# An integrated Architecture for Business Intelligence support from Application down to Storage

André Brinkmann, Sascha Effert,
Michael Heidebuer, Mario Vodisek
University of Paderborn, Germany
Email: brinkman@hni.upb.de

Henning Baars

University of Stuttgart, Germany
Email: henning.baars@wi.uni-stuttgart.de

## Abstract

*Recent developments both in the business and the technological domain have led to a significant increase in demand for Business Intelligence (BI) infrastructures that can handle huge amounts of data in small time frames. BI applications are increasingly used by large user bases on all management levels; support tasks spanning the complete value chain are based on transactional data and are directly coupled with operational systems in "closed loop" approaches.*

*To effectively handle the resulting data volume turns out to be an extremely challenging task which encompasses a variety of issues on different levels. We propose an integrated multi layer tool for monitoring, benchmarking, analyzing, and optimizing the performance of such BI infrastructures.*

*Inside this paper we give a coarse outline of the tool's architecture and demonstrate the value of distinct measurement points at operating system layer. For that purpose we introduce a kernel based benchmark environment and present first measurement results. The gathered data clearly indicates that a meaningful analysis of performance benchmarks without kernel trace points is of limited value - which shows the necessity to consider a separate component within the tool's architecture.*

## 1 Introduction

The term *Business Intelligence* (BI) refers to an integrated approach to Management Support. BI infrastructures nowadays comprise of *Data Warehouses*, which provide a semantically and syntactically coherent data pool, *Data Marts* with application specific data extracts, and a multitude of analytic Management Support applications, e.g. with functionality for Online Analytical Processing (OLAP), data mining, and reporting [12].

In the last years the scope of Business Intelligence has significantly widened - regarding the involved business functions and processes, the number of users and the supported hierarchy levels. Following the common trend to an increased enterprise application integration, BI infrastructures now also provide support for lower management levels and more and more blur with operational system which are tightly coupled with BI systems [9], e.g. in the field of Customer Relationship Management (CRM).

Alongside these developments comes a growing demand for higher data granularity as well as for a more timely data provision. This leads to data warehouses which are supposed to keep data on transactional level and that are expected to provide this data in real time.

New technologies for automatic identification, esp. the emerging EPC-standards for RFID tags, will accumulate data on the time and location of individual article units. This results in data warehouses of the size of several hundreds of TByte instead of few gigabytes [10]. In such environments it becomes vital to be able to closely and continuously monitor, steer and optimize the extraction, transformation and loading *(ETL)* of data as well as data provisioning and analysis.

By now some interesting insights have been gained on efficient logical data models (see [13] for an overview), on physical database design (e.g. by data warehouse partitioning [8]), and on the optimization of typical BI data base queries (e.g. in [18]). All these approaches are limited in scope and can not adequately meet the before mentioned demands: Performance has to be managed on several levels in parallel - from the choice of physical disks up to the policies that regulate usage behavior [5]. To effectively and efficiently handle this task, an integrated tool is required that is able to provide all relevant data for BI performance management. This tool should automatically adjust relevant settings wherever possible.

The outline of this paper is as follows. In section 2 we present a multi layer architecture for a data warehouse performance monitoring and optimization tool. The paper will

focus on demonstrating the necessity of a distinct component on the operating system layer for storage performance measurement and optimization. After presenting the storage management environment V:DRIVE in section 3, section 4) discusses the issues of performance measurement inside virtualized SAN environments and introduces kernel based measurement points for storage benchmarking. Section 5 presents concrete measurement results that clearly indicate the importance of those measurement points

## 2 Architecture

As discussed in the previous section, there is a growing demand for data warehouses of the size of several hundred TByte. To handle these enormous amounts of data within reasonable time frames, it is necessary to closely couple, monitor, and optimize the BI application and the underlying business processes.

### 2.1 High Level Architecture of an integrated Approach to support Business Intelligence Applications

As discussed before, performance management has to span several distinct layers. Critical to the performance are:

- The storage system
- The operating system and the network environment,
- The storage application,
- The database system including physical and logical data base design,
- The queries to access the data base system from the analytical applications, and
- The Policies that channel usage behavior

A failure on any of these components can easily lead to insufficient system performance and decrease the possibility to appropriately react to changes of the business process.

The foundation of any optimization process is a solid measuring and analysis of the effects leading to bottlenecks inside the system. The challenging tasks is to detect the interdependencies between the different process layers of the application framework (see Fig. 1). Knowledge on placement of database records can e.g. easily decrease storage access times; the requirement is that this knowledge is submitted consistently from the higher level application to the storage management environment.

Database applications tended to use this high level information by taking direct influence on the data placement

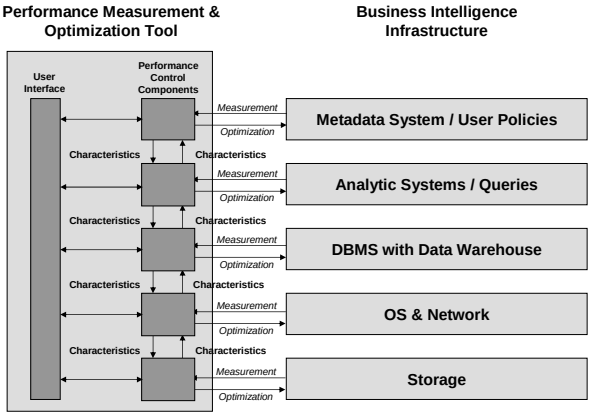



**Figure 1. Integrated Architecture.**

itself by using its own cache system and accessing disk subsystems directly. The transfer from smaller, disk based environments to large scale storage networks removes this ability of the database application. By using storage virtualization environments, the direct view on storage layout is changed to an abstract usage of storage capacity, requiring new interfaces between the different application layers, being integrated in a framework environment.

## 3 V:DRIVE storage management environment

Inside this section we give a brief introduction into the V:DRIVE storage virtualization environment for Linux 2.4 and Linux 2.6. This introduction is restricted to the properties of V:DRIVE which are required for the understanding of this paper. For a more detailed description of V:DRIVE, see e.g. [3].

In V:DRIVE, physical volumes are grouped in storage pools. These storage pools are not accessed directly, but by the abstract concept of virtual volumes which are exported to the accessing servers. The properties of a virtual volume have not to be related to the properties of the underlying storage pool. It is possible to create a virtual volume with a capacity much bigger than the capacity of the storage pool, unless the used capacity of the set of virtual volumes does not exceed the physically available capacity of the storage pool. Each virtual volume can be concurrently used by an arbitrary number of servers.

The capacity of each disk in a storage pool is partitioned into minimum sized units of contiguous data blocks, so called *extents*. The extents are distributed among the storage devices according to the randomized *Share* strategy which is able to guarantee an almost optimal distribution of the data blocks across all participating disks in a storage pool

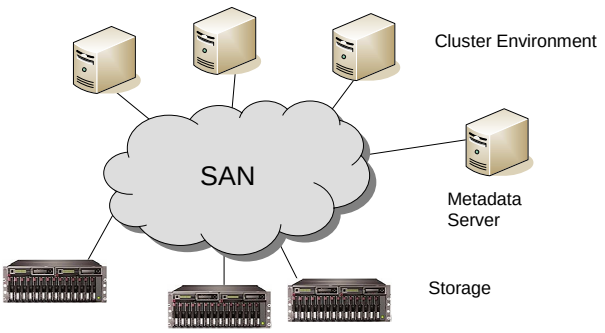



**Figure 2. V:DRIVE environment.**

(see [4]). The typical size of an extent varies between 4 MByte and 512 MByte.

The core component of V:DRIVE is a clustered metadata appliance that stores and distributes information about the SAN environment. This information includes all physical volumes accessible from attached servers, the storage pools and virtual volumes, and the set of extents which are already assigned to the different virtual volumes (see Fig. 2).

Inside the V:DRIVE environment, each Linux server is running a small driver module that presents its virtual volumes to the host operating system. Each time the server accesses an address of a virtual volume that belongs to a new extent, the server has to send a request for the location of the extent to the metadata appliance. The access is delayed until the reception of a valid extent location answer from the metadata appliance. To speed up following request to the extent, the extent location is stored inside an extent cache. The communication between the metadata appliance and server is done through standard TCP/IP sockets.

## 4 Measuring Storage Performance inside SAN environments

Inside this paper we will concentrate on measuring and analyzing storage performance inside our performance optimization framework for Business Intelligence applications and we will show why standard measurement environments are not suited to analyze **and** optimize storage performance for large scale storage environments.

### 4.1 Restrictions of standard Benchmarking Environments for large scale storage environments

Storage benchmarks environments can be distinguished in general in synthetical benchmarks, producing synthetic I/O, and in application benchmarks, which are based on real world programs and measurements (see e.g. [11] [17] [6] [15] [1] [16]). Most of these environments for measuring storage performance are restricted to be set up on top of the file system or block level interfaces of an operating system. Working on top of these interfaces, the benchmarks do not only measure the performance of the storage subsystem, but also the quality of the caching mechanisms inside the operating systems, being dependent on the size and speed of the computer memory subsystem. These measured values help to identify performance problems inside a storage environment, but they are only of restricted value to analyze the reasons for these bottlenecks inside the storage environment, especially in large scale SAN environments (see section 5).

Main restrictions of standard benchmarking environments are:

1. Caching effects are not taken into account
2. The influence of operating system based pre-fetching can not be analyzed without changing OS parameters themselves
3. The influence of storage virtualization and data distribution schemes can not be analyzed

The third item becomes especially important in large scale storage environments. Most of these environments are based on storage virtualization that builds pools of storage devices and presents them as a single, large device to the operating system. The storage virtualization itself can be based inside the host, inside the network or inside the storage devices. In each case, for benchmarking these environments it is important not to see the virtual volumes as a single device, but as a collection of devices with different characteristics and potentially different loads.

The reason can be seen in fig. 3. Being set up on top of a virtual device can help to identify the fact that performance

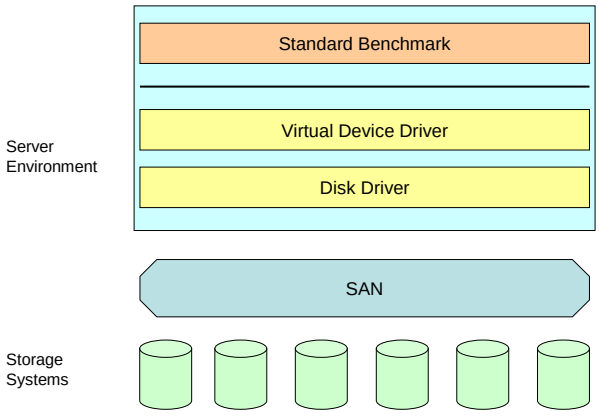



**Figure 3. Standard Benchmark Environment.**

bottlenecks exists; it is not helpful on identifying underlying reasons, especially if they are based on an insufficient data distribution about the physical storage systems [2]. The benchmark environment is not able to detect, which real device is performing a data access, it is only able to see the resulting access time for the whole environment.

### 4.2 Block Level Interfaces inside Linux 2.6

To be able to measure the influence of data distribution, caching effects and pre-fetching, it is necessary to enhance existing data structures inside the kernel by dedicated measuring information.

Inside this section we focus on data structures inside the Linux Kernel 2.6 environment which help to develop benchmarking environments. The described interfaces are also available, under other names, but in similar contexts, in most modern operating systems.

We assume in the following that the measuring is done as part of the virtualization environment itself. All of the described schemes can also be implemented inside of pseudo-drivers, sitting between the virtual device driver and the disk driver.

Each access to a storage device inside the Linux 2.6 kernel is described as a `bio`-structure [7]. This structure does not only contain information about the physical and logical parameters of a block access, but gives the kernel programmer also the ability to integrate its own fields, holding valuable information about the path of the access through the storage hierarchy (see Structure 1).

The `bi_private` field is a dedicated field of type `void * ` for integrating driver specific information inside the bio structure. The measuring environment requires a struct `bench_info` with five fields (see Structure 2).

The first field `start_time` contains the information about the start time of this access. The `start_time` is given in so called *jiffies* after system start, where a jiffie is defined as the time between two timer interrupts (see e.g. [14], Chapter 10).

The vddev field points to the virtual device of this request

**Structure 1** Buffer IO description inside Linux 2.6

```
struct bio{
      sector_t              bi_sector;
      struct block_device   *bi_bdev;
      unsigned long         bi_rw;
      ...
      bio_end_io_t          *bi_end_io;
      ...
      void                  *bi_private;
      ...
};
```

**Structure 2** Additional private information

```
struct rad_bio_private {
      unsigned long start_time;
      rad_vddev     *vddev;
      rad_rdev      *rdev;
      bio_end_io_t  *end_io_old;
      void          *private_old;
};
```

and the field rdev points to the underlying physical device that has served the request.

The fourth field contains a pointer of type `bi_end_io` and the fifth field contains a pointer of type `void * ` to a private field. These fields are necessary to store the information of the driver originally creating this request, e.g. information stored by the Linux md RAID driver.

The field `bi_end_io` inside structure 1 points to a function that is called after the request defined inside the bio structure has finished. The standard `bi_end_io` function inside the kernel checks the result of the storage access and, if the I/O has really finished, returns the bio structure to a pool of pre-allocated memory. Redirecting this function to a function inside the virtualization module enables the module to analyze the outcome of the request and to store the gathered information inside internal data structures.

After these tasks the new end_io function has to reset the preserved fields `bi_end_io` and `bi_private` and to redirect the bio structure back to the calling instance.

### 4.3 Calculating statistical values inside V:DRIVE

Based on the annotated requests and the number of these requests, the following key values are calculated inside V:DRIVE:

- Latency: The access latency is defined as the time frame from receiving the request inside the V:DRIVE environment until the endio() function is called. This time can be calculated by a simple subtraction.
- Total accessed storage capacity: How much data has been written to / read from the storage systems in an interval / since the beginning of the measurements. The access size can be increased by a simple add.
- Throughput: The throughput for a given interval can be directly calculated by taking the total access size and divide it by the size of the interval.

To measure performance as exact as possible, it is necessary to be aware of the performance impact of the measurements themselves. In the following, we present the calculations done inside the V:DRIVE environment each time the endio() function is called.

Latency and total accessed volumes can be calculated by simple sub / add commands and the maximum latency can be received by a simple comparison. The throughput can be directly calculated from the accessed storage capacity by dividing the accessed capacity by the time required to access the capacity. This division can be performed offline.

The most costly calculation is the standard deviation $\sum_{i=1}^{N}(x_i - \bar{x})^2$ for the latency. Inside V:DRIVE it is calculated based on the standard calculation formula for running sums:

$$\sigma_x = \sqrt{\frac{N \times \sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}{N \times (N-1)}},$$

where $x_i$ is the latency of the currently finished access and $N$ is the total number of accesses finished so far. This calculation only requires to add $x$ and $x^2$ to their corresponding sums for each access and to evaluate the formula if required. The evaluation of the complete formula is only necessary when the statistical values are presented to the /proc-file system of the Linux operating system (see next section). Even in this case, the calculation of the square root is omitted and only the variance $\sigma_x^2$ is calculated.

All information required to calculate the benchmark information is stored inside a `bench_info` structure. This structure is instantiated three times for each virtual volume and for each physical volume; once for the continuously updated information, once for the last time interval that is presented as actual value, and once for the current time interval.

The time interval can be chosen arbitrarily, based on a smallest time interval unit of a single jiffy. In the following, we will set it always to 10 seconds.

### 4.4 Presenting, manipulating and analyzing the gathered information

The gathered information can be exported and manipulated in many different ways. In the following subsection, we will present the interfaces to the kernel based storage benchmarking environment inside V:DRIVE:

**/proc-file system** The /proc-file system inside the Linux kernel is an interface to export information about the system. V:DRIVE uses the /proc-file system to present the gathered benchmarking information in two different files:

The proc-entry `vdrive` exports information about all storage pools and the physical volumes assigned to them:

```
rivercola:~ # more /proc/vdrive/vdrive
Storage Pool ID: 1458
Capacity: 58630 MByte
Virtual Volumes using Storage Pool: 1
Disks: 5
Extent size = 32 MB
   Disk ID: 18
        Disk (18) Capacity = 11727 MByte
   Statistical Values Average
        access number = 78396
        accessed capacity = 4802 MByte
        max latency = 1911 ms
        average latency = 290 ms
        latency variance = 113806 ms^2
        average throughput = 20 MByte/s
   Statistical Values last Interval
        access number = 2842
        accessed capacity = 324 MByte
        max latency = 1753 ms
        average latency = 851 ms
        latency variance = 246945 ms^2
        average throughput = 32 MByte/s
   Disk ID: 21
        Disk (21) Capacity = 11727 MByte
    ...
```

In the example that will be used throughout this paper, only a single storage pool is used. The physical capacity of the pool is 58 GByte, provided from 5 different physical volumes. The extent size is 32 MByte and only a single virtual volume is accessing the pool.

This /proc-file system entry presents the most important statistical values for all of its physical volumes, including the access number, the accessed capacity, the maximum and average latency, the variance of the latency and the average throughput. All these information is given for the physical volumes as average for the time since creating the storage pool and for the last interval.

The proc-entry `virtualvolumes` contains information about all virtual volumes accessible for the server:

```
rivercola:~ # more /proc/vdrive/virtVolumes
VDrive configuration:
  virtual device(0): /dev/rada
        id = 2175
        capacity = 51200 MB
        pool id = 1458
    Statistical Values Average
        access number = 801035
        accessed capacity = 57419 MByte
        max latency = 3080 ms
        average latency = 315 ms
        latency variance = 184851 ms^2
        average throughput = 96 MByte/s
```

Inside our example, only a single virtual volume is used. Besides configuration information for the virtual volume, the presented statistical values are similar as given for the physical volumes. It is important to consider that each virtual volume is able to access a large number of different physical volumes and each physical volume can be accessed

by more than a single virtual volume. Therefore, information gathered from the /proc-entry `vdrive` and the /proc-entry `virtualvolumes` are correlated with each other.

**IOCTL** An I/O control call can be send in general from an arbitrary user space program to any kernel module. Inside V:DRIVE, I/O control is used to set the properties of the benchmark environment. It is e.g. possible to reset all benchmark information.

# 5 Measurements

Inside this section we present measurement results indicating the value of benchmark extensions for the Linux kernel environment. After evaluating the overhead of the benchmark extensions, we will give two examples, where only kernel extensions help to analyze the storage behavior: First in analyzing the influence of the buffer cache and second in analyzing the distribution of data accesses among different physical disks which are grouped to a single virtual disk.

## 5.1 Measurement Environment

The measurements are based on the V:DRIVE storage management environment, presented in section 3. The metadata-appliance is running a 1 GHz Pentium III processor and has 1 GByte main memory and 30 GByte disk. The benchmark server is running a dual-processor 2.4 GHz Xeon processor and has 1 GByte main memory and a 40 GByte hard disk. Both servers are using a SUSE Linux 9.3 Professional operating system with kernel version 2.6.11.

The benchmark server is connected to a SAN environment through a Brocade Silkworm 3800 Fibre Channel switch. The disks accessed during the following measurements are 78 GByte Seagate ST 37320FC Fibre Channel disks which are enclosed inside a Transtec 3000 JBOD array. All disks are partitioned into 3 12 GByte partitions.

All measurements presented inside this section are based on the same storage configuration already introduced in section 4.4: A single storage pool, build from 5 physical 12 GByte partitions from different physical disks.

The benchmark environment IOMeter version 2004.07.30 is used as external load generator and to compare the kernel benchmarking results received from within V:DRIVE with the performance visible to a user application.

## 5.2 Measuring Benchmark Overhead

Performance measurements always have a direct impact on the performance of the devices under test. As outlined in section 4, the V:DRIVE has to perform around 50 additional operations for each successful read or write. These operations only include move-operations, sub- and add-operations, comparisons and one 64 bit multiplications.

To measure the overhead of the kernel benchmarking extensions, we compare IOMeter benchmark results with and without the kernel benchmark extensions enabled. The size of the accessed virtual volume is 58 GByte; the accessed file size is 50 GByte. IOMeter is started with a single worker thread enabled, a maximum of 4 outstanding I/Os and a block size of 4 KByte. Each of the tests runs for five minutes.

The results of these tests for pure sequential I/O are given in Fig. 4. Throughput and average latency values are given for different write percentages. In each case, the differences for the results with and without the benchmark extensions enabled are negligible and only in the range of standard measurement errors. Sometimes even the environment with kernel benchmark extensions enabled is able to achieve better throughput and/or lower latencies, indicating that the impact of the extensions is below the threshold that is measurable. The same behavior has been received when measuring random I/O performance.

## 5.3 Influence of Caching on System Performance

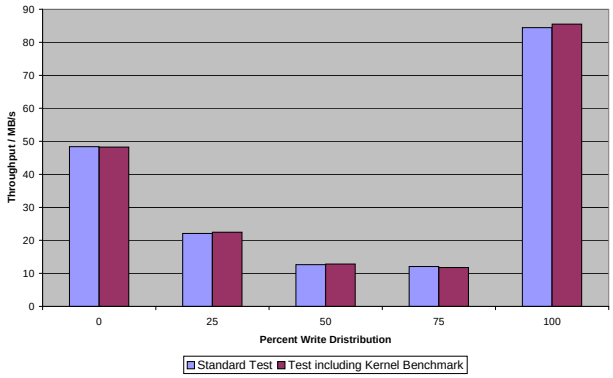
Cache size, cache behavior and prefetching algorithms have an important impact on many benchmarks and real-world applications. Nevertheless, this impact is difficult to measure by pure user-space benchmarks. In these benchmark types, it is difficult to distinguish between cached and non-cached accesses.

Figure 5 shows combined measurement results for the IOMeter benchmarking environment and the V:DRIVE kernel benchmarking extensions. In all cases, a single worker thread is reading data from a virtual volume purely sequential and only a single outstanding I/O is allowed. The virtual volume is built according to the definitions at the beginning of this section. The size of the accessed data varies between 400 MByte and 40 GByte.

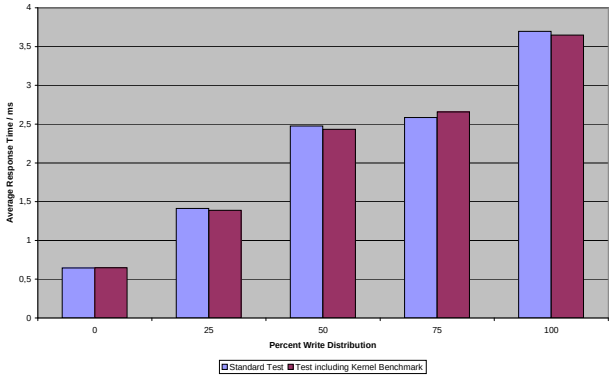
Enhancing the information gathered from IOMeter by the information gathered from the kernel extensions, it is easy to see that up to a file size of 800 MByte only the cache performance has an impact on system performance[1].

Increasing the file size to 1 GByte, and therefore above the available cache size, the buffer cache is still able to be of benefit, even for pure sequential accesses. The measured throughput of 53 MByte/s inside IOMeter is still more than twice the throughput measured by V:DRIVE, indicating that

[1]The small total throughput value is due the single outstanding I/O. Increasing the number of allowed outstanding I/Os to 16 increases the throughput to 640 MByte/s

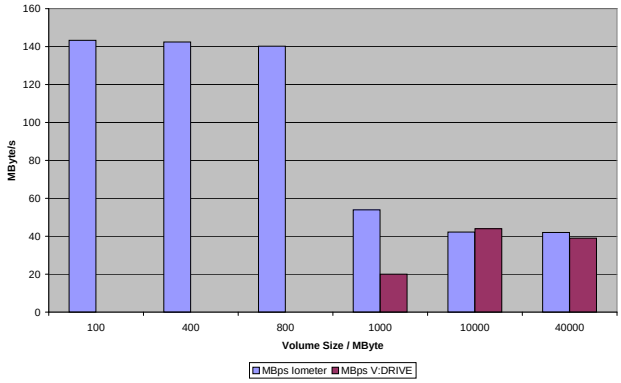



(a) *Throughput Sequential.*

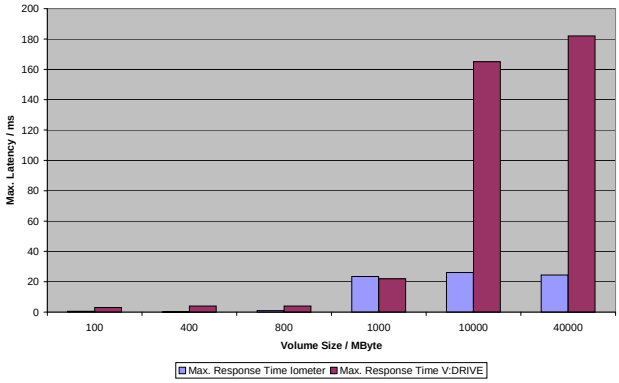



(b) *Average Latency Sequential.*

**Figure 4. Performance comparison for V:DRIVE with and without kernel benchmark enabled.**

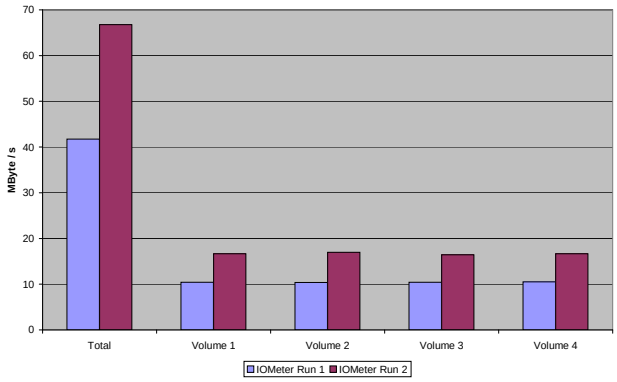



(a) *Sequential Read throughput.*

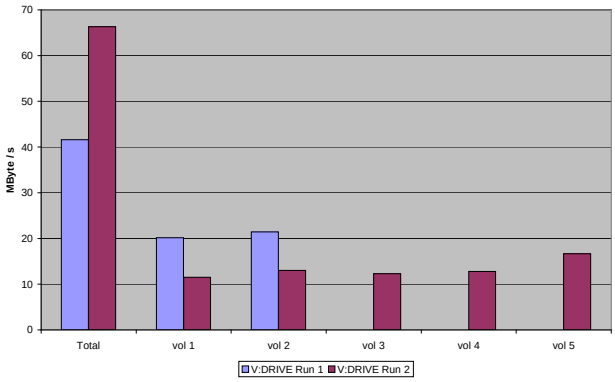



(b) *Maximum Latency.*

**Figure 5. Sequential performance results from IOMeter and from kernel environment.**




(a) *IOMeter results for different virtual volumes.*




(b) *V:DRIVE results for different physical volumes.*

**Figure 6. Influence of data distribution on storage performance.**

still a lot of requests are directly performed from the buffer cache.

When accessing 10 GByte or more sequentially, the influence of the cache can be neglected, all data has to be read from the physical disks. In this case the maximum latency is of interest (see Fig. 5(b)). By prefetching data from disk, the buffer cache inside the Linux environment is able to mask latencies of up to 180 ms. The maximum latency seen inside the IOMeter environment is still below 30 ms.

### 5.4 Impact of Data Distribution

In this subsection we will present measurement results indicating the influence of data distribution on storage results and on how to detect bottlenecks inside the system.

We have prepared two different measurements based on the IOMeter benchmarking environment. In both cases, we have used four virtual volumes with a capacity of 2 GByte each. One benchmark thread is assigned to each of the virtual volumes. Data is accessed again purely sequential with a maximum number of 16 outstanding I/Os.

From the perspective of IOMeter the environment looks the same for both measurements. Nevertheless, the performance gap between the measurements is significant (see Fig. 6(a)). In this case, the additional information gathered from the kernel benchmark extensions is able to resolve the reason for this gap very fast. In the first case, accesses are only distributed about the first two physical volumes belonging to the storage pool, in the second case, data is distributed about all five physical volumes, enabling a much better performance (see Fig. 6(b)).

## 6 Conclusions

The aim of this paper has been to outline the demand for an integrated architecture to benchmark, monitor, and optimize BI application environment. The gathered results clearly support the chosen layered architectural approach for performance measurement and optimization. They especially proof the value of integrating benchmarking results from the operating system layer and the file system layer: With the chosen measurement points concrete optimization possibilities in the fields of buffer cash design and distribution of data accesses are rendered visible which would have otherwise been masked by overlying effects. Similar optimization potential is expected for all other layers which will be closely evaluated in further research.

## References


[1] IoMeter User's Guide. Technical report, Intel Corp., 2003.

[2] A. Brinkmann, S. Effert, M. Heidebuer, and M. Vodisek. V:DRIVE - Parallel Storage Performance Redefined. Technical report, University of Paderborn, 2005.

[3] A. Brinkmann, M. Heidebuer, F. M. auf der Heide, U. Rueckert, K. Salzwedel, and M. Vodisek. V:Drive - Costs and Benefits of an Out-of-Band Storage Virtualization System. In *Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST)*, pages 153 – 157, College Park, Maryland, USA, April 2004.

[4] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, adaptive placement schemes for non-uniform distribution requirements. In *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 53–62, Aug. 2002.

[5] A. S. C. Rautenstrauch. Improving the performance of a database-based information system. a hierarchical approach. In *Proceeding of the Conference of the Association of Management / International Association of Management, San Diego*, 1999.

[6] P. Chen and D. Patterson. Storage performance—metrics and benchmarks. *Proceedings of the IEEE*, 81(8):1151–1165, August 1993.

[7] J. Corbet, A. Rubini, and G. Kroah-Hartmann. *LINUX Device Drivers*. O'Reilly, 3rd edition, 2005.

[8] P. Furtado. Experimental evidence on partitioning in parallel data warehouses. In *Proceedings of the DOLAP'04*, pages 23–30, New York, USA, 2004.

[9] M. Golfarelli, S. Rizzi, and I. Cella. Beyond data warehousing: What's next in business intelligence? In *Proceedings of the DOLAP'04, New York*, 2004.

[10] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive rfid data sets. Technical report, University of Illinois, 2005.

[11] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 1(1):51–81, Feb. 6.

[12] H. Kemper, C. Unger, and W. Mehanna. *Business Intelligence - Grundlagen und Praktische Anwendungen*. Vieweg, 2004.

[13] R. Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons, 1996.

[14] R. Love. *Linux Kernel Development*. Novell Press, 2nd edition, 2005.

[15] C. Ruemmler and J. Wilkes. Unix disk access patterns. In *Proceedings of the Winter'93 USENIX Conference*, pages 405 – 420, Jan. 1993.

[16] T. Ruwart. Xdd - User's Guide. Technical report, I/O Performance, Inc. - Version 6.3, 2005.

[17] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The sequoia 2000 storage benchmark. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 2–11, Washington, D.C., USA, May 1993.

[18] D. Theodoratos. Exploiting hierarchical clustering in evaluating multidimensional aggregation queries. In *Proceedings of the DOLAP'03*, pages 63–70, New York, USA, 2003.