SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2016, Vol. 59 092104:1–092104:14
doi: 10.1007/s11432-015-5383-x

Characterizing and optimizing TPC-C workloads on large-scale systems using SSD arrays

Jidong ZHAI, Feng ZHANG, Qingwen LI, Wenguang CHEN & Weimin ZHENG*

Department of Computer Science and Technology, Tsinghua University, Beijing 100084*, China*

Received November 23, 2015; accepted April 19, 2016; published online August 23, 2016

Abstract Transaction processing performance council benchmark C (TPC-C) is the de facto standard for evaluating the performance of high-end computers running on-line transaction processing applications. Differing from other standard benchmarks, the transaction processing performance council only defines specifications for the TPC-C benchmark, but does not provide any standard implementation for end-users. Due to the complexity of the TPC-C workload, it is a challenging task to obtain optimal performance for TPC-C evaluation on a large-scale high-end computer. In this paper, we designed and implemented a large-scale TPC-C evaluation system based on the latest TPC-C specification using solid-state drive (SSD) storage devices. By analyzing the characteristics of the TPC-C workload, we propose a series of system-level optimization methods to improve the TPC-C performance. First, we propose an approach based on SmallFile table space to organize the test data in a round-robin method on all of the disk array partitions; this can make full use of the underlying disk arrays. Second, we propose using a NOOP-based disk scheduling algorithm to reduce the utilization rate of processors and improve the average input/output service time. Third, to improve the system translation lookaside buffer hit rate and reduce the processor overhead, we take advantage of the huge page technique to manage a large amount of memory resources. Lastly, we propose a locality-aware interrupt mapping strategy based on the asymmetry characteristic of non-uniform memory access systems to improve the system performance. Using these optimization methods, we performed the TPC-C test on two large-scale high-end computers using SSD arrays. The experimental results show that our methods can effectively improve the TPC-C performance. For example, the performance of the TPC-C test on an Intel Westmere server reached 1.018 million transactions per minute.

Keywords TPC-C benchmark, OLTP workload, performance analysis, performance optimization, high-end computers

Citation Zhai J D, Zhang F, Li Q W, et al. Characterizing and optimizing TPC-C workloads on large-scale systems using SSD arrays. Sci China Inf Sci, 2016, 59(9): 092104, doi: 10.1007/s11432-015-5383-x

1 Introduction

High-end computers [1] play an important part in running IT's most mission-critical applications, such as finance, telecommunication, and banking. The market for typical high-end computers is dominated by several big company models, such as the HP Superdome series, IBM x3850, and so on, which feature large processor and memory scales, and high levels of reliability, availability, and serviceability. For example,

* Corresponding author (email: zwm-dcs@tsinghua.edu.cn)

HP Superdome-2 servers use highly reliable Intel Itanium 9500 processors and support up to 256 processor cores, as well as have 8 TB of memory.

The on-line transaction processing (OLTP) workload [2] is an important type of application that runs on high-end computers. For instance, telecommunication companies use OLTP applications to handle tens of thousands of telephone communication service orders every second. To evaluate the performance of high-end computers executing OLTP workloads, the Transaction Processing performance Council (TPC) releases the TPC-C benchmark [3]. Due to its widespread usage, TPC-C has become the de facto standard for evaluating OLTP performance on high-end computers.

The TPC-C benchmark is different from other standard benchmarks such as SPEC CPU2006 [4]. The TPC organization is only responsible for creating TPC-C evaluation standards, but does not provide any specific implementation or optimization strategies for a given high-end computer. Computer manufacturers need to design and implement the TPC-C benchmark according to the TPC-C standard specification. Also, the evaluation process should be in strict compliance with the standard, and evaluation results should be certified by the TPC organization before release.

Despite previous efforts, it is still a challenging task to get optimal performance results for TPC-C evaluation on large-scale high-end computers. The TPC-C workload has large computational and memory requirements, frequent I/O accesses for small file blocks, and large-scale concurrent user requests. All of these features make it challenging to obtain optimal results. Below is a list of the main challenges of the TPC-C evaluation.

• **High demand on system capability of I/O processing.** TPC-C workloads feature a large quantity of I/O accesses, each for very small file blocks. For example, when the TPC-C performance is about 1 million transactions per minute (TPMC), there are about 370000 physical I/O accesses per second. According to the TPC-C standard, these I/O requests should be completed within a specified time. Therefore, the requirement for I/O operations per second is very high. Traditional TPC-C evaluation systems typically require tens of thousands of hard disks to meet the test requirements. As a result, the total cost of the traditional test systems is very high.

• **Large-scale test data.** According to the TPC-C standard, a large quantity of test data must be preloaded into the storage systems to achieve a given testing objective. For example, the top-ranked Oracle SPARC system in the TPC-C released results requires approximately 250 TB of pre-loading test data [5]. The effective organization of the test data to make full use of the underlying storage systems is a challenging problem.

• **High demand for computing and memory resources.** TPC-C workloads also place large demands on computing and memory resources, requiring systems to meet tens of thousands of concurrent transaction requests per second from different users. It is a hard problem to effectively use the system's memory resources and reduce the processing overhead.

To address the above problems, we designed a large-scale TPC-C evaluation system using solid-state drive (SSD) storage devices and further proposed a series of system-level optimization methods. In summary, we make the following contributions:

1. We propose an approach based on SmallFile table space to organize the test data in a round-robin method on all of the disk array partitions. This can make full use of the concurrent processing capabilities of the underlying disk arrays.

2. According to the characteristics of the TPC-C workload, we propose using a NOOP-based disk array scheduling algorithm to reduce the utilization rate of processors and to improve the average service time of each I/O process.

3. To improve the translation lookaside buffer (TLB) hit rate and reduce the processor overhead, we take advantage of the huge page technique to manage a large amount of memory resources in high-end computers.

4. We propose a new interrupt mapping strategy to improve system performance. According to the locality of interrupt requests, we map different types of interrupts on different processors based on the asymmetry characteristic of the Non-Uniform Memory Access (NUMA) systems to reduce the interrupt

interference on user processes.

In this study, we performed a number of experiments on different high-end servers to validate the above optimization strategies. The experimental results show that our methods can effectively improve the TPC-C performance. For example, the performance of the TPC-C test on an Intel Westmere server reached 1.018 million TPMC.

This paper is organized as follows: Related work is discussed in Section 2; An overview of our TPC-C evaluation system is given in Section 3; The TPC-C workload characteristic is described in Section 4; We present our optimization strategies in Section 5; Experimental results are reported in Section 6; And our conclusion is presented in Section 7.

2 Related work

TPC-C implementations. There are some open-source implementations of the TPC-C benchmark, such as Hammerora [6], TPCC-UVa [7], and DBT-2 [8]. Commercial implementations include Benchmark Factory [9], and Orastress [10].

TPCC-UVa [7] is a TPC-C implementation developed by the University of Valladolid. TPCC-UVa only supports the PostgreSQL database system [11], and the current implementation cannot perform the TPC-C test on large-scale systems. DBT-2 [8] was developed by Portland State University. DBT-2 is similar to TPCC-UVa and currently only supports the PostgreSQL database system. Hammerora [6] is another open-source implementation of the TPC-C benchmark and supports most popular database systems, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. However, it does not use any database middleware to manage database transactions between clients and database servers. User requests are directly dispatched to the database server without needing to queue up for better utilization of server resources, making it difficult to extend to larger-scale high-end computers. London Linxcel Europe Company's Orastress Suite [10] is a TPC-C commercial implementation, which is primarily designed as an Oracle database evaluation software. However, this software only realizes two read-only transactions in the TPC-C test, Stock-Level and Order-Status. At the same time, Orastress requires all the test data to be stored in the memory, which is not in accordance with the TPC-C specifications. Benchmark Factory [9] was developed by QUEST Company. It is similar to Hammerora in that it does not use any database middleware management technology, and the simulation clients only support Windows platforms.

OLTP analysis and optimization. Hsu et al. [2] analyzed the characteristics of the production database workloads of 10 of the world's largest corporations and also compared them with TPC-C and TPC-D. However, their evaluation platform is based on hard disk drives, whileas our evaluation system only uses SSD devices. Delimitrou et al. [12] proposed a modeling and generation framework that greatly reduces the time it takes to set up and perform storage experiments in large-scale instances of TPC benchmarks. Barham et al. [13] described a Magpie system, which can take stand-alone events generated by the operating system, middleware, and application components, correlate related events, and finally produce a workload model. Kim et al. [14] proposed a hybrid storage architecture, Hetero Drive. HeteroDrive can actively reshape random writes to sequential writes.

Chen et al. [15] compared the I/O access patterns of both TPC-C and TPC-E using two disk traces. They found that TPC-E is more read intensive than TPC-C. In their work, they only analyzed I/O patterns for both benchmarks. However, we optimized the performance of TPC-C on a large-scale SSD machine. Ash et al. [16] investigated how to optimize traditional database storage structures to take advantage of the new characteristic offered by SSDs. However, they only analyzed two basic operations in OLTP workloads. Yao et al. [17] conducted a performance evaluation of two public Clouds. They used TPC-C to evaluate three types of instances provided by the two public Clouds, respectively, and found that both Clouds could provide elastic processing ability to meet the demand of increasing workload. Zhai et al. [18] proposed a performance prediction model for parallel programs. Chen et al. [19] introduced a functional workload model for building a big data benchmark from the TPC-C benchmark. The authors

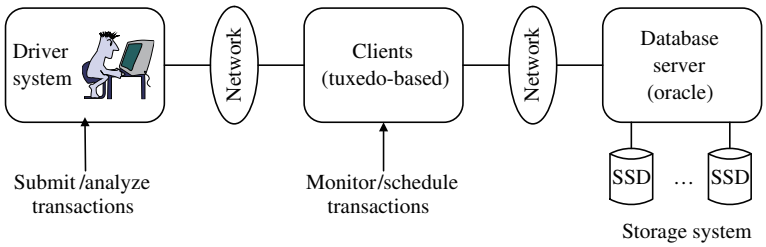

Figure 1 TPC-C system framework.

Figure 2 TPC-C evaluation system.

argued that a big data benchmark built with this model could be representative, portable, scalable, and relatively simple. Tozun et al. [20] studied the evolution of the OLTP benchmarks throughout the decades, including TPC-B, TPC-C, and TPC-E. Their results showed that TPC-E exhibited similar microarchitectural behavior to TPC-B and TPC-C.

In this study, we analyzed the characteristics of the TPC-C workload on a real large-scale system using only SSD arrays and performed a series of system-level optimization strategies on this system. To the best of our knowledge, our paper is the first to implement a real TPC-C evaluation environment using only SSD devices.

3 TPC-C overview

We have designed and implemented a TPC-C evaluation system based on the latest TPC-C standard specification of Version 5.5. The framework of the TPC-C evaluation system is shown in Figure 1, which consists of several main modules, a driver system, clients, a database server, and a storage system.

Driver system. The driver system is used to simulate the behavior of end-users, which includes filling out an order list, submitting a transaction request, receiving processing results, and recording the processing time and statistical results for each transaction.

Clients. Clients receive transaction requests from different end-users, then submit the transactions to database servers and wait for processing results from the servers. At the same time, they are also responsible for scheduling transactions. The final results from the database servers are returned to the end-users. Our system currently uses Tuxedo middleware to monitor and schedule database transaction requests.

Database server. It executes various database system software and handles requests from end-users. To support large-scale data processing and evaluation on high-end computers, our system uses the Oracle database system. The driver system, client, and database server are connected by an Ethernet network.

Storage system. It is used to store testing data for the TPC-C evaluation. Normally, disk storage arrays are widely used, which are connected to the database server through fiber switching devices.

Figure 2 shows our TPC-C evaluation system, which includes dozens of clients and driver systems, and

Table 1 Data operation statistics of five transactions in the TPC-C database tables (R means read, W means write)

DB Tables	NEW-ORDER	PAYMENT	ORDER-STATUS	DELIVERY	STOCK-LEVEL
WAREHOUSE	R	R,W	–	–	–
DISTRICT	R,W	R,W	–	–	R
CUSTOMER	R	R,W	R	R,W	–
HISTORY	–	W	–	–	–
NEWORDER	W	–	–	R,W	–
ORDER	W	–	R	R,W	–
ORDER-LINE	W	–	R	R,W	R
ITEM	R	–	–	–	–
STOCK	R,W	–	–	–	R

a set of SSD storage systems. Client and database servers are connected by a Gigabit Ethernet network. Each storage array consists of 24 60 GB Single-Level Cell (SLC) SSD disks. There are a total of 26 disk arrays, which are connected to the database server by fiber switches.

4 TPC-C workload characteristics

To optimize the TPC-C evaluation system, we analyzed the workload characteristics of the TPC-C benchmark, including database transaction types, database operation modes for different tables, and underlying data access patterns.

4.1 TPC-C transactions

TPC-C is a complex OLTP evaluation program, which contains a large number of disk I/O operations. The data distribution of transaction accesses is also very complex. The TPC-C benchmark simulates a wholesale supplier owning a number of geographically distributed warehouses. It includes five types of database transactions, New-Order, Order-Status, Payment, Delivery, and Stock-Level.

• **New-Order.** The New-Order transaction completes the entire order creation through a complete database transaction. It is a middle-weight frequent read write operation transaction, and its proportion in the TPC-C test is about 45%. The number of new-order transactions completed per minute successfully represents the final TPC-C testing performance (TPMC).

• **Payment.** The Payment transaction is used to update the account balance of customers, and it reflects the changes in sales of different warehouses. It is a light-weight transaction with a mix of reading and writing operations. Its proportion in the TPC-C test is about 43%.

• **Order-Status.** The Order-Status transaction is used to query the state of customer orders. It is a middle-weight read-only database transaction. The execution frequency is relatively low and accounts for 4% of all TPC-C transactions.

• **Delivery.** The Delivery transaction is a batch transaction process and processes 10 new orders each time. Each processing is a mixed read write transaction. The frequency of the delivery transaction is relatively low and accounts for about 4% of total transactions. The delivery transaction is implemented through a delayed queue. The other four transactions are implemented through interactive modes.

• **Stock-Level.** The Stock-Level transaction analyzes whether the inventory status for recent sales of the products (the latest 20 selling products) is below a given threshold and also calculates the quantity of the commodity below that threshold. It is a read-only database transaction, and its percentage in the TPC-C evaluation test is about 4%.

Table 1 shows the data operation results of five types of transactions on the TPC-C database tables. The transactions of New-Order need to operate on almost all of the database tables and it is a mixed read write transaction. According to the TPC-C standard, a certain percentage of New-Order and Payment transactions should access remote database tables randomly, which will reduce the locality of data access to simulate real-life conditions. Both Order-Status and Stock-Level are read-only transactions.

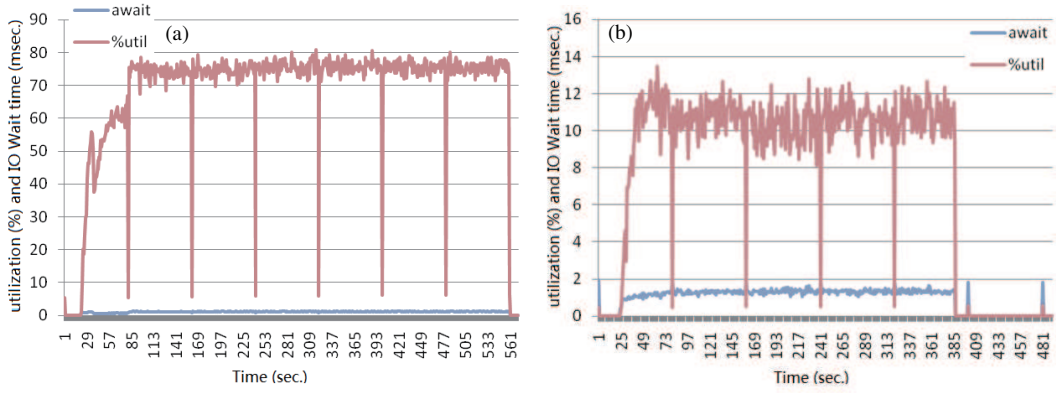

Figure 3 Disk utilization for the data partition and the log partition. (a) The data partition; (b) the log partition.

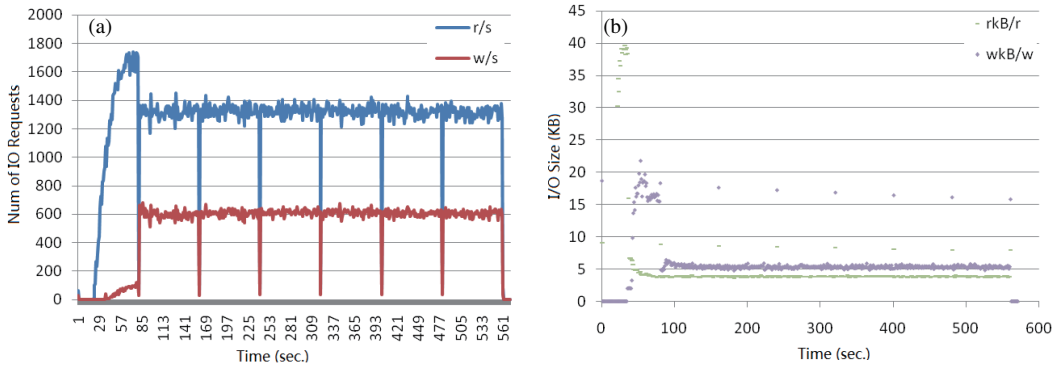

Figure 4 The data access patterns for the data partition. (a) I/O access frequency; (b) the block size for each I/O operation.

4.2 TPC-C data access pattern

In this section, we will analyze the data access pattern of the TPC-C workload. In the following analysis, the data in the TPC-C test were only stored on a disk array, which contained 24 SLC SSD disks and 2 disk controllers, A and B. The redo and undo log files were stored on 8 SSD disks, and the data files and index files were stored on 10 SSD disks. Controller A was used to manage log files and controller B was used to manage the data files and index files. The database server has 4 Itanium processors and 128 GB of memory. We perform a small-scale TPC-C test to analyze main characterization of the TPC-C workload and further present a large-scale TPC-C test in Subsection 6.5.

Figure 3 shows the disk utilization for the data partition and the log partition. It can be seen that the disk utilization of the data partition is about 80%, whileas the disk utilization of the log partition is only about 12%. This indicates that the TPC-C workload is an I/O-intensive OLTP application. In the following analysis, we focus on the data access patterns for the data partition.

Figure 4(a) shows the I/O access frequency for the data partition. We can see that the I/O access frequency is relatively high for the data partition. There are about 1300 reading operations per second on the data partition, and 600 writing operations per second. The number of reading operations is about twice that of writing operations.

Figure 4(b) shows the distribution of the read/write block size for each I/O operation and the I/O volume on the data partitions. From the figure, we can see that the average I/O read size is about 4 kB and that the average I/O write size is about 5 kB; this block size is relatively small. There is about 4 MB of data per second read from the disk array and 3 MB of data per second written into the disk array.

In summary, the data access frequency on the disk partitions is relatively high. Most of the accessed

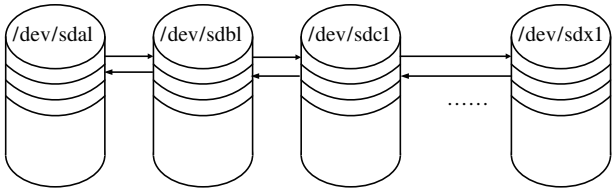

Figure 5 SmallFile table space-based storage management.

data blocks are small, ranging from 4 kB to 5 kB. The TPC-C workload is a mixed read write operation application, but the reading operations account for most of the I/O operations. The average number of reading operations is twice the number of writing operations.

5 Optimization strategies

5.1 SmallFile table space based storage management

Since the TPC-C test requires a large amount of testing data and a large number of storage devices, it is critical to effectively manage and organize these to get optimal TPC-C test results. Logical volume manager (LVM) [21] is widely used in Linux systems to manage a number of physical disks, and it organizes the underlying physical disk partitions using a hierarchical method. The main modules in the LVM architecture are physical volumes, volume groups, and logical volumes. LVM can organize a group of physical disk partitions into a single large logical partition. Users can create their own logical partitions and file systems under LVM volume groups, and they can also rename and repartition the logical partition according to their needs. However, the management functions of LVM are implemented in a software layer. This can result in large performance overhead during the TPC-C test.

To reduce the system overhead caused by disk management software, we propose using raw devices to manage the underlying disk partitions. Raw devices are also called raw partitions, which are special character devices. Applications perform read/write operations directly from the raw partitions without going through the file systems. Using raw devices, data can be directly sent from disk partitions to database buffers, skipping the operating system layer. As a result, we can effectively improve the data access efficiency.

To effectively leverage the parallelism of the underlying disk arrays and improve the I/O access efficiency, we propose a new method to manage the underlying storage devices, called SmallFile table space. We created the small partitions of the same size on all the disk arrays and used the round-robin method to build the database table spaces on these small partitions. The partition size that we used was 8 GB.

As shown in Figure 5, our method has two main advantages: (1) Using the round-robin method to create the table spaces on all the disk partitions, the parallelism of the underlying storage arrays can be fully exploited and (2) the database table spaces can be distributed uniformly on all the disk arrays. This will allow all the disk partitions to achieve better load balance during the TPC-C test.

5.2 NOOP-based I/O scheduling for SSD devices

There are a large number of disk I/O requests during the TPC-C test, and so the efficient scheduling of I/O requests can greatly improve associated performance. Currently, the default disk scheduling strategy on Linux systems is completely fair queuing (CFQ) [22]. CFQ creates separate queues to manage all the I/O requests generated by different processes. CFQ uses a time slice mechanism to guarantee that each process can be assigned proper I/O bandwidth. The length of the time slice for each process and the number of I/O requests allowed in each process depend on its priority.

CFQ scheduler is efficient on systems running multiple tasks that require equal access to I/O resources. However, it can create a bottleneck when used in the TPC-C test, which has a strict deadline for each transaction. To achieve fair scheduling, each process is assigned the same time slot in CFQ, but in the TPC-test, what is actually needed is a short response time and a fast I/O processing speed. Also, the I/O

Table 2 The number of memory pages compared with common pages and huge pages

Server name	Architecture	Memory size (GB)	Number of common pages	Number of huge pages
DELL T710	X86_64	72	9×2^{21}	9×2^{12}
Itanium server	IA64	256	1×2^{26}	1×2^{10}
Intel Westmere server	X86_64	1024	1×2^{28}	1×2^{19}

processing time of the underlying SSD storage devices is very short, and therefore, complex scheduling algorithms can lead to higher processor overhead [23].

To improve the response time of each I/O process and reduce the processor overhead caused by the complex scheduling algorithm, we propose using the NOOP scheduling mechanism to manage all the I/O requests. NOOP submits all the I/O requests to the underlying storage system in accordance with the principle of first in first out (FIFO). At the same time, the NOOP scheduler also merges adjacent I/O requests based on FIFO, thereby not completely satisfying the FIFO principle.

To keep the fairness for different tasks, the TPC-C clients can guarantee that each task generates equal I/O requests for the evaluated server. We do not need extra fairness mechanism in the disk scheduling. In this study, we compared the performance of both the NOOP scheduler and the CFQ scheduler on SSD storage devices and found that the NOOP scheduler was better than the CFQ scheduler in terms of the system overhead and I/O response time. There is not any fairness issue in our TPC-C implementation.

5.3 Huge-page-based memory system optimization

The TPC-C workload can consume a large amount of memory resources. For example, the TPC-C test on Oracle's SPARC Super Cluster can use around 13.5 TB in memory resources [5]. On current Linux systems, the default page size is 4 kB. According to the management mechanism of the virtual memory, a smaller page size can result in a larger number of page numbers for a given program [24]. For a memory-intensive application, frequent translations between virtual addresses and physical addresses can significantly increase the processor overhead. At the same time, a smaller page size means that many more translations between virtual addresses and physical addresses must reside in the system, which will increase the TLB miss rate and cause the application's performance to decrease further.

To reduce the processor and memory overhead and to also reduce the TLB miss rate, we propose using the huge page technique during the TPC-C test. In Linux kernel 2.6, the maximum page size on the x86_64 platform is 2 MB, which is about 500 times the size of the default page. For the Itanium platform, the maximum page size is 256 MB. Therefore, the huge page technique can reduce the number of pages in the TPC-C test and effectively increase the TLB hit rate. Table 2 shows the comparative number of pages when using common pages versus huge pages across three different servers. It can be seen that the huge page technique can effectively reduce the number of actually used memory pages.

5.4 Locality-aware interrupt mapping

During the TPC-C test, since database transactions require frequent access of disk arrays, large interrupt requests are generated by the I/O devices. Also, a large number of clients commit database transactions through network interfaces, so the database server will receive a number of network interrupt requests. During the TPC-C test on the Itanium server (when the row number of the "warehouse" table was 16000), we recorded the number of interrupt requests per second processed by the database server. The results show that the system processes about 140000 interrupt requests per second from the I/O devices and network interfaces.

A large amount of interrupt requests can not only interfere with the execution of user processes but also affect TPC-C test performance. In the Linux system, the interrupt scheduling strategy dispatches all the interrupt requests uniformly to each processor core. However, we find that this strategy does not take into account the locality of the interrupt requests and possible interference with user processes.

We propose a locality-aware interrupt mapping strategy to handle the above problems. Our strategy is described as follows: First, we map the interrupt requests from different devices to particular processor

Table 3 The configurations of database servers used in the TPC-C test

Server name	Processor	# of cores	Mem size	OS	Storage array	Fiber card
Itanium server	4 Intel Itanium 9350	16	256 GB	Redhat EL 5.3	8 sets of disk arrays	4 8GB fiber cards
Intel Westmere server	4 Intel Westmere-EX	40	1024 GB	Redhat EL 6.1	11 sets of disk arrays	4 8GB fiber cards

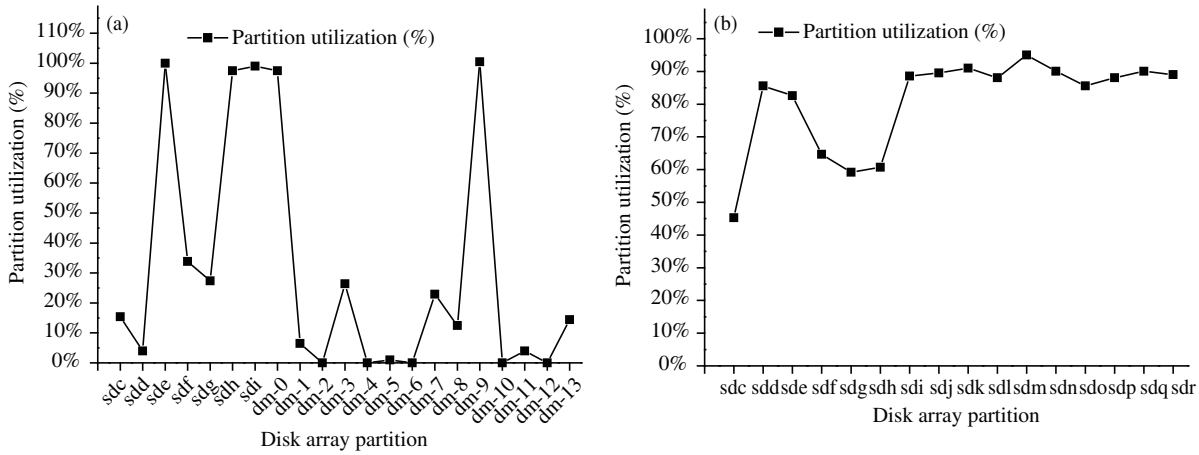

Figure 6 The disk partition utilization for both BigFile mode and SmallFile mode. (a) BigFile mode; (b) SmallFile mode.

cores separately, thus the interference resulting from interrupts from different devices can be reduced; Second, according to the characteristics of the NUMA system, we map the interrupts to their closest processor cores; And lastly, we map the interrupt requests to the idle processor cores as much as possible to reduce interference with critical user processes. From our experimental results, it can be concluded that we can effectively improve the server interrupt processing efficiency using the above methods.

6 Evaluation

In this section, we will report on a series of experiments conducted to verify the performance of the previously mentioned optimization strategies on real systems.

6.1 Experimental platforms

Here, we give a detailed description of our experimental platforms. The clients and driver systems of the TPC-C test were deployed on the same servers in our evaluation. The clients and driver systems used two-way Intel Xeon X5650 processors, with a memory size of 32 GB. The network was Gigabit Ethernet. There were two database servers in our evaluation, and their detailed configurations are shown in Table 3. Each disk array included 24 SLC SSD disks and each disk drive was 60 GB. The database used in our servers was Oracle 10g.

6.2 Storage system optimization

We set up two storage systems for the TPC-C test. In the first storage environment, we used LVM to manage the underlying disk partitions, and Oracle BigFile to organize all the data files. Each table space in the Oracle database was allocated to a set of disk arrays according to the size of the database tables (we use BigFile to denote this mode in the figures). In the second storage environment, we used our proposed SmallFile table space method to manage the underlying disk arrays. Each disk array was partitioned into a number of 8 GB blocks. We used the round-robin method to establish table spaces on all the small blocks as described in Subsection 5.1 and managed the underlying storage devices using the raw devices approach (we use SmallFile to denote this mode in the figures).

Figure 6 shows the disk partition utilization for both BigFile mode and SmallFile mode during the TPC-C test. For BigFile mode, the disk utilization shows great inconsistency. The utilization of disk

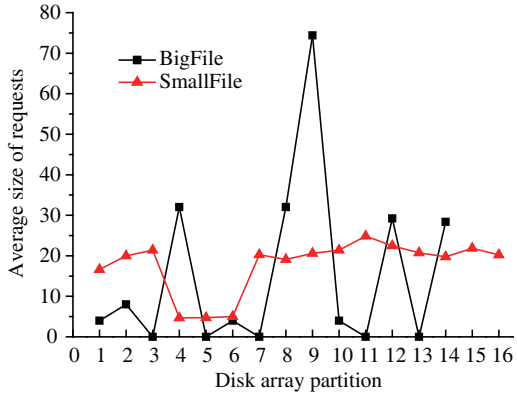

Figure 7 (Color online) The average number of I/O requests of each disk partition for both storage modes.

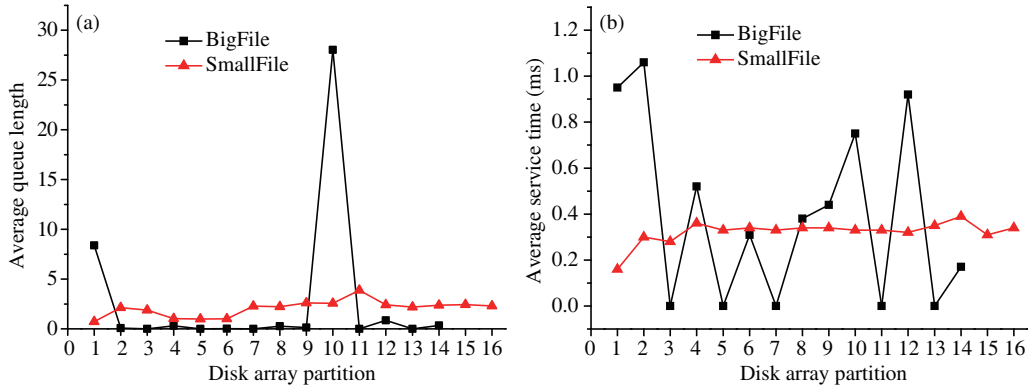

Figure 8 (Color online) The average I/O queue length (a) and service time (b) of each disk partition for both storage modes.

partitions ranges from 5% to 100%, for which it is difficult to make full use of all the underlying disk arrays, and which can lead to relatively large latency when reading data from certain disk arrays. In our proposed storage system, the utilization of all the disk partitions is well balanced. The utilization for most of the disk partitions was about 90%, and only the utilization of the sdf, sdg, and sdh disk partitions was relatively low. The sdc partition was used to store redo logs in the Oracle database and experiences only sequential write operations, which is why its utilization was relatively low.

Figure 7 shows the average number of I/O requests in both storage modes. We can see that for the BigFile mode, the number of I/O requests from each disk partition is non-uniform, ranging from 0 to 75. However, for the SmallFile mode, the number of I/O requests from all the disk partitions is well balanced. The average number of I/O requests in SmallFile mode was about 20.

Figure 8 shows the average I/O queue length and the average I/O service time of each disk partition for both storage modes. We can see that the SmallFile mode outperforms the BigFile mode in both aspects. For example, the average service time for the SmallFile mode was about 0.3 ms, whereas for the BigFile mode, it ranged from 0.01 to 1.06 ms.

Note that the average queue length of disk partition 10 in the left part of Figure 8 is about 10 times larger than those of other disk partitions. This is because the BigFile mode cannot uniformly distribute data on all the disk partitions, the disk partition with relatively high user requests will generate a longer waiting queue. Based on our analysis, the BigFile mode would randomly cause a long waiting queue for each disk partition. In contrast, the SmallFile mode can use all the disk partitions uniformly.

With our proposed storage optimization strategy, the disk I/O requests can be uniformly distributed between all the disk partitions. Each disk partition demonstrated good performance in terms of the average I/O queue length and the average I/O service time. The performance of underlying disk storage arrays can be maximized using our method.

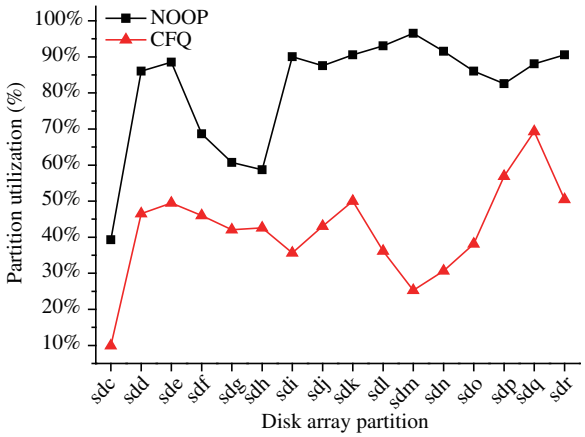

Figure 9 (Color online) The disk partition utilization for different scheduling strategies.

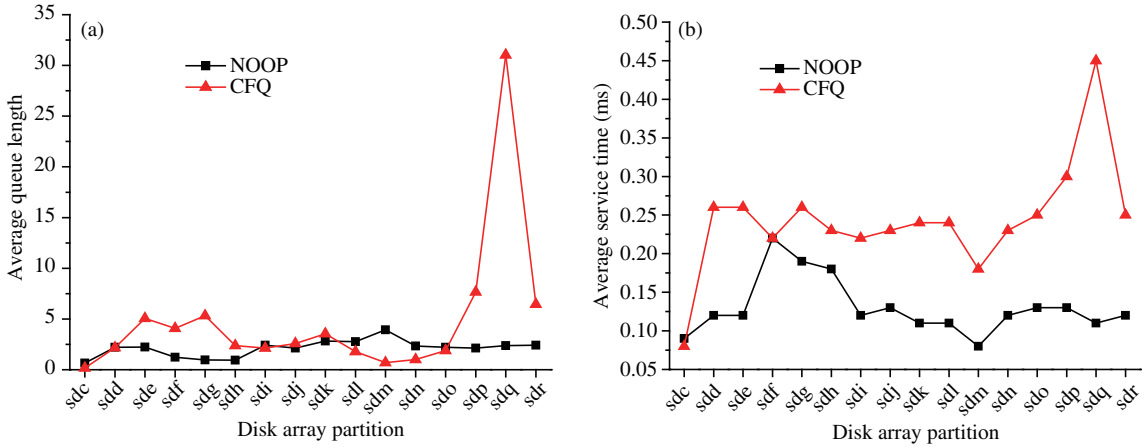

Figure 10 (Color online) The average I/O queue length (a) and service time (b) for the different scheduling strategies.

6.3 I/O scheduling optimization

In this section, we analyze the impact of both CFQ and NOOP I/O schedulers on TPC-C performance. We performed the TPC-C test on an Itanium server, and the row number of the warehouse table was 16000. Except for the I/O scheduling strategies, the other parameters of the operating systems remained the same for both configurations.

Figure 9 shows the disk partition utilization for both CFQ and NOOP scheduling strategies during the TPC-C test. We can see that the average disk partition utilization was about 40% and the maximum was 69.3% for the CFQ scheduling strategy, whereas for the NOOP scheduling, the disk partition utilization was 80% on average and the maximum was 96.5%. In summary, the NOOP scheduling strategy can effectively improve the utilization of each disk partition.

Figure 10 shows the average I/O queue length and service time of each disk partition for the two scheduling strategies. We can see that the I/O queue length was too long for some disk partitions using the CFQ scheduling strategy. It can also be seen that long I/O queues randomly appeared on different disk partitions during the test process. This is mainly because the CFQ strategy needs to submit a group of I/O requests to the disk devices according to the time slice. As a result, in a short time, the disk partitions accumulate a number of disk I/O requests. The average queue length for the NOOP scheduling strategy was about two, which is much shorter than the CFQ strategy. Also, the average service time for the NOOP scheduling strategy was 0.13 ms, but was 0.24 ms for the CFQ strategy.

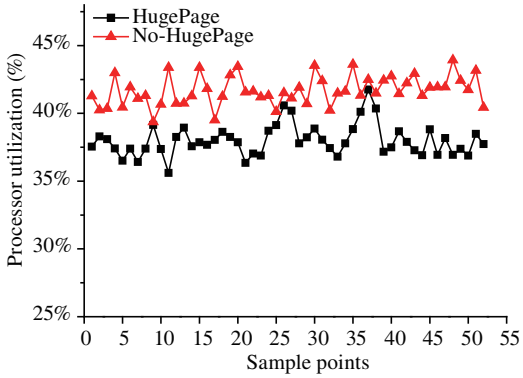

Figure 11 (Color online) Processor utilization for different page management strategies.

6.4 Memory optimization

We analyzed the performance of different memory optimization strategies on the Itanium server. The row number of the warehouse table in the TPC-C test was 16000. On the Itanium platform, the huge page was 256 MB and we used 990 huge pages in total. Figure 11 shows the processor utilization in the user mode for different page management strategies. Using the huge page technique, the average processor utilization for user processes was 37.8%. When the huge page was disabled, the average processor utilization increased to 42.1%. In our system, the huge page optimization technique can reduce processor utilization by an average of 4.3%. As a result, the system TPC-C performance can be improved by saving processor resources.

When we analyzed the completion time for each TPC-C transaction, we found that the huge page technique can effectively reduce the processing time of each transaction. The experimental results show that the average processing time for each transaction was less than 1 s when we took advantage of the huge page technique, but was more than 1 s when it was disabled.

6.5 TPC-C results

Lastly, we investigated the TPC-C results on two high-end computers when all of the above optimization methods were used.

Figure 12 shows the TPC-C test results on the Itanium server. This server includes four Itanium processors with 256 GB of memory and eight sets of disk storage arrays. When the row number of the warehouse table was 24000, the TPC-C test reached its maximum performance of 291000 TPMC. The processor utilization in user mode, kernel mode, and iowait was about 56%, 23%, and 14%, respectively. Also, idle processor resources were about 7%. The main bottleneck of this server was memory latency. Through analyzing the Oracle Automatic Workload Repository report, we found that the hit rate of the latch hit event was very low for the Oracle database.

Figure 13 shows the TPC-C test results for the Intel Westmere server, which is a 4-way Xeon server with 1 TB of memory and 11 sets of disk storage arrays. When the row number of the warehouse table was 85700, the TPC-C test reached its maximum performance of 1.018 million TPMC.

Figure 14 shows the average number of I/O requests per second for each disk partition on the 11 disk arrays during the TPC-C test on the Intel Westmere server. We found that the average number of read requests per second was about 10000, and that the average number of write requests per second was about 5000. The number of read requests was twice the number of write requests. The disk partition sdc was used for redo logs in the Oracle database and there were only sequential write requests for this partition.

The processor utilization of the Intel Westmere server in user mode, kernel mode, and iowait was about 55%, 19%, and 18%, respectively. Idle processor resources were about 8%. The TPC-C performance on this server can be improved further. Figure 15 shows the disk partition utilization of the Intel Westmere server. We can see that the distribution of the disk utilization of each disk partition using our proposed

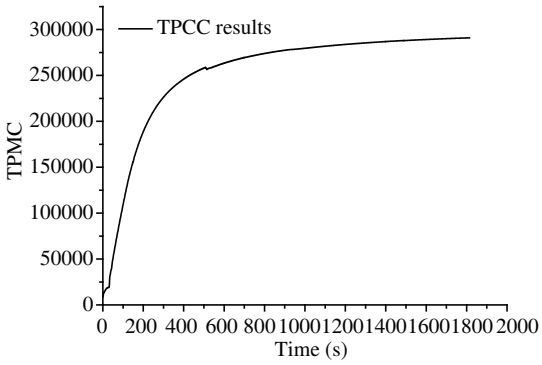

Figure 12 TPC-C test results for the Itanium server (warehouse = 24000).

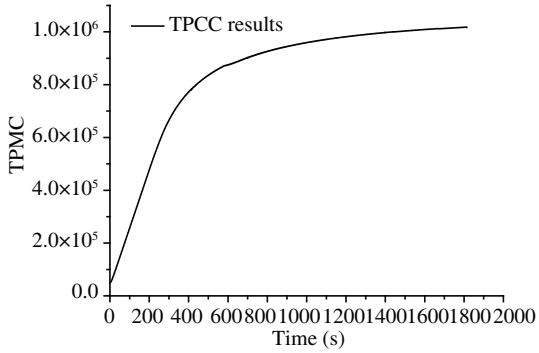

Figure 13 TPC-C test results for the Intel Westmere server (warehouse = 85700).

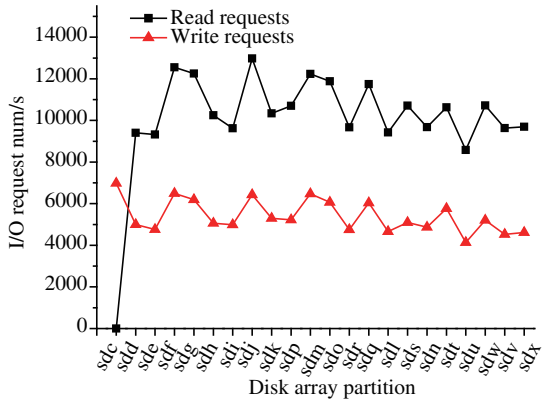

Figure 14 (Color online) I/O requests number per second for each disk partition on the Intel Westmere server.

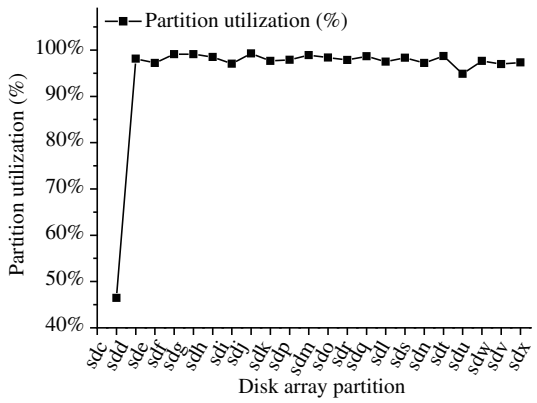

Figure 15 The utilization of each disk partition on the Intel Westmere server.

SmallFile table space method was very uniform. Note that the partition sdc was used to store log files in the Oracle database, and so the utilization for this partition was relatively low.

7 Conclusion

In this paper, we designed and implemented a TPC-C benchmark based on the latest TPC-C standard specification and built a large-scale TPC-C evaluation environment using SSD arrays. Through analyzing the TPC-C workload characteristics, we proposed a series of system-level optimization methods and performed the TPC-C test on two high-end computers. The experimental results show that our proposed methods can effectively improve TPC-C performance, with the performance of the Intel Westmere server, for example, reaching 1.018 million TPMC. In summary, we make the following key conclusion.

- Effective data distribution is a key factor for a large-scale TPC-C test. Due to a large amount of test data stored in storage devices, uniformly partitioning all the data on the storage devices is necessary. The SmallFile table space-based approach can effectively organize and use underlying disk arrays.
- A NOOP-based disk scheduling mechanism is more effective than a complex one, such as CFQ, to reduce processor overhead and improve the average service time for each I/O process in the TPC-C test.
- For an application using a large amount of memory, the huge page technique can effectively improve processor utilization, as well as the TLB hit rate.
- On NUMA systems, a fine-grained interrupt mapping strategy is necessary to reduce interrupt interference. We need to adjust interrupt mapping according to both the locality of interrupt requests and the asymmetry characteristics of NUMA systems.

What is more, in-memory processing system [25] has become more popular recently, but TPC-C is an I/O intensive benchmark. According to the TPC-C specification, it is very difficult to load all the data into memory on current commodity servers. For example, there are about 8 TB of TPC-C data on our Intel Westmere server whose memory size is already 1 TB. Although we can use more servers for the TPC-C test, the performance of current distributed database system needs to be significantly optimized.

Acknowledgements This work was supported by National High Technology Research and Development Program of China (863) (Grant No. 2013AA01A213), National Natural Science Foundation of China (Grant No. 61472201, 61170008), and Tsinghua University Initiative Scientific Research Program.

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Gostin G, Collard J F, Collins K. The architecture of the HP superdome shared-memory multiprocessor. In: Proceedings of the 19th Annual International Conference on Supercomputing, Cambridge, 2005. 239–245
2 Hsu W W, Smith A J, Young H C. Characteristics of production database workloads and the TPC benchmarks. IBM Syst J, 2001, 40: 781–802
3 Transaction Processing Performance Council. TPC Benchmark C, Standard Specification Version 5.11. http://www.tpc.org. 2010
4 Henning J L. SPEC CPU2000: measuring CPU performance in the new millennium. Computer, 2000, 33: 28–35
5 Transaction Processing Performance Council. TPC-C results by performance. http://www.tpc.org. 2013
6 Shaw S. HammerDB Installation and Troubleshooting Guide Version 2.5, 2013
7 Llanos D R. TPCC-UVa: An open-source TPC-C implementation for global performance measurement of computer systems. ACM SIGMOD Record, 2006, 35: 6–15
8 Wong M, Meredith M E. Open Source Development Labs Database Test 2 User Guide Version 0.21, 2002
9 Quest Software Inc. Benchmark Factory for Databases User Guide Version 6.9.2, 2013
10 Clark M. Installation and Configuration Guide for Orabm and Orastress Version 2.1, 2006
11 PostgreSQL Global Development Group. PostgreSQL 7.1 Reference Manual, 2001
12 Delimitrou C, Sankar S, Khessib B, et al. Time and cost-efficient modeling and generation of large-scale TPCC/TPCE/TPCH workloads. Topics in Performance Evaluation, Measurement and Characterization. Berlin: Springer, 2012, 7144: 146–162
13 Barham P, Donnelly A, Isaacs R, et al. Using magpie for request extraction and workload modelling. In: Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI), San Francisco, 2004, 4: 18
14 Kim S H, Jung D, Kim J S, et al. HeteroDrive: reshaping the storage access pattern of OLTP workload using SSD. In: Proceedings of 4th International Workshop on Software Support for Portable Storage (IWSSPS), Seoul, 2009. 13–17
15 Chen S, Ailamaki A, Athanassoulis M, et al. TPC-E vs. TPC-C: characterizing the new TPC-E benchmark via an I/O comparison study. ACM SIGMOD Record, 2011, 39, 5–10
16 Ash S M, Lin K I. Optimizing database index performance for solid state drives. In: Proceedings of the 18th International Database Engineering Applications Symposium. New York: ACM, 2014. 237–246
17 Yao J H, Ng A, Chen S P, et al. A performance evaluation of public cloud using TPC-C. In: Service-Oriented Computing-ICSOC Workshops. Berlin: Springer, 2013, 7759: 3–13
18 Zhai J, Chen W, Zheng W. Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node. ACM Sigplan Notices. 2010, 45: 305–314
19 Chen Y P, Raab F, Katz R. From TPC-C to big data benchmarks: a functional workload model. Specifying Big Data Benchmarks. Berlin: Springer, 2014, 8163: 28–43
20 Tozun P, Pandis I, Kaynak C, et al. From A to E: analyzing TPC's OLTP benchmarks: the obsolete, the ubiquitous, the unexplored. In: Proceedings of the 16th International Conference on Extending Database Technology, Genoa, 2013. 17–28
21 Teigland D, Mauelshagen H. Volume Managers in Linux. In: Proceedings of USENIX Annual Technical Conference, FREENIX Track, Boston, 2001. 185–198
22 Seelam S R, Teller P J. Fairness and performance isolation: an analysis of disk scheduling algorithms. In: IEEE International Conference on Cluster Computing, Barcelona, 2006. 1–10
23 Iyer S, Druschel P. Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O. In: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Banff, 2001. 117–130
24 Raina S. Virtual Shared Memory: A Survey of Techniques and Systems. University of Bristol Technical Report, 1992
25 Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, Berkeley, 2010. 10