향을 받는다. 즉, 각 FMC 코어에서 처리된 결과의 크기가 클수록, iSSD의 FMC 코어의 수가 많아질수록, 그리고 병합 작업이 복잡할수록 전체 병합 비용이 커진다.

3.2.3 전체 수행 비용

앞에서 설명한 데이터 접근 비용과 데이터 처리 비용을 모두 고려한 데이터 마이닝 알고리즘의 IHP와 ISP의 수행비용은 각각 수식 9 그리고 10과 같다. 우리는 아래의 두 식을 통해 데이터 마이닝 알고리즘을 iSSD에서 수행할 때의 성능 향상 정도를 계산할 수 있다.

- $total\_\mathrm{cost}(IHP)$
$$= data\_access\_\mathrm{cost}(IHP) + data\_processing\_\mathrm{cost}(IHP) \quad (9)$$

- $total\_\mathrm{cost}(ISP)$
$$= data\_access\_\mathrm{cost}(ISP) + data\_processing\_\mathrm{cost}(ISP) + merge\_\mathrm{cost}(ISP) \quad (10)$$

3.3 성능 평가

3.3.1 실험 환경

본 절에서는 위에서 수립한 비용 모델을 기반으로 iSSD에서의 데이터 마이닝 알고리즘들의 성능 평가 결과를 소개한다. 이 때, 사용한 매개변수는 표 2와 같

표 2. 매개 변수
Table 2. Parameters

	Parameter	Value(s)
iSSD	#FMC cores	32-256
	#way	8
	SSD core rates (MHz)	400
	FMC core rates (MHz)	200
	SSD memory Size (MB)	400
	FMC memory Size (KB)	8
	CELL read time (us)	50
	CELL write time (us)	1,200
	Page size (B)	8,192
Interface	Bandwidth (Gbps)	3
Host	Host CPU rates (GHz)	2.5
	HOST memory size (GB)	4

다. 표 2에서 굵은 글씨체의 변수 값은 해당 매개 변수를 대표하는 값이다. 특정한 하나의 매개 변수 값을 변화시켜 가는 동안 다른 매개 변수 값들은 표에서 굵게 표시된 값으로 고정된다. 실험에 사용된 데이터는 합성 데이터를 사용하였다 [1]. 본 실험에서 사용할 k-means[13], PageRank[14], Apriori[15] 총 세 가지 대표적인 데이터 마이닝 알고리즘들을 사용한다. 수식 9와 10의 데이터 접근 비용을 계산하기 위해서 Samsung SSD datasheet를 사용하였다[3]. 수행비용 모델을 통해 성능을 측정하기 위해서 먼저 각 알고리즘의 함수들을 도출하여야 한다. 각 함수를 수행하는데 발생하는 cycles per instruction (CPI)와 instructions의 수 (# of INS)는 V-tune[4]을 사용하여 추출 가능하다. 이를 통해 식 ((CPI * #instructions) / CPU rates)을 이용하여 수행 시간을 계산할 수 있다.

3.3.2 비용 모델 정확도 검증

수립한 비용 모델의 정확도는 실험 결과의 신뢰도에 큰 영향을 미친다. 우리는 우선 수립한 비용 모델의 정확도를 검증하기 위해 Gem 5 시뮬레이터[16]을 사용하였다. Gem 5 시뮬레이터는 가상의 컴퓨팅 환경을 구축하여 그 위에서 cycle-level의 정확도로 프로그램을 수행할 수 있으며, 가상의 처리 장치들을 검증할 때 주로 사용된다[16]. Gem 5 시뮬레이터를 기반으로 iSSD를 구현하였다. 구체적인 매개 변수 설정 방안은 본 논문의 지면 제약으로 생략한다[5]. 한 가지 언급할 점은 Gem 5 시뮬레이터는 설정된 환경의 성능을 측정하는데 매우 많은 시간을 요구한다. 그 예로 0.1초를 요구하는 함수에 대해 Gem 5 시뮬레이터는 약 12시간의 소요시간을 요구한다. 따라서 Gem 5 시뮬레이터를 이용하여 빅 데이터에 대한 iSSD의 성능을 직접 측정하는 것은 불가능하다. 따라서 우리는 비용 모델의 정확도를 작은 데이터 셋으로 검증하고, 큰 데이터 셋에 대한 성능 평가는 비용 모델을 이용한다.

표 3은 Gem 5 시뮬레이터로 측정한 수행 시간과 수행비용 모델에 의해 계산된 수행 시간 간의 pearson correlation coefficient (PCC)를 나타낸다. 실험 결과, PCC는 약 0.999 이상으로 우리의 Gem 5 시뮬레이터와 수행비용 모델이 매우 유사하다 것을 알 수 있다. 이를 통해, 우리의 iSSD의 수행비용 모델은 정확도를 신뢰할 수 있다.

표 3. Gem 5 시뮬레이터와 수행비용 모델의 PCC
Table 3. PCC between Gem 5 simulator and the cost model

Algorithms	PCC
Apriori	0.9989
k-means	0.9997
PageRank	0.9998

3.3.3 실험 결과

실험 1. iSSD의 매개 변수 변화에 따른 성능 평가: 첫 번째 실험은 iSSD의 FMC 코어수를 32, 64, 128, 그리고 256으로 늘려가면서 ISP의 성능 변화를 측정한 결과이다. 이 때, 다른 매개 변수들은 표 2와 같이 설정하였다. 그림 3는 실험 결과를 나타낸다. 실험 결과, 채널수가 증가함에 따라 ISP의 성능이 큰 폭으로 상승함을 알 수 있다. k-means와 PageRank의 경우, 채널수가 증가함에 따라 성능이 향상 되었는데 이는 iSSD에서의 채널 확장의 효용성을 잘 보여주는 결과이다. 반면, 전체 병합을 수행하는 함수가 병목이 되어 Apriori에서는 FMC의 코어의 수가 증가함에 따라 성능의 향상 폭이 상대적으로 적었다. 이외에 다른 매개변수에 대한 실험을 수행하였고 이들의 수행시간 변화의 경향도 위와 유사한 경향을 보였다. 그러나 분량의 제약 상으로 생략한다.

실험 2. IHP와 ISP의 성능 비교: 본 실험은 iSSD의 잠재력을 정량적으로 보이기 위해, k-means, PageRank, 그리고 Apriori의 IHP와 ISP의 성능을 비

3) http://www.samsung.com/semiconductor/products/flash-storage/
4) https://software.intel.com/en-us/intel-vtune-amplifier-xe/
5) iSSD Gem 5 시뮬레이터 관련해서는 다음의 논문에서 확인 가능하다[17].

교한 결과이다. IHP를 위해, (1) 하드 디스크를 저장 장치로 사용한 IHP(HDD)와 (2) SSD를 저장 장치로 사용한 IHP(SSD)에 대해 성능을 측정하였다. ISP를 위해, (1) 현재 typical한 SSD를 기반으로 한 iSSD (iSSD_C: 32개 채널, 200MHz 채널 CPU)와 (2) 내부 대역폭과 처리 능력이 향상된 미래형 iSSD (iSSD_F: 256개 채널, 800MHz 채널 CPU)에 대해 실험을 수행하였다. 그림 4은 실험 결과를 나타낸다. 실험 결과, k-means와 PageRank에서는 모든 경우에서 ISP의 성능이 IHP에 비해 우수하였다. 이를 통해, iSSD가 빅 데이터 마이닝 알고리즘들을 효율적으로 수행할 수 있음을 알 수 있다. 전체 병합이 발생의 비용이 큰 Apriori의 경우, ISP(iSSD_C)는 IHP(SSD)와 비교하여 오히려 성능이 좋지 않았으며, ISP(iSSD_F)에서도 성능 향상은 크지 않았다. 이것으로 Apriori의 경우, 전체 병합을 처리하는 SSD 코어가 새로운 병목점이 된다는 것을 알 수 있다.

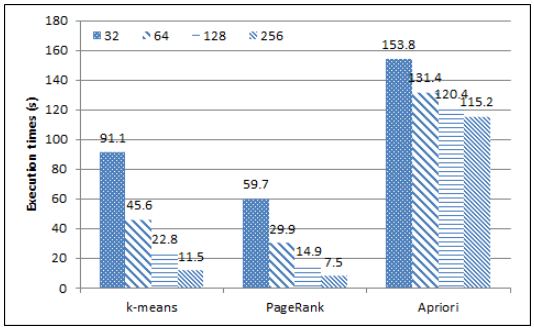


그림 3. iSSD내 FMC 코어의 수 변화에 따른 ISP 성능.
Fig. 3. Performance of ISP with different numbers of the FMC cores in iSSD

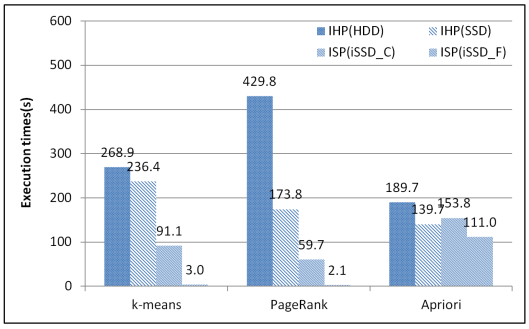


그림 4. IHP와 ISP의 성능 비교
Fig. 4. Performance comparison of the ISP with the IHP.

Ⅳ. 협업을 통한 데이터 마이닝 알고리즘 수행

이전 장에서는 iSSD만을 이용하여 데이터 마이닝 알고리즘을 수행하는 방안에 대해 소개하였다. 그러나 iSSD 내부 개개별 컴퓨팅 자원들은 호스트 CPU와 비교하면 그 성능이 매우 낮다. 수행 과정에서 전체 병합이 필요한 데이터 마이닝 알고리즘 특성 상, 해당 과정은 성능이 높은 호스트 CPU에서 처리하면 성능을 향상시킬 수 있을 것이다. 더 나아가 호스트 CPU 뿐만 아니라 최근 대용량의 데이터를 병렬적으로 처리하기 위해 널리 사용되는 graphic processing units (GPU)를 활용할 경우, 성능 향상을 더 기대할 수 있다. 연구자들은 이미 호스트 CPU 뿐만 아니라 GPUs, digital signal processors (DSPs), field-programmable gate arrays (FPGAs) 등 이질 (heterogeneous) 컴퓨팅 자원들을 함께 이용하여 데이터를 효율적으로 처리하는 방안에 대해 많이 연구하였다. 대표적인 예가 OpenCL이다. 그러나 iSSD는 아직 해당 컴퓨팅 환경에 포함되지 않았다. 본 장에서는 데이터 마이닝 알고리즘의 성능을 극대화하기 위해 iSSD와 호스트 CPU, GPU 등 이질 컴퓨팅 자원과의 협업 처리 방안을 소개한다[6,11].

4.1 배경지식: GPU 구조

그림 5는 GPU의 구조를 나타낸다. GPU는 고유의 프로세서 구조와 다양한 계층의 메모리 구조를 갖는다[18]. GPU의 프로세서는 스트리밍 멀티프로세서 (streaming multiprocessor: SM)의 집합으로 이루어져 있으며, 각 SM은 스트리밍 프로세서 (streaming processor: SP)의 집합으로 이루어져 있다. 또한, 크기와 접근 속도에 따라 글로벌 메모리, 공유 메모리, 레지스터 등의 메모리가 존재한다[18]. GPU는 single instruction multiple thread (SIMT) architecture를 사

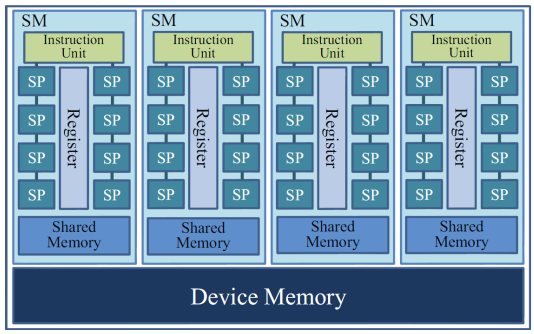


그림 5. GPU 구조
Fig. 5. Architecture of GPU

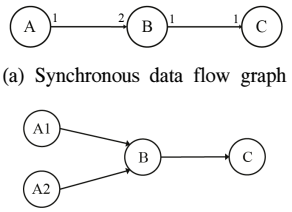
용하며, SM내의 모든 SP들은 동시에 동일한 명령어를 처리한다. 이러한 구조로 인해 GPU는 간단한 연산들을 반복적으로 수행하는 알고리즘을 처리 하는데 매우 효과적이다[19].

4.2 BIL 스케줄링

협업 처리 환경은 다양한 프로세서들이 존재하며, 이들은 데이터를 처리할 때 필요한 각자의 메모리 공간을 가지고 있다. 또한, 프로세서들 간의 통신(inter-process communication, IPC)이 발생한다. 이러한 협업 처리 환경은 이질 환경으로 분류된다. 이 때, 협업 성능을 높이기 위해서 각 프로세서가 어떠한 함수를 어떠한 시점에 처리할지 결정하기 위한 스케줄이 필요하다. 우리는 이질 환경에 적합한 스케줄을 획득하기 위해 대표적인 이질 스케줄링 알고리즘인 best imaginary level (BIL) 스케줄링[20]을 도입하였다. BIL 스케줄링은 주어진 프로세서들에서 가장 오랜 수행시간을 갖는 함수 (level이 높은 함수)를 우선 선택하고 이를 가장 빠르게 처리할 수 있는 프로세서에 스케줄링하며, 이 과정을 모든 함수에 대해서 반복적으로 수행한다[20].

BIL 스케줄링의 입력 값으로 (1) 알고리즘의 함수들의 관계를 모델링한 그래프 (2) 알고리즘의 각 함수 수행시간 및 (3) IPC 시간 정보가 필요하다. 그래프로 알고리즘을 표현할 때, 각 알고리즘의 함수를 노드로, 함수의 호출 관계는 노드들 간의 간선으로 표현된다. 각 함수의 수행시간은 그래프 내의 노드들의 수행시간을 나타내며, IPC 시간은 주어진 처리 환경의 프로세서간의 통신시간을 의미한다. 이때, 동일한 알고리즘도 그래프의 구성에 따라 스케줄이 달라질 수 있다. 서로 다른 스케줄은 알고리즘의 수행시간에 영향을 미친다. 즉, 협업 처리 환경에 적합하게 알고리즘을 그래프로 모델링 하는 것은 가장 중요한 일이다.

먼저 알고리즘을 그래프로 모델링하기 위해서 synchronous data flow (SDF)을 사용한다[21]. SDF는 단순히 알고리즘의 함수들 간의 선후 관계만을 표현한 그래프이다. SDF 그래프에서는 각 노드는 알고리즘의 함수를 의미하고, 간선은 함수간의 관계를 의미한다. 하나의 노드 양 끝에 두 값은 함수를 수행하기 위해 필요한 데이터의 양 (IO token)과 함수의 수행으로 생성된 데이터의 양을 의미한다. 그림 6(a) 는 임의의 알고리즘을 SDF 그래프로 표현한 예시이다. 예시의 알고리즘은 총 3개의 노드로 이루어져 있다. 노드 A는 한 번의 수행으로 1개의 IO token을 생성한다. 노드 B는 한 번 수행되기 위해 2개의 IO token을 필요로 하고, 수행 후 1개의 IO token을 생성한다. 노드 C는 생성하는 IO token은 없으며 수행되기 위해 1개의 IO token이 필요하다. 실제 스케줄링에 적용하기 위해서는 주어진 입력 데이터에 대해서 알고리즘을 수행시키기 위해 노드가 호출되는 수만큼을 표현한 그래프인 acyclic precedence expanded graph (APEG)가 필요하다 [20]. 따라서 SDF를 노드의 IO token에 맞추어 노드의 수만큼의 APEG로 변환하여야 한다. 그림 6(b)는 그림 6(a)의 SDF를 변환한 APEG의 예시이다. SDF를 따라 알고리즘을 수행하기 위해서는 노드 A를 2번 호출해야 하고 나머지 노드들은 각 한 번씩 호출되기 때문에 총 4개의 노드로 구성된다.



(a) Synchronous data flow graph

(b) Acyclic precedence expanded graph

그림 6. 그래프 예시
Fig. 6. Example of graphs

4.3 iSSD 협업을 위한 그래프 모델링 전략

4.3.1 그래프 설계

협업 처리 환경에 적합한 SDF 그래프 (이를 기반으로한 APEG) 를 생성할 때 다음과 같은 이슈가 발생한다. 첫째, 함수와 대응되는 노드의 granularity (function granularity) 을 결정하여야 한다. 협업 처리 환경에서 GPU와 iSSD는 성능은 낮지만, 많은 수의 코어들이 존재하여 병렬처리에 용이하다. 반면 호스트 CPU는 성능이 좋지만 그 수는 적기 때문에 병렬처리보다는 병합처리에 더욱 용이하다. 따라서 노드들은 다음과 같이 분류될 수 있다: (1) 병합처리 필요 노드, (2) 병렬처리가 가능한 노드. 이와 같이 분류된 노드들을 기반으로 다양한 조합의 SDF 그래프를 생성하고, 이들 중 가장 효과적인 조합을 찾는 것이 필요하다. 둘째, 각 함수의 IO token의 크기 (data granularity) 를 결정해야 한다. 협업 처리 환경에서 GPU의 코어들은 iSSD내의 코어들, CPU 코어들과 달리 SIMT architecture이다[19]. SIMT architecture는 여러 개의 코어 (또는 스레드) 들이 주어진 데이터에 대해서 모두 동시에 동일한 작업을 수행하도록 설계

된 프로세싱 구조이다. 즉, 호스트 CPU 코어나 iSSD 내의 코어들은 하나의 블록 (또는 페이지)단위로 데이터를 처리하도록 설계가 되어 있는 반면, GPU는 코어의 수에 비례한 데이터의 양을 동시에 처리하도록 설계되어 있다. 따라서 GPU를 효과적으로 사용하기 위해서는 GPU에 적합하도록 data granularity를 조율하는 것이 필요하다. GPU에서 수행되기 위한 함수는 수식 11를 따른다. IO token의 수는 $n_{IOtoknes}$, GPU의 코어 수를 n_{cores}, GPU 메모리 공간을 S_{mem}, 각 IO token의 크기를 $S_{IOtokens}$ 라고 표기한다. 이 때, IO token의 크기의 총합은 GPU 메모리 공간보다 작거나 같아야 하며, IO token의 수는 GPU의 코어수의 배수여야 한다.

$$n_{IOtokens} = m \times n_{cores}$$
$$n_{IOtokens} \leq S_{mem} / S_{IOtokens} \quad (11)$$

스케줄링 알고리즘은 개발자 및 설계자가 구성한 SDF 그래프의 function과 data granularity를 바탕으로 자동으로 스케줄을 생성한다. 따라서 개발자 및 설계자가 직접 각 함수의 function과 data granularity를 조정해 주어야한다. 우리는 각 알고리즘에 대해 가장 효율적인 스케줄을 생성할 수 있는 그래프의 function과 data granularity를 결정하기 위해서 다음과 같은 과정을 수행한다. 먼저 각 알고리즘를 그래프로 구성할 때, 함수를 가능한 한 가장 작은 단위(fine-grained)의 노드들로 구성하고 노드를 분류한다. 구성된 노드들에 대해 GPU에서 수행되길 원하는 노드를 선택하여 수식 11에 맞추어 data granularity를 조정한다. 이후 분류 된 노드들을 조합하여 function granularity가 다른 SDF 그래프(coarse-grained)를 생성한다. 이렇게 생성된 그래프들에 대해 스케줄을 생성하고 이들 중 수행시간을 최소화 할 수 있는 granularity의 그래프를 선택한다.

4.3.2 스케줄의 호환성

입력 데이터의 크기가 변하는 경우, 결과 도출을 위해 함수를 호출하는 횟수가 변할 수 있으며, 이에 따라 주어진 SDF 그래프의 granularity에 맞춰 APEG의 구조가 변하게 된다. 특히, 정적 스케줄링은 알고리즘 수행 이전에 얻은 스케줄을 그대로 사용해야 하므로, 그래프의 노드 수가 바뀌는 경우 동일한 스케줄을 활용할 수 없다. 이를 해결하기 위해, 입력 데이터 크기 변화량에 따라 data granularity를 동일한 비율로 변경함으로써 기존 그래프를 동일하게 재사용할 수 있다. 이것이 가능한 이유는 주어진 협업 처리 환경의 코어들의 수는 변화하지 않기 때문이다. 먼저, 가장 초기에 주어진 입력 데이터를 기반으로 base 스케줄을 생성한다. 이후 입력 데이터의 크기 변화에 따라 각 노드의 data granularity를 조정함으로써 기존의 수행시간과 IPC 시간을 예상 값으로 변경한다. 예를 들어, base 스케줄을 생성한 데이터에 비해 2배 큰 데이터가 입력된다면, base 스케줄의 data granularity 또한 2배로 높여 사용한다. 또한, 이에 따라 base 스케줄의 각 노드의 수행시간과 IPC 시간을 2배씩 증가하여 스케줄을 재구성한다[6].

입력 데이터 변화에 맞추어 base 스케줄을 호환하여 사용하는 것은 실제 스케줄 보다 비효율적일 수 있다. 그러나 대량의 데이터를 처리하는 데이터 마이닝 알고리즘의 특성상 같은 함수를 반복적으로 처리해야 하므로, 각 프로세서에 함수가 배치되는 경향은 크게 변하지 않을 가능성이 높다. 또한 한 번 생성된 스케줄을 재사용함으로써 스케줄을 생성하는데 발생하는 시간을 제거 할 수 있다. 이는 주어진 granularity에 따라 스케줄을 생성하는 시간이 알고리즘 수행시간 보다 더욱 많은 수행시간을 요구할 수 있기 때문이다[6].

4.4 성능 평가

본 실험에서는 총 세가지 (k-means[13], PageRank[14] SimRank[22]) 알고리즘을 수행한다. 이전 장과 달리 본 실험에서는 Apriori 실험이 제외되었다. Apriori의 경우, 알고리즘의 특성상 데이터의 특성 및 크기에 따라 생성되는 중간 결과물이 매우 불규칙적이며, 그 크기를 가늠하기가 어렵다. 따라서 본 실험에서는 이전 장에서 수행된 알고리즘에서 Apriori를 대신 SimRank를 추가하여 실험을 수행한다. 각 알고리즘에 대한 SDF 그래프는 지면 관계상 생략한다[11]. 협업 처리 환경은 이전 장과 동일하게 수행비용 모델을 기반으로 성능을 평가한다[1]. 하드웨어 매개 변수는 표 4와 같다. 실험 데이터는 100GB의 임의로 생성한 데이터를 사용한다. 데이터는 iSSD내에 각 셀에는 데이터가 고르게 저장되어 있어, 각 FMC 코어에서 균등한 데이터 IO가 발생한다고 가정한다.

본 실험에서는 다음의 총 5가지의 컴퓨팅 환경에 대해서 성능을 비교한다: (1) Host CPU only (CPU), (2) iSSD only (iSSD), (3) Host CPU + iSSD (CPU/iSSD), (4) Host CPU + GPU (CPU/GPU), (5) 모든 컴퓨팅 자원을 다 사용하는 방안 (ALL). 참고로 BIL 스케줄링은 이질 스케줄링 기법이므로, IHP에는

표 4. 하드웨어 관련 매개 변수
Table 4. Parameters related to hardware

Parameter	Value
# of host CPU cores	4
host CPU rate (GHz)	3
# SSD cores	4
SSD core rate (MHz)	800
# FMC cores	32
FMC core rate (MHz)	400
# of GPU cores	3,000
GPU core rate (MHz)	900

적용 불가능하다. 표 5는 각 알고리즘 별 서로 다른 처리 환경에서의 수행시간을 빠른 순서대로 정렬한 결괴이다. 실험 결과, 알고리즘의 수행시간은 협업 처리 환경이 Host CPU나 iSSD만 사용하는 경우보다 더욱 좋은 성능을 보였다. 특히, 모든 컴퓨팅 리소스를 활용한 협업 처리 환경에서 가장 좋은 성능을 보였다.

실험 결과를 통해 우리는 iSSD 기반의 ISP는 빅 데이터 마이닝 알고리즘들을 효과적으로 처리할 수 있다는 사실을 확인 할 수 있으며, 더 나아가 iSSD 뿐만 아니라 다른 컴퓨팅 자원을 이용한다면 이들에 대한 처리 성능을 더욱 향상 시킬 수 있을 것으로 기대한다.

표 5. 각 처리 방안별 수행시간(초)
Table 5. Execution time for each processing environment

	k-means	PageRank	SimRank
ALL	402.4 (1)	1902.9 (1)	3791.6 (1)
CPU/GPU	419.9 (2)	2328.3 (2)	5980.0 (3)
CPU/iSSD	650.8 (3)	2482.6 (3)	5884.3 (2)
iSSD	977.2 (4)	3807.5 (4)	10488.2 (4)
CPU	1264.71 (5)	3796.69 (5)	13612.3 (5)

V. 결 론

최근, 데이터가 저장된 장소와 가장 가까운 곳에서 데이터를 처리하고자 하는 NDP는 기업체와 학계 모두가 큰 관심을 갖고 있다. 특히, SSD 내부에서 데이터를 처리할 수 있는 iSSD는 NDP의 극한이라 할 수 있다. 본 논문에서는iSSD의 등장 배경과 효율적인 빅 데이터 처리를 위한 iSSD 내부 구조를 소개하였다. 또한, iSSD에서 데이터 마이닝 알고리즘을 효과적으로 수행하기 위한 전략들을 소개하였으며, 이를 정확도 높은 수행비용 모델을 설계하였다. 끝으로, iSSD와 더불어 호스트 CPU와 GPU 등 이질 컴퓨팅 자원을 모두 활용하여 데이터를 더욱 효율적으로 처리하는 방안에 대해서 소개하였다.

본 논문은 하나의 머신에 하나의 CPU와 GPU 그리고 iSSD만이 장착되어 있는 환경을 가정하고 있다. 추후 연구로, 우리는 하나의 머신에 여러 개의 CPU와 GPU 그리고 iSSD가 장착된 협업 환경의 성능을 평가하고자 한다. 더 나아가 분산 환경에서 각 머신이 협업 환경으로 이루어진 상황에 대한 성능을 평가하고자 한다.

References

[1] D. Bae et al., “Intelligent SSD: a turbo for big data mining,” in *Proc. ACM CIKM 2013*, pp. 1553-1556, 2013.

[2] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[3] K. Shavachiko, et al., “The hadoop distributed file system,” in *Proc. IEEE MSST 2010*, pp. 1-10, 2010.

[4] R. Greenwald, et al., *Achieving extreme performance with Oracle Exadata*, McGraw-Hill, 2011.

[5] P. Francisco, *The netezza data appliance architecture: A platform for high performance data warehousing and analytics*, IBM Redbooks 3, 2011.

[6] Y. Jo et, al., “On running data intensive algorithms with intelligent SSD and host CPU: a collaborative approach,” in *Proc. ACM SAC 2015*, pp. 2060-2065, 2015.

[7] J. Do, et al., “Query processing on smart SSDs: opportunities and challenges,” in *Proc. ACM SIGMOD 2013*, pp. 1221-1230, 2013.

[8] S. Kim, et al., “Fast, energy efficient scan inside flash memory SSDs,” in *Proc. ADMS 2011*, Seattle, WA, Sept. 2011.

[9] E. Riedel, G. Gibson, and C. Faloutsos, “Active storage for large-scale data mining and multimedia,” in *Proc. VLDB 1998*, pp. 62-73, 1998.

[10] D. Kim and S. Hwang, "An efficient wear-leveling algorithm for NAND flash SSD with multi-channel and multi-way architecture," *J. KICS*, vol. 39, no. 7, pp. 425-432, 2014.

[11] Y. Jo, et al., "Collaborative processing of data intensive algorithms with CPU, intelligent SSD, and GPU," in *Proc. ACM SAC 2016*, pp. 1865-1870, 2016.

[12] J. Zhang, M. Shihab, and M. Jung, "Power, energy and thermal considerations in SSD-Based I/O acceleration," in *Proc. USENIX Workshop HotStorage*, Philadelphia, PA, Jun. 2014.

[13] D. Shin, et al., "Malicious traffic detection using k-means," *J. KICS*, vol. 41 no. 02, pp. 277-284, 2015.

[14] L. Page, et al., *The PageRank citation ranking: bringing order to the web*, Technical Report, Stanford University, 1999.

[15] J. Kim and K. Park, "Personalized group recommendation using collaborative filtering and frequent pattern," *J. KICS,* vol. 41, no. 07, pp. 768-774, 2016.

[16] N. Binkert, et al., "The Gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1-7, 2011.

[17] Y. Jo, et al., "Data mining in intelligent SSD: simulator-based evaluation," in *Proc. BigComp 2016*, pp. 123-128, 2016.

[18] V. Volkov and J. Demmel, "Benchmarking GPUs to tune dense linear algebra," in *Proc. Int. Conf. Supercomputing*(SC 2008), pp. 1-11, 2008.

[19] S. Ryoo, et al., "Optimization principles and application performance evaluation of a multi-threaded GPU using CUDA," in *Proc. ACM SIGPLAN 2008*, pp. 73-82, 2008.

[20] H. Oh and S. Ha, "A static scheduling heuristic for heterogeneous processors," in *Euro-Par Parall. Process.*, pp. 573-577, 1996.

[21] E. Lee and D. Messerschmitt, "Synchronous data flow," in *Proc. IEEE*, vol. 75, no. 9, pp. 1235-1245, 1987.

[22] G. Jeh and J. Widom, "SimRank: a measure of structural-context similarity," in *Proc. ACM SIGKDD 2002*, pp. 538-543, 2002.

조 용 연 (Yong-Yoen Jo)

2011년 2월 : 한양대학교 컴퓨터공학과 졸업
2013년 2월 : 한양대학교 전자컴퓨터통신학과 석사
2013년 3월~현재 : 한양대학교 컴퓨터·소프트웨어학과 박사과정
<관심분야> Data Mining, Graph Processing, SSD, GPGPU, Social Network Analysis

김 상 욱 (Sang-Wook Kim)

1989년 2월 : 서울대학교 컴퓨터공학과 졸업
1991년 2월 : 한국과학기술원 전산학과 석사
1994년 8월 : 한국과학기술원 전산학과 박사
1995년 3월~2003년 2월 : 강원대학교 정보통신공학과 부교수
2003년 3월~현재 : 한양대학교 공과대학 컴퓨터공학부 교수
<관심분야> Data Mining, Recommender Systems, Social Network Analysis, Databases

배 덕 호 (Duck-Ho Bae)

2006년 2월 : 한양대학교 정보통신공학과 졸업
2008년 2월 : 한양대학교 전자컴퓨터통신공학과 석사
2013년 8월 : 한양대학교 전자컴퓨터통신공학과 박사
2013년 9월~현재 : 삼성전자 메모리사업부 책임연구원
<관심분야> SSD-based DBMS, Graph DB, Social Network Analysis