Characterization of the E-commerce Storage Subsystem Workload

Xi Zhang
The Ohio State University
Columbus, OH 43210
zhanxi@cse.ohio-state.edu

Alma Riska
Seagate Research
Pittsburgh, PA 15222
alma.riska@seagate.com

Erik Riedel
Seagate Research
Pittsburgh, PA 15222
erik.riedel@seagate.com

Abstract

This paper characterizes the workload seen at the storage subsystem of an e-commerce system. Measurements are conducted on multi-tiered systems running three different benchmarks, i.e., TPC-W, TPC-C, and RUBiS. In this environment, TPC-W and RUBiS are used to represent web-based e-commerce applications (i.e., on-line shopping and auctioning). They generate mostly READ-dominated workloads. The TPC-C benchmark, although not directly an e-commerce benchmark, is used to represent an e-commerce system under heavy on-line transactions processing activity. Different from the TPC-W and RUBiS benchmarks, TPC-C generates WRITE-dominated workloads. For all three benchmarks, in addition to the system load, the workload mix causes the system resources such as memory to saturate, IO traffic to increase, and, consequently, overall system throughput to reduce. Generally, when a workload shifts from web-site browsing (i.e, reading) to transaction processing (i.e writing) the IO load reduces but the footprint of the IO working set increases, which slows down the IO subsystem. File system and device driver scheduling represent elements in the IO path that for a given set of system resources further improve user-level throughput. Their impact is visible for medium to high utilization and diminishes for light load or overload.

1 Introduction

In the last few years, the world wide web has evolved from a global information repository into an ubiquitous interface for people to interact and conduct daily activities. Accordingly, the popularity of on-line e-commerce sites, such as Amazon and E-Bay, has increased rapidly and their performance represents one of the most important factors for their success. Therefore, it becomes critical to understand the characteristics of their workloads and their implications on overall system performance for effective resource management and capacity planning.

Typically an e-commerce site is supported by a multi-tiered system, which consists of a front-end web server, an application server, and a back-end database server. The database server stores all the data made available through the e-commerce site and executes all the queries necessary to dynamically generate the information requested by the clients. A critical component of the database server is its storage subsystem whose behavior under the e-commerce workloads is the focus of this work.

Our evaluation of the IO subsystem behavior in a multi-tiered architecture supporting e-commerce applications is based on measurements conducted on three benchmarks that emulate a range of e-commerce services. Specifically, we deployed TPC-W [16], an on-line bookstore (such as Amazon.com), RUBiS [1], an on-line auction site (such as Ebay.com), and TPC-C [16], an on-line transaction processing service such as inventory systems or banking. Although an OLTP service such as the one emulated by TPC-C qualifies only *indirectly* as an e-commerce application, we include it in our evaluation to represent e-commerce applications during periods of high transactions (i.e. ordering and bidding type) volumes. Our intention is to emulate the normal operation of an e-commerce site with the TPC-W and RUBiS benchmarks and the occasional high activity periods with the TPC-C benchmark. As a result, our work is different from previous ones [8, 7] that evaluate TPC-C with the goal of understanding the behavior of production databases where the normal operation is characterized by heavy online-transaction processing.

Although we conduct measurements throughout the IO path of the last tier (i.e., database one) of an e-commerce system, we focus only on the general block-level characterization of the workload, i.e., well below the file system, where the application semantics are not available. Our goal is to understand what can be done at the IO subsystem to optimize for such workloads in terms of request scheduling, request merging, or utilization of idle times. As such, the evaluation presented in this paper is different from those that focus on evaluating IO behavior under any of the above benchmarks with the goal of addressing a specific feature

978-0-7695-3360-5/08 $25.00 © 2008 IEEE
DOI 10.1109/QEST.2008.44

either in the IO path or somewhere above it [3, 6].

Our measurements indicate that the memory size of the database server (relative to the database size) largely determines the load seen by the IO subsystems under e-commerce workloads. Nevertheless, there are general observations that can be drawn. An important observation is that e-commerce IO workloads are dominated by READs rather than WRITEs. This is an outcome of the browsing activity carried by the majority of web site users (rather than purchasing). Browsing generates more IO activity than ordering or bidding and utilizes more the IO subsystem. However, as overall user activity switches from browsing to ordering/bidding, the footprint of the IO working set increases because the e-commerce supporting databases are design to have tables that handle exclusively ordering/bidding activity and they are rarely accessed if users browse. In general, IO workload randomness increases as user activity switches from browsing to ordering/bidding. Yet, under e-commerce applications, locality of WRITEs is higher than locality of READs.

The choice of the file system and the IO scheduler impacts mostly cases when the IO subsystem is under heavy load. If the IO subsystem operates under light load or overload then the user-level performance gap between different file systems and IO schedulers is minimal. In the cases that matter, ReiserFS gives the best performance among the three file systems evaluated and the Deadline IO scheduler performs best among the IO schedulers evaluated.

The rest of the paper is organized as follows. Section 2 discusses related work. In Section 3, we describe the benchmark specifications, the evaluated workloads, as well as our experimental environment. Section 4 gives the user-level performance of the benchmarks under different loads and workloads. Section 5 discusses IO workload characteristics under e-commerce applications. We conclude our paper with Section 6, where we summarize our results.

2 Related work

With the ubiquitous deployment of e-commerce services that support daily commercial and personal activities, the complexity of the underlying systems has increased and various studies have been conducted to understand the main characteristics of such systems [2, 13, 22, 21]. Better understanding of the behavior and workload in e-commerce systems has resulted in new and more effective resource management policies that are tailored for such applications and systems [4, 11, 12, 14].

Because it is difficult to obtain data from real e-commerce sites, one can only resort to synthetic workload generators to study such systems, with the most prominent ones being the benchmarks of the Transaction Processing Council [16], such as TPC-C and TPC-W, respectively, modeling an OLTP (on-line transaction processing) database server and an on-line bookstore. Although an OLTP service such as the one emulated by TPC-C qualifies only *indirectly* as an e-commerce application, it can be used to represent e-commerce applications during periods of high transactions (i.e. ordering and bidding type) volumes. Other benchmarks are proposed to model on-line e-commerce services such as RUBiS [1] that models an on-line auction server. Various workload characterization studies are based on measurements on these benchmarks [5, 20, 8, 7, 3, 6] with some further generalizing the behavior via analytic models [18].

Our work differs from previous ones, because our focus is on the detailed characterization of the block-level workload in the underlying system that supports e-commerce applications rather than general understanding of such systems workloads as in [1, 5, 21]. We stress that the databases supporting e-commerce systems differ from the production-level ones on the way that they are accessed from the users. As a result, the IO e-commerce workloads are READ dominated and the IO online transaction processing workloads are WRITE dominated. Our work explains the difference and uses the TPC-C benchmark to capture occasional heavy load situations on e-commerce sites. Consequently, our work differs from previous work that focuses solely on the behavior of the TPC-C benchmark and OLTP workloads [8, 7, 3, 6].

3 Measurement System

3.1 Benchmarks Evaluated

In this paper, three benchmarks are evaluated; TPC-W and RUBiS capturing the average behavior and TPC-C capturing the occasional heavy-transaction behavior in an e-commerce site. The benchmarks themselves define how clients interact with the system and the database structure that supports a specific e-commerce application.

Interaction	Browsing	Shopping	Ordering
Browse	95%	80%	50%
Order	5%	20%	50%

Table 1. Frequencies of the various Web interactions for the TPC-W workload mixes

TPC-W is a transactional benchmark that models an online bookstore. There are 14 Web interactions in TPC-W classified as either *browsing* or *ordering*. The browsing interactions mostly read data from the database while the ordering interactions read and write data to the database. Three different workload nixes, i.e., browsing, shopping,

and ordering are defined in TPC-W. Table 1 gives the percentage of the two main Web interaction types for each workload mix.

Interaction	Browsing	Bidding	High Bidding
Browse	100%	85%	60%
Bid	0%	15%	40%

Table 2. Frequencies of the various Web interactions for the RUBiS workload mixes

RUBiS benchmark [15] models the core functionalities of an auction site where browsing, bidding, and selling of auction items are possible. Items for sale in an auction site are available for bidding only within a pre-defined time period. RUBiS's web interactions are classified as either browsing or bidding. Browsing interactions mostly read from the database, while bidding interactions read and write into the database. The RUBiS benchmark defines only two workload mixes, i.e., browsing and bidding. For our experimental purpose, we also define a third workload mix (i.e., high bidding) where the bidding rate is even higher than in the default bidding workload, as shown in Table 2.

Transactions	Standard Mix	High Write Mix
New Order	45%	45%
Payment	43%	7%
Order Status	4%	4%
Delivery	4%	40%
Stock Level	4%	4%

Table 3. Frequencies of the transactions in the TPC-C workload mixes

TPC-C benchmark models a medium complexity online inventory system which facilitates warehouse items ordering and delivery, as well as the maintenance of the items stock level. There are five transaction types in TPC-C. The *new order* placing transaction, the *payment* transaction, and the *delivery* transaction read and write into the database. The *order status* and the *stock level* transactions only read from the database. The original benchmark defines a standard workload mix. We define a second mix with more database writing activity than the standard workload mix. We refer to this new workload mix as *"High Write"*. Table 3 gives the transaction percentage for each of the TPC-C workloads.

3.2 Experimental Configuration

TPC-W and RUBiS are set-up as multi-tiered systems with a few clients accessing a Web server (Apache 2.0), which forwards all the traffic to an application server (Tomcat 4.1), and down to the database server (MySQL 4.1). The TPC-C benchmark, although it can be used in a similar setting, is implemented as a single-tier benchmark, i.e., clients communicate directly with the database server via SQL queries. The hardware used in our experimental set-up is described in Table 4.

We use openly available benchmark distributions. Specifically, we use the TPC-W distribution from the PHARM project at University of Wisconsin [10], RUBiS distribution from Rice University [15], and the TPC-C distribution from OSDL [17]. Since our focus is the storage subsystem behavior, then we scaled each benchmark such that the underlying databases do not fit entirely into the available memory of the database server. Hence, our focus is on the out-of-memory databases, which characterize Web sites with large inventory of available items. The sizes of our databases range from more than 512MB to 2GB (see Tables 5 6 7). Given that our available memory in the database server is only 512MB (see Table 4), it results that the database sizes are from 1 to 4 times the size of the available memory. Similar studies, for even larger databases, have also maintained a ratio of 1:1 to 1:4 between memory and database size. Consequently, we consider our set-up a realistic one.

Table	Cardinality	Row Size	Table Size
Customer	1,446,126	260 Bytes	364 MB
Address	2,886,116	154 Bytes	301 MB
Orders	1,296,000	73 Bytes	85 MB
Order Line	3,888,203	132 Bytes	338 MB
CC Trans.	1,296,000	80 Bytes	106 MB
Items	1,000,000	520 Bytes	510 MB
Authors	250,000	370 Bytes	87 MB
Country	92	70 Bytes	3.1 KB

Table 5. Sizes of TPC-W database tables

The TPC-W database has eight tables, i.e., customer, address, order, order line, credit card transactions, item, author, and country. TPC-W defines how the sizes of these tables scale relative to each-other. We opt to emulate a Web site with 1,000,000 items in stock which results in a database size of 1.7GB. The table sizes in the TPC-W database are given in Table 5.

The database in RUBiS consists of seven tables i.e., users, items, bids, buy_now, comments, categories, and regions. Similar to TPC-W, the benchmark defines the relative scaling between the tables in the database. The database size is 0.5 GB and the table sizes are shown in Table 6.

The TPC-C database consists of nine tables: customers, items, new_order, orders, order_line, stock, warehouse, district, and history. The database size is 2.1 GB and the table sizes are shown in Table 7.

Component	Software	Processor	Memory	OS Kernel
Clients	Emulated Web Browser	Pentium 4, 2GHz	256MB	Linux 2.4.20
Application Server	Apache 2.0, Tomcat 4.1	Pentium III, 1.3GHz	2GB	Linux 2.4.18
Database Server	MySQL 4.1	Intel Xeon, 1.5GHz with HT	512MB	Linux 2.6.16
Disk	Seagate ST373453LC, SCSI, 15,000rpm, 73GB			

Table 4. Software and hardware configuration of the experimental environments

Table	Cardinality	Avg. Row Size	Table Size
Users	1,000,974	109 Bytes	104 MB
Items	533,507	3,347 Bytes	177 MB
Bids	3,786,484	33 Bytes	119 MB
Buy now	729	25 Bytes	18 KB
Comments	398,534	292 Bytes	111 MB
Categories	20	26 Bytes	516 B
Regions	62	26 Bytes	1.5 KB

Table 6. Sizes of RUBiS database tables

Table	Cardinality	Avg. Row Size	Table Size
Customer	900,000	608 Bytes	522 MB
Items	100,000	82 Bytes	7.8 MB
New order	89,405	40 Bytes	3.4 MB
Orders	1,110,328	37 Bytes	39 MB
Order Line	10,746,128	72 Bytes	733 MB
Stock	2,975,627	316 Bytes	895 MB
Warehouse	30	113 Bytes	3.3 KB
District	300	120 Bytes	35 KB
History	977,725	58 Bytes	54 MB

Table 7. Sizes of TPC-C database tables

Among all databases that we experimented with, RUBiS is the smallest one, then TPC-W, and TPC-C. Some of the results, presented later in the paper are related to this fact. In particular, the small number of block-level requests for RUBiS, as well as the effect that workload changes have on the IO subsystem behavior.

In this set-up, we instrumented collection of traces from various logs. Specifically, we use

1 - the HTTP clients logs that contain the round-trip times for each web request sent to the server and are used to calculate user-level throughput,

2 - the application server servlet logs that contain the round trip time for each servlet request sent to the database server,

3 - the MySQL server logs that record the response time for each SQL query

4 - `strace` logging of the block-level activity in the IO subsystem of the database server. We use VMWare [19] to host the database server and collect the IO trace by running `strace` on the host system where the IO traffic of the database server appears as a single process thread.

4 Overall system Performance

For our measurements the IO subsystem is configured with its default settings, i.e., the Ext3 file system, the Anticipatory [9] IO scheduler, a maximum queue length of 4 at the disk, and enabled WRITE-caching at the disk. Figure 1 plots the user-level throughput, measured in transactions per minute, as a function of the number of clients (i.e., sessions) in the system, for each benchmark and each workload defined in Subsection 3.1. The results show that the workload mix determines the capacity of the system, i.e., the system has different capacities for different workloads. As a general observation, the browsing workload mix for both TPC-W and RUBiS and the standard workload for TPC-C achieve lower throughput than the other workload mixes in the benchmarks. Under browsing mix the IO traffic is more read-intensive than the other workloads in each benchmark and queries execute over a large set of data which easily saturates the database server memory and its IO subsystem despite the fact that browsing is mostly contained within only a few tables in the database. The workloads with additional WRITE activity, such as ordering and bidding achieve higher throughput because individual queries are executed over small sets of data and the IO subsystem is configured to improve WRITE performance (i.e, the disk has write-cache enabled).

RUBiS benchmark experiences a sharp increase in the user-level throughput as the number of clients increases from 16 to 64 (see Figure 1), because RUBiS's small database mainly fits in the memory and adding more clients increases workload locality and consequently the cache hit ratio. The same effect is not seen in the TPC-W and TPC-C benchmarks where out-of-memory databases experience higher IO traffic as the number of clients increases in the system (flattening the user-level throughput early on).

For the TPC-W and RUBiS benchmarks, where the system is multi-tiered, we calculate the contribution of each tier, i.e., Web server (as seen by the client), application server, and database server in the average user-level request round trip time. Figure 2 presents such decomposition of round-trip times for the three workload mixes of the RUBiS benchmarks as a function of the number of clients (i.e.,

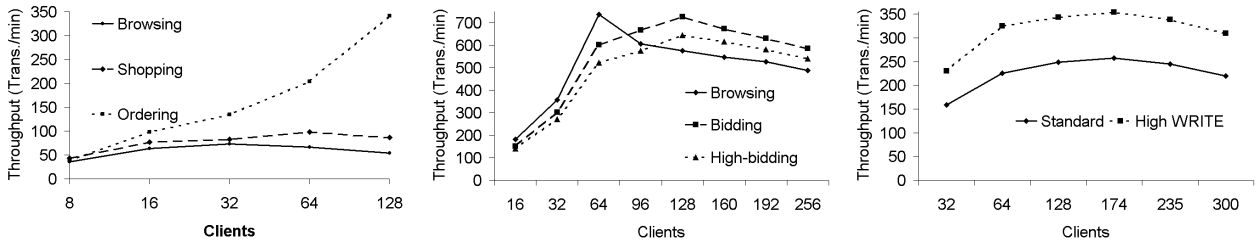

Figure 1. Benchmarks scalability from left to right; TPC-W, RUBiS, and TPC-C.

Figure 2. Decomposition of round trip times among the three tiers for the three RUBiS workload mixes (from left to right; browsing, bidding, and high bidding).

sessions) in the system. Results for TPC-W and TPC-C are qualitatively the same and are omitted here for sake of brevity.

The browsing workload mix for RUBiS utilizes mostly the database server as, in average, every request spends more than 80% of its time at the database server (see the leftmost plot in Figure 2). As the number of clients increases in the system the time spent at the database server waiting for service approaches 100% of the overall round trip time. For the other two RUBiS workloads, i.e., bidding and high bidding mixes (see the middle and the rightmost plots, respectively, in Figure 2), the database represents a lower, but still high, overhead. In particular, for a small number of clients, the application server contributes significantly to the overall round trip time and only when more clients in the system cause the IO subsystem to be the bottleneck, the relative application server overhead is reduced.

The user-level performance of all three benchmarks suggests that system performance depends on the optimization of the database tier where storage is an important component. In the following, we show detailed characterization of the IO subsystem workload under a range of user-level workloads.

5 Storage Subsystem Behavior

An important aspect of storage performance is the access pattern at the block level. It determines how effective various IO optimization techniques such as request merging and reordering are. We record the block (LBA) level trace for each benchmark and workload mix using the `strace` tool and plot the LBA of each IO request as a function of the request issued time. Figures 3, 4, and 5 capture the disk access patterns for the different workload mixes in TPC-W, RUBiS, and TPC-C, respectively, for only one load level (i.e., number of clients in the system).

For the TPC-W benchmark (see Figure 3), the majority of block-level activity is associated with accesses to the items table (i.e., the thickest horizontal band of dots in each plot). The distinction between the different workloads in TPC-W, with regard to the block access pattern, reflects the changes in user activity. Specifically, when switching from browsing (the leftmost plot in Figure 3) to shopping (the middle plot in Figure 3, to ordering (the rightmost plot in Figure 3) the footprint of the working set increases because ordering activities require to access almost all tables in the database which are not commonly accessed with the browsing activities. This causes in-memory data in the shopping and ordering mixes to be evicted faster than under browsing. Consequently, there are more sequential scans of the

items table (see the slanted lines in the thickest horizontal band of each plot) for the shopping and ordering than for the browsing workload.

For the RUBiS benchmark (see Figure 4), there are fewer disk accesses than for the TPC-W benchmark (because the database size is smaller than the TPC-W database resulting in higher cache hit ratio). However, similarly to the TPC-W access pattern, there are many accesses for the browsing workload mix (leftmost plot in the figure) in the items table. As bidding becomes more dominant in the bidding and high-bidding workloads, then the working set increases because more tables dealing with orders become active.

The TPC-C benchmark (see Figure 5), is designed such that the queries access mostly randomly all database tables. Generally, under TPC-C, the queries do not generate the answer based on a large set of data as the browsing queries in TPC-W and RUBiS and although the database is the largest among the three tested, we do not find the long sequential scans as in the TPC-W browsing mix. Because visually there is no difference between the two TPC-C workloads Figure 5 plots the access pattern only for the standard mix.

The block-level trace is further analyzed to extract additional metrics that characterize the IO workload for all three benchmarks, in addition to the high-level visualization of the access pattern plots. Specifically, we estimate the number of sectors read and written, average request interarrival times, length of sequential streams of READs and WRITEs that can be detected and exploited for optimization purposes at the disk level, and the sequentiality of READ and WRITE traffic (after the Anticipatory IO scheduler has merged consecutive requests). Sequentiality is measured as the ratio of the number of sequential requests over the total number of requests (i.e., a fully random workload has zero sequentiality and the larger the sequentiality metric the larger the number of sequential streams detected in the workload). We calculate such metrics for the READ-intensive workloads i.e., browsing and the standard mix for TPC-W/RUBiS and TPC-C, respectively, and WRITE-intensive workloads, i.e., ordering, high bidding, and high WRITE mixes for TPC-W, RUBiS, and TPC-C, respectively. We present our findings in Tables 8, 9, for the light load READ-intensive and WRITE-intensive workload mixes, respectively, and in Tables 10 and 11 for the heavy load READ-intensive and WRITE-intensive workload mixes, respectively.

Tables 8, 9, 10 and 11 indicate that READ traffic dominates the overall disk traffic for all benchmarks. The only exception is the high WRITE workload that we defined for the TPC-C benchmark (as an extreme workload mix dominated by WRITEs). For TPC-C workloads the ratio between READs and WRITEs oscillates between 60%/40% for the standard workload to 40%/60% for the high WRITE workload. For browsing workload mixes in TPC-W and RUBiS this ratio is respectively 99.9% /0.1% and 99%/1%,

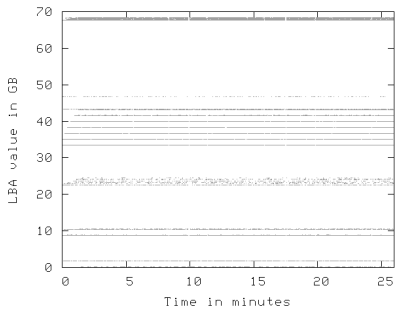

Figure 5. Disk access pattern for the TPC-C benchmark under the standard workload mix with 30 warehouses and 7 clients.

i.e., READs overwhelmingly dominate the workload. Intensifying the WRITE activity via either ordering or bidding for the two benchmarks increases the WRITE portion of the workload 10 times for TPC-W and 20 times for RUBiS (the respective READ/WRITE ratios are 99%/1% and 81%/19%). As load in the system increases (i.e., number of clients or sessions) then the READ/WRITE ratios remain the same for TPC-W and TPC-C but change for the RUBiS benchmark. Specifically, under high load, RUBiS's READ traffic increases much more than the WRITE traffic, for both workload mixes, but in particular for the READ-intensive mix.

Changing the user activity in the workload from browsing to ordering/bidding reduces the overall traffic in the IO subsystems in particular for the RUBiS benchmark by 4 and 10 times for light and high load, respectively. Similarly the TPC-W IO traffic reduces by 2 times for light load when workload changes from browsing to ordering but increases by 10% when the same change happens under high load. TPC-C IO traffic increases by 30% and 40% when the standard workload is replaced by a WRITE-dominated one for light and heavy load, respectively. The available memory plays an important role in the changes seen at the block-level trace when the workload mix switches from browsing to ordering/bidding. In general this increases the footprint of the working set and if the memory can accommodate the increase, as it is the case for RUBiS, the overall IO traffic reduces otherwise it increases, as it is the case for TPC-C benchmark.

Workload locality is measured via the metric of sequentiality. With a few exceptions, load increase is associated with higher locality in the workload (mostly for READs than WRITEs) for the three benchmarks. This is expected because a workload mix determines the working set footprint in the block-level trace and even if only random requests are added to that mix (i.e,, higher load) it will only improve workload sequentiality (given that the database

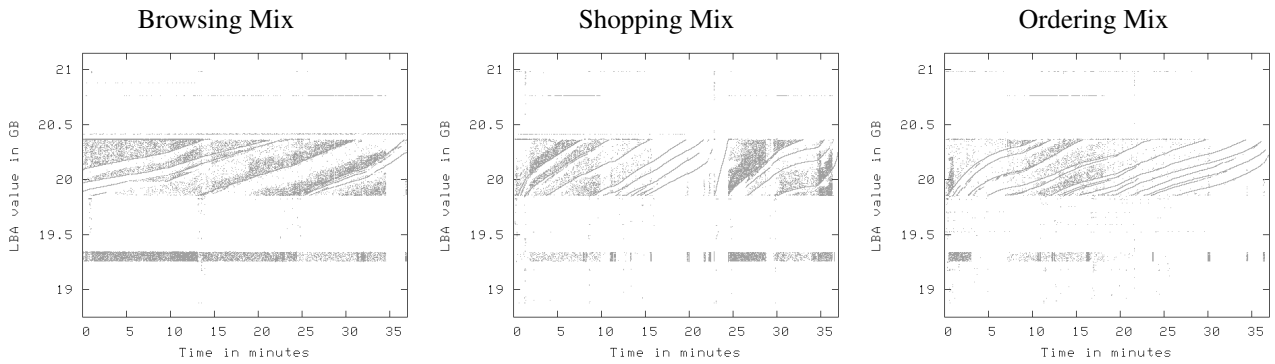

Figure 3. Disk access pattern for the TPC-W benchmark with 64 clients.

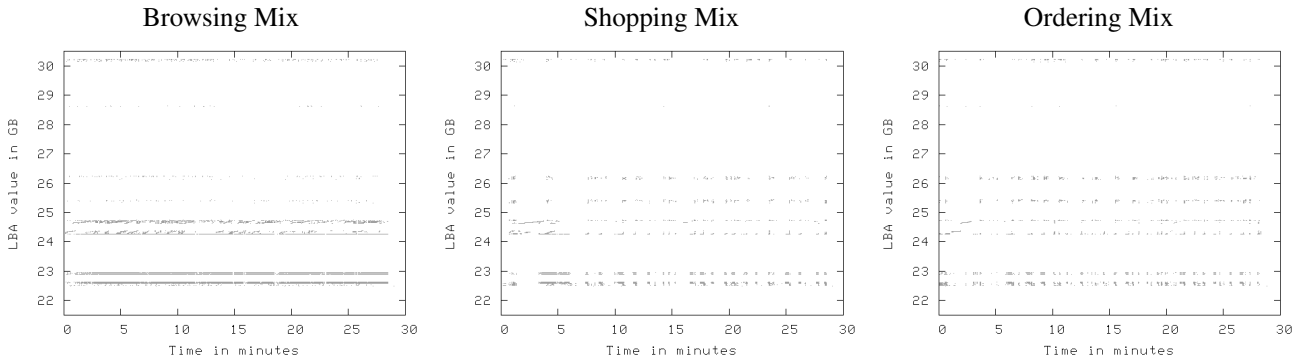

Figure 4. Disk access pattern for the RUBiS benchmark with 32 clients.

Benchmark	Sectors read	Sectors written	Avg. Read Stream Size	Avg. Write Stream Size	Read Seq.	Write Seq.	Avg. Interarrival Time
TPC-W	11,327,714	11,352	36.69	11.62	0.46	0.88	3.06
RUBiS	659,653	7,400	9.48	11.97	0.09	0.99	11.97
TPC-C	1,356,551	940,036	12.58	8.79	0.14	0.07	7.83

Table 8. Block-level trace characteristics under light load and READ-intensive workloads (i.e, browsing, browsing, and standard mixes for TPC-W, RUBiS, and TPC-C, respectively).

Benchmark	Sectors read	Sectors written	Avg. Read Stream Size	Avg. Write Stream Size	Read Seq	Write Seq	Avg. Interarrival Time
TPC-W	6,751,063	20,840	35.99	14.02	0.49	0.33	4.21
RUBiS	118,549	27,184	9.11	11.82	0.05	0.22	40.33
TPC-C	1,159,496	1,592,638	14.40	9.69	0.12	0.09	6.32

Table 9. Block-level trace characteristics under light load and WRITE-intensive workloads (i.e, ordering, high bidding, and high WRITE mixes for TPC-W, RUBiS, and TPC-C, respectively).

Benchmark	Sectors read	Sectors written	Avg. Read Stream Size	Avg. Write Stream Size	Read Seq	Write Seq	Avg. Interarrival Time
TPC-W	10,075,511	12,688	51.93	11.19	0.53	0.79	4.51
RUBiS	4,018,828	7,752	21.31	11.43	0.35	0.99	5.79
TPC-C	1,049,287	875,084	15.03	9.50	0.15	0.07	8.78

Table 10. Block-level trace characteristics under heavy load and READ-intensive workloads (i.e, browsing, browsing, and standard mixes for TPC-W, RUBiS, and TPC-C, respectively).

Benchmark	Sectors read	Sectors written	Avg. Read Stream Size	Avg. Write Stream Size	Read Seq	Write Seq	Avg. Interarrival Time
TPC-W	13,460,872	26,640	56.53	13.51	0.56	0.31	3.23
RUBiS	440,200	21,880	13.18	14.01	0.19	0.31	15.99
TPC-C	1,368,204	1,652,168	14.00	10.84	0.12	0.14	6.04

Table 11. Block-level trace characteristics under heavy load and WRITE-intensive workloads (i.e, ordering, high bidding, and high WRITE mixes for TPC-W, RUBiS, and TPC-C, respectively).

size is fixed). RUBiS has the highest locality gain as expected because the footprint is smaller as the database is smaller. As workload switches from READ-intensive to WRITE-intensive the footprint of the working set increases and this causes, generally, for the workload locality to reduce for all workloads and in particular for the WRITE traffic. The exceptions to this observation are related to the database size and the cache hit ratio for the specific workload.

The request interarrival times are short for most TPC-W and TPC-C scenarios captured in Tables 8, 9, 10 and 11. RUBiS experiences lighter loads than TPC-C and TPC-W, with the only exception of READ-intensive and heavy load case where the average interarrival times are short for all three benchmarks.

5.1 File System

We further evaluate storage subsystem performance under the three benchmarks when different file systems manage the IO traffic. Ext2, Ext3, and ReiserFS are evaluated as three popular Linux file systems, available in all recent Linux distributions. The high-level distinction between the file systems is

- **Ext2** is a standard FFS-like file system, which uses cylinder groups for placement and single/double/triple indirect metadata blocks.
- **Ext3** has data structures that are backwards compatible with Ext2, but it uses a journal to enhance data reliability and consistency.
- **ReiserFS** also has a journal, which is a single contiguous file, and it uses a B^+-tree structure to manage metadata.

Figure 6 presents the user-level throughput measured in transactions per minute as a function of the file system and workload mix for the three benchmarks under our evaluation. The measurements of Figure 6 are conducted under heavy load scenarios, i.e, 64, 128, and 300 clients for TPC-W, RUBiS and TPC-C, respectively. Results with light load are available as well and qualitatively similar to those presented in Figure 6. These results indicate that the file system does not effect user-level throughput when system is in overload as it is the case of TPC-W benchmark (recall that the interarrival times are short - between 3ms and 4ms). This observation holds also for the bidding and high-bidding workloads in RUBiS but the reason is that RUBiS with 128 clients represent heavy IO load only for the browsing mix (i.e,, interarrival times of 5.79 ms) and light IO load for the other two mixes (i.e, 15ms average interarrival times). Similarly to browsing mix of RUBiS, the TPC-C standard and high WRITE workloads represent heavy loads but not overloads (as the TPC-W case) with interarrival times 6 ms and 9 ms, respectively. As a result, the file system does not effect user-level throughput for the IO light or overload cases but it makes a difference for the heavy load cases where effective optimization matters when it comes to overall system performance.

The main difference between the three file systems is journal maintenance. The lack of the journal for Ext2 means that it has, in general, less work to complete, while Ext3 and ReiserFS maintain a journal of the written data enhanc-

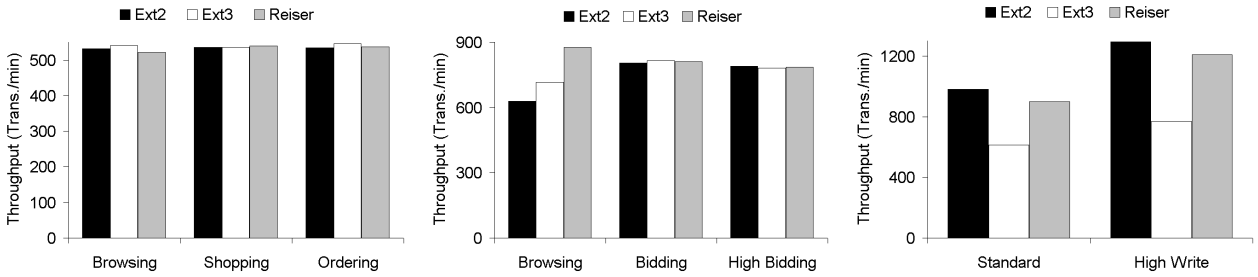

Figure 6. Throughput (in user transactions per minute) under different file systems and workload mixes for the three benchmarks. From left to right plots represent TPC-W with 64 clients, RUBiS with 128 clients, and TPC-C with 64 clients. IO scheduling is Anticipatory.

ing data consistency and reliability. Efficient management of the journal as it is the case of ReiserFS under heavy loads results on higher user-level throughput under ReiserFS than Ext3 and Ext2 (under RUBiS). For TPC-C, with more WRITEs and more metadata in the journal to be maintained, Ext2 performs slightly better than ReiserFS and by factoring in the added data consistency and reliability in ReiserFS, we consider it the best performing file system among the three evaluated.

5.2 Effects of I/O Scheduling Algorithm

Finally, we evaluate the impact that various I/O scheduling algorithms have on user-level throughput. We measure performance of the four I/O schedulers supported in Linux kernel 2.6:

- **No-OP**: first come first served with the benefit of request merging (only with the last request, to preserve fairness).

- **Deadline**: behaves as a standard elevator, unless reads and writes have been waiting longer than respective predefined thresholds.

- **Anticipatory** (default): same as deadline, however sometimes pauses, in order to avoid seeking, while waiting for more sequential read requests (Non-Work Conserving) to arrive.

- **CFQ**: elevator that attempts to give every process the same number of I/Os (one queue per process)

All experiments use ReiserFS as file system and the user-level throughput for the TPC-C benchmark is given in Figure 7. Figure 7 shows that the choice of the IO scheduling algorithm impacts user-performance for the TPC-C benchmark with Deadline being the best performing IO scheduler.

The user-level throughput of TPC-W and RUBiS is not effected by the IO scheduler choice. While one factor is the load in the system, i.e, TPC-W and RUBiS generate extreme loads (either very light or very heavy), another cause might be the multi-tiered structure of the TPC-W and

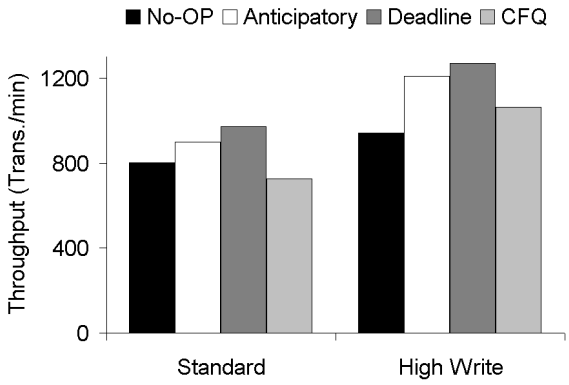

Figure 7. Throughput (in user transactions per minute) for different disk schedulers and workload mixes for the TPC-C benchmark with 300 clients. File system is ReiserFS.

RUBiS benchmarks. The user-level throughput for TPC-C is measured at the database clients while the user-level throughput for the TPC-W and RUBiS is measured at the Web clients (i.e., two tiers up in the hierarchy). The service at the Web server and application server in TPC-W and RUBiS diminishes the effect of the IO scheduler in the user-level throughput.

6 Conclusions

We conducted measurements on three different benchmarks, i.e., TPC-W, TPC-C, and RUBiS, with the purpose of emulating the behavior of the storage subsystem under e-commerce applications. Our measurements indicated that there are commonalities in the behavior of the storage subsystem under such applications, that can be used in system design and capacity planning. Generally, when a workload

shifts from web-site browsing (i.e, reading) to transaction processing (i.e writing), the IO traffic reduces, while the footprint of the IO working set increases. Consecuently, the block-level locality of requests under the ordering and the bidding activities is reduced when compared with the request locality under the browsing workload mix. As a result, the workload mixes dominated by transactions (ordering, bidding, shipping) demand more work from the storage subsystem than the workload mixes dominated by browsing, although the former consistes of fewer requests overall. The file system and the device driver scheduling represent elements in the IO path that for a given set of system resources improve user-level throughput even further, under medium or high load. These effects diminis for light load or overload.

References

[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *Proceedings of the 5th IEEE Workshop on Workload Characterization*, Austin, TX, USA, Nov. 2002. ACM Press.

[2] M. Arlitt, D. Drishnamurthy, and J. Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology*, 1(1):44–69, 2001.

[3] Z. Chen, Y. Zhou, and K. Li. Eviction-based cache placement for storage caches. In *USENIX Annual Technical Conference, General Track*, pages 269–281, 2003.

[4] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *Proceedings of the 13th international conference on World Wide Web*, pages 276–286, New York, NY, USA, 2004. ACM Press.

[5] D. Garcia and J. Garcia. TPC-W E-Commerce Benchmark Evaluation. *IEEE Computer*, pages 42–48, Feb. 2003.

[6] S. Gurumurthi, J. Zhang, A. Sivasubramaniam, M. T. Kandemir, H. Franke, N. Vijaykrishnan, and M. J. Irwin. Interplay of energy and performance for disk arrays running transaction processing workloads. In *ISPASS*, pages 123–132, 2003.

[7] W. W. Hsu, A. J. Smith, and H. C. Young. I/o reference behavior of production database workloads and the tpc benchmarks an analysis at the logical level. *ACM Trans. Database Syst.*, 26(1):96–143, 2001.

[8] L. Huang and T. cker Chiueh. Charm: An i/o-driven execution strategy for high-performance transaction processing. In *USENIX Annual Technical Conference, General Track*, pages 275–288, 2001.

[9] S. Iyer and P. Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. In *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[10] Java TPC-W Distribution. *PHARM Project*. `http://www.ece.wisc.edu/pharm/`, ECE / CS Departments, University of Wisconsin-Madison.

[11] A. Kamra, V. Misra, and E. Nahum. Controlling the performance of 3-tiered web sites: modeling, design and implementation. In *Proceedings of SIGMETRICS, the International Conference on Measurement and Modeling of Computer Systems*, pages 414–415, New York, NY, USA, 2004. ACM Press.

[12] D. McWherter, B. Schroeder, N. Ailamaki, and M. Harchol-Balter. Priority mechanisms for oltp and transactional web applications. In *20th International Conference on Data Engineering (ICDE 2004)*, Boston, MA, Apr. 2004.

[13] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: Findings and implications. In *Proceedings of ACM SIGCOM'02*, Sweeden, Aug. 2000.

[14] S. Pandey, K. Ramamritham, and S. Chakrabarti. Monitoring the dynamic web to respond to continuous queries. In *The Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003.

[15] RUBiS Implementation. `http://rubis.objectweb.org/`.

[16] Transaction processing performance council. `http://www.tpc.org/`.

[17] TPC-C Implementation, `http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/`.

[18] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proceedings of SIGMETRICS*, pages 291–302, 2005.

[19] VMWare INC. *VMWare Workstation*. http://www.vmware.com.

[20] H. W.Cain, R. Rajwar, M. Marden, and M. H. Lipasti. An architectural evaluation of Java TPC-W. In *The Seventh International Symposium on High-Performance Computer Architecture*, Jan. 2001.

[21] Q. Zhang, A. Riska, and E. Riedel. Workload propagation - overload in bursty servers. In *Proceedings of the Second Conference on the Quantitative Evaluation of Systems (QEST)*, pages 179–188, Torino, Italy, Sept. 2005.

[22] Q. Zhang, A. Riska, E. Riedel, and E. Smirni. Bottlenecks and their performance implications in e-commerce systems. In *WCW*, pages 273–282, 2004.