2012 IEEE 4th International Conference on Cloud Computing Technology and Science

Value-Based Tiering Management on Heterogeneous Block-Level Storage System

Chai-Hao Tsai
Department of Computer Science
National Tsing Hua University, TW
Email: cmj0121@cs.nthu.edu.tw

Jerry Chou
Department of Computer Science
National Tsing Hua University, TW
Email: jchou@cs.nthu.edu.tw

Yeh-Ching Chung.
Department of Computer Science
National Tsing Hua University, TW
Email: ychung@cs.nthu.edu.tw

***Abstract*—As the scale of datacenter continues to grow, it is hard to keep servers homogenous, with the same hardware and performance characteristics. Today's datacenters commonly operates on several generations of servers from multiple vendors, and mix both high-end and low-end devices together to deliver service quality requirement with lowest cost. However, the heterogenous environment also complicates the management of the datacenters, especially in terms of resource allocation. In this paper, we focus on the resource allocation of a tightly unified block-level storage with SSD and HDD. We conduct experiments to quantify the performance of difference access patterns on each type storage devices. Then formulate our resource allocation problem into a ILP (Integer Linear Programming), and proposed data migration algorithms based on the observations. We evaluate our solution by implementing a heterogenous storage consist of HDD, SDD and iSCSI HDD, and show the data access response time can be reduced by 27%.**

***Index Terms*—Tiering management, SSD, heterogeneous storage, performance**

I. Introduction

As the scale of datacenter continues to grow, it is hard to keep servers homogenous, with the same hardware and performance characteristics. It is well known that many of today's datacenters are deployed incrementally and operated on several generations of servers from multiple vendor. It is also commonly seen for the cloud-based datacenters to mix both high-end and low-end devices together, so that they could satisfy the service quality requirement from varied web applications/services with lowest cost. As a result, heterogeneous environment has become increasingly common and popular for datacenter design. However, it also complicates the management of the datacenters, especially in terms of resource allocation, because variant of hardware has to be considered as different types of resources with their own properties and performance.

In this paper, we attempt to solve the resource allocation problem of a heterogenous block-level storage for cloud computing. The system infrastructure of cloud computing is commonly consist of multiple tiers, and block-level storage is one of the tier for providing the fundamental storage service. Specifically, this paper considers a heterogenous block-level storage that is consisted of both hard disk (HDD) and solid-state drive (SSD) from varied vendors.

Many recent research considers SSD and HDD as the typical model of hierarchical or multi-layer heterogeneous storage system. Based on the difference properties between solid-state and traditional disk, many harbingers try to find the mathematical model for heterogeneous storage system [1]. There has been much research in optimizing the heterogeneous or hybrid storage system with both solid-state and disk storage, even on the crossed network environment. Some of them try to integrate solid-state and disk into a hybrid storage system [2]; other try to make the solid-state disk as the traditional storage cache or even make the traditional disk as the write buffer such that solid-state disk can reduce the erase times [5] and increase the lifetime [4]. More details on those approaches are in Section II.

In contrast, this paper attempts to propose a general framework that manages any forms of heterogenous storage device based on the value of data. In general, we use the hot data as the most valuable data in our assumption. For example, the most commonly used data would be the hot data under the usage assumption and these hot data should be placed into high-end device such that can improve the performance on the full system. We give the value for each data and identify the popularity or importance such that we can use these information to model the heterogeneous problem. This model is based on the value we given and calculate the overall value to decide the placement for each data.

We choose the random access as our performance subject. Compared with popularity which can identified by the number of usage, random is hard to be defined in block-level environment. Unlike in file-level, we can guess the video or music files should be sequence read and the metadata or data index would be random access. When random access pattern can be valuation and be given the value for each type of access pattern, we can model the problem into the mathematical problem. Based on this valuation we can classify the data blocks and placement data into the particular disk.

The rest of this paper is organized as follows. In Section 2, we introduce some related work. In Section 3, we describe the model on our system and propose our methodology to solve the hybrid storage device environment. Section 4 present the detailed architecture and show the experiment results. We conclude in Section 5 where we summarize our results.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

II. Related Work

The solid-state drive (SSD) is a storage technology based on NAND flash memory with single-level cell (SLC) or multi-level cell (MLC). Comparing to traditional hard disk drive (HDD), SSD does not have the overhead for disk seek, spin or rotation. Therefore, SSD has lower power consumption and higher access speed for most I/O operations, in particularly for random read. However, SSD wasn't popular in the past because it is much more expensive than disk [3], and it only allows limited number of writes in its lifetime. But, as the price of hardware continue decreasing, many modern datacenters are building their storage systems with the mixing of SSD and HDD.

Several storage architectures and management strategies are designed specifically for heterogenous storage device consisted of SSD and HDD. One common approach is to reduce disk access by caching data in SSD, because SSD has better performance for most I/O operations. But Kim et al found SSD could have worse performance than disk in random write. Thus, they proposed HeteroDrive [5] to redirect random write requests into disk as the buffer for SSD. Another similar approach is I-CASH [2] proposed by Jin Ren et al. I-CASH puts all the modify data into disk continuously and write back into SSD periodically. This gains the benefit of sequential write on disk and reduce the times of write on SSD. [4] also shows using HDD as the write buffer can extend the lifetime of SSD. In contrast, this paper attempts to propose a general framework that manages any forms of heterogenous storage device based on the value of data.

Our work is also close related to the concept of information lifecycle management (ILM) [12]. ILM comprises the policies, processes, practices, and tools to address the management problem in large scale storage system according to the business value of information. Today, most of IT companies develop their storage management product and architecture based on the concept of ILM. Thus, there are also studies on data valuation for hierarchical storage system management. In [1], Xiaonan Zhao et al build a valuation model of heterogeneous storage model by analyzing the frequency, granularity, distribution and association of block-level read/write requests. [10], [11] also build information valuation model at the file-level. In this paper we determine the data value by considering the data access pattern of random I/O requests on various storage device, such that data can be placed into more a suitable type of storage device.

III. Mathematical model and Methodology

In this section, we introduce the mathematical model and some definitions we would use in the later sections, and then model the tiering problem. First, we give some variable and some mathematical model that is described our environment and problem. Second, based on above model we propose our method to solve this problem such that we can gain the benefit of heterogeneous storage system.

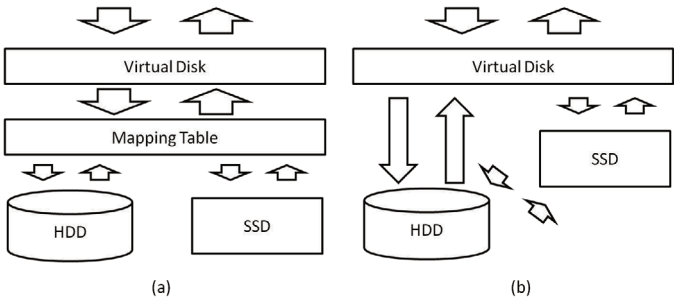


Fig. 1. The usage SSD in two scenario: Scenario (a) is the case which both SSD and HDD are combined into one virtual disk. The mapping table handle the re-direction of request. Scenario (b) is the case which SSD is the storage cache of HDD.

block	popularity	Access Pattern	Factor	Valuation
a	100	Sequence	$\times$ 1	100
b	50	Full Random	$\times$ 2	100
c	80	Partial Random	$\times$ 1.5	120

TABLE I
Example of Blind Spot for Placement: Popularity is not only the factor using on placement policy.

A. Mathematic model

There are various usage scenario to integrate SSD and HDD into a heterogenous storage device. Most of them [4], [5] consider SSD and HDD as different layers in the storage hierarchy as shown in Figure 1 (b). But in our work, we consider a more tightly integration as shown Figure 1 (a) where different storage devices are mixed into a single unified hybrid device through technique, such as mapping table.

For the rest of paper, we use the following variables to define our storage model.

Definition 3.1: Given n disks and m data blocks:

- $x_{ij} \in \{0, 1\}$ indicates whether block j is placed on disk i or not.
- v_{ij} is the value if block j is placed on disk i.
- n_j is the access frequency of block j.
- R_j is the number of replica for block j.
- C_i is the capacity of disk i.

The v_{ij} is the one of the key variables on our model. It is used to identify which block is more suitable to be placed into solid-state disk than traditional disk or verse vice. The precise extents of this value can significantly affect the result. In general, frequent I/O or hot data should be placed into solid-state disk instead of into traditional disk because of the high speed and low latency properties of solid-state. But with this coefficient, it is out of our estimation. Given the example on Table I, if our placement is based on the popularity in Table I, we can find that the priority of placement into solid-state disk is $a > c > b$. But considered the affection of access pattern, the best choice should be c instead.

Based on these variables, our target is to find the best selection such that the system can get the maximal benefit (i.e. aggregated value). Thus, we formulate our problem as follows:

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

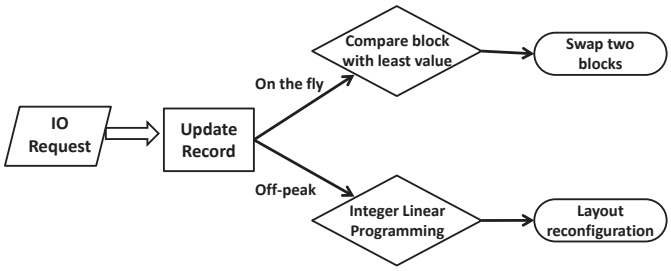


Fig. 2. The workflow of the trigger policy: When the new IO request coming, the system update the record and identify the request is hot or not. In the situation of on the fly, system only migration the block which value exceeds the threshold. In the situation of off-peak, the system will consider the full blocks and re-allocate into the suitable placement.

Definition 3.2: The target function:
Max $\{\sum\sum v_{ij} * x_{ij} * n_j\}$, which the subject functions are

$$\text{Subject to: } \begin{cases} \sum_{i=1}^{n} x_{ij} = R_j, \text{ for block j} \in [1,m] \\ \sum_{j=1}^{m} x_{ij} \leq C_i, \text{ for disk i} \in [1,n] \\ x_{ij} \in \{0,1\} \end{cases}$$

There are two major conditions for this equation. One is the disk capacity which traditional disk has larger capacity and solid-state disk has smaller one; another is the number of replica for each data. Disk capacities is the factor that limited the number of the data in disk. If the disk is full, we need to swap out the data and swap in the more valuable data into disk. The number of replica is used for the security and back-up. If the system need to guarantee the quality of service (QoS) of data, the number of replica should be set larger than 1.

In general, this problem is called the linear programming. Linear programming (LP) is a method for determining the best choice in the given mathematical model for some requirements as linear relationships. LP is widely used in business and economics. Many researchers focus on how to solve the LP problem more efficiently. Simplex algorithm [13] is widely used to solve LP problem and it can solve LP in polynomial time in random problem. In the worse-case, Simplex algorithm solves the LP problem in exponential time, but usually, simplex method can provide the polynomial time complexity.

If the unknown variables are all required to be integers, the problem is called an integer linear problem (ILP). Compared with LP problem which can be solved efficiently in the worse case, ILP are generally NP-hard. The binary ILP is the special case for ILP where variable are required to be 0 or 1, and this problem is one of the Karp's 21 NP-complete problems [6], [8]. In our case, our model is the binary ILP and it is NP-hard problem. Therefore, we use a local maximal method to solve ILP in our case as describe in Section III-B1.

B. Migration Policies

To maximize the storage performance descried in Def. 3.2, our tiering management system has to migrate data among blocks. However, data migration could be a costly operation in storage system and could cause severe performance degradation to users. Thus, as shown in Figure 2, depending on the system loading, our tiering management system make migration decision in two occasions: on the fly and off-peak.

Upon a I/O request arrives, if the system load is high, migration in large data quantity could serious degrade user performance. Thus, we simply attempt to increase the overall system value by swapping the storage location of the requested data with on of the lowest value data in the system. On the other hand, if the system is under utilized, we consider to reallocate all blocks into more suitable location by solving the equations in Def. 3.2. The two migration algorithm is described in more detail in below.

1) Online Swapping Algorithm: Here we consider the system is busy, and we propose Algo 1 that attempt to increase the overall system value by swapping two individual data upon each arriving request. More specifically, upon the arrival of each request, our Algo 1 performs two operations. First is to update a data list L that records the top-K data currently having the lowest value. Second is to increase the total system value by swapping the storage location of the requested data with one the data from the top-K.

To maintain the top-K list, we compare the value of each arriving requested data with the data in the list. If the requested data has lower value than one of the data in the list, we insert the requested data in the list and remove the the highest value data from the list. Thus, the complexity of this operation is $O(K)$.

To swap the data position, we need to compute the new value gained from swapping the requested data with each of the data in the top-K list. Then we pick the data block from the list that can result in the maximum value gain. The the complexity of this operation is again $O(K)$. As a matter of fact, we could combine the two operations together into one loop that scans through the top-K list as shown in the pseudo-code of our Algo 1.

Notice that K is a constant parameter to our algorithm. Thus, the two operations only causes constant overhead to each I/O request. As the value os K become larger, the constant overhead increases, but the swapping benefit could also be greater, because there are more candidate data for swapping.

Finally, since there is still a performance penalty from swapping the data between disks, we only swap the data if the overhead is less than some threshold. The value of threshold is used to make the trigger not too sensitive to migration frequently. If the system is not busy, the threshold can set larger and vice versa.

2) Off-peak Reconfiguration Algorithm: When the system load is light, it gives us an opportunity to reconfigure all the block locations by solving the optimization problem in Eq. 3.2. However, as state previously, the problem is NP-complete, and it is still time consuming to solve it using existing ILP solver. Therefore we propose a linear complexity heuristic method for the problem as shown in Algo 2.

Our Algo 2 is constructed based on the following two observations. (1). SSD has better performance than disk even on the crossed network environment as shown in Figure 4. (2).

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

Algorithm 1 Online Swapping Algorithm

Require: x_A: the current requested block; x_B: the block with the lowest value at its current placement.
Ensure: Compared with the value between original statue and state after swap.

```
1: Update the record of the new request.
2: Maintain the minimal value list.
3: Set V = v_ij * n_j be the value of the block.
4: Compared the value after the swapping V_A and original value V_O
5: if then (V_T - V_O)/V_O ≥ Threshold
6:     Swap
7: else
8:     No operation
9: end if
```

Algorithm 2 Off-Peak Reconfiguration Algorithm

Require: Given the blocks of data **Blocks**:$\{x_1, x_2, \dots\}$.
Ensure: The new placement of each block.

```
1: List_i ← φ for any disk i.
2: Blocks.sort(s) by the key ← number n_i.
3: while Blocks is not Empty do
4:     x = Blocks.pop(1)
5:     for list in List.order(key=priority) do
6:         if list is not Full then
7:             Insert list ← x and Break
8:         else
9:             y = list.Smallest
10:            if n_x * v_x ≤ n_y * v_y then
11:                list.pop(y)
12:                Insert list ← x
13:                Insert Blocks ← y and Break
14:            end if
15:        end if
16:    end for
17: end while
```

More frequent accessed data contributes to higher percentage of the total system value as defined in model in Def. 3.2. Therefore, Algo 2 is consisted of the following steps.

First, in line 2 of Algo. 2, we sort all block by the number of access and place them into storage system in order. Then, we allocate into the particular disk according by the value defined by Def. 3.2. If block wants to insert into solid-state which is full, the module will choose the smallest value on the solid-state, compared with the value between two. If the new block is more suitable than original one, swap two block and original would be re-allocate again. The detail are in line 5-13 of Algo. 2.

IV. Implement and Experiment

In this section, we first briefly explain the implementation of our work. Then we discuss how to build our data valuation model. Finally, we evaluate the performance results from our valuation model and data placement algorithms.

Server			
CPU	Intel Xeon L5640		
Memory	24GB		
OS	CentOS 5.8 with kernel version 2.6.18-274.e15		
Network	D-link DGS-1210-24		
Make and model	Capacity	solid-state	Network
WD-2003ABYX	2T		
Intel-SSDSA2M080G2GC	80G	Y	
KINGSTON-SV100S264G	64G	Y	
M4-CT128M4SSD2	128G	Y	Y
Hitachi-HDS723020BLA642	2T		Y

TABLE II
IN OUR EXPERIMENTS, WE BUILT A HETEROGENOUS STORAGE THAT CONSISTS OF FIVE DIFFERENT TYPES OF DISKS

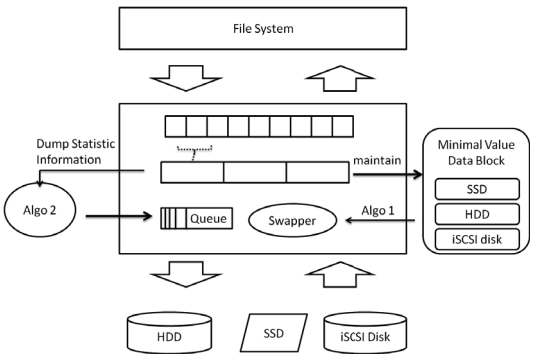


Fig. 3. The implementation of our heterogenous storage system management mechanism.

A. Implementation

To evaluate our storage management mechanism. we built a heterogenous system consisted of diverse range of devices. As summarized in Table II, our heterogenous environment includes five different disks: one local traditional disk, two local SSDs, one remote traditional disk, and one remote SSD. The three remote disks are placed on a remote server that connected to our local server by ethernet cable though the iSCSI protocol. We implement our block-level data management mechanism as a kernel module in CentOS 5.8 with kernel version 2.6.18. The architecture of the implementation is shown in Figure 3.

First of all, as we know, OS handles I/O requests in the unit of logic block address (LBA). However, in practice, it would be too costly to record information and manage data in such fine granularity. Thus, in our implementation, we group many logic blocks into a single "*Unit*" as the definition of data for our management mechanism. As illustrated by Figure 3, our mechanism controls the device mapping table in OS to map each *Unit* of data blocks onto different disks. Upon the arrival of a I/O request from file system, it is then redirected to the raw disk that holds the *Unit* of the data block.

Our kernel module implements two key components. One is the monitor that records the information of each "*Unit*" for our data migration algorithms. Currently, we records the access pattern and access frequency. The other one is the controller that either swap data on-the-fly using Algo 1, or reconfigure the whole data placement using Algo 2. Based on the new data placement position, the controller then migrates data among disks, and modifies the colorred OS LBA mapping table.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

B. Valuation Modeling

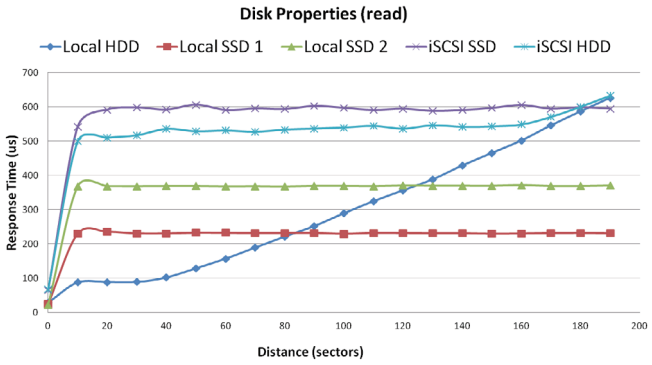
As describe in Section III, our approach is based on a data valuation model that captures the value $v_{i,j}$ of placing data i on disk j. So here, we describe how to determine the value of v_{ij} for our experiments in the next subsection.

In the experiments, we decide to use the I/O response time as our performance metric. As shown in previous studies, SSD and traditional disk have different response time between random access and sequential access, or between write and read requests. But the differences may actually depend on the degree of randomness and the performance of devices. Therefore, our objective is to quantify the correlation between response time, access pattern and types of disk.

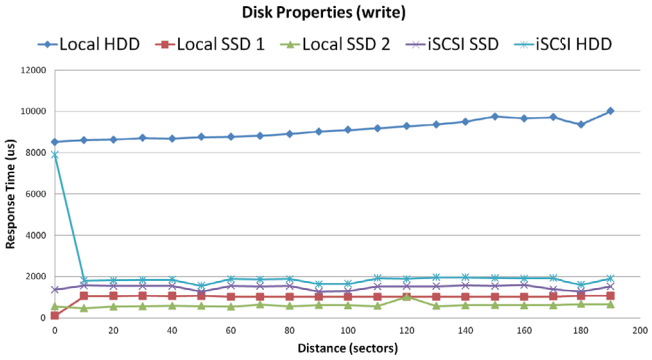
Specifically, we consider the degree of randomness by the distance between two consecutive requests. For the experiments, we modified a commonly used disk profile *hdparm* [9] to generate I/O requests separated in a given distance (i.e. number of sectors), and collect the average response time of the requests. Figure 4 reports our I/O response time measurements versus the distance (i.e. randomness) for each of the five types of disk used in our storage system. We varied the distance from 0 sector to 190 sectors, and we conducted the experiments for both read and write requests. As can be seen in the figure, the read and write requests must be considered separately, because they have quite different characteristic and behavior. According to the results, given the randomness of a data and type of request (i.e. read or write), we can determine its corresponding value in each type of disks.

From Figure 4 we also have the following observations. For the read request, the performance of solid-state is pretty stable when the access distance is more than 10 sectors. Compared with the solid-state, the response time of traditional disk is almost linear growth by the distance, while the iSCSI disk has the same result after controlling the distance large than 60 sectors. Integrated experimental result between solid-state, disk and iSCSI, we can find that the traditional disk is batter than solid-state disk when the distance between requests closed is smaller than about 50 sectors, and the iSCSI does not affect the performance significantly on this case.

On the other hand, for the write request shown in Figure 4 (b), we found the performance of solid-states is about 8 times of the performance of traditional disk. The iSCSI also have much better performance than traditional disk likely due to the network buffering effect. As we observers that the iSCSI disk would return the finish signal even if the data has not written into the remote disk.



(a) Disk read request response time versus randomness (i.e. distance between consecutive requests)



(b) Disk read request response time versus randomness (i.e. distance between consecutive requests)

Fig. 4. We control the distance of two consecutive requests and report its performance impact on the average response time for different types of disks denoted in Table II.

C. Performance Evaluation

We use the block-level OLTP1 data access trace log from *UMass Trace Repository* [7] to drive our evaluation. The trace log is collected from a financial service server. Our evaluation was conduct on a heterogenous storage previously described in Section IV-A. We show the performance improvement of our approach by comparing the request response time with and without our tiering management mechanism. Without our tiering management mechanism, we assume the storage system simply combines all disk linearly in an order, and no data migration would occur.

In our experiment, we re-play the I/O request in the trace log and report the average response time of every 500 requests as a stage over the time. The Figure 5 and Table III summarize the results. For the first two stages of requests, we have lower improvement because our system was in a learning stage to collect the access pattern of new records and cause more data migrations. The information inaccuracy and the migration overhead limit the performance improvement from our approach. For the next four stages of requests, the system became more stable, and the hot data could be more accurately identified. Thus the system performance was clearly improved comparing to the results without our tiering management. Finally, we found the access pattern of data blocks was changed for the last stage of requests. As a result, the performance improvement was decreased. However, as our system continues to profiling the access pattern of data blocks, we expect the improvement to increase once again after the access pattern become more stable. Overall, we observed an average improvement in I/O request response time from $3.8\% \sim 27.2\%$.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

stage	without tiering management	with tiering management	improvement
1	800.25	794.81	6.8%
2	734.71	731.71	3.8%
3	733.85	659.78	11.3%
4	884.88	705.39	25.4%
5	901.05	708.47	27.2%
6	667.18	575.12	16.0%
7	737.87	734.54	4.5%

TABLE III
THE AVERAGE REQUEST RESPONSE TIME AND PERFORMANCE IMPROVEMENT REPORTED FROM OUR EXPERIMENTS USING THE OLTP1 TRACE.

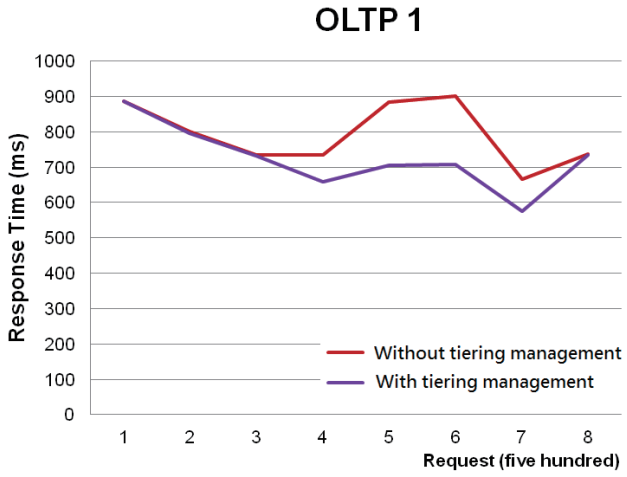


Fig. 5. The average response time of every hundred requests from the trace log of OLTP1.

V. FUTURE WORK AND CONCLUSIONS

We model the heterogeneous resource problem for a block-level storage in distributed and heterogeneous environment. By modeling the problem, we can find the theoretically best solution and find the useful information that can classify the data by the degree of randomness.

Using the modified benchmark tool hdparm, we built our data valuation model based on the relationship between the randomness of data, the type of disk, and the performance of I/O access. Based on the valuation model, we proposed an on-line and an off-line data placement strategies. Upon receiving each I/O request, we use a data swap strategy to improve system performance without introducing too much overhead. The periodically, we use a off-line reconfiguration algorithm to maximize the performance by finding the more suitabable for all data.

Using our work, the heterogeneous storage can gain the benefit of the solid-state drive on performance and the benefit of the disks on the capacities. Using the solid-state accounted the 10% capacities of all, we can improve the performance on response time for about $3.8\% \sim 27.2\%$.

In the future work, we will continue our work in several directions. (1) In this work, we determine the data value based on the data access randomness. But in the future, we would like to explore other more parameters. (2) We would like to keep improving our heuristic algorithm for data reconfiguration. (3) We will attempt to apply our approach at the file-level. In file-level environment, we could collect more information to classify the data, include the last time of usage or modify, the type or the owner of file, even the size of the file, which can more helpful on the decision.

REFERENCES

[1] Xiaonan Zhao, Zhanhuai Li and Leijie Zeng. "A Hierarchical Storage Strategy Based on Block-Level Data Valuation". *Fourth International Conference on Networked Computing and Advanced Information Management, IEEE*, 2008, Page(s): 36–41
[2] Jin Ren and Qing Yang. "I-CASH: Intelligently Coupled Array of SSD and HDD". *High Performance Computer Architecture (HPCA), IEEE*, 2011, Page(s): 278–289
[3] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety and Antony Rowstron. "Migrating Server Storage to SSDs: Analysis of Tradeoffs". *European conference on Computer systems (EuroSys), ACM*, 2009, Page(s): 145–158
[4] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan and Ted Wobber. "Extending SSD Lifetimes with Disk-Based Write Caches". *USENIX conference on File and storage technologies (FAST), ACM*, 2010, Page(s): 8–8
[5] Sang-Hoon Kim, Dawoon Jung, Jin-Soo Kim and Seungryol Maeng. "HeteroDrive: Reshaping the Storage Access Pattern of OLTP Workload Using SSD". *International Workshop on Software Support for Portable Storage (IWSSPS)*, 2009, Page(s):13–17.
[6] bitprog, Solve binary integer programming problems. *http://www.mathworks.com/help/toolbox/optim/ug/bintprog.html*. MATLAB.
[7] UMass Trace Repository. *http://traces.cs.umass.edu/index.php/Storage/Storage*.
[8] KHACHIAN, L G. Polynomial algorithms in linear programming. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 1980, Vol. 20, Page(s) 51-68
[9] hdparm - get/set SATA/IDE device parameters. *http://linux.die.net/man/8/hdparm*.
[10] Ying Chen, information valuation for information lifecycle management, Proceedings of international Conference on Autonomic Computing (ICAC 05), June 2005, pp. 135-146.
[11] Muzhou Xiong, Hai Jin, Song Wu, Information Lifecycle Management in Multi-Gird Environment, Proceeding of the fourteenth national conference on information and storage techniques, September 2006, pp. 175-184.
[12] ILM Definition and Scope an ILM framework, July,2004. *http://www.snia.org/forums/dmf/programs/ilmi/DMF-ILMVision2.4.pdf*.
[13] J. A. Nelder and R. Mead. "A simplex method for function minimization". *The Computer Journal*, 1965, Page(s): 308-313.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE