A Quantitative Comparison between Raw Devices and File Systems for implementing Oracle Databases

Oracle/HP White Paper

April 2004

1. Introduction

The debate on whether to use a file system or raw devices for implementing databases still rages among Oracle database administrators. Most database administrators are familiar with the UNIX file system implementation, and prefer the ease of management of a file system as compared to a raw implementation. In addition, all UNIX commands and tools for file manipulation can still be used with a file system implementation. This is not the case for a raw database. Therefore, DBAs frequently choose file systems for implementing databases.

However, there is considerable performance overhead when using a file system to implement a database. The file locking inherent to UNIX file systems (files are exclusively accessed during write operations) and double buffering (once in kernel space, once in user space) introduce significant performance overhead. This paper attempts to quantify these overheads.

Our results indicate that database implementations using raw devices have the best performance, with 92% transactional throughput improvement over file-system based implementations (log option). We believe that administrators will progressively migrate to raw databases as database management tools, such as Oracle Enterprise Manager (OEM), continue to improve.

We used an OLTP workload for our evaluation. We report performance measurements of the Oracle database using raw logical volumes and using file system and explore the effect of using asynchronous and synchronous I/O operations for the raw device implementation. We also investigate various file system optimizations to determine their effect on database performance. We also vary the number of Oracle database writer processes and Oracle I/O slaves processes to get the best performance for a given configuration. The experiments tend to cover all the different combinations possible when implementing an Oracle database. The Oracle Stripe And Mirror Everything (SAME) methodology was used for the database layout for all the tests.

2. Hardware configuration

2.1. Client configuration

We used 2 HP N-class servers, 1 client and 1 database server configured as followed:

8 CPU 440Mhz
16 GB RAM
10 1 Gb Fibre Channel Host Bus Adapters
1 1Gb network card for client access
HP-UX 11i

2.2. Database server configurations

Oracle9i Database Release2
230 GB database
Raw based database
File system based database
Archiving was turned off
12GB SGA
8KB Oracle block size

2.3. Disk configuration

We used 4 HP virtual arrays (VA7400), in Direct Attached Storage (DAS) mode to store the Oracle database. The VA7400 is a mid range array with two controllers, both the controllers and array backend are fiber based. The Virtual Array series is built-in with the SAME concept.

The HP VA is a virtual array -- there is no notion of spindles when allocating space for an object. All the physical disks in the array are put into a storage pool. All files allocated in the VA are striped across all the spindles in the pool. The storage pool allocation unit is 256KB block. The data protection is provided at the array physical extent level. Both RAID01 and RAID6 (double parity) are available. The VA7400 has a maximum of 105 disks with dual active controller. The array cache on the VA is controller-based for a maximum size of 1 GB.

2.3.1 VA7400 configuration

4 VA7400
2 active controllers per VA7400
1GB cache per controller
45 disks per VA7400
36GB/15k rpm disk drives

3. Workload description and test methodology

The workload is an OLTP application with 40% reads and 60% writes. The IO size varies between 2KB and 16KB and the number of clients associated with this workload is 150. The workload was designed to put a reasonable load on both the storage and database server. However, the file system configuration was not able to absorb this load, and, as a result, we also experimented with a medium workload using 50 clients.

During warm up, the workload was run for a period of 80 minutes. The official measured runs lasted 150 minutes. Each test was repeated twice, obtaining consistent results across the runs. The database was refreshed using snapshot technology before each test case via Virtual Array Business Copy and the snapshot was taken off line after the database was successfully created and loaded.

The disk configuration remained constant between the different test cases. We ran our experiments with two different database configurations, raw and file system based and also introduced additional testing for each configuration. The raw database setup was tested with both asynchronous and synchronous IO (default on HP-UX 11i v1). In the case of the file system, we used a combination of multiple IO slaves and file system mount options. The Oracle online logs were stored in raw volumes using HP Logical Volume Manager (LVM) in both raw and file system configuration because internal experiments have demonstrated that logs will take advantage of the absence of file locking and double buffering (data will be copied directly from user space to disk) that are inherent in Unix file systems.

For all the experiments, the Oracle data files and logs were automatically striped across the entire available spindles in any given array and controllers. Each array was configured in RAID01. The HP Logical Volume Manager (LVM) striping was used to provide IO load balancing across the 4 VA7400. This means every logical volume was equally distributed across the 4 VA7400 arrays. The LVM stripe size was 64KB, but in RAID1 configuration the stripe size does not matter for the OLTP application we used in our tests.

4. Performance measurements

We used Oracle and HP performance tools to collect statistics during the entire run for each test case. We compared the different run results using the application throughput measured in number of transactions per minute, the **log_file_parallel_write**, **d**db_file_sequential_read, IO throughput and CPU utilization.

5. Experiments

5.1. Raw-device based database

5.1.1 Asynchronous IO

On HP-UX, asynchronous IO is only supported with a raw device (raw disk partition or raw logical volume), although this will change with HP-UX 11i v3 (internally known as 11.31). With asynchronous IO configured, Oracle can submit multiple IO requests in parallel. In some cases it may be necessary to use multiple database writer processes (**DBWR**) to stress the backend, but we were able to achieve this with only 2 **DBWR** processes. In our case, we did not see any improvement beyond 2 database writer processes. The right number could be determined through the Oracle IO statistics but is beyond the scope of this paper. The usage of asynchronous IO will require HP-UX kernel changes and modification of the Oracle initialization parameters file, init.ora. For more information, consult the Oracle Installation and Administrators Reference Guide documentation on HP.

5.1.2 Synchronous IO

This configuration is not recommended with raw-device implementation of the database. In this case, all the IOs are serialized. The only gain is the elimination of double buffering. We run this experiment only to compare it with the best File System scenario.

5.2. File system based database

HP-UX 11i JFS/OnLine Journaling file system (OEM'd VERITAS VxFS) today only supports synchronous IO. This will change in the next major release of HP-UX 11i. HP's developed Advanced File System (formerly from Tru64 UNIX) will be built-in to HP-UX 11i v3 with async IO capabilities and is targeted for release second half of 2005. In addition, HP will also have a clustered file system option with HP-UX 11i v3.. It is generally recommended to use multiple IO slaves with a file system database because there is a possibility that one **DBWR** may not be sufficient to keep up with a heavy workload. In this case the **DBWR** becomes an efficient scheduler and dispatches the IO between different IO slaves. We used 40 IO slaves in our test after experimenting with different values.

The file system based database has a lot of limitations. The biggest one is the exclusive file locking for write operations. In Unix, files are locked for writes to guarantee data integrity and consistency. This does not allow parallelization of writes. Another limiting factor is the double buffering. The file system has a buffer cache and Oracle also maintains its own cache. The buffers are being copied from kernel space (file system buffer cache) to the user space (Oracle buffer cache). This operation generates a lot of overhead. The file system will also require the management of its metadata and data structures, which adds extra costs.

In order to improve the performance of a file system based database, new mount options were introduced to eliminate the double buffering and limit the overhead induced by file system metadata management. These mount options are only available on HP-UX 11i with the HP OnLine Journaling File System (JFS) option. HP JFS by default does not include the OnLine option, which is part of the Enterprise Operating Environment or an add-on product.. We tested the file system with two different sets of options. The first set included **nodatainlog** and **delaylog** while the second set was comprised of **nodatainlog, delaylog, mincache=direct** and **convosync=direct**. The **nodatainlog** option logs only the inode update information in the intent log and writes the data directly to the file. In the **delaylog** option, some system calls return before the intent log is written. With this mode the writes in the intent log are delayed. The **mincache=direct** and **convosync=direct** allow data to be transferred directly from Oracle buffer cache to disk and disk to Oracle buffer cache. This avoids the double buffering by bypassing the file system buffer cache. For more information on the mount options please consult the man page for mount_vxfs(1m).

We also evaluated the case where the **log** option is used instead of the **delaylog**. This measures the trade-off between no data loss and a potential data loss in case of system crash. We should remind the reader that the delaylog option offers a similar guarantee as the traditional UNIX file system. Log and delaylog options are only meant for file system metadata, but they can influence the amount of data loss during system crash. Our tests showed that the transactional throughput difference between **log** and **delaylog** option is between 8% (medium workload) and 10% (heavy workload).

6. Test results

6.1. File System based Database with multiple IO slaves

6.1.1 Medium Workload (50 clients) vs. Heavy Workload (150 clients) using delaylog and nodatainlog options

	MEDIUM WORKLOAD	HEAVY WORKLOAD
TRANSACTION THROUGHPUT	9051	9847
IO THROUGHPUT	1513	1605
LOG_FILE_PARALLEL_WRITE	1 ms	1 ms
DB_FILE_SEQUENTIAL_READ	12 ms	14 ms
CPU UTILIZATION	51%	52%

6.1.2 Medium Workload vs. Heavy workload using delaylog, nodatainlog, mincache=direct and convosync=direct options

	MEDIUM WORKLOAD	HEAVY WORKLOAD
TRANSACTION THROUGHPUT	19097	20305
IO THROUGHPUT	3048	3203
LOG_FILE_PARALLEL_WRITE	2 ms	2 ms
DB_FILE_SEQUENTIAL_READ	14 ms	19 ms
CPU UTILIZATION	84%	85%

61.3 Medium Workload vs. Heavy Workload using log, nodatainlog, mincache=direct and convosync=direct options

	MEDIUM WORKLOAD	HEAVY WORKLOAD
TRANSACTION THROUGHPUT	17654	18421
IO THROUGHPUT	2595	2904
LOG_FILE_PARALLEL_WRITE	2 ms	2 ms
DB_FILE_SEQUENTIAL_READ	16 ms	18 ms
CPU UTILIZATION	86%	88%

6.2. Raw-device based database with multiple DBWR

6.2.1 Medium Workload vs. Heavy Workload in Raw based database without async IO driver

	MEDIUM WORKLOAD	HEAVY WORKLOAD
TRANSACTION THROUGHPUT	23135	28168
IO THROUGHPUT	3316	3898
LOG_FILE_PARALLEL_WRITE	9 ms	9 ms
DB_FILE_SEQUENTIAL_READ	9 ms	19 ms
CPU UTILIZATION	80%	88%

6.2.2 Medium Workload vs. Heavy Workload in Raw based database with async IO driver

	MEDIUM WORKLOAD	HEAVY WORKLOAD
TRANSACTION THROUGHPUT	31961	35367
IO THROUGHPUT	4877	5392
LOG_FILE_PARALLEL_WRITE	4 ms	8 ms
DB_FILE_SEQUENTIAL_READ	12 ms	26 ms
CPU UTILIZATION	79%	91%

7. Result Analysis

The results clearly demonstrate that a file system based database with the mount options (log, and nodatainlog) would be the worst case. There are several reasons for this. The first cause for this result is the double buffering: copying data buffers between the file system buffer cache and Oracle buffer cache is causing a high overhead. Second, a write to a file is exclusive due to the inherent properties of the Unix file system. Third, all the writes are serialized, one IO request after the other. The only parallelism for the writes in this case is obtained using multiple data files (in our tests we had 42 datafiles).

When the double buffering is avoided using **mincache=direct** and **convosync=direct** in conjunction with **nodatainlog** and **delaylog** or **nodatainlog** and **log**, the performance improvement is considerable. Both the IO and the transactional throughput improved by 80% to 100% for the Medium and Heavy workloads. This indicates that double buffering must be avoided if the only choice available to a customer is a file system based database implementation. The issue with the exclusive locking for writes operations still remains. A good recommendation would be to create multiple data files to increase the number of concurrent IOs and lessen the impact of the exclusive file locking inherent to Unix File System. For write intensive applications, the tablespaces impacted should be spread across multiple datafiles. The number of IO slaves should be proportional or close to the number of datafiles. .

We should acknowledge that running with the **delaylog** option represents a certain risk in the sense that there is a potential data loss if a system crashes. The metadata information is not synchronously written in the intent log. Our experience has shown at most 10% transactional degradation with the **log** option.. This is workload dependent. If data loss is unacceptable one should always run with the **log** option, but the performance price is not negligible .

The raw database transactional throughput with async IO has a performance improvement for the Heavy Workload of 92% over the file system database with the following options:**log, nodatainlog, mincache=direct and convosync=direct.** . The ease of management should not justify the choice of a file system database. Today, Oracle Enterprise Manager simplifies the complexity of database management regardless of the choice of a raw or file system database.

We expect that customers will always use async IO with raw databases. Our intent was to show the improvement provided by the async IO driver. It would be interesting to evaluate the HP implementation of asynchronous IO on top of an HP VxFS file system. The expectation is to offer performance close to raw with async. This is a clear indication that the best choice is to use raw device with async IO.

We would like to acknowledge that we chose this write intensive workload to show the great differences that could exist between raw and file system. Whenever ease of management is not an issue and there is no application constraint, customers should always implement raw based database with async IO on HP-UX.

8.References

For more information, consult the following references:

Oracle Technology Network
http://otn.oracle.com/deploy/performance/pdf/opt_storage_conf.pdf

Oracle technical white papers
http://technet.oracle.com/deploy/availability/techlisting.html

9.Acknowledgements

We would like to acknowledge Ahmed Alomari, Mamdouh Ibrahim, Vineet Buch, Herve Lejeune and Mustafa Uysal's collaboration on this paper.

Authors: Baila Ndiaye, Xumin Nie, Umesh Pathak, Margaret Susairaj