대용량 공유 스토리지 시스템을 위한 토큰 버킷기반 성능 제어 기법

남 영 진[†] · 박 찬 익[††]

요 약

스토리지 시스템의 성능과 용량이 증가됨에 따라서, 스토리지 시스템은 이제 다양한 사용자에 의해서 공유되어 사용되는 것이 일반화 되고 있는 추세이다. 본 논문은 네트워크에서 널리 이용되는 토큰 버킷을 이용하여 공유 스토리지 시스템 상에서 각 사용자 원하는 스토리지 입출력 성능을 확률적으로 보장하는 기법을 설계한다. 또한, 다양한 형태의 작업부하를 이용한 시뮬레이션을 통해서 제안된 알고리즘의 동작 및 성능을 검증한다.

키워드 : 성능제어, 공유 스토리지, 토큰 버킷

Token-Bucket-based Performance Control Techniques for Large-Scale Shared Storage Systems

Young Jin Nam[†] · Chanik Park[††]

ABSTRACT

With the increase in I/O performance and capacity, it becomes commonplace that storage systems are shared by multiple storage users. This paper proposes an efficient token-bucket-based performance control technique for the shared storage that can statistically guarantee demanded storage performance from different users. It also verifies the operational characteristics and performance of the proposed scheme under various types of I/O workloads.

Key Words : Performance Control, Shared Storage, Token Bucket

1. 서 론

SAN(Storage Area Network)[1] 등의 출현 등으로 인해서, 스토리지 시스템(이하, 스토리지)의 입출력 성능과 용량은 큰 폭으로 증가하게 되었다. 이로 인해, SAN으로 연결된 물리적인 스토리지들이 다수의 사용자들에 의해 공유되고[2, 3], 스토리지 가상화(virtualization)를 통해서 각 사용자에게 논리적인 스토리지를 제공하는 것이 보편화되고 있는 추세이다[2, 4]. 반면에, 각 사용자는 여전히 자신이 혼자 그 스토리지를 독점하여 사용하는 것처럼 보장받기를 원한다. 이때 각 사용자가 요구하는 스토리지 성능을 단순화된 형태의 스토리지 QoS[2]라 할 수 있다. 즉, 스토리지 QoS는 성능 이외에 스토리지 용량, 신뢰도, 가격 등과 같은 다양한 요소들이 포함될 수 있지만, 가장 중요한 요소는 스토리지의 입출력 성능이라 할 수 있다[2, 3].

공유된 스토리지에 대하여 각 사용자가 원하는 입출력 성능을 보장해주기 위한 여러 가지 연구들이 다양한 각도에서 진행되었다. 우선, 단일 디스크에 대해서 다양한 입출력 특성을 갖는 각각의 사용자에게 특정 성능을 보장해주는 기법들이 제안되었다[2, 5, 7, 8]. 이들 기법들은 공통적으로 하부 디스크의 구조적 특성을 파악하고, 각 입출력을 서비스할 때 발생하는 오버헤드를 고려하여 스케줄링하고자 하였다. 하지만, SAN기반 RAID 시스템과 같이 대용량 스토리지의 경우에는 단일 디스크에서와 같이 구조적 특성을 파악하는 것은 현실적으로 불가능하며, 따라서 기존의 디스크를 위해서 개발된 알고리즘들을 직접 적용하는 것은 불가능하다. 대용량 공유 스토리지에 대해서도 다양한 성능 제어 기법들이 제안되었다[9, 10]. 기존 연구[9]에서는 사용자로부터 도착하는 각 입출력 요구에 대해서 마감시간(deadline)을 부여하고 해당 사용자 큐에 대기시킨 후에, EDF 방법을 이용하

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학IT연구센터 지원사업의 연구 결과로 수행되었으며, BK21 사업을 통하여 포항공과대학교 전자.컴퓨터공학부에 주어진 교육부의 재정지원과 한국과학재단 특정기초과제(R01-2003-000-10739-0) 및 국제협력연구(F01-2005- 000-10241-0), 그리고 정보통신 선도기반기술개발사업(5NN0501101)을 통한 ETRI로부터의 연구지원을 받았음.

† 정 회 원 : 대구대학교 컴퓨터 · IT 공학부 전임강사
†† 정 회 원 : 포항공과대학교 컴퓨터공학과 교수

논문접수 : 2005년 5월 20일, 심사완료 : 2005년 9월 27일

여 마감시간이 가장 임박한 입출력 요구를 선택하여 스케줄링하는 Facade 구조를 제안하였다. Facade 구조에서는 각각의 입출력 요구에 대해서 항상 성능 제어를 수행하며, 사용자로부터 도착하는 입출력 요구 속도가 SLO(Service Level Objective)[9]에서 정의한 값보다 큰 경우에는 마감시간을 무한대로 설정함으로써 SLO에서 정해놓은 응답시간을 보장하지 않는다. Facade에서는 적어도 한 사용자의 입출력 성능 요구사항이 만족되지 않을 때, 즉시 하부 스토리지와 동시에 처리될 수 있는(outstanding) 입출력 요구 수 값을 감소시켜준다. 이는 성능상의 문제를 발생시킨 실제 사용자 뿐만 아니라, 모든 사용자의 입출력 성능을 감소시키는 결과를 낳을 수 있는 문제점도 함께 존재한다. 기존 연구 [4, 10]에서는 네트워크에서 사용되는 Fair Queuing 알고리즘들을 스토리지 환경에 적합하게 확장한 기법들을 제안하였다. Fair Queuing 방법을 이용할 경우에 가상 클록 관리 뿐만 아니라, 여러 개의 사용자 큐들 중에서 가장 적절한 입출력 요구를 선택할 때 오버헤드가 존재한다.

본 논문에서는 이러한 기존 접근 방법과 달리 네트워크에서 널리 이용되는 토큰 버킷[6]만을 이용하여 대용량 스토리지를 공유하는 각 사용자의 성능 요구사항을 만족시켜주는 기법을 설계한다. 기존 기법들에 비해서 제안된 기법은 처리된 입출력 요구의 응답시간 관측을 통하여 성능제어를 선택적으로 수행함으로써, 사용자가 요구한 응답시간을 만족시켜주는 것과 동시에 초당 처리된 평균 입출력 요구량을 높일 수 있도록 하였다. 또한, 토큰 버킷을 통해서만 성능 제어 작업을 수행함으로써 기존 기법들에 비해서 구현상 비교적 간단하다는 장점도 존재한다. 본 논문은 다음과 같이 구성되어 있다. 2장에서는 본 논문에서 제안하는 토큰 버킷 기반 성능제어 기법을 설명한다. 3장에서는 대용량 스토리지 시뮬레이터를 이용하여 제안된 기법의 동작과 성능 평과 결과를 보여준다. 끝으로, 4장에서 결론 및 향후 연구 방향을 기술하도록 한다.

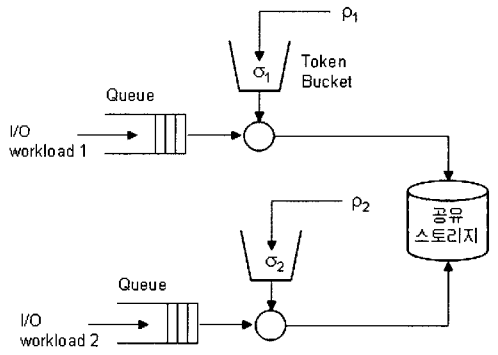
2. 제안된 기법

제안된 기법을 설명하기에 앞서 필요한 정의, 용어 및 가정들을 설명한다. 첫째, 각 사용자의 입출력 패턴은 미리 알려져 있다고 가정한다. 둘째, 각 사용자 입출력 요구는 하부 스토리지 내에 존재하는 데이터를 임의형태로(randomly) 접근한다고 가정한다. 사용자 입출력 패턴은 기존 연구 [11]에서 제안한 방법을 통하여 자동적으로 추출가능하다. 두번째 가정은 본 논문에서 제시하는 알고리즘의 동작에는 영향을 미치지 않으며, 대신 사용자 입출력 요구 패턴을 바탕으로 초기에 어떤 스토리지에서 서비스를 담당하게 할 것인가를 결정할 때 영향을 미친다[2, 3].

한 스토리지를 공유하여 사용하는 사용자 입출력 작업부하(workload)들의 집합을 WS = $\{W_1, W_2, W_3, ..., W_n\}$으로 표기한다. 이때, 각 사용자 입출력 작업부하 W_k는 $\{iops_k, size_k, rt_k\}$로 표기되며, 여기서 $iops_k$는 초당 평균 입출력 요구수, $size_k$는 평균 입출력 요구 크기, 그리고 rt_k는 사용자가 요구하는 입출력 응답시간이다. 본 논문에서 제안하고자 하는 기법의 최종목적은 공유 스토리지에 주어지는 WS에 속한 각 W_k에 대해서, $iops_k$와 $size_k$가 주어질 때 각 입출력 요구에 대한 응답시간이 rt_k보다 크지 않게 하는 것이다. 한 사용자 입출력 작업부하에 대해서 여러 쌍의 $\{iops_k, size_k, rt_k\}$ 값을 갖도록 정의 할 수도 있으며, 이는 본 연구의 향후 연구과제로 남아 있다. 다음으로, 네트워크에서 널리 이용되는 토큰 버킷을 이용한 공유 스토리지 성능 제어를 위한 기법을 설계하도록 한다.

2.1 TA1 기법

본 기법은 네트워크에서 사용하는 토큰 버킷 기능을 거의 수정 없이 그대로 이용한다. (그림 1)에서와 같이 각 사용자별로 독립적인 입출력 요구 큐를 두며, 각 큐에 존재하는 입출력 요구들이 하부 공유 스토리지로 보내지기 전에 토큰 버킷으로부터 $size_k$에 해당하는 토큰을 소비해야 한다. 만일, 충분한 토큰이 존재하지 않은 경우에는 입출력 요구는 큐에 대기해야 하며, 차후에 토큰이 마련된 후에야 토큰 버킷을 통과 할 수 있다. 네트워크와 다른 한 가지는 스토리지에서는 입출력 요구를 임의로 드롭(drop)시키지 말아야 한다는 점이다. 즉, 해당 입출력 요구 큐에 대기시켜야 한며, 대기 큐가 꽉차서 입출력 요구를 드롭시켜야 할 경우에는 호스트에 명시적으로 통보해주어야 한다. (그림 1)에서 ρ_k는 사용자 k의 토큰 버킷에서 토큰이 채워지는 속도를 나타내며, σ_k는 사용하지 않은 토큰이 쌓여 있을 수 있는 버킷의 최대 크기를 나타낸다. TA1 기법은 구현이 단순하다는 장점이 존재한다. 하지만, 어떤 사용자가 하부 스토리지를 이용하고 있지 않을 때에, 그 남은 자원을 다른 사용자가 이용할 수 없는 문제가 존재한다. 결과적으로 낮은 스토리지 사용률을 야기한다. 또한, 스토리지에 도착하는 요구들의 패턴은 토큰을 토큰 버킷에 채우는 패턴과 일치하지 않을 수 있으며, 이로 인한 지연이 발생할 수 있다. 이러한 문제점들을 TA2 기법 설계를 통해서 해결해 보도록 한다.

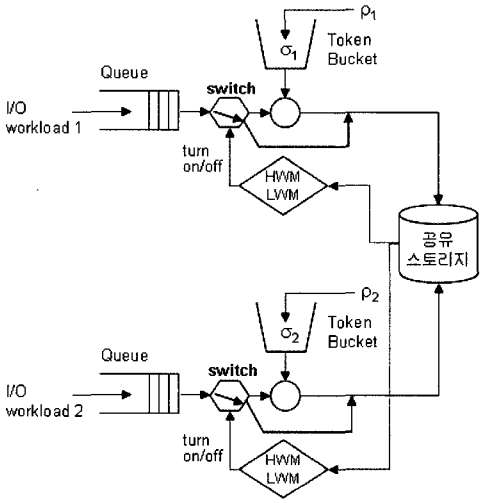


(그림 1) TA1 기법 구조

2.2 TA2 기법

만일 공유 스토리지가 WS에 속한 사용자들의 입출력 요

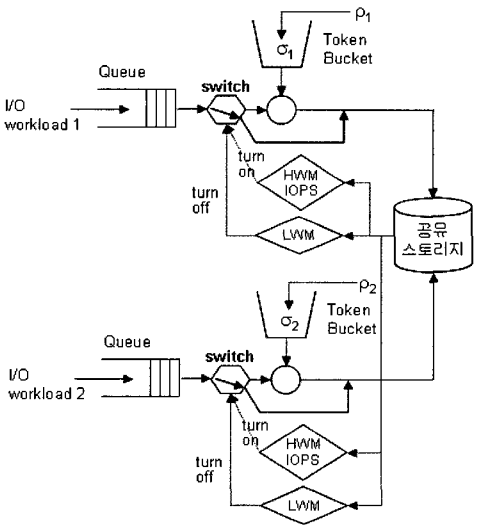
구를 충분히 서비스 할 수 있다면, 굳이 토큰 버킷을 작동시킬 필요가 없다. 이때, 충분히 서비스 할 수 있다 혹은 없다는 스토리지에서 서비스된 입출력 요구의 응답시간을 각 사용자가 요구한 응답시간 rt_k와 비교함으로써 판단할 수 있다. 이러한 아이디어를 수용하기 위해서, TA2 기법은 (그림 2)에서와 같이 TA1 기법에 토큰 버킷을 거치지 않고 통과할 수 있는 경로를 제공한다. 경로 변경을 위해서 일종의 스위치를 두는데, 현재 하부 스토리지 시스템에서 관측되는 응답시간이 각 사용자가 원하는 응답시간 rt_k를 만족하는 경우에 토큰 버킷을 거치지 않도록 스위치를 오프(off)한다. 그 반대의 경우에는 스위치를 온(on)함으로써 토큰 버킷이 동작하게 만든다. 스위치가 너무 빈번히 온-오프되는 것을 방지하기 위해서 고수준(HWM) 및 저수준(LWM) 값을 두었다. 만일, 스위치가 오프 상태에서 관측되어진 응답시간이 각 사용자의 토큰 버킷에 설정된 HWM 값보다 크면, 스위치가 온 상태로 되며 뒤따르는 모든 사용자 입출력 요구는 토큰 버킷을 통과한 후에 하부 공유 스토리지로 내려가게 된다. 반대로, 스위치가 온 상태에서 관측되어진 응답시간이 각각의 LWM 값보다 적어지면, 스위치가 오프 상태로 되며 뒤따르는 모든 사용자 입출력 요구는 토큰 버킷을 통과하지 않고 하부 스토리지로 내려간다. 이렇게 동작함으로써 부가적으로 한 사용자가 사용하지 않는 스토리지 자원을 다른 사용자가 사용할 수 있는 기회가 존재하게 된다. 예를 들어, 두 사용자 W_1과 W_2가 존재하고, W_1이 입출력을 발생시키지 않을 때, W_1과 W_2예의 sz_k가 비슷한 경우에 W_2는 초당 최대 $iops_1$+$iops_2$까지 입출력 요구를 처리함으로써 스토리지 사용률을 높일 수 있다. TA1을 사용할 경우에는 항상 최대 $iops_2$까지만 처리할 수 있다. 결과적으로, TA2 기법은 TA1에서 지적되었던 두 문제점들을 어느 정도 극복할 수 있다. 하지만, TA2는 관측된 응답시간이 HWM 값보다 커졌을 때, Facade 구조[9]에서와 같이 무작정 모든 사용자에 장착된 토큰 버킷을 한꺼번에 동작시킨다는 문제점이 여전히 존재한다. 즉, 실제적으로 문제를 일으킨 사용자에 대해서만 토큰 버킷을 동작시켜서, 그 외 잘 동작중인 사용자들에 대해서는 토큰 버킷을 동작시키지 않고서도 스토리지 입출력 서비스를 받을 수 있게 할 수 있도록 해야 한다.



(그림 2) TA2 기법 구조

2.3 TA3 기법

본 기법은 (그림 3)에서와 같이 TA2 기법에서 관측된 스토리지 응답시간에 대해서 각 사용자의 HWM 값과 비교할 때에, 현재 각 사용자의 입출력 요구 도착률인 $iops_k$도 함께 비교한다. 이렇게 함으로써 실제적으로 응답시간을 높게 만든 사용자가 누구인지를 밝히고, 해당 사용자의 스위치만 온 상태로 만든다. 결과적으로, 초기에 주어진 형태로 입출력 요구를 발생시키지 않고 이를 위반함으로써 문제를 일으킨 사용자에 대해서만 토큰 버킷을 통과하게 함으로써, 위반을 하고 있지 않은 다른 사용자에게는 토큰 버킷에 의한 어떠한 형태의 지연도 발생하지 않도록 한다. 또한, TA2에서와 마찬가지로 사용자 W_k의 입출력 요구 도착률이 $iops_k$보다 높더라도, 다른 사용자들로부터의 입출력 요구가 발생하지 않거나 혹은 상대적으로 낮아서 W_k의 토큰 버킷이 동작되지 않은 채로 스토리지에 대한 입출력 서비스가 이루어질 수 있다.



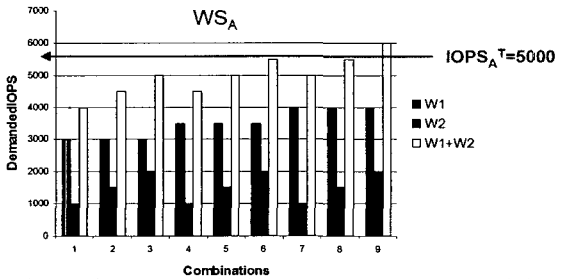
(그림 3) TA3 기법 구조

3. 성능 평가

대용량 공유 스토리지의 성능 제어를 위해서 제안된 세 가지 기법들에 대한 성능 평가를 수행할 목적으로 스토리지 입출력 시뮬레이션 환경을 구축하였다. 공유 스토리지는 호스트 시스템과 두개의 100MB/s 파이버채널(Fibre Channel) HBA(Host Bus Adaptor)을 통해서 연결되어 있다. 스토리지 내부에는 10,000RPM이고 73GB인 SCSI 디스크 8개가 4개의 80MB/s SCSI HBA를 통하여 연결되어 있다. 또한, 사용자 데이터들은 내부 디스크에 RAID0의 형태로 분산되어 저장된다. RAID0의 스트라입 유닛(stripe unit) 크기는 128KB

〈표 1〉 성능평가를 위한 사용자 작업부하 세트

workload set	WS_A		WS_B	
parameter	W_1	W_2	W_1	W_2
$size_k$	4KB	4KB	32KB	4KB
$iops_k$	3500	1500	2500	1000
rt_k	41ms	41ms	40ms	40ms
access pattern	random	random	random	random



(그림 4) W_1 및 W_2의 각 $iops_k$ 값 변화에 따라 구성된 9개의 작업부하 조합

이며, 시스템 내에는 캐쉬가 없는 것을 가정하였다. 각 사용자 작업부하 W_i에 대한 토큰 버킷과 관련된 파라메터로 버킷에 토큰이 채워지는 속도에 해당하는 ρ_i 값과 버킷의 크기에 해당하는 σ_i 값이 존재한다. 또한, 모든 토큰 버킷에 대해서 얼마나 자주 토큰을 버킷에 채워줄 것인가를 나타내는 τ 값이 존재한다. 본 실험에서는 τ=1000으로 설정하였으며, 이 때 ρ_i 값은 $iops_i/\tau$로 설정된다. 또한, 토큰버킷의 크기 σ_i 값은 $1.5\rho_i$로 설정하였다.

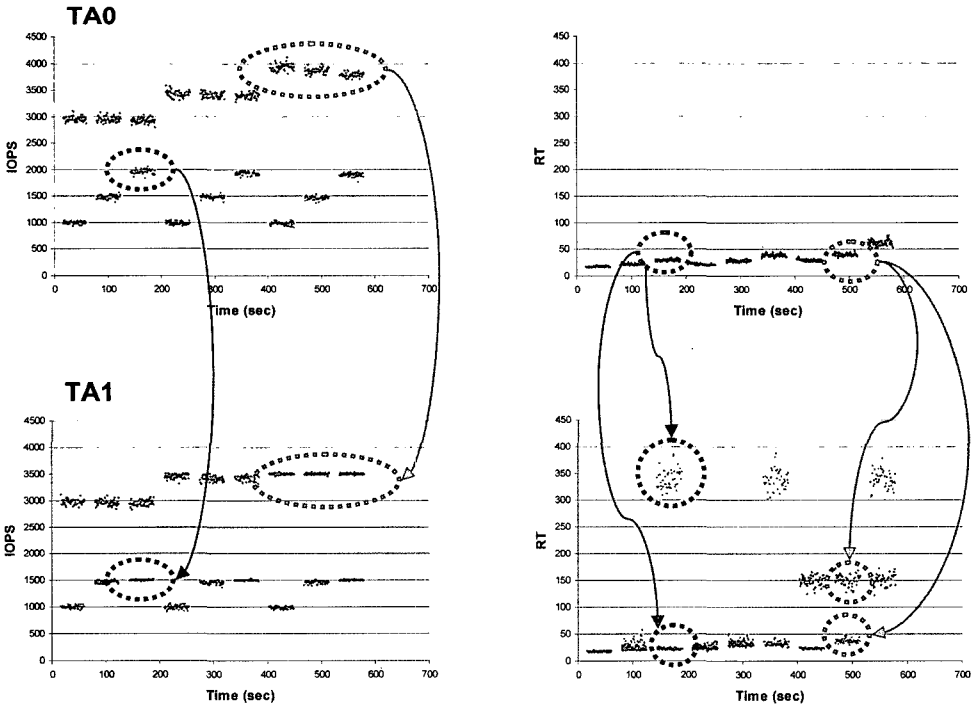
성능 제어 시험을 위해서 입출력 요구의 크기가 작은 DB 응용들로만 구성된 입출력 작업부하 세트 WS_A와 DB 응용 하나와 입출력 요구의 크기가 큰 응용으로 구성된 입출력 작업부하 세트 WS_B를 사용하였다. 참고로, 본 논문에는 작업부하 세트 WS_A의 결과만을 포함하였으며, WS_B에 대해서도 비슷한 형태의 결과를 얻을 수 있었다. 각 작업부하 세트에서 요구하는 응답시간 rt_k은 각 사용자가 $iops_k$ 형태로 입출력을 발생시킬 경우에 하부 공유 스토리지에서 만족을 시켜줄 수 있도록 값을 설정하였다. WS_A의 경우에 두 사용자의 $iops_k$의 합이 5,000일 때, 하부 스토리지는 두 사용자 모두에게 41msec를 보장할 수 있다.

성능 평가를 위한 파라메터로 각 작업부하 세트 내에 존재하는 모든 W_k의 초당 처리된 평균 입출력 요구량과 각 W_k에서의 요구된 응답시간(rt_k) 만족도를 이용한다.

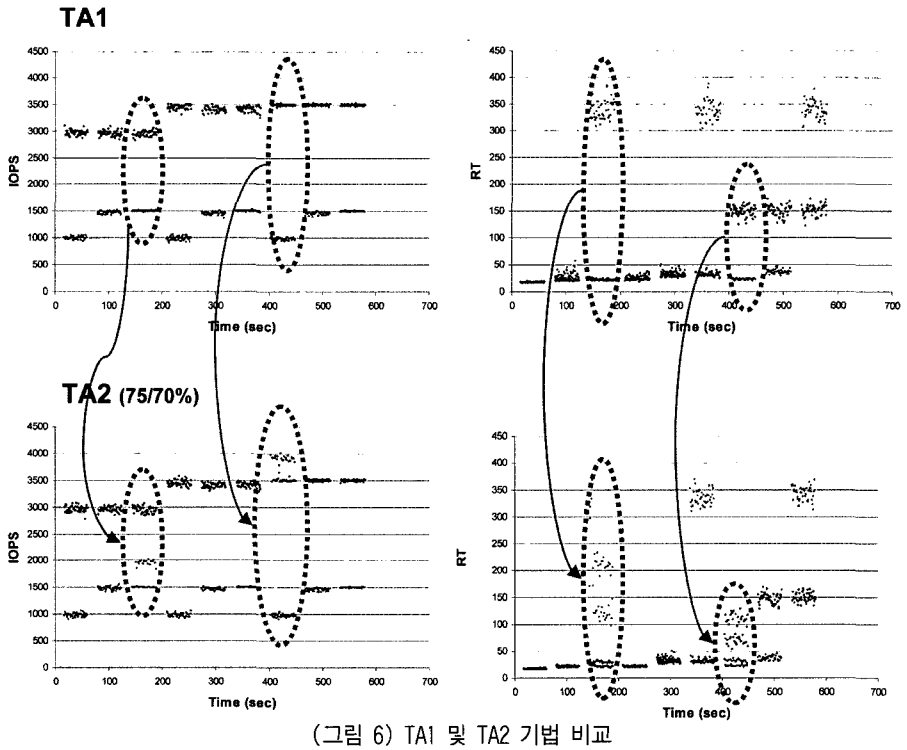
다음으로, (그림 4)에서와 같이 성능 평가를 위하여 각 작업부하 세트 내에 포함된 W_1과 W_2의 $iops_k$ 값을 조정함으로써 총 9개의 조합으로 이루어진 작업부하를 만들었다. W_1의 경우에 구간 1~3에서는 $iops_1$보다 낮은 입출력 요구 발생률을 갖고, 구간 4~6에서는 설정된 $iops_1$과 동일하며, 구간 7~9에서는 $iops_1$ 보다 큰 값을 갖는다. W_2의 경우에는 구간 1, 4, 7에서 설정된 $iops_2$ 보다 낮은 입출력 요구 발생률을 갖고, 구간 2, 5, 8에서 $iops_2$와 동일하며, 구간 3, 6, 9에서는 $iops_2$ 보다 큰 값을 갖는다.

3.1 시간에 따른 입출력 요구 처리량 및 응답시간 변화 비교

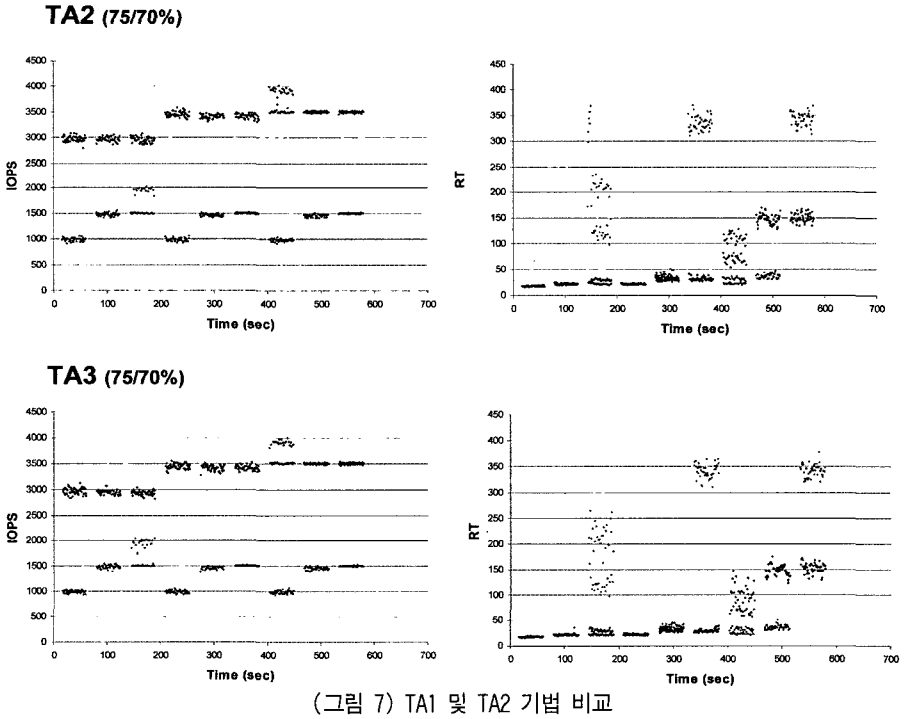
(그림 5)는 토큰 버킷을 적용하지 않았을 경우에 해당하는 TA0 기법과 토큰 버킷이 항상 온 상태에 있는 TA1 기법을 비교한다. TA0 기법을 사용할 경우에 W_1은 영역 7, 8, 9에서 그리고 W_2는 영역 3, 6, 9에서 $iops_k$에서 정한 값보다 높은 속도로 입출력 요구가 도착하게 된다. 따라서, W_2의



(그림 5) TA0 및 TA1 기법 비교



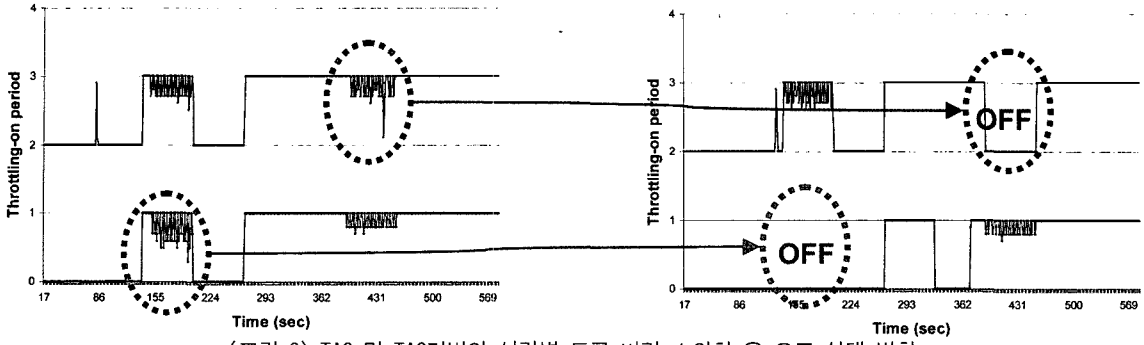
(그림 6) TA1 및 TA2 기법 비교



(그림 7) TA1 및 TA2 기법 비교

경우에 영역 7과 8에서의 응답시간이 rt_2 보다 커지는 문제가 발생하며, W_1의 경우에는 영역 3과 6에서의 응답시간이 rt_1보다 커지는 문제점이 존재한다. 하지만, TA1는 토큰 버킷을 통하여 각 사용자에 대해서 초당 입출력 요구 도착률을 $iops_k$보다 커지지 못하게 항상 제어함으로써, 주어진 rt_k를 만족시켜줌을 볼 수 있다. 응답시간 만족에 대한 구체적인 결과는 3.3절에서 살펴보도록 한다.

(그림 6)은 TA1과 TA2 기법에 대한 실험결과를 보여준다. TA2 기법을 이용할 경우에 두 사용자로부터 도착하는 입출력 요구들에 대한 응답시간이 좋을 때, 즉 HWM 값보다 높지 않을 때는 토큰 버킷을 거치지 않는다. 대신, 측정된 응답시간이 HWM 값보다 커지는 순간 스위치를 온 상태로 만들어 토큰 버킷을 동작시켜준다. 이 동작을 구간 3과 7에서 확인할 수 있다. 구간 3과 7에서는 W_2와 W_1 각



(그림 8) TA2 및 TA3기법의 시간별 토큰 버킷 스위치 온-오프 상태 변화

각에 의해서 스위치가 주기적으로 온/오프 상태로 변화는 것을 볼 수 있다((그림 8) 참조). 이에 따른 응답시간의 변화로 함께 관찰할 수 있다.

(그림 7)은 TA2과 TA3 기법에 대한 실험 결과를 보여준다. TA2 기법이 모든 사용자에 장착된 토큰 버킷을 동시에 동작시키는 반면에, TA3에서는 문제를 일으킨 사용자의 토큰 버킷 만을 선별적으로 동작시키는 기능이 추가되었다. 따라서, (그림 8)에서 보여주는 바와 같이 구간 3과 7에서 실제적으로 문제를 일으킨 W_2와 W_1에 대해서만 토큰 버킷을 동작시켜주는 것을 볼 수 있다. 이렇게 함으로써, TA3는 TA2보다 W_1와 W_2 각각에 대해서 보다 나은 응답시간을 보장 할 수 있다.

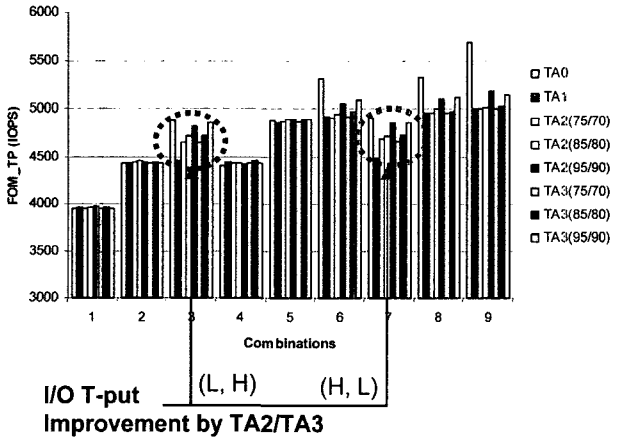
3.2 초당 처리된 평균 입출력 요구량 비교

(그림 9)는 TA1, TA2, TA3 기법에 대해서 주어진 작업부하를 발생시켰을 때 측정한 초당 처리된 평균 입출력 요구량을 보여준다. TA2와 TA3 기법은 서로 다른 세 개의 HWM와 LWM 쌍을 이용하였다. 즉, rt_k에 대한 75% 및 70%, 85% 및 80%, 그리고 95% 및 90% 쌍을 이용하였다. 우선, TA2 및 TA3 기법은 구간 3과 7에서 TA1 기법에 비해 높은 성능 향상이 존재함을 볼 수 있다. 이 구간에서는 한 사용자가 다른 사용자가 사용하지 않는 자원을 이용하며, 이에 따른 성능상의 이득이 존재한다. TA1의 경우에는 계속적으로 토큰 버킷이 동작됨으로써, 여분의 자원을 이용할 수 없게 된다. HWM 및 LWM 값이 높이 설정될수록 보다 높은 성능상의 이득을 볼 수 있음을 알 수 있다. HWM와 LWM를 75% 및 70%로 설정할 경우에도 계속해서 성능 향상이 존재함을 볼 수 있다. 하지만, 주어진 응답시간 만족도 측면에서는 반대의 결과가 있음을 예측할 수 있다.

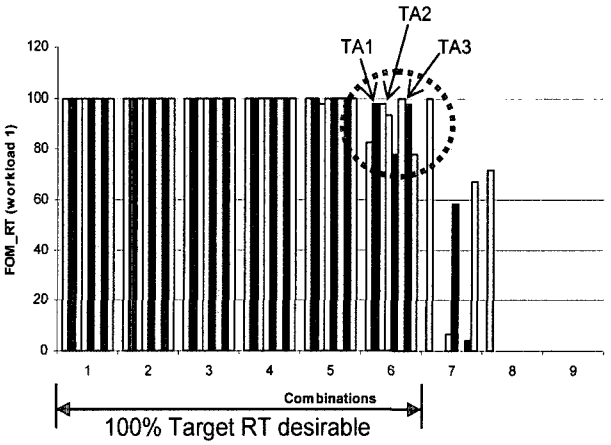
3.3 요구된 응답시간(rt_k) 만족도 비교

(그림 10)은 W_1 작업부하에 대해서, TA1, TA2, TA3 기법을 적용하였을 때에 요구된 응답시간 rt_1을 얼마나 잘 만족시킬 수 있는가에 대한 결과를 보여준다. 우선, W_1의 경우에는 구간 1에서 6까지 요구된 응답시간이 만족되기를 원한다. 결과에 따르면, 구간 5까지 발생된 모든 입출력 요구들에 대해서 rt_1을 100% 만족함을 볼 수 있다. 구간 6에서는 TA2와 TA3의 HWM와 LWM를 75% 및 70%로 설정하였을 경우에는 TA1에서 제공하는 만족도와 동일함을 볼 수 있다. 이 두 값을 점차적으로 크게 설정될수록 만족도가 조금씩 낮아지는 것을 볼 수 있다. 구간 7, 8, 9에서는 입출력 도착률이 $iops_1$ 보다 크기 때문에 요구된 응답시간을 만족하지 않아도 상관없다.

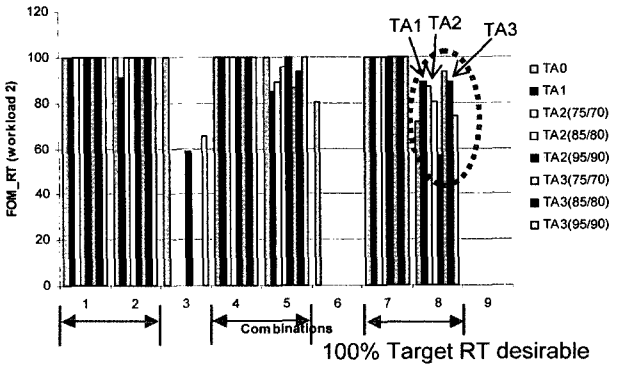
(그림 11)은 W_2 작업부하에 대한 결과를 보여주고 있으며, (그림 10)에서와 유사한 형태의 결과를 얻을 수 있었다. 특히, 구간 5에서는 HWM 및 LWM 값이 커질수록 응답시



(그림 9) 주어진 작업부하에 대한 TA1, TA2, TA3 기법의 초당 평균 입출력 산출량 비교



(그림 10) 주어진 작업부하 하에서 W_1에 대한 TA1, TA2, TA3 응답시간 만족도 비교



(그림 11) 주어진 작업부하 하에서 W_2에 대한 TA1, TA2, TA3 응답시간 만족도 비교

간 만족도가 점차 좋아지는 경향을 보여주고 있다. 그 이유는 W_1과 W_2 모두 주어진 $iops_k$에 맞추어 입출력을 발생시키고 있기 때문에, 토큰 버킷을 되도록 적게 동작시켜주는 상태에서 높은 응답시간 만족도가 발생되었다. 대신, W_1과 W_2의 초당 입출력 요구 도착률의 합이 $iops_1$+ $iops_2$ 보다 큰 구간 8에서는 앞의 (그림 10)에서와 같은 형태의 결과를 볼 수 있다.

3.4 기존 기법과의 비교

토큰 버킷을 기반으로 한 성능제어 기법과 기존 기법들[4, 9, 10]과의 기능 및 성능적인 측면에서 비교 결과를 표 2에서 보여주고 있다. Fair-Queuing기반 알고리즘[4, 10]은 각 사용자에게 하부 스토리지 자원의 대역폭을 비례적으로 나누어 사용하도록 하는 제어만을 수행한다. 즉, 각 입출력 요구 도착률에 대해서 응답시간을 직접적으로 제어하지 않는다. 따라서, 입출력 작업부하 {$iops_k$, $size_k$, rt_k}가 주어질 때, 입출력 요구의 도착률이 $iops_k$ 이하인 경우에 대해서 응답시간이 rt_k보다 크지 않도록 제어하기에는 부적절한 방법으로 알려져 있다[9].

반면에, Facade 구조는 사용자로부터 도착하는 입출력 요구 각각에 대해서 항상 성능 제어를 수행한다. 사용자에서 도착하는 입출력 요구의 속도가 정해놓은 값, $iops_k$를 초과하는 경우에는 다른 사용자의 입출력 요구량에 상관없이 처리 마감시간(deadline)을 항상 무한대의 값을 설정함으로써 입출력 처리의 우선순위를 최하위로 낮춘다. 따라서, Facade 구조는 앞서 기술한 TA1 기법과 동작 및 성능면에서 유사하다는 것을 알 수 있다. 또한, Facade는 사용자로부터 도착하는 각 입출력 요구에 마감시간을 부여하고 이를 EDF 스케줄링 알고리즘에 따라서 스케줄링하도록 구현해야 하는 복잡성이 존재한다.

제안된 기법의 경우에는 사용자로부터 도착하는 입출력 요구률이 $iops_k$ 이하이고 입출력 요구의 응답시간이 설정된 rt_k 값보다 커질 때만 성능제어를 수행한다. 불필요한 성능 제어 작업을 제거함으로써, 사용자가 요구한 응답시간을 만족시켜줄 수 있을 뿐 아니라 초당 처리된 평균 입출력 요구량을 높일 수 있도록 하였다. 또한, 토큰 버킷을 통하여 성능 제어 작업을 수행함으로써 기존 기법들에 비해서 구현상 비교적 간단하다는 장점도 존재한다.

〈표 2〉 기존 성능 제어 기법과 비교

	Fair-Queuing기반 알고리즘 [4, 10]	Facade 구조[9]	제안된 기법(TA3)
성능 제어 대상	스토리지 자원 대역폭	입출력 속도(IOPS)에 따른 응답시간	입출력 속도(IOPS)에 따른 응답 시간
응답시간 제어여부	제어하지 않음	제어함	제어함
성능 제어 시점	항상	항상	필요시(주어진 응답시간 불만족 경우)
구현상 복잡성	복잡 (가상클록관리, 마감시간 부여, Fair Queuing)	비교적 복잡 (마감시간 부여, EDF)	간단 (토큰버킷)
입출력 성능 및 응답시간 보장	-	TA1 기법과 유사	TA3 기법

4. 결론 및 향후연구방향

본 논문에서는 공유 스토리지 상에서 각 사용자 원하는 스토리지 입출력 성능을 확률적으로 보장하기 위하여 네트워크에서 많이 이용되는 토큰 버킷을 기반으로 설계한 성능 제어 기법을 제안하였다. 토큰 버킷을 그대로 이용하며 항상 성능 제어를 수행하는 TA1 기법이 공유 스토리지 상에서 각 사용자가 원하는 성능, 즉 응답 시간을 보장해 줄 수 있음을 여러 형태의 시험을 통해서 우선 확인하였다. TA1 기법에 비해 현재 관측되는 스토리지 응답시간에 따라서 동적으로 토큰 버킷 스위치를 온-오프 상태로 변경함으로써, 필요시에만 성능제어를 수행하는 TA2 및 TA3 기법이 보다 입출력 요구 처리률이 높은 것을 확인하였다. 또한, 실제적으로 문제를 발생시키는 사용자 입출력에 대해서만 토큰 버킷 스위치를 온 상태로 만드는 TA3 기법이 TA2 기법보다 조금 나은 응답시간 만족도를 보이는 것을 확인하였다. TA3에 대해서는 다양한 값을 갖는 HWM 및 LWM 값을 사용하였으며, 그에 따른 영향도 함께 조사하였다. 본 논문에서 사용한 입출력 환경에서는 HWM와 LWM를 85%와 80%로 설정하였을 때, 만족된 결과를 얻을 수 있었다. 제안된 기법은 기존 기법들에 비해서 구현상 간단하다는 장점이 있을 뿐만 아니라, 처리된 입출력 요구의 응답시간 관측을 통하여 성능 제어를 선택적으로 수행함으로써 전체적인 입출력 요구 처리률을 높여주었다는데 기존 기법과의 차별성이 존재한다.

향후 연구로 실제적인 스토리지 환경에 제안된 기법을 적용하는 것과, 사용자의 스토리지 성능에 대한 요구사항을 여러 쌍으로 이루어진 입출력 크기, $iops_k$에 요구되는 응답시간 rt_k로 기술하고 이를 만족시키도록 제안된 기법의 기능을 확장하는 것을 계획하고 있다.

참 고 문 헌

[1] T. Clark, Designing Storage Area Networks: A Practical Reference for Implementing Fibre Channel SANs, Addison Wesley, 1999.

[2] Y. Nam, Dynamic Storage QoS Control for Storage Cluster and RAID Performance Enhancement Techniques, Ph.D Dissertation, POSTECH, 2004.

[3] J. Wilkes, "Traveling to rome: QoS specifications for automated storage system management," Proceedings of International Workshop on QoS, Jun., 2004.

[4] L. Huang, G. Peng, T. Chiueh, "Multi-dimensional storage virtualization," Proceedings of the Joint International Conf. on Measurement and Modeling of Computer Systems, 2004.

[5] Y. Nam and C. Park, "An efficient fair queuing for guaranteed disk bandwidth," Lecture Notes in Computer Science(Proceedings of Euro-Par2004), Springer-Verlag, Sept., 2004.

[6] J. Turner, "New directions in communications," IEEE Communications, Oct., 1986.

[7] J. Bruno, et al., "Disk scheduling with quality of service guarantees," Proceedings of the IEEE International Conf. on Multimedia Computing and Systems, Jun., 1999.

[8] P. Shenoy and H. Vin, "Cello: A disk scheduling framework for next-generation operating systems," Proceedings of SIGMETRICS, Jun., 1998.

[9] C. Lumb, A. Merchant, and G. Alvarez, "Facade: Virtual storage devices with performance guarantees," Proceedings of Conf. on File and Storage Technologies, Mar., 2003.

[10] W. Jin, F. Chase, J. Kaur, "Interposed proportional sharing for a storage service utility," Proceedings of the Joint International Conf. on Measurement and Modeling of Computer Systems, 2004.

[11] A. Veitch and K. Keeton, "The Rubicon workload characterization tool," Technical Report HPL-SSP-2003-13, Hewlett Packard, Mar., 2003.

남 영 진

e-mail : yjnam@daegu.ac.kr

1992년 경북대학교 전자공학과(학사)

1994년 포항공과대학교 전자전기공학과 (공학석사)

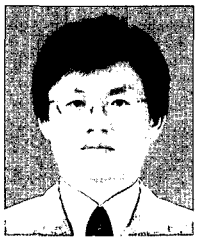
2004년 포항공과대학교 컴퓨터공학과 (공학박사)

1994년~1998년 한국전자통신연구원 연구원

1995년~1996년 미국 Novell사 방문연구원

2004년~현재 대구대학교 컴퓨터·IT공학부 전임강사

관심분야 : 저전력 임베디드 시스템, 스토리지 QoS, 임베디드 OS



박 찬 익

e-mail : cipark@postech.ac.kr

1983년 서울대학교 전자공학과(학사)

1985년 한국과학기술원 전기전자공학과 (공학석사)

1988년 한국과학기술원 전기전자공학과 (공학박사)

1992년~1992년 미국 IBM Thomas J. Watson 연구소 방문연구원

1999년~2000년 미국 IBM Almaden 연구소 방문연구원

1989년~현재 포항공과대학교 컴퓨터공학과 교수

관심분야 : 임베디드 시스템, 시스템보안, 차세대 스토리지 시스템