Rethinking Designs for Managing Multi-Tenant OLTP Workloads on SSD-based I/O Subsystems

Ning Zhang [#1] [*], Junichi Tatemura [*2], Jignesh M. Patel [#3], Hakan Hacıgümüş [*4]

[#]*Computer Sciences Department, University of Wisconsin-Madison, USA*
[1]nzhang@cs.wisc.edu [3]jignesh@cs.wisc.edu

[*]*NEC Laboratories America, USA*
[2]tatemura@nec-labs.com [4]hakan@nec-labs.com

ABSTRACT

Multi-tenancy is a common practice that is employed to maximize server's resources and reduce the total cloud operation costs. The focus of this work is on multi-tenancy for OLTP workloads. Several designs for OLTP multi-tenancy have been proposed that vary the trade-offs made between between performance and isolation. However, existing studies have not considered the impact of OLTP multi-tenancy designs when using an SSD-based I/O subsystem. The focus of this work is on examining and comparing a range of multi-tenancy designs for OLTP workloads on an SSD-based I/O subsystem.

We compare three designs using both open-source and proprietary DBMSs. Our study reveals that in contrast to the case of an HDD-based I/O subsystem, *VM-based designs* have fairly competitive performance to the non-virtualized designs (generally within 1.3–2X of the best performing case) on SSD-based I/O subsystems. Whereas previous studies were based on traditional hard disk-based environments, our results indicate that switching to a pure SSD-based I/O subsystem requires rethinking the trade-offs for multi-tenancy for OLTP workloads.

1. INTRODUCTION

Today, it is common to consolidate multiple database workloads onto a single server to reduce the total operating cost. This practice is called "multi-tenancy." Several multi-tenancy designs are available for online transaction processing (OLTP) workloads, such as, the "share-everything" design [9, 14], where a *single* database management system (DBMS) instance is used to process the requests on behalf of multiple database workloads. Another design is to install multiple DBMS instances on the same machine, with each DBMS instance dedicated to serve the requests from one database client [9]. We call this design as the "dedicated" design. The third design employs a virtual machine monitor (VMM) to create multiple virtual machine (VM) instances, and an independent DBMS instance is run in each VM [16].

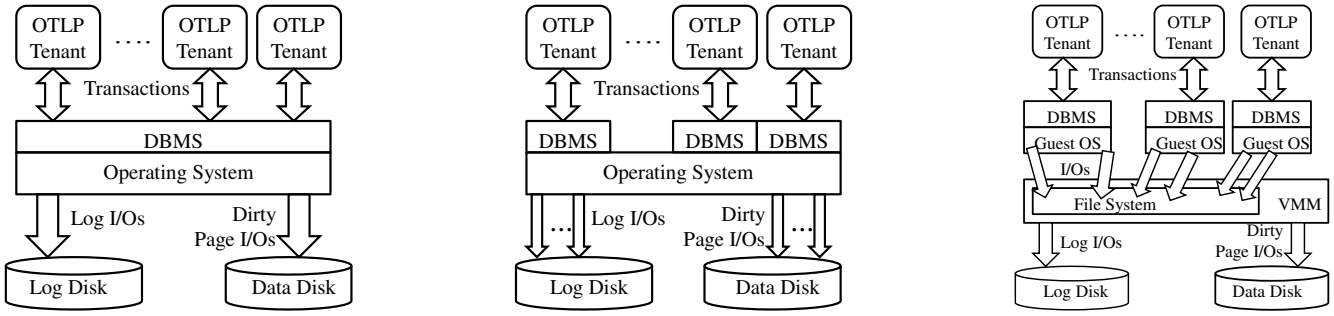
As a motivating scenario, consider the provisioning of private clouds for multi-tenant OLTP applications. The cloud designers usually face a critical design choice: Although they tend to prefer the VM-based design for better management and isolation, they are concerned about potential performance penalties associated with running a database service in a VM, especially since previous work has pointed out that the VM-based design leads to a severe performance penalty [9]. Our key motivation in this paper is to conduct a performance study to help engineers make well-informed decision regarding multi-tenancy DBMS designs.

Our work aims to fill two research needs in this area. First, while there is existing work studying various multi-tenancy OLTP designs (e.g. [9, 14, 16]), a comprehensive direct comparison of these designs is missing. Second, there is a critical move in the industry towards servers with pure SSD-based storage [5], and most previous works have examined multi-tenant OLTP designs using hard disk drives (HDD) based I/O subsystems. Hence, it is not clear if these previous results hold with I/O subsystems that use SSDs.

In this paper, we identify key factors that are crucial to the performance of three multi-tenant OLTP designs (share-everything, dedicated, and VM-based) on a modern SSD-based I/O subsystem. We also compare these designs with each other on both HDD and SSD-based environments, and show that switching to a pure SSD-based I/O subsystem produces different results than what previous studies have found when using an HDD-based I/O subsystem.

To the best of our knowledge, this is the first paper that presents results from a sophisticated evaluation of three dominant multi-tenant OLTP designs on a modern SSD-based I/O subsystem. In this work, we use both open-source and proprietary VMMs and DBMSs.

[*]The work was done while the author was at the NEC Laboratories of America.



(a) Shared Design (b) Dedicated Design (c) VM-based Design

Figure 1: Overview of the different multi-tenant OLTP designs

The key contributions of this paper are as follows:

1. We show that the VM-based designs show fairly competitive performance to the non-virtualization based designs (from 1.3X to 2X performance degradation) on a modern SSD-based I/O subsystem. While Curino et al. [9] reported that the shared-everything design outperformed the VMware-based design by 6-12X, we show that when switching to an pure SSD-based systems, the performance penalty that holds for traditional HDD-based systems does not hold on a modern SSD-based I/O subsystem.
2. Our experiments show that the shared design still maintains a performance advantage over the VM-based designs on an SSD-based I/O subsystem, as long as it can minimize the contention on the shared resources, such as locks on internal DBMS data structures.
3. The dedicated design with PostgreSQL generally presents the best performance compared to the other designs, in our settings. This is primarily because it can avoid the resources contention problem in the shared designs [11, 14], and the performance penalties associated with the VM-based designs [9]. Various factors prevented us from running the dedicated design with the commercial DBMS, likely indicative of a broader trend that actual production deployment of this setting is challenging.

The remainder of this paper is organized as follows: Section 2 describes the multi-tenant OLTP designs that we consider in this paper. The experimental setup and results are reported in Section 3. Related work is presented in Section 4, and Section 5 contains our conclusions for this paper, and points to some directions for future work.

2. MULTI-TENANT OLTP SOLUTIONS

In this section, we describe the multi-tenant OLTP designs that we consider in this paper.

2.1 Shared Design

The shared design, which is also called as the "share-everything" technique in [9, 14], is to install a single DBMS instance on a physical server to process the transactions on behalf of all the tenants. Figure 1(a) illustrates the shared design. Most DBMS internal structures are shared across the tenants; for example, one Write-Ahead-Logging (WAL) writer process and one background writer process is shared across all the tenants when writing the DB logs, and when flushing dirty pages from the buffer pool.

2.2 Dedicated Design

The dedicated design, as introduced in [9], installs multiple DBMS instances on a single physical server, and each tenant only interacts with its own DBMS instance. Figure 1(b) shows the dedicated design. The internal structures for each instance are not shared across the tenants. For example, there is a dedicated WAL writer process, and a dedicated background writer process for each DBMS instance. However, from the log and data I/O device's perspectives, multiple concurrent log and dirty page I/Os are issued.

2.3 Virtual Machine Design

The Virtual Machine (VM) based design, as shown in Figure 1(c), shares the underlying hardware resources by creating multiple VMs. Here, each tenant is associated with a DBMS instance that is installed in an independent VM. Different VM implementations have different characteristics about how they virtualize the resources (CPU, memory, network and IO resources), and here we consider the two leading VM systems: Xen [1] and VMWare [2].

3. EXPERIMENTAL RESULTS

In this section, we experimentally compare the three multi-tenant OLTP designs presented above. For our evaluation, we used an open-source implementation of the TPC-C benchmark [6], which has been used before in an number of studies in this area, including [12, 13].

IOPS	H-SSD	SSD RAID
Sequential Read	20,863	6,062
Random Read	13,692	3,084
Sequential Write	17,085	6,118
Random Write	12,623	2,701

Table 1: IOPS of the H-SSD and the SSD RAID devices for four different I/O patterns. These IOPS numbers are observed in CentOS 6.3, and generated by issuing synchronous 8KB I/Os against a 20GB file.

3.1 Hardware and Software Specifications

3.1.1 Hardware Configuration

Our experimental platform is a server with a 2.26GHz Intel(R) Xeon CPU E5520 with 8 physical cores, 64 GB ECC memory, a SATA 7200 RPM hard disk drive (HDD) to hold the OS, a SSD RAID 0 system (**SSD RAID**), and a high-end (**H-SSD**). The SSD RAID subsystem consists of three identical Intel Series 320 SSD 120 GB in a RAID 0 configuration, and the H-SSD subsystem consists of one 80GB Fusion IO ioDrive. Their raw I/O performance of these devices is shown in Table 1. As discussed below, the data and log placements across these two subsystems are optimized for each design.

In keeping with the key theme of this paper – i.e. focusing on impact of SSD-based I/O subsystem for OLTP multi-tenancy – all the results presented in this paper, except for those in Section 3.7, place both the data and the log files on the SSD devices. To close the loop with previous work [9] that used only HDDs, in Section 3.7 we consider the case when both the data and the log files are placed on HDDs.

3.1.2 Multi-tenancy Design Settings

The server described above is used as the hardware for each design. For the DBMS software, we use an open-source DBMS and a popular commercial DBMS. The open-source DBMS is PostgreSQL 9.2.4 (referred to as PostgreSQL). The proprietary DBMS is called DBX. (The proprietary DBMS name has been anonymized to respect the licensing terms.)

The software setting for each design is discussed below.

The **dedicated design** is implemented on a CentOS 6.3 server with multiple independent DBMS instances running concurrently. Each tenant stores its data and runs its queries within a dedicated DBMS instance.

The **shared design** runs on a CentOS 6.3 server with only one DBMS instance. All tenants run their queries and store their data within this shared DBMS instance.

The **VMware design** employs VMware ESXi 5.1.0 as the VMM and we set it up by following the standard best practices guideline [7]. Each guest OS (CentOS 6.3) has three virtual disks: one HDD holds the OS and the DBMS binaries, one for the SSD RAID, and the last one for the H-SSD. Each VM runs one dedicated DBMS instance.

The **Xen design** runs Xen 4.1.2, and is set up by following its best practices guide [4]. Each guest OS (CentOS 5.9) is installed with para-virtualization and runs one dedicated DBMS instance. We note that para-virtualization has been used in a similar context before [10] to consolidate and run database workloads. As in the VMware design, each guest OS has three attached virtual disks: one HDD to hold the OS and the DBMS binaries, one SSD RAID, and one H-SSD.

3.1.3 Data and Workload Settings

Each tenant generates TPC-C workload exclusively against one TPC-C database, and with no think time. The actual size of the TPC-C database varies depending on the specific experiment. Each tenant runs on a separate server, which is connected to the experimental platform via a 1Gbps network. All tenants run simultaneously, and the first 10 minutes are used to ramp up their databases. Then, we collect and report the average aggregate performance over the next 60-minute measurement window.

Note that in OLTP environments (and in other database environments [15]), tenants are often consolidated so that their working sets mostly fit in the main memory, so that the extra data-related I/O does not degrade performance significantly. For example, one of the key contributions of [9] is a profiling method to identify the effective size of the working sets (hence the optimal degree of multi-tenancy). In this paper, we largely focus on the case when the working sets of all tenants reside in the main memory, but we also conduct additional experiments (see Section 3.6) to consider the case when the working set is larger than the main memory. We recognize that dealing with workload fluctuations is an important and related aspect to multi-tenancy design, but it is also an active area of work. To keep this paper focused, we only consider the scenario described above.

Finally, we note that even when the working sets fit in the main memory, there is still significant write I/O traffic because of the continual checkpointing process(es) and log writes.

We place the log files and the data files for the tenant databases on separate disks (or virtual disks), using either the SSD RAID to save the log files and the H-SSD to save the data files, or vice versa, in order to find highest-performing disk assignment for each design. For simplicity, we call the disk that saves the log files as the *log disk* and the disk saves the data files as the *data disk*.

3.1.4 Database Parameter Settings

In our experiments, we use the optimized configuration parameters published at [3] for PostgreSQL, and the default parameters for DBX.

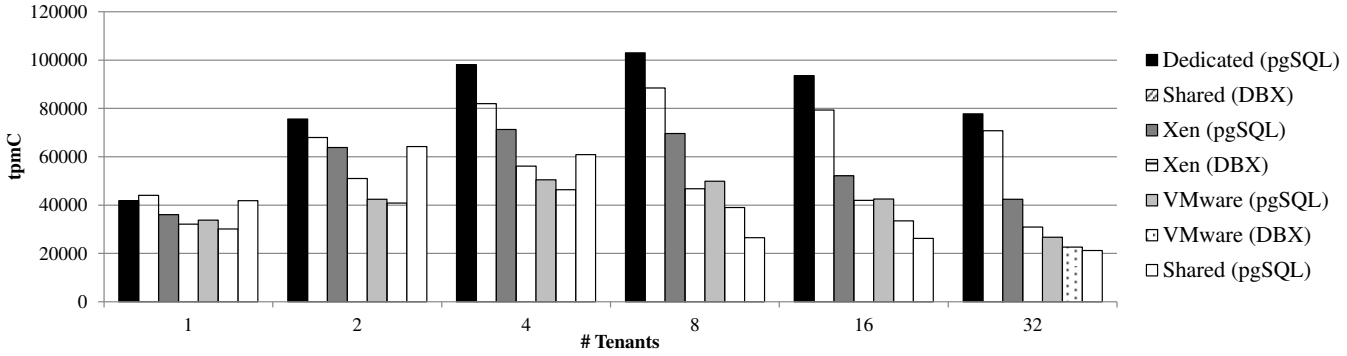


Figure 2: Varying the number of tenants: Comparison of all designs, each at their best performance configuration

Design	Log Disk	Data Disk
Dedicated design with PostgreSQL	H-SSD	SSD RAID
Shared design with DBX	SSD RAID	H-SSD
Xen design with PostgreSQL	H-SSD	SSD RAID
Xen design with DBX	H-SSD	SSD RAID
VMWare design with PostgreSQL	H-SSD	SSD RAID
VMWare design with DBX	H-SSD	SSD RAID
Shared design with PostgreSQL	SSD RAID	H-SSD

Table 2: Best location of the data and the log for each design

3.2 Key Result: Overview of the Evaluation

Based on the three multi-tenancy designs introduced in Section 2, we compare the seven designs listed in the first column of Table 2. Note that we could not get the "Dedicated design with DBX" to work reliably as it involves deploying multiple DBX instances in the same server. This configuration is not supported by the vendor, and there are various stability issues with running this configuration. Licensing costs further complicate this scenario. Hence, we do not report any results for this configuration.

Before we analyze each design, we first present a direct comparison of these seven designs based on the configuration (the location of the data and the log files) that produce their individual best performance. Table 2 shows the optimal configurations for each design.

In Figure 2, we present a direct comparison of these seven designs. The comparison is based on the aggregate throughput performance metric, i.e. the New-Order transactions per minute (**tpmC**), over 1, 2, 4, 8, 16 or 32 homogeneous tenants. These tenants all share the same physical machine based on the specific multi-tenancy design. Each tenant has 15 warehouses (1.5 GB database) and creates 15 concurrent threads to generate the TPC-C workload.

From Figure 2, we observe that the dedicated design with PostgreSQL (black bars in the figure) outperforms all other designs, and in all cases. The reason for this behavior is as follows: (1) Using the SSD-based I/O subsystem significantly speeds up the transaction log I/O and checkpointing (dirty page) I/O processing, which in turn pushes up the transaction commit rate significantly. (2) Compared to the VM-based designs, the dedicated design has no virtualization-related overheads (CPU, cache, memory, I/O). Thus it can utilize all the hardware resources in the server more effectively. (3) The dedicated design isolates the tenants at the DBMS-level by using multiple PostgreSQL instances, so that it can eliminate potential contentions that are incurred when sharing the same DBMS instance across different tenants. Note that given the nature of TPC-C, which is a very partitionable workload, the contention is usually for shared internal data structures and metadata (as opposed to actual data contention.)

Surprisingly, the four VM-based designs (Xen with PostgreSQL, Xen with DBX, VMware with PostgreSQL, VMware with DBX) are not significantly slower than the dedicated design. For example, Xen with PostgreSQL is only 1.3X slower than the dedicated design at their best performance point (i.e., when the number of tenants is 8). This is an interesting result as it shows that an SSD-based system has different tradeoffs (for multi-tenant OLTP workloads) compared to an HDD-based I/O subsystem. In [9], the authors showed that the shared design has 6X-12X better (tpmC) performance than the VMware design, mostly because the guest OSs in the VMware design generate multiple independent and random I/Os for the log and the data file-related writes (on the HDD), while the shared design sequentially writes to one log file and coordinates data writes on the HDD. However, SSDs have far better I/O performance (than HDDs) especially for random I/Os, and as a result, using a VM-based design with SSDs is fairly competitive.

In addition, as seen in Figure 2, the shared design generates significantly different results when using the open-source versus using the proprietary DBMSs. With the proprietary DBX, the shared design outperforms all other designs except the dedicated design. However, when using the shared design with PostgreSQL, we observed the worst per-

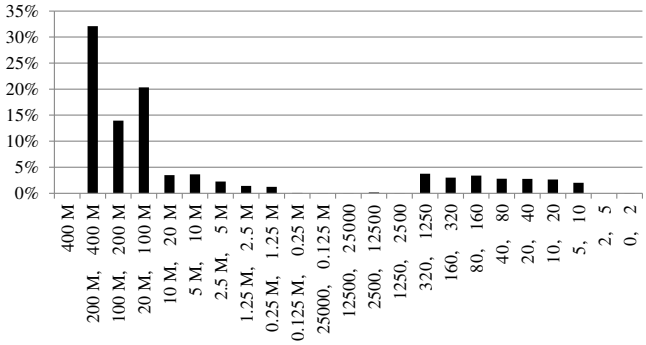


Figure 3: Dedicated Design with PostgreSQL: Distribution of distance (in LBNs) between two successive log I/Os ("**M**" denotes **Million** in x-axis

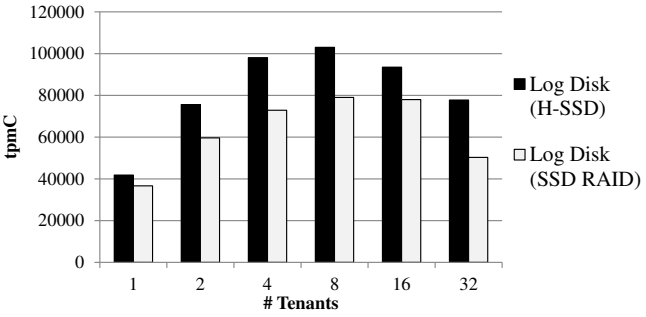


Figure 4: Dedicated Design with PostgreSQL: Comparing the two different disk assignments

Parameter	Definition
svctm	The average service time (in milliseconds, excluding the queuing time) for I/O requests
util%	The disk bandwidth utilization (disk saturation occurs when this parameter is 100%)

Table 3: Disk I/O statistics

formance (as seen in Figure 2) when more than 4 tenants (e.g. 8, 16, 32) are consolidated. This is because when all the tenants run on one shared PostgreSQL, lightweight locks are needed to serialize the accesses to the shared PostgreSQL resources (e.g. WAL buffer pool). When the I/O subsystem is fast (because of SSDs), the contention on these lightweight locks becomes very prominent. However, with DBX, we observed that the lock contention problem is not significant, so the shared design yields good performance. In other words, the internal locking overhead has been better optimized in DBX compared to PostgreSQL, making DBX a better candidate to exploit the higher I/O performance associated with SSD-based I/O subsystems, in the shared design.

Having presented the main results above, in the following sections, we describe the configuration for each design that results in its best performance (as shown in Figure 2), and present the challenges associated with each design.

3.3 Dedicated Design

In this section, we first describe how we achieved the best performance on the dedicated design with PostgreSQL.

Recall that the dedicated design isolates the tenants at the DBMS-level by using multiple PostgreSQL instances. Now, since the performance of the TPC-C workload is mainly determined by the speed of log I/O processing, having multiple PostgreSQL instances inevitably leads to multiple log files co-existing on the same log disk. Logically, each log file is written in a sequential manner. But, from the log disk's point of view, in this case, the physical I/O pattern is close to a random I/O pattern, which can potentially degrade the overall performance.

To verify this hypothesis, we plot the distribution of the distance between two successive log I/Os; here the distance is the spread in the logical block numbers (LBNs) of the blocks. This result is shown in Figure 3. From this figure, we see that most of the log I/Os are far apart (in terms of the logical block layout on disk) from the previous log I/Os. For example, about 32% of log I/Os are more than 200 and less than 400 million logical blocks away from the previous I/Os. So, as observed from Figure 3, many concurrent (independent and sequential) log I/Os generate a physical random I/O pattern on the log device.

Given the random I/O pattern on the log disk, now we consider the impact of assigning the log file to either the SSD RAID or the H-SSD devices, and vice versa for the data file(s). As can be seen in Figure 4, performance is higher when using the H-SSD device as the log disk, across all multi-tenancy levels. The reason for this behavior is that the log I/O speed is an important factor that impacts the performance of TPC-C workload, and the H-SSD is far more efficient than the SSD RAID in handling the log I/O requests that are random and synchronous. To better understand this behavior, we used the Linux "sar" command to gather the disk statistics described in Table 3.

Now we can compare the disk statistics between these two disk assignments. In Table 4, we notice that when the SSD RAID device is used as the log disk, the average service time for the log disk (`svctm`) and the disk utilization (`util%`) is much higher than the case when the log disk is on the H-SSD device. For example, when there are 4 tenants, the disk utilization (`util%`) for the log disk (H-SSD) is only 47.8%, compared to 91.95% when using the SSD RAID device as the log disk. Essentially, using the H-SSD device as the log disk results in better performance, as in this case, the log disk performance is the dominant factor in determining overall performance.

As can be observed in Figure 2, the performance of the

	Log disk (H-SSD)						Data disk (SSD RAID)					
# tenants	1	2	4	8	16	32	1	2	4	8	16	32
svctm	0.12	0.09	0.10	0.19	0.38	0.41	0.15	0.13	0.11	0.07	0.06	0.18
util%	29.6	39.4	47.8	75.6	88.9	95.9	2.7	6.4	8.5	14.2	18.8	88.5

(a) Disk I/O statistics when using H-SSD as the log disk

	Log disk (H-SSD)						Data disk (SSD RAID)					
# tenants	1	2	4	8	16	32	1	2	4	8	16	32
svctm	0.25	0.28	0.24	0.35	0.45	0.47	0.04	0.03	0.03	0.04	0.02	0.11
util%	57.2	85.3	91.95	99.5	99.7	100	1.1	1.9	3.35	7.96	15.2	70.5

(b) Disk statistics when using SSD RAID as the log disk

Table 4: Dedicated Design with PostgreSQL: Disk statistics observed from the Linux OS

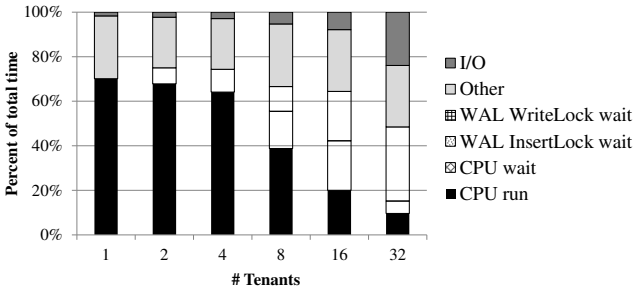


Figure 5: Dedicated Design with PostgreSQL: Time-breakdown of a server thread

dedicated design is relatively stable from 4 to 16 tenants, and decreases when the number of tenants is 32. To explain this behavior, we choose to analyze the PostgreSQL server threads, which are created to execute the requests for each tenant. We broke down the execution time of each server thread into the components that are listed in Table 5.

The two crucial variables in Table 5 are the two *pre-defined* lightweight locks in PostgreSQL, namely the `WAL InsertLock` and the `WAL WriteLock`. For the `WAL InsertLock`, its wait time depends on the number of active threads that are concurrently inserting WAL records into the WAL buffer. While for the `WAL WriteLock`, its wait time is dependent on the time it takes to flush/write the WAL records from the WAL buffer to the log disk.

The time-breakdown analysis in the dedicated design is shown in Figure 5, across all the multi-tenancy levels. Since all the tenants are homogeneous, we randomly select a tenant for the time-breakdown analysis, and show these results in Figure 5. (But we have looked at the time-breakdown for all the tenants individually, and, as expected, these characteristics are similar across all the server threads.)

In Figure 5, we notice that when there are 8 and 16 concurrent tenants, the `CPU wait` time component appears in the time-breakdown, which means the workload is largely CPU-bound. This is because in this case, the I/O device has sufficient resources to "keep up" with the workload that is generated. As can be seen in Table 4 (a) the disk utilization (`util%`) for both the log and the data disk is below 90%. Thus, with 8 and 16 concurrent tenants, the overall throughput is largely determined by the speed with which the CPU can process the workload.

Then, in Figure 5, we notice that at 32 concurrent tenants, the `CPU wait` time component of the breakdown is now no longer the dominant portion (the `CPU run` and the `CPU wait` times are less than 30% of the total time). At this point, the system is slowed down by the log and data disk. As can be seen in Table 4 (a), at 32 tenants, the data disk utilization is 88.5%. However, it is only 18.8% when there are 16 tenants. Similarly, the log disk is also close to 100% busy (95.9%) when there are 32 tenants.

Finally, in Figure 5, we note that the `WAL WriteLock` wait time component is relatively small (compared to the other components). This is because when the log is on the H-SSD device, it can effectively handle the random concurrent log writes to the log disk. This behavior can be observed in Table 4 (a) that shows the log disk utilization (`util%`) is always below 90% when the degree of concurrency increases from 1 to 16. Thus, when the log is on the H-SSD device, the `WAL WriteLock` wait time component is a small fraction of the total runtime. In other words, the log disk is not a bottleneck in most cases.

To summarize, for this design: (1) Because concurrent log I/O requests can generate large amounts of *random* and *synchronous* writes to the log disk, choosing the H-SSD as the log disk produces a higher performance configuration. (2) The checkpointing (dirty page) I/Os (sent to the data disk) are asynchronous, which means that the dirty page write requests are not as critical as the log writes. So using SSD

Components	Definition
CPU run time	The time spent when a server thread is executing code on a CPU core.
CPU wait time	The time spent when a server thread waits in some CPU wait queue.
I/O time	The time spent when a tenant workload waits for an I/O to complete.
`WAL InsertLock` **wait**	The time spent when a server thread wants to acquire a `WAL InsertLock` to insert a WAL record into the WAL buffer, but needs to wait for another thread to release the `WAL InsertLock` first.
`WAL WriteLock` **wait**	The time spent when a server thread wants to acquire a `WAL WriteLock` to dump the WAL records in the WAL buffer to the log disk when a transaction commits (or the WAL buffer is full), but needs to wait for another thread to release the `WAL WriteLock` first.
Other	The time spent when the server thread is in a state other than the above states.

Table 5: Components measured in the time-breakdown of a server thread

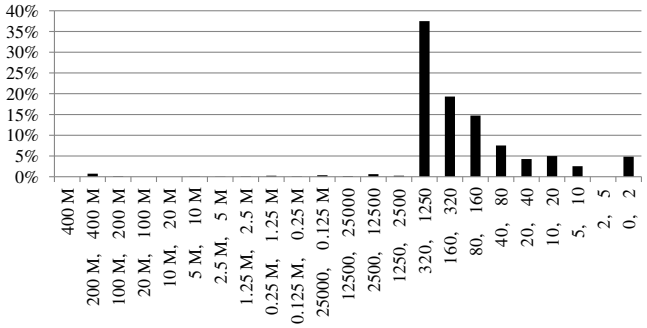


Figure 6: Shared Design with PostgreSQL: Distribution of distance (in LBNs) between two successive log I/Os ("**M**" denotes **Million** in x-axis)

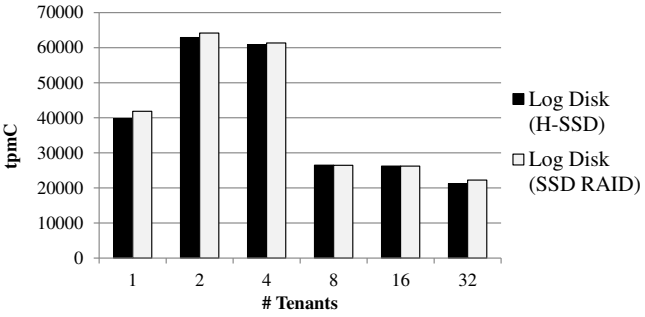


Figure 7: Shared Design with PostgreSQL: Comparing the two different disk assignments

RAID as the data disk and H-SSD as the log disk results in the best performing configuration.

3.4 Shared Design

For the shared design, we first analyze the case when it is deployed with PostgreSQL.

3.4.1 Shared Design with PostgreSQL

Similar to the method described in Section 3.3, we first look at the log I/O pattern, which is shown in Figure 6. We compare this pattern to the log I/O pattern in the dedicated design (shown in Figure 3). We notice that the log I/O pattern in the shared design is far more sequential than the dedicated design, since the shared design only installs one DBMS (one log writer) in the system.

Next, we compare the performance when assigning the log file to either the SSD RAID or the H-SSD devices, and vice versa for the data file(s). These results are shown in Figure 7. As can be seen from this figure, the performance of these two configurations is similar, which implies that the disk I/O devices are not the key performance bottlenecks.

More importantly, as seen in Figure 2, the shared design performs the worst when the number of tenants is larger than 4. To explain this behavior, we did a time-breakdown analysis of the execution time for this design, which is shown in Figure 8. From this figure, we observe that when the number of tenants is larger than 4, the `WAL InsertLock` wait time component, as shown by the *dotted white bar*, starts to dominate the total runtime. The reason for this behavior is as follows: the shared design has only one PostgreSQL instance and all the tenants (workload generators) concurrently try to insert the WAL records into the shared WAL buffer pool. Now, the `WAL InsertLock`, which protects shared access to the WAL buffer pool, becomes a key source of contention and a bottleneck. When the number of tenant is 8 or more, the performance starts to decline (as can be seen in Figure 2), as the contention for the `WAL InsertLock` is very high.

3.4.2 Shared Design with DBX

Compared to the result generated by the shared design with PostgreSQL, the shared design with DBX is the overall second highest-performing design (recall Figure 2).

First, we study the impact of assigning the log file to either the SSD RAID or the H-SSD devices, and vice versa for the

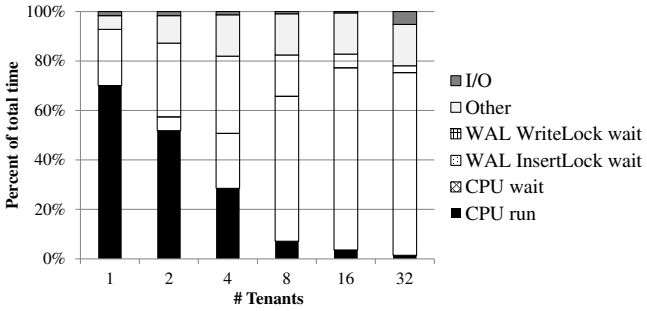


Figure 8: Shared Design with PostgreSQL: Time-breakdown of a server thread

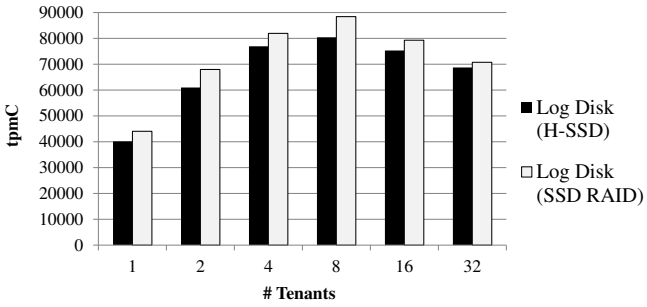


Figure 9: Shared Design with DBX: Comparing the two different disk assignments

data file(s).

The performance with these two disk assignments is shown in Figure 9. Notice that using the SSD RAID device as the log disk outperforms the case when using the H-SSD device as the log disk. (This is the opposite of the best disk assignment in the dedicated PostgreSQL design.) The reason for this behavior is that the shared design does not generate a very random log I/O pattern (as seen in Figure 6), allowing the SSD RAID device to keep up with the log I/O traffic. In addition, in this configuration, using the H-SSD device as the data disk is more efficient (than using the SSD RAID device as the data disk) in handling the checkpointing (dirty page) I/Os.

As one might imagine, it was harder to carry out a detailed time-breakdown analysis of the lock waits in this case (since DBX is a closed-source product). Instead, in Figure 10, we use "Locks" to denote the time that is spent waiting for all types of locks. Comparing this figure to Figure 8, the lock time shown in Figure 10 is much smaller than the shared design with PostgreSQL. For example, when the number of tenants is 16, the shared design with PostgreSQL spends 70% of the total time waiting for the `WAL InsertLock` (shown in Figure 8), while the shared design with DBX only spends 40% of the total time waiting for locks (shown in Figure 10). Thus, lock contention is not a key bottleneck

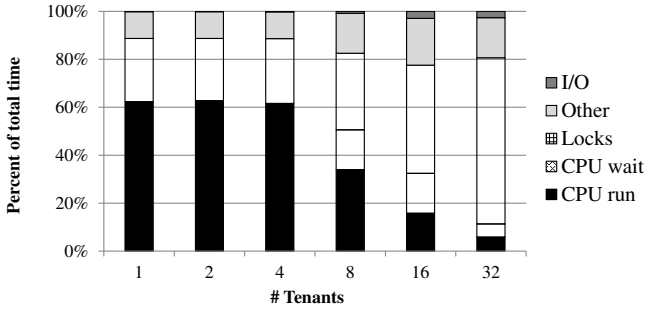


Figure 10: Shared Design with DBX: Time-breakdown of a server thread

for the shared design with DBX.

3.5 Virtual Machine Design

For all the designs based on virtual machines, the highest performing configuration was always the case when the H-SSD device was used as the log disk and the SSD RAID device was used as the data disk, as shown in Table 2. The reason for this behavior is similar to the dedicated design – since the physical log I/O pattern produced by the multiple VMs is random, using the H-SSD device is more effective in handling the log writes, which improves the overall transaction commit rates.

First, we analyze the VMware design with PostgreSQL.

3.5.1 VMware Design with PostgreSQL

The VMware design with PostgreSQL is similar to the one used in [9], in which one DBMS instance was installed in each VM that was supported by VMware ESXi.

As before, we start with a time-breakdown analysis, which is shown in Figure 11. As the number of tenants increases, the sum of the `WAL WriteLock` wait time and the `CPU wait` time constitute a bulk of the total runtime. In our previous analysis (in Section 3.3), we explained that the length of the `WAL WriteLock` wait time is determined by the log disk performance. Here, we present statistics about the log disk I/O, as observed from the guest OS, to explain the relationship between the `WAL WriteLock` wait time and the log disk performance. Table 6 shows the log disk statistics. In this table, we notice that both the average service time (`svctm`) and disk utilization (`util%`) keeps increasing as the degree of multi-tenancy increases from 1 to 32. This means that the log disk gets busier and its performance gradually degrades as the number of tenants increases, which also explains why the `WAL WriteLock` wait time component grows as a fraction of the total runtime (as seen in Figure 11).

Meanwhile, starting from a degree of multi-tenancy of 8, the `CPU wait` time component in Figure 11 starts becoming significant. We speculate that the appearance of the `CPU wait` is partially due to the performance penalty associated

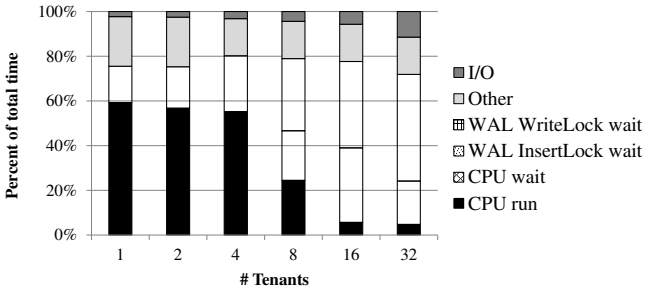


Figure 11: VMware Design with PostgreSQL: Time-Breakdown of a server thread

# tenants	1	2	4	8	16	32
svctm	0.14	0.32	0.39	1.95	3.35	5.83
util%	29.7	46.2	67.7	83.3	98.6	99.4

Table 6: VMware Design with PostgreSQL: Log disk I/O statistics as observed from the guest OS

with virtualizing the CPU resources by VMware. To quantify this penalty, we conducted two benchmarks: (1) a CPU benchmark, and (2) a cache benchmark, which we describe next.

First, we benchmark the CPU by executing the prime numbers test from SysBench [8]. This test is a single-threaded workload that simply computes prime numbers up to 30,000. In this test, we vary the degree of multi-tenancy by concurrently running 1, 2, 4, 8 or 16 instances of this benchmark in the following three configurations: (1) a standard Linux OS with no virtualization, (2) each benchmark instance runs in a VM that is supported by VMware, and (3) each benchmark instance runs in a VM that is supported by Xen.

To quantify the performance penalty associated with this CPU computation, we measured the average time to run this benchmark for each configuration described above (the variance was close to zero, so using the average is a good measure for this test). The results for this experiment are presented in Figure 12. We note that when the number of benchmark instances is larger than 4, the VMware setup is the slowest, and is about 25% slower than the native Linux implementation. In contrast, the Xen method has very low overhead (less than 5%) over the native Linux implementation, likely due to the use of *para-virtualization*, which is designed to reduce the CPU computation overhead associated with virtualization.

Next, we run another benchmark to examine the impact of virtualization on the CPU cache and main memory. This test is a single-threaded random array-lookup test. It first creates an integer array, and then randomly reads one integer from the array at a time. We repeat the "random read" 20,000 times. Since our CPU has a 64 KB L1 cache, a 256 KB L2

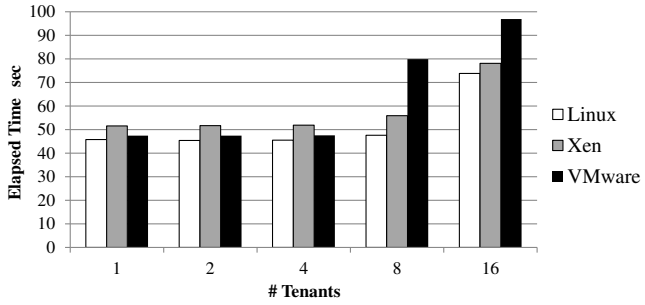


Figure 12: The CPU prime numbers benchmark runs on standard Linux, VMware, and Xen

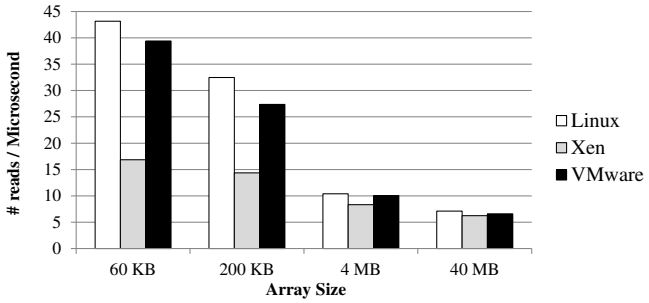


Figure 13: The cache benchmark on standard Linux, VMware, and Xen

cache, and a 8 MB L3 cache, we want to test the "cache" hierarchies (L1 $\rightarrow$ L2 $\rightarrow$ L3 $\rightarrow$ memory) level by level. This benchmark uses 4 different array sizes – 60 KB (less than the L1 cache size), 200 KB (bigger than the L1 cache size, but less than the L2 cache size), 4MB (bigger than the L2 cache size, but less than the L3 cache size), and 40MB (bigger than the L3 cache size, and in-memory array). To fully load all the 8 CPU cores in the server, we concurrently run 8 instances of this benchmark in the following three configurations: (1) a standard Linux OS with no virtualization, (2) each benchmark instance runs in a VM that is supported by VMware, and (3) each benchmark instance runs in a VM that is supported by Xen.

To quantify the performance penalty on the CPU caches and the main memory, we measured the average number of reads per microsecond. The results for this benchmark are shown in Figure 13. Compared to the VMware and naive Linux, Xen has a large overhead when accessing the L1 and L2 caches. However, for the L3 cache and the main memory access cases, Xen is only marginally slower than the VMware and the native Linux configurations.

Note that the VMware design with PostgreSQL is at most 2X worse than the dedicated design (shown in Figure 2). This may be a positive message for cloud operators who prefer using VMware to consolidate the database workloads because of its high isolation between VMs, but are concerned

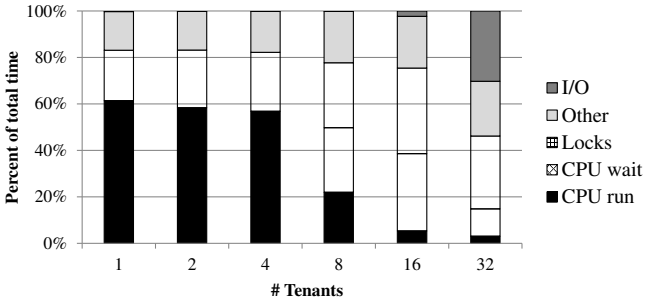


Figure 14: VMware Design with DBX: Time-breakdown of a server thread

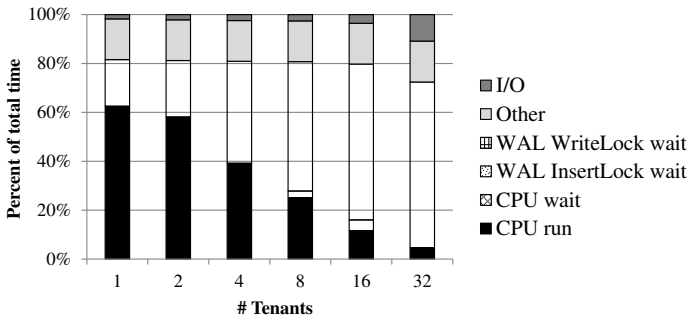


Figure 15: Xen Design with pgSQL: Time-breakdown of a server thread

about the huge virtualization penalty (e.g. the 6X-12X reported in previous studies).

3.5.2 VMware Design with DBX

The VMware design with DBX is a popular deployment with operators who want to use the proprietary VMM and DBMS for their advanced features, better manageability, and higher reliability.

The time-breakdown analysis for this design is shown in Figure 14. Similar to the behavior as seen in Figure11, with 8 or more tenants, the CPU wait time and the lock time contribute a large portion to the total runtime. The appearance of the CPU wait time shown in Figure 14 is primarily because of the CPU virtualization overhead associated with VMWare, which has been described in Section 3.5.1. When there are 32 tenants, the bottleneck starts to shift from the CPU to the I/O, as can be seen in Figure 14.

When considering the performance penalty associated with this design, Figure 2 shows the VMware design with DBX is within 2X slower than the shared design with DBX, mostly because of the CPU virtualization penalties. Previous study [9] shows 6X-12X performance penalty when running the TPC-C workloads in VMware on traditional HDD-based subsystems. However, an interesting finding of this work is that when switching to the SSD-based I/O subsystem, the new challenge with this VMM design is how to reduce the CPU virtualization-related penalties.

3.5.3 Xen Design with pgSQL

Contrasting to the VMware design with DBX, the Xen design with pgSQL is a low upfront-cost deployment which uses the open-source VMM and DBMS.

As seen in Fig. 2, the Xen design with pgSQL is the best VM-based design which is only about 1.3X slower than the dedicated design when the number of tenants is less than 16. To explain this result, we look at the time-breakdown of a server thread, as shown in Fig.15. In this figure, we notice that the cpu wait time is much smaller that the case when using the VMware as the VMM (as seen in Fig.11).

# tenants	1	2	4	8	16	32
svctm	0.15	0.18	0.25	0.35	0.49	0.94
util%	19.3	38.1	57.6	79.7	89.3	99.9

Table 7: Xen Design with pgSQL: Log disk I/O statistics as observed from the host OS

In Fig. 12, we have compared the CPU virtualization overhead of Xen and VMware with the Linux, and our conclusion is: Xen only has very low overhead over the Linux on CPU computation, primarily because the host OS uses the *para-virtualization* technology.

As seen in Fig.15, when there are more than 4 tenants, the `WAL WriteLock` takes most of the runtime and this is mostly caused by the degraded log I/O performance. As the CPU virtualization overhead is relatively smaller (compared to the VMware designs), the workloads begin to stress the log disk (H-SSD), which is critical to the performance. To better understand the log disk performance degradation in the Xen design, we show the log disk statistics from the host OS level in Table 7, which are comparable to the log disk statistics observed from the dedicated design (shown in Table.4(a)).

Compared to the log disk statistics (in Table 4(a)) observed from the dedicated design, the log disk in the Xen design shows higher average service time (`svctm`) and disk utilization (`util%`) in most cases. For example, when there are 4 tenants, the log disk utilization in the Xen design is 57.6%. While for the dedicated design, it is 47.8%. When consolidating 32 tenants, the average service time (`svctm`) of the log disk in the Xen design is 0.94, which increases about 2X from the 16 tenants (0.49).

In conclusion, since Xen has low CPU virtualization penalty, it shows very competitive performance (1.3X overhead) to the dedicated design. Now a new challenge for the Xen design is how to efficiently virtualize the (fast) I/O device (H-SSD), when it is ported as many virtual disks to handle the random and synchronous log I/Os..

3.6 Increasing the Database Size

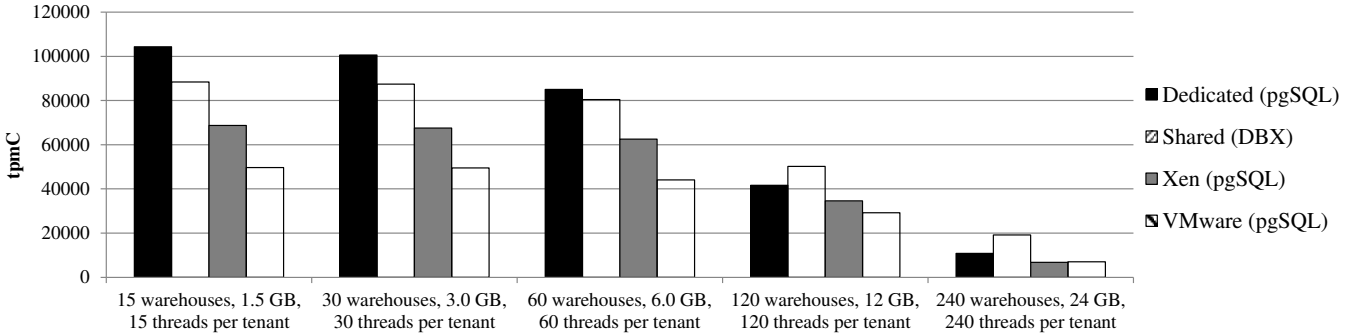


Figure 16: Vary the tenant's database size (8 tenants): Comparison of four designs, each at their best performance configuration

So far we have considered the case when the database working sets across the tenants fits in server memory, which is generally the case with high-performance OLTP workloads. In this section, we relax this assumption and consider the impact of disk-resident data access.

We consolidate 8 tenants on the same server (because most designs show their best performance with 8 tenants, as shown in Figure 2), and the data size for each tenant is varied from 1.5 GB with 15 threads to 24 GB with 240 threads. The size of each warehouse remains the same at 100MB (as in the experiments above).

Here we use four best-performing designs as example, and show their performance in Figure 16. These four designs are: the dedicated design with PostgreSQL, the shared design with DBX, the Xen design with PostgreSQL, and the VMware design with PostgreSQL.

In Figure 16, we show the performance of the four designs with different tenant database sizes. When the database grows from 15 to 30 warehouses, the main memory is still larger than the working set, and we observe that the performance of each design is not strongly affected by this data growth.

As the data size (i.e. # warehouses) continues to grow beyond 60 warehouses, the working set size becomes larger than the main memory size, and query execution now needs to read data from the disks. As can be seen in Figure 16, the performance of each design starts to drop at 60 warehouses. At 120 and 240 warehouses, the data disk I/O (including the read I/O) becomes the bottleneck in each design. In Table 8, we present both the log and data disk I/O statistics for the dedicated design with PostgreSQL and the Xen design with PostgreSQL (as representative results).

As seen in Table 8, at 120 and 240 warehouses, the utilization (`util%`) of data disk is almost 100%, indicating that the system is bottlenecked at the data disk. Since the data disk is now the bottleneck, the log disk is less busier. For example, as can be seen in Table 8(b), with Xen, the log disk utilization (`util%`) decreases from 84.7% to 41.8% when the number of warehouses increases from 60 to 240. Thus,

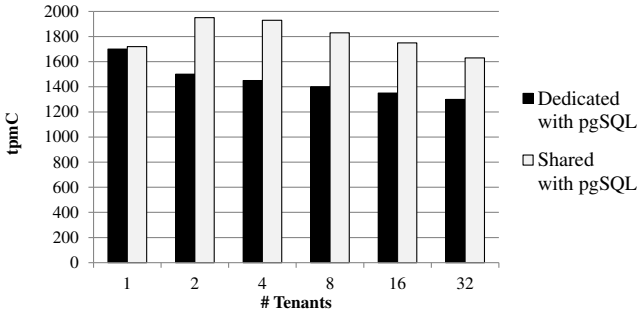


Figure 17: Comparing the shared and the dedicate designs with an HDD-based I/O subsystem

in this setting, when the working set is not main memory resident, the data disk is prone to becoming the bottleneck (because of the large amount of read I/Os).

Additionally, when the number of warehouses is 120 or 240, as seen in Figure 16, the shared design with DBX outperforms all other designs. We infer that this is because when the (checkpointing and read) I/O traffic to the data disk I/Os becomes the dominant performance factor, the DBMS buffer management becomes very important for overall performance. The shared design with DBX manages a single uniform buffer pool, so that it can better utilize it to cache the most important data, to minimize the I/O traffic, and to generate a better pattern of write to the I/O device. While for the other designs (e.g. dedicated and VM-based), each DBMS manages its own buffer pool and potentially loses this global view.

3.7 Shared and Dedicated Designs with PostgreSQL and HDD-based I/O subsystem

To connect with the previous study [9], we compare the shared and dedicated designs with PostgreSQL when using an HDD-based I/O subsystem. We use two identical SATA-based HDDs (1TB 7200 RPM WD Caliver Black disks), and put the log and the data files separately on each drive. Here the tenant has 1.5 GB data with 15 server threads.

	Log disk (H-SSD)					Data disk (SSD RAID)				
# warehouses	15	30	60	120	240	15	30	60	120	240
svctm	0.19	0.21	0.43	0.60	0.67	0.07	0.08	0.12	0.10	0.07
util%	75.6	91.8	98.6	83.4	61.2	14.2	31.3	69.1	96.5	100

(a) Dedicated design with PostgreSQL: Log and disk I/O statistics as observed from Linux

	Log disk (H-SSD)					Data disk (SSD RAID)				
# warehouses	15	30	60	120	240	15	30	60	120	240
svctm	0.35	0.47	0.68	0.59	0.45	0.05	0.09	0.13	0.08	0.07
util%	79.7	83.4	84.7	68.2	41.8	10.4	35.8	77.2	98.9	100

(b) Xen design with PostgreSQL: Log and disk I/O statistics as observed from host OS

Table 8: Increasing the database size: Disk I/O statistics

From Figure 17, we notice that when using an HDD-based I/O subsystem, the shared design performs better than the dedicated design. By observing and comparing the I/O statistics collected from the two designs, we reached similar conclusion as [9], namely that the shared design can reduce the I/O loads on both the log and the data disks by coordinating the I/O requests across all tenants. On the other hand, since each tenant in the dedicated design makes independent decisions on when to flush the dirty pages, and the I/O pattern to the log disk is random, the dedicated design performs poorly with an HDD-based I/O subsystem.

Recall that when analyzing the shared design in Section 3.4, we found that the lock contention becomes a significant problem when the I/O subsystem is not the bottleneck (because of the SSDs). However, for the HDD-based I/O subsystem, we did not observe serious lock contention because the "new" performance bottleneck is now the I/O subsystem.

In [9], the authors observed that the shared design has 6X-12X higher throughput than the VMware design with MySQL. We draw similar conclusions with PostgreSQL, and believe that this behavior is true with HDD-based I/O subsystem, especially since the VMware design introduces additional overheads due to virtualization.

4. RELATED WORK

There has been a flurry of recent work on examining various aspects of multi-tenancy for database workload, in large part driven by the move towards cloud database services. For example, Soror et al. [16] proposed to use VMs to consolidate mixed workloads and provided a virtualization design advisor to split the physical computing resources, and provision each VM in order to achieve the best aggregate performance. However, that work does not consider the performance overheads associated with using VMs, or compare the performance of their VM-based solution to the solutions that do not use a VM.

There are two leading work that are closely related to this paper. Curino et al. [9] proposed to consolidate multiple DB workloads using a "share-everything" solution (a single DBMS instance) without virtualization. They proposed a consolidation technique, called Kairos, to measure the hardware requirements of the workloads and predict the combined resource utilization of all workloads. The authors compared Kairos with a VMware-based solution, and concluded that Kairos has a factor of 6X-12X higher throughput than the VMware-based solution when using traditional HDDs. Their reason for this performance gap was the overhead of using multiple VMs for log and data writes, as well as the frequent and expensive context switches. In this paper, our results match their conclusion when the I/O subsystem uses traditional HDDs. However, a big motivation of our work is to revisit this issue when considering pure SSD-based I/O subsystems.

The other work [14] performed a detailed performance analysis of OLTP deployments in servers with multiple cores per CPU (multicore) and multiple CPUs per server (multi-socket). They compared different database deployment strategies where they varied the number and size of independent database instances running on a single server, from a single shared-everything instance to fine-grained shared-nothing configurations. Similar to their work, we systematically compare and evaluate OLTP deployments on a single server. However, their work focused on optimizing OLTP deployments on different CPU architecture designs, while our work focuses on comparing the popular multi-tenant OLTP deployments on an SSD-based I/O subsystem.

Recent work has also examined the impact of running database workloads in a VM. Minhas et al. [10] conducted an experimental study to examine the overhead of running a single-tenant TPC-H workload in a VM supported by Xen, and showed that the average overhead was less than 10%, compared to the case without virtualization. Our work is complementary to their work since we focus on the multi-tenant OTLP workloads instead of single-tenant OLAP workload.

To our best knowledge, there isn't a comprehensive previ-

ous study on evaluating popular multi-tenant OLTP designs on SSD-based I/O subsystems.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have examined various multi-tenancy designs for OLTP workloads on a modern SSD-based I/O subsystem. We make the following key observations: (1) The VM-based design shows fairly competitive performance to the non-virtualization based design (from 1.3X to 2X performance degradation) on a modern SSD-based I/O subsystem, which is far lower than shown in previous studies that considered traditional HDD-based I/O subsystems. Thus, when switching to an pure SSD-based I/O systems, the performance penalty that holds for traditional HDD-based I/O systems may not hold. (2) The shared design has a performance advantage over the VM-based designs on an SSD-based I/O subsystem, as long as it can minimize the contention on the shared resources among the workload threads. (3) The dedicated design with PostgreSQL presents the best performance compared to all other designs, for the TPC-C like workload that we considered in this paper (which is easily partitionable). This is mostly because it can avoid the resources contention problem in the shared design [11, 14], and the performance penalties associated with the VM-based design [9].

There are a number of directions for future work, including dealing with dynamic workloads, expanding to more complex and mixed workloads, examining mixed devices (HDDs and SSDs), and extending this study to conduct a full price v/s performance analysis of various solutions that includes aspects such as amortized hardware costs and software licensing costs.

6. REFERENCES

[1] `http://www.xen.org/`.

[2] `http://www.vmware.com/`.

[3] `http://oltpbenchmark.com/experiments/dbms_config/postgresql.conf`.

[4] Best practices for xen. `http://wiki.xensource.com/xenwiki/XenBestPractices`.

[5] Flash drives replace disks at amazon, facebook, dropbox. `http://www.wired.com/wiredenterprise/2012/06/flash-data-centers/`.

[6] Oltpbenchmark. `http://oltpbenchmark.com/wiki/index.php?title=Main_Page`.

[7] Performance best practices for vmware vsphere 4.0, vmware esx 4.0 and esxi 4.0, vcenter server 4.0. `http://www.vmware.com/pdf/Perf_Best_Practices_vSphere4.0.pdf`.

[8] Sysbench: a system performance benchmark. `http://sysbench.sourceforge.net/`.

[9] C. Curino, E. P. C. Jones, S. Madden, and H. Balakrishnan. Workload-aware database monitoring and consolidation. In *SIGMOD Conference*, pages 313–324, 2011.

[10] U. F. Minhas, J. Yadav, A. Aboulnaga, and K. Salem. Database systems on virtual machines: How much do you lose? In *ICDE Workshops*, pages 35–41, 2008.

[11] I. Pandis, P. Tözün, R. Johnson, and A. Ailamaki. Plp: Page latch-free shared-everything oltp. *PVLDB*, 4(10):610–621, 2011.

[12] A. Pavlo, C. Curino, and S. Zdonik. Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In *SIGMOD*, pages 61–72, 2012.

[13] A. Pavlo, E. P. Jones, and S. Zdonik. On predictive modeling for optimizing transaction execution in parallel OLTP systems. *Proc. VLDB Endow.*, 5:85–96, 2011.

[14] D. Porobic, I. Pandis, M. Branco, P. Tözün, and A. Ailamaki. Oltp on hardware islands. *PVLDB*, 5(11):1447–1458, 2012.

[15] J. Schaffner, T. Januschowski, M. Kercher, T. Kraska, H. Plattner, M. J. Franklin, and D. Jacobs. Rtp: robust tenant placement for elastic in-memory database clusters. In *SIGMOD Conference*, pages 773–784, 2013.

[16] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *SIGMOD Conference*, pages 953–966, 2008.