RJ10514 (ALM1310-009) October 25, 2013
Computer Science

IBM Research Report

Visualizing Block IO Workloads

Ohad Rodeh, Haim Helman, David Chambliss
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
USA

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home .

Visualizing Block IO Workloads

Ohad Rodeh, Haim Helman, David Chambliss

October 18, 2013

Abstract

Massive block IO systems are the work horses powering many of today's largest applications. Databases, healthcare systems, and virtual machine images, are examples for block-storage applications. The massive scale of these workloads, and the complexity of the underlying storage systems, makes it difficult to pinpoint problems when they occur. This work attempts to shed light on workload patterns through visualization, aiding our intuition.

We describe our experience in the last three years of analyzing and visualizing customer traces from XIV, an IBM enterprise block storage system. We also present results from applying the same visualization technology to Linux filesystems.

We show how visualization aids our understanding of workloads, and how it assists in resolving customer performance problems.

1 Introduction

Block storage systems are used today by a vast number of applications. They provide an easy to use block read/write interface, while scaling from a single disk to multi-rack complexes. A lot of complexity is hidden behind the simple interface, and when performance problems occur, it is difficult to understand the root causes.

Our work uses visualization to better understand storage system behavior, and in particular, how applications use storage. We present a novel technique allowing the viewing of multiple aspects of a workload in time and space on a two-dimensional grid. This technique is part of a visualization toolkit, developed at IBM Research. It has been used for the last three years by enterprise customers, to gain insight into their workloads, and to show the results of what-if scenarios.

The data we work with is block traces, containing a short record for each read or write request that arrives at the storage target from the hosts. For an enterprise storage system, it is not unusual to see a billion IOs in a single day. These are large data sets that defy standard summarization procedures. There are no parametric statistical distributions describing such data, and it is too much information for a person to manually sift through. Our initial attempts at finding good statistical methods for summarizing the data were unsuccessful. Instead, we found that visualization was a good middle ground. It allowed presenting large amounts of non parametric data on a screen, in a manner that is easily digestable by non experts.

Through visualization, we show how read/write, sequentiality, and cache hits rates change in time and space. This is done per volume, per host, or for an entire rack. We also calculate and present application footprints, IO histograms, latency graphs, and cache predictions. These all allows users to drill down, and find the issues that most concern them. Most intriguing, we find that access patterns in time and space tend to have similar structure across applications. Studying this is left for future work.

This document is structured as follows: Section 2 describes related work, and Section 3 describes the XIV controller. Section 4 goes through a quick tour of the visualization. Section 5 describes issues in the design and implementation. Section 6 compares Linux filesystems in a graphical way, and Section 7 examines customers issues. Section 8 summarizes, and Section 9 adds acknowledgments. Many commercial products are referenced in this article, we list all the relevant trademarks at the end of Section 9.

2 Related work

As a rough generalization, all storage products have visualization tools. Plots of IOps, throughput, and latency, as a function of time, are fairly standard. What is new in the images that we generate, is the ability to effectively view access patterns in time and space using a two dimensional display. A major obstacle is that the space axis is very large; the addressable disk space is enormous. How would one present the space axis effectively? Here, we describe the solutions used by others for time/space viewing.

On Linux, `blktrace` [9] is the standard IO trace collection tool, it has complementary analysis and visualization tools: `btt` [2] and `Seekwatcher` [4]. Generally speaking, running `blktrace` on a Linux machine creates a binary file with detailed records describing ongoing IOs for the trace duration. `blkparse` processes the output, and can present it in human readable form. `Seekwatcher` can visualize the trace using a conventional two dimensional graph, or with an animation. The main focus of Seekwatcher is presenting the disk head movement. For example, Figure 1 shows a comparison between two programs, `acp` and `tar`, that walk a directory tree. `btt` creates an animation of the trace where the X axis is time, the Y axis is the start LBA, and the Z axis is number of bytes transferred. Solaris has a roughly similar tool called `taztool`.

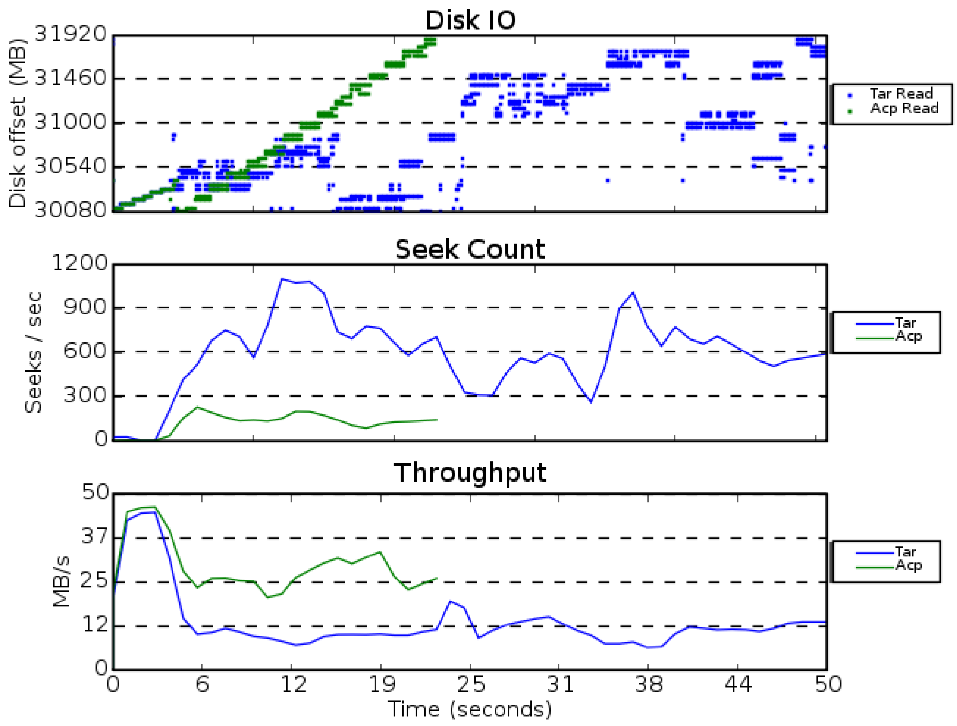


Figure 1: A Seekwatcher visualization of the disk seeks involved in packaging a directory recursively into a tar file. Two tools are compared: `acp` and `tar`.

The Sun Storage 7000 series uses an interesting performance visualization toolkit [8]. The product line is based on the Solaris operating system, ZFS filesystem, and DTrace kernel tracing facility. A web server exports performance data to connected browsers. In particular, a heat map where the X axis is time, the Y axis is latency, and color codes are used for IOps intensity ranges is used to present system performance. This heat map representation does not display a space/time continuum, however, it does represent three dimensional data in a two-dimensional display.

IBM Redbook [3] provides a good description of XIV version 11.1. It describes the SSD caching, IO analytics, and visualization.

3 XIV

XIV is an enterprise storage system, supporting block level access using FCP and iSCSI protocols. Externally, it presents a single system image, with multiple FCP and Ethernet ports, which a customer can connect to the SAN. Data volumes can be created on the system, allowing a user to store data. XIV is typically used by higher end customers, for storing VM images, databases, health care information, ERP, etc.

The system comprises a rack of modules; see Figure 2. Each module holds a server class CPU, 24GB of RAM, and 12 SAS disks. An Infiniband network connects the modules, allowing fast low latency communication. Disks are not shared; each is connected to a single module. Systems with flash disks also have 512GB of SSD per module, offering a total system flash capacity of 7.5TB. With 3TB disks, the system has 243TB of usable disk space.

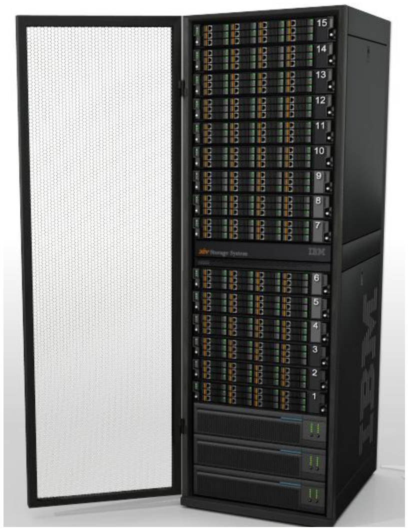


Figure 2: A XIV system with a full rack comprising 15 modules.

Customer data is striped across modules and disks in a declustered RAID-1 pattern. This ensures fault tolerance to single disk and module faults, while allowing fast rebuild. The slice-based data distribution places the responsibility for different megabyte units on different modules, in a manner that is rebuilt and rebalanced very rapidly when hardware is added, removed, or lost to a failure. Writes are replicated to two modules using

NVRAM, improving write speed. Small IOs are coalesced in NVRAM, and written asynchronously to disk in large extents.

3.1 Caching

The controller includes hundreds of gigabytes of read-write RAM cache and a second, read-only, cache residing in terabytes of SSD. The SSD cache is located at the lowest level of the IO stack, right above the disks. The cache for each module operates on its data only, independently from other modules. The cache is consulted prior to any disk read, and in most cases is populated immediately after a disk read or write. The cache design is focused on servicing small, random reads. It is engineered with write shaping and other flash-aware features to ensure high performance and good device lifetimes, even with lower-cost SSDs that do not themselves provide sustained high performance for random writes. Eviction decisions recognize groups of pages that are likely to reside on the same flash erase blocks. Another feature is that large and sequential IOs do not get populated into the cache. Several goals are met by this design:

1. Avoiding early SSD wear out
2. Making good use of high disk throughput
3. Reducing disk seeks

3.2 Trace Collection

The trace collection facility used in this study is built upon circular buffers of IO operation history to support engineering diagnostics. Trace collection operates in an external system that repeatedly polls the XIV modules to efficiently retrieve the new content from the buffers. The collection system processes the records from multiple nodes into a unified time-coherent log of all operations. Each operation record has seven fields:

num_blocks: number of 512-byte sectors

is_read: is this a read or a write request?

LBA: Logical Block Address

time: time when the request completed, in microseconds

latency: latency in microseconds

volume: volume ID as a 64-bit number

initiator_id: An identifier for the request initiator

The collection facility also gathers configuration metadata which includes a volume table, an initiator list, and an overall statistics table. The volume table has per volume information like name, capacity, group, owner, and snapshot metadata. The initiator list holds the set of initiators connected to the storage target. The statistics table contains average latency, iops, and hit-rates, per five minute time slot. This is crucial for cache hit rate prediction.

The trace collection procedure has no discernible impact on the XIV system performance, which was essential for applying this to customers' production systems. It normally succeeds at collecting 100% of the IO records. Some records might be dropped if the collection process faces serious network congestion or scheduling congestion on its host machine. In early stages of this work we explored collection-time sampling to reduce data size, for example by collecting for one minute out of every 5-minute window. Such sampled data can yield some useful results, but we found in most cases it cannot provide the accuracy or confidence for cache prediction that we require. Thus it was necessary to optimize the procedure to achieve 100% collection without performance impact.

The traces we work with are mostly from enterprise customers, whose systems can perform a massive amount of IO. In one day, we have seen traces ranging from hundred million to 2 billion IOs.

3.3 Cache simulation

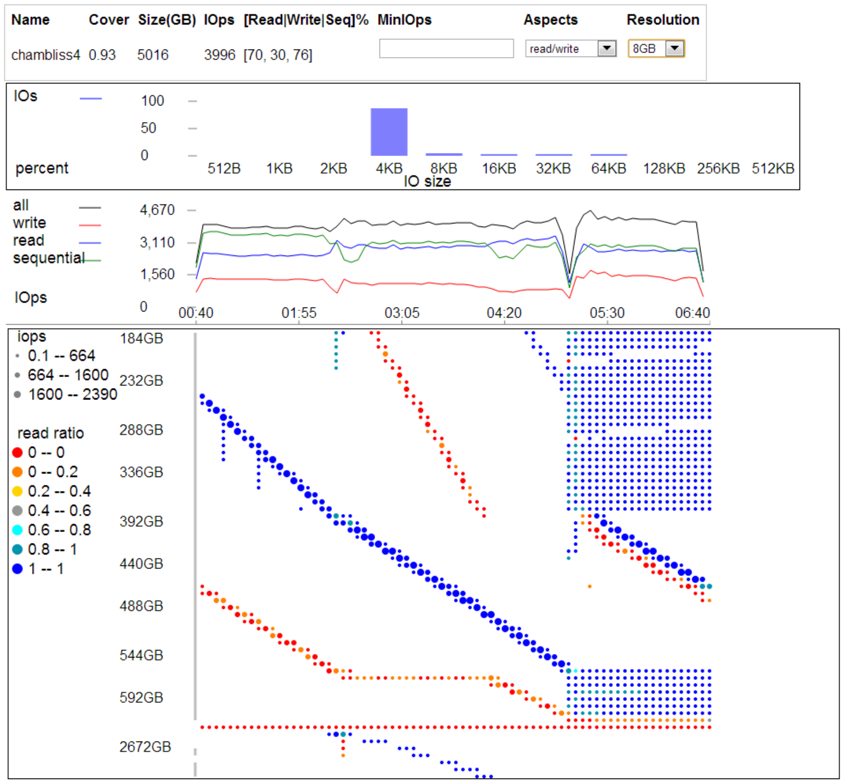
Based on collected traces, a cache simulator was built. It implements a much simplified version of the XIV cache, takes traces, and outputs a prediction for the read cache hit rate. The simulator takes a configuration file specifying how much RAM and SSD the machine has. This allows running what-if scenarios for customers that want to purchase flash cache, or additional modules.

4 A quick tour

A good visualization allows the user to quickly make sense of a large amount of data by presenting it in visual form. The human mind is very good at making sense of visual patterns. If data is presented correctly, complex patterns are immediately recognizable.

A key difficulty is that the screen is two dimensional, with a limited number of pixels. The data is multi-dimensional, with enormous resolution. Hence, screen real estate must be used sparingly, to present the important information. Deciding what is important is a key question.

The IO trace was split into time/space cells of size 1GB $\times$ 5-minutes. This was done for each user visible volume. Several features were calculated for each cell: average IOps, ratio of read vs. write, distribution of IO sizes, sequentiality, etc. The main feature in the presentation is a heat map based on these features; see Figure 3 for an example. The heat map occupies a bit more than the bottom half of the figure. The x-axis is time, the y-axis is space, i.e., the logical address within the volume. The hot gigabytes are presented as rows, as much as can fit in a screen. Cold gigabytes, those with low IOps, are omitted. A light gray vertical line appears on the left hand side, it connects rows with adjacent addresses. Each circle has two properies: radius and color. The radius encodes IOps intensity; the larger the circle, the higher the IOps. The encoding of IOps is non-linear and data-dependent, based on quantiles to enhance the contrast of variations. Areas with low-IOps are omitted. In the default view, color encodes read vs. write ratio. Red circles have a pure write workload, blue circles see only reads.



Name	Cover	Size(GB)	IOps	[Read|Write|Seq]%	MinIOps	Aspects	Resolution
chambliss4	0.93	5016	3996	[70, 30, 76]		read/write	8GB

Figure 3: A database volume, where the dominant patterns are sequential read and writes scans. These appear as diagonal lines in the map. Total trace duration is six hours, row granularity is 8GB, and the coverage is 93%.

There are three other panels in Figure 3. The top panel provides overall information in textual form, the volume name is *chambliss4*, the size is roughly 5TB, and the average IO rate is about 4000 IOps. The second panel depicts the histogram of IO sizes in logarithmic buckets. We can see that the major component is 4KB IOs. The third panel is a timeline with separate lines for total, read, write, and sequential IOps rates.

Figure 3 presents a DB2 database workload, where the dominant patterns are sequential read and write. The heat map shows sequential scans as diagonals, and random access areas as rectangular shapes (5:00 – 6:40). Each row shows 8GB of disk space, and the coverage is 93%. This means that we are depicting the hottest 93% of the total space on the volume. Put another way, 7% of the IO is not presented, because it consists of low density accesses to large volume areas.

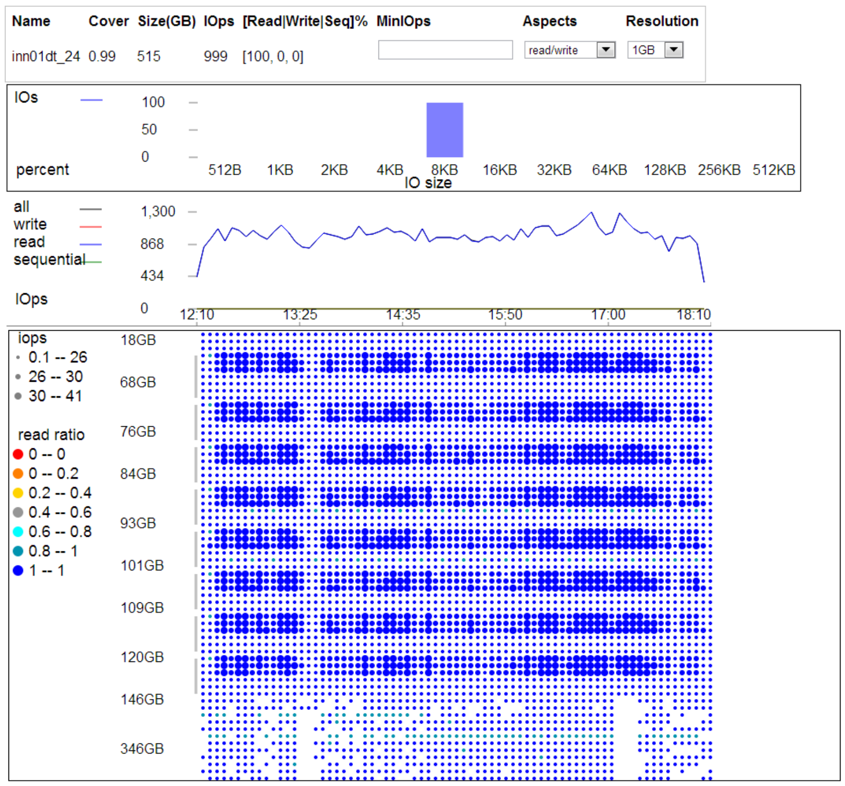
The number of rows in the screen might not be sufficient to present all the hot gigabytes. In that case, a single row presents average values for a range of contiguous gigabytes. Similarly, the number of columns might be larger than those available on the screen. In that case, multiple columns are merged using averaging. Normally, it is not possible to present all the accessed areas. To give the user an idea for how much of the IO pattern is presented, the *coverage* is calculated. This is the ratio between the IOs presented on the screen, and the total IO performed on the volume.

There are many ways to define sequentiality. Our calculation is based on examining 64KB regions (called *grains*) accessed inside a cell. The following heuristic inferences rules are used:

- If all the grains in a MB are read in 1 minute, then the MB is read sequentially.
- If all the grains in a MB are written in 1 minute, then the MB is written sequentially.

If a MB is accessed sequentially, then all the IO in that minute is counted as sequential. For each cell, the sequentiality score is the percent of sequential IO, out of the total amount of IO.

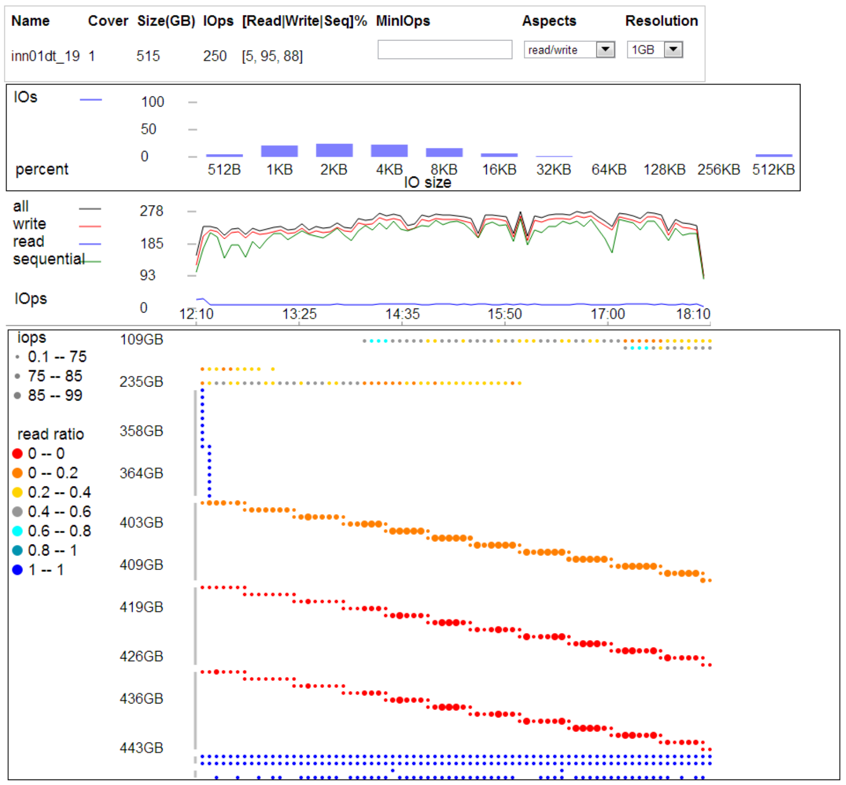
Figure 4 shows an Oracle database volume, with a steady OLTP workload. There are 100% reads over the whole volume, and access is split into eight distinct bands. Cold gigabytes are omitted, causing breaks in the address space. In interpreting the heat map one must pay attention to the legend. The scale for IOps is selected to increase contrast and make features visible. In this case a cell with the smallest circle might represent IOps only 15% lower than another cell with the largest circle. In a live version one can click on a circle and see a pop up with exact numbers.



Name	Cover	Size(GB)	IOps	[Read|Write|Seq]%	MinIOps	Aspects	Resolution
inn01dt_24	0.99	515	999	[100, 0, 0]		read/write	1GB

Figure 4: A randomly accessed database volume. The rows are 1GB, and coverage is 99%.

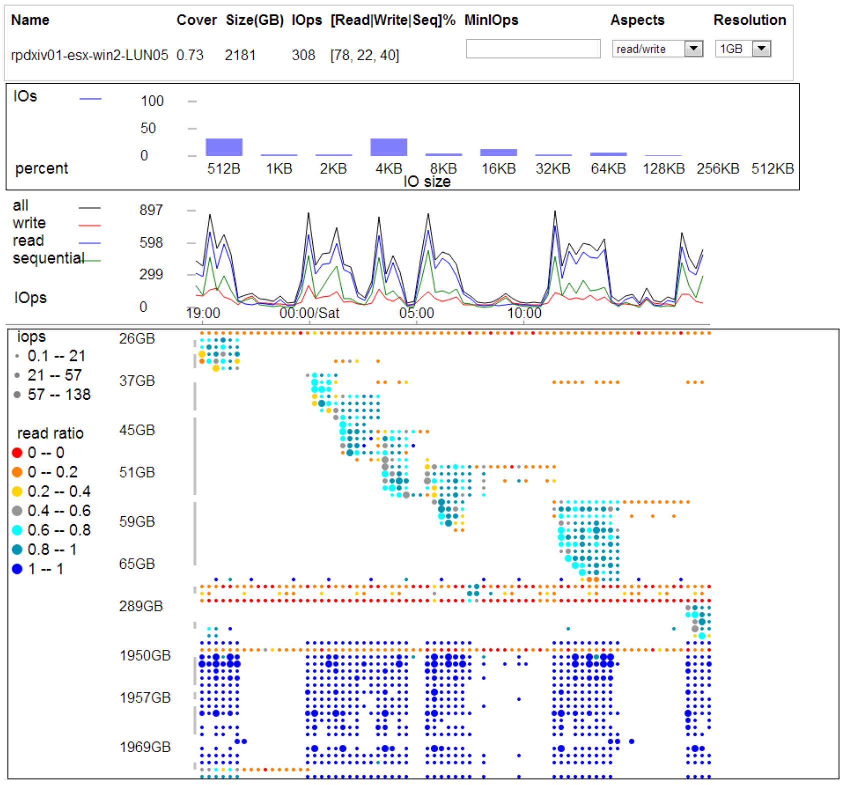
Figure 5 presents an Oracle volume that is written sequentially in three parallel streams, to three separate disk areas. A pure read workload is applied to two other areas. The IOps graph shows that the workload is stable, and consists mostly of sequential writes.



Name	Cover	Size(GB)	IOps	[Read|Write|Seq]%	MinIOps	Aspects	Resolution
inn01dt_19	1	515	250	[5, 95, 88]		read/write	1GB

Figure 5: A sequentially accessed volume, with three sequential streams. Rows are 1GB, and coverage is 100%.

Figure 6 depicts a complex access pattern for a volume that holds multiple VM images. VMWare ESX is the hypervisor and manager.



Name	Cover	Size(GB)	IOps	[Read|Write|Seq]%	MinIOps	Aspects	Resolution
rpdxiv01-esx-win2-LUN05	0.73	2181	308	[78, 22, 40]		read/write	1GB

Figure 6: A VM volume, hosting multiple operating system images. It has a complex access patterns. Row granularity is 1GB, with 73% coverage.

In the browser, clicking on a circle presents detailed information about the IO pattern on that cell.

The visualization also includes more ordinary *line graphs*. A line graph is a traditional X-Y plot of a continuous variable. Several variables could be plotted together. For example, Figure 7 presents IOps during a day in the life of a storage system for VMs. Note that several kinds of operations are presented compactly together: reads, writes, and sequential reads and writes. There are three distinct activity peaks, at 18:00, 20:00, and 1:00. There are few sequential writes, but about a third of the reads are sequential.

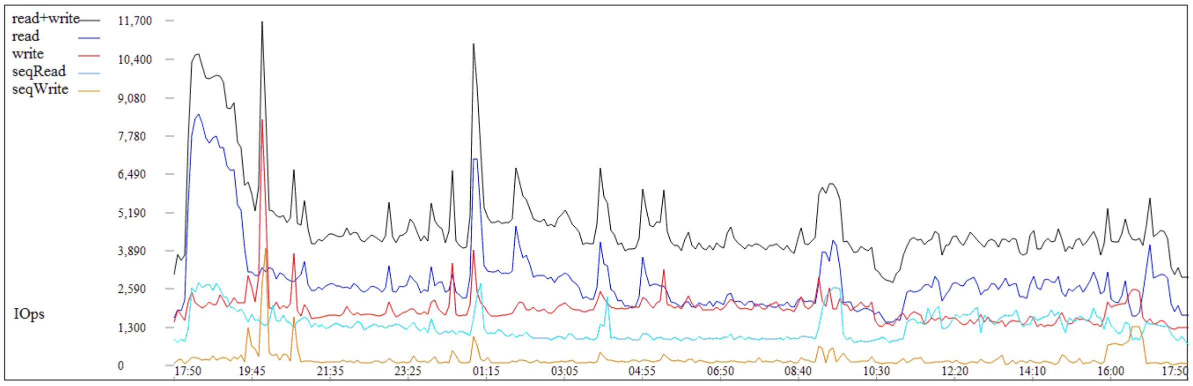


Figure 7: IOps graph for one day in the life of a VM system.

A footprint graph plots the disk area that the user applications access as a cumulative distribution. It gives a quantitative view, over time, of the heat skew in a workload, which is useful in understanding the effectiveness of system features such as caching. Grains (64KB) are sorted from hottest to coldest, and percentiles are calculated. In Figure 8, the *Footprint_64KB_60min_ALL* graph below looks at time windows of 60 minutes, and examines all IO. The application was a commercial database. At Tuesday/5am, the hottest 60% of the IO covered a 154GB on-disk area, and the hotest 80% of the IO covered a 240GB area. The entire data set fit inside a 1TB area. The bottom graph, *Footprint_64KB_60min_RANDOM_ALL* counts only the randomly accessed grains. In the first half of the trace, random access accounts for the majority of the footprint, in the second half, it account for very little.

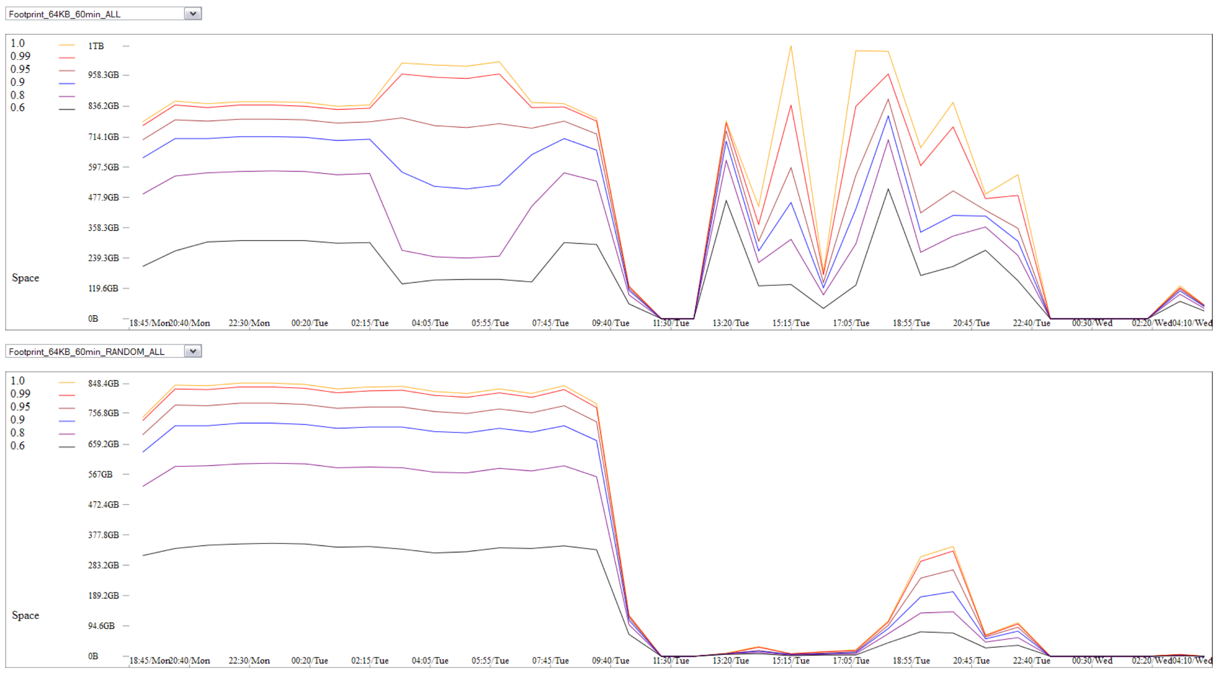
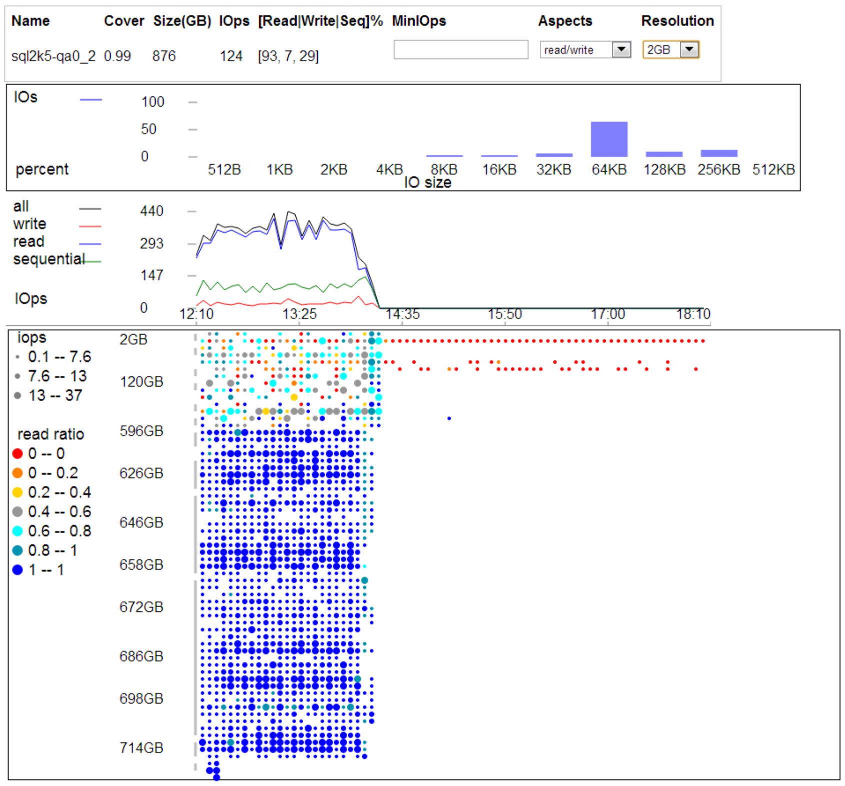


Figure 8: The footprint of a database workload

Footprints are important when trying to estimate how much cache would benefit an application.

Figure 9 shows six hours of activity on a volume holding a Microsoft SQL 2005 database. The top graph is a line plot showing IOps. The bottom graph is heat-map showing an access pattern with two distinct regions. The area 2GB – 162GB, at the beginning of the volume, is read and written in mixture of accesses. A much higher region, 600GB – 715GB, is accessed in a random read-only pattern. We speculate that the high area holds queried database records. Since we do not have the client application nor data, there is no way for us to know for certain. Note that having the IOps graph does not give an indication as to the spatial distribution of IO. For example, knowing that there are 20 write IOps at 1pm does not inform us that these IOs are occurring primarily to the region [2-162]GB of the volume.



Name	Cover	Size(GB)	IOps	[Read|Write|Seq]%	MinIOps	Aspects	Resolution
sql2k5-qa0_2	0.99	876	124	[93, 7, 29]		read/write	2GB

Figure 9: A SQL 2005 database. Row size is 2GB, coverage is 99%.

5 Design and implementation

In our setting, customers send us data, and we send back a visualization. For this to work well, we needed a compact visualization, that would not require special viewing software on the customer side. We did not want to create and ship a complimentary client GUI executable, which would have to work across multiple OSes.

After deliberation, we chose the HTML5 format and the JavaScript programming language. From a customer trace, our toolkit creates a directory of HTML5 and JavaScript that is around 20MB in size. It can be easily viewed by any HTML5 compliant web-browser, including smart phones. To work well, the graphs and grids are all constructed with Scalable Vector Graphics (SVG). Thus, rotating the screen, and zoom in/out works without loss of resolution. The user could interact with the browser, and filter through the information.

For internal use, we also support a servlet based interface. A server machine runs a java servlet with a backend database that holds the heat-maps. A separate front end GUI is written in javascript and runs on a browser.

Our heat-maps are based on a the idea of a grid of circles. This is a matrix of circles of varying size and color, that convey an overall impression. We experimented with many color palettes, and circle sizes. We had also consulted Edward Tufte's books [5],[6], [7]. The literature recommends no more than 20-30 colors, more than that, and the human mind has trouble distinguishing shades. We found that seven colors were the maximum we could use. This is a different problem than cartographic map building, where colors designate height. Height is a continuous metric, and using many shades is standard practice. In our setting, read cache hit rates, sequentiality, and read/write ratios are not continuous. Adjacent circles could have any combination of colors, and the user would like to recognize this fact with a single glance at the screen.

Choosing the legend of circle sizes presented a different challenge. We ended up using only three sizes, because using more would have reduced the number of rows. Seven colors, and three sizes, provide 21 configurations for a circle in a grid. We found that this was a good choice for the kinds of workloads we saw in practice. Normally, volume heat maps have a distinct pattern that is easily distinguishable. In cases where there is no obvious pattern, many times, this is because the volume is part of a larger stripe group, or as in VMware ESX, it might be home to tens of virtual machines.

6 Linux filesystems

As part of an effort to compare contemporary Linux filesystems [10], we configured a standalone Linux Ubuntu machine with three mainstream filesystems. We ran several benchmarks, and extracted IO traces during run time. Trace collection was based on the `blktrace` tool. When run, blktrace has a discernible impact on a Linux system, up to 5% CPU in our experiments.

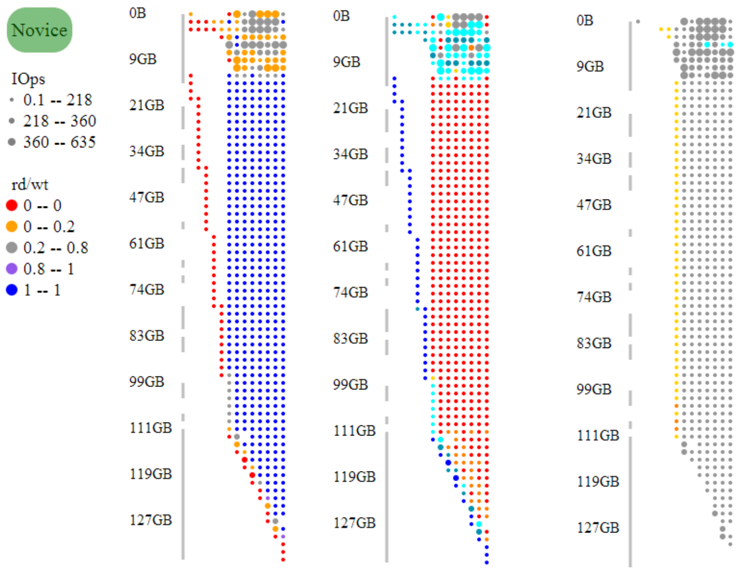
The machine used was an Intel Core i7-2600 3.40GHz CPU. It has a single socket with eight cores, and 16GB of RAM. The memory was limited to 2GB by setting a Linux boot parameter. The OS was Ubuntu 12.10, with a 3.6.6 Linux kernel. The flash disk (*SSD*) was an Intel SSD-320 Series, 300GB capacity, with R/W throughput of 270 MB/s / 205 MB/s, and R/W IOps of 39.5K/23K.

Default mount options and `mkfs` programs were used. Per file system, the following options were employed:

filesystem	options
BTRFS	relatime,space_cache,rw,ssd
Ext4	data=ordered,relatime,rw
XFS	attr2,noquota,relatime,rw

The FileBench [1] toolkit was used for benchmarking, with the mail personality. A FileBench run starts with an empty volume, and preallocates a filesystem tree. Once that is done, the chosen workload is executed while carefully measuring performance. This workload mimics an electronic mail server. It creates a flat directory structure with many small files. It then creates 100 threads that emulate e-mail operations: reading mail, composing, and deleting. This translates into many small file and metadata operations.

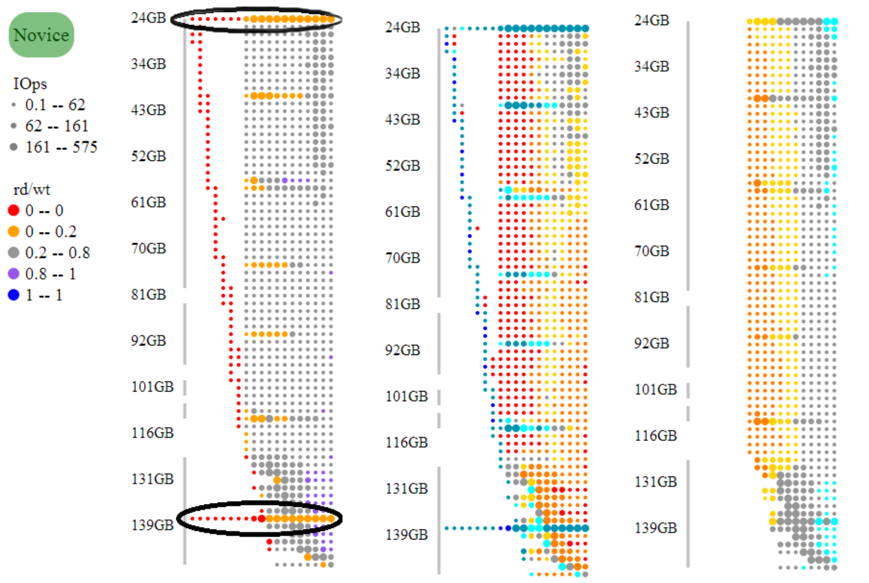
BTRFS is a copy-on-write filesystem that uses b-trees to represent all of its on disk metadata. It's behavior is shown in Figure 10. The test lasts about an hour, after preallocation completes. We can see side by side heatmaps showing (a) read/write ratio, (b) sequentiality, and (c) read cache hit rates. From (a) we can see that BTRFS writes files at the edge of the allocated area, and then reads previously written areas. There is an area of mixed access at the beginning of the disk, our guess is that it holds metadata. The preallocation phase is characterized by pure sequential writes, whereas steady state involves a mixed read/write workload. Figure (b) shows that the writes are purely sequential, whereas the reads are random. Finally, Figure (c) shows that read cache hits slightly improve over time; the cache starts empty, and fills up, but never reaches more than 80% hits. We estimate cache hits from read IO latency, a read that takes less than 3ms is assumed to be a cache hit. The circles are colored with the percent of fast reads. A circle colored red means zero hits, a circle colored blue has 100% read cache hits.



(a) read/write (b) sequentiality (c) read cache hits

Figure 10: BTRFS

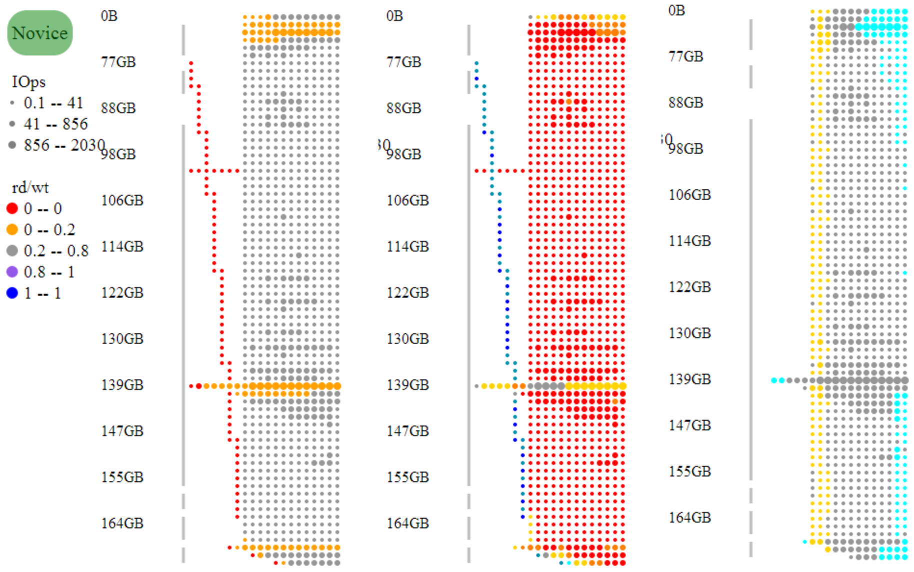
XFS is a write-in-place filesystem that uses a write-ahead-log. It's behavior is depicted in Figure 11. Panel (a) shows that after a pure write preallocation phase, we get mixed read/write access to most disk areas. This is in contrast with BTRFS, where there is good segregation of reads from writes. From (a) and (b) we can see two areas where writes are mostly sequential: the edge of the filesystem and at the logs. The log areas are, most likely, $\{24GB, 139GB\}$. This is because they have a consistently write dominated write workload throughout the test. The rest of the disk absorbs mostly non sequential read/write IOs. There are areas with a mostly write sequential workload, $\{40GB, 58GB, 75GB, 89GB\}$, we suspect these store metadata. Figure (c) shows that the cache warms up with time, reaching 70-80% cache hits at the end of the run.



(a) read/write (b) sequentiality (c) read cache hits

Figure 11: XFS. The two areas marked with black ellipses have a dominant write sequential workload.

EXT4 is the standard Linux filesystem. It uses an update in place policy, with a write-ahead-log for crash recovery. Its disk behavior, shown in Figure 12, is fairly similar to XFS.



(a) read/write (b) sequentiality (c) read cache hits

Figure 12: EXT4

7 Customer issues

The most important use of the visualization is to solve customer performance issues. An example is an enterprise customer that had an Oracle database with XIV storage, used for OLTP. The read cache hit rates were hovering around 30%, and he was wondering if SSDs were going to improve this situation. A trace was conducted on the system. We then visualized it, and ran our cache simulation. Figure 13 shows the cache hit situation. The prediction is for 80%, after about six hours. This number was validated using a replay in the lab, shown in blue.

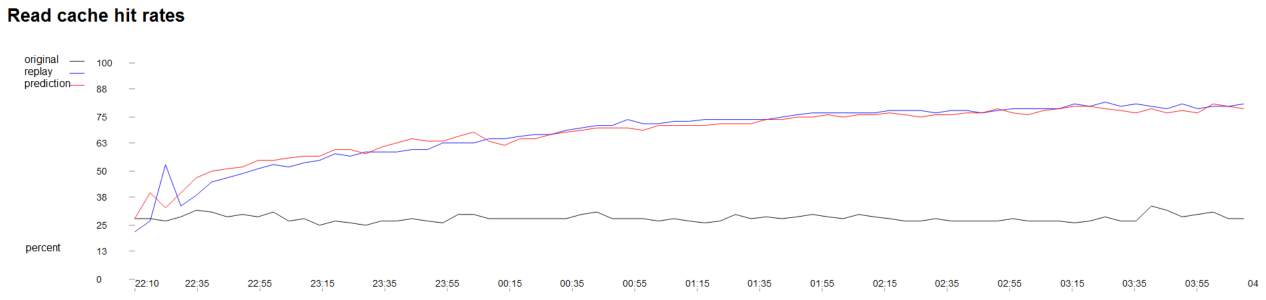
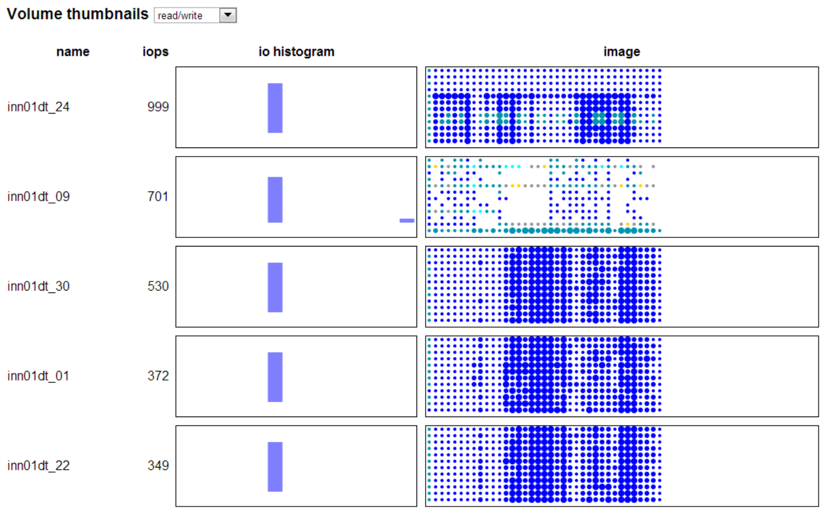


Figure 13: Read cache hit rates. The current hit rate is around 30%. The prediction with SSD is around 80%.

To get a better understanding of the workload, we examined the hottest volumes, see Figure 14. We saw that they all are issued 8KB random reads. This matched the description of a database workload. Hence, the pertinent question is whether the footprint fits in the 7.5TB SSD cache.


Volume thumbnails read/write

name	iops	io histogram	image
inn01dt_24	999		
inn01dt_09	701		
inn01dt_30	530		
inn01dt_01	372		
inn01dt_22	349		

Figure 14: Thumbnails of the hottest five volumes.

Figure 15 shows that the footprint of random reads in a window of four hours is 1.4TB. This indicates that the working set is going to fit in the SSD cache.

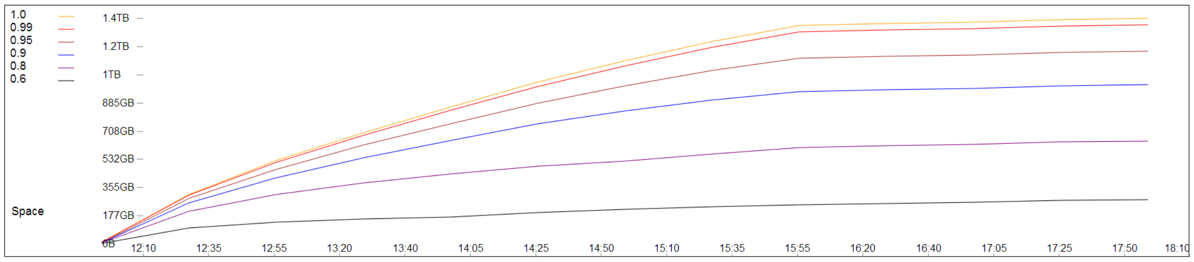


Figure 15: The footprint of random reads in a four hour window.

Our recommendation to the client was that purchasing SSDs would significantly improve performance, and it did.

Another example is a customer, that owned an XIV box with RAM cache, but no SSD cache. The storage was used by a Microsoft Exchange e-mail server, and performance was adequate. The question was whether purchasing SSDs as cache was going to improve performance.

We traced the production workload, visualized it, and ran a cache prediction algorithm on it. The prediction was that SSDs were not going to improve performance, and the customer should forgo the expense. It turned out, that the customer already knew this, but wanted to see what we could tell him about the workload.

Figure 16 shows the read cache hit rate in black, and the prediction in red. The prediction is based on a cache simulation, which takes a few hours to warm up. Towards the end of the day, the prediction converges to the current performance, which is around 75-80%.

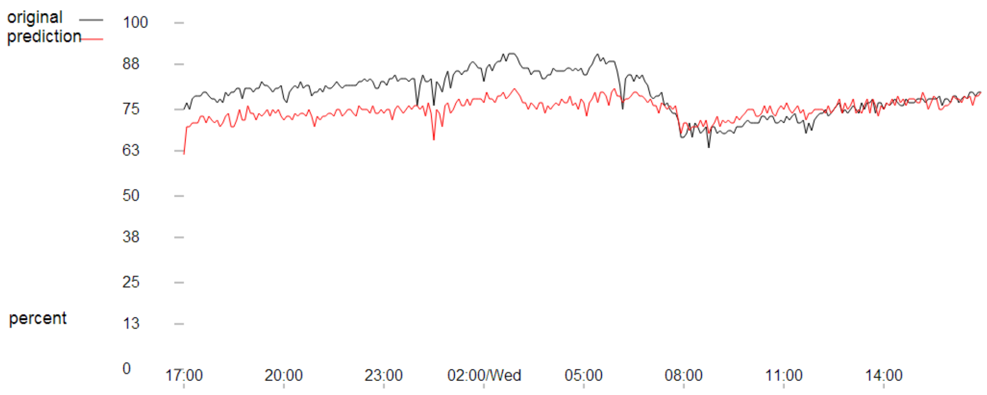


Figure 16: Read cache hit rates

It is rare that SSDs do not improve performance at all. To shed light on this mystery, we plotted the histogram of IO sizes, see Figure 17. Most reads are 256KB, which bypass the XIV SSD cache. This answered half the question, but we were still wondering what sort of application issues so many large reads.

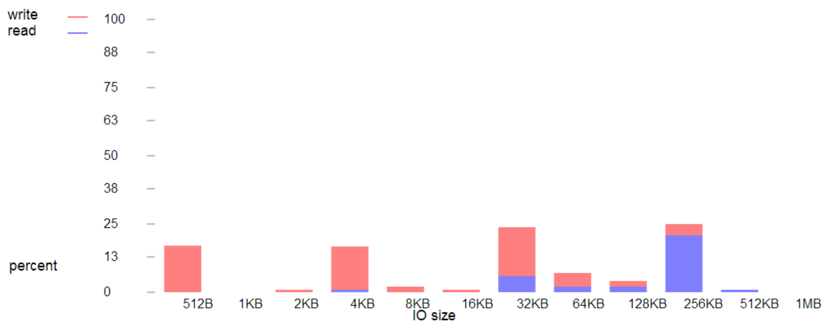


Figure 17: IO histogram size

Figure 18 shows a Exchange volume typical for this customer, indeed, typical of all Exchange volumes that we have seen. During the night, a sequential scan is performed, this scan is done with 256KB read IOs. We believe this is a virus scan or a backup. This behavior accounts for almost all of the read IO on the box, and is not cache friendly. Hence, SSDs are not going to help this workload. The different Exchange volumes stagger the sequential scans throughout the day, so that they do not overwhelm the storage rack.

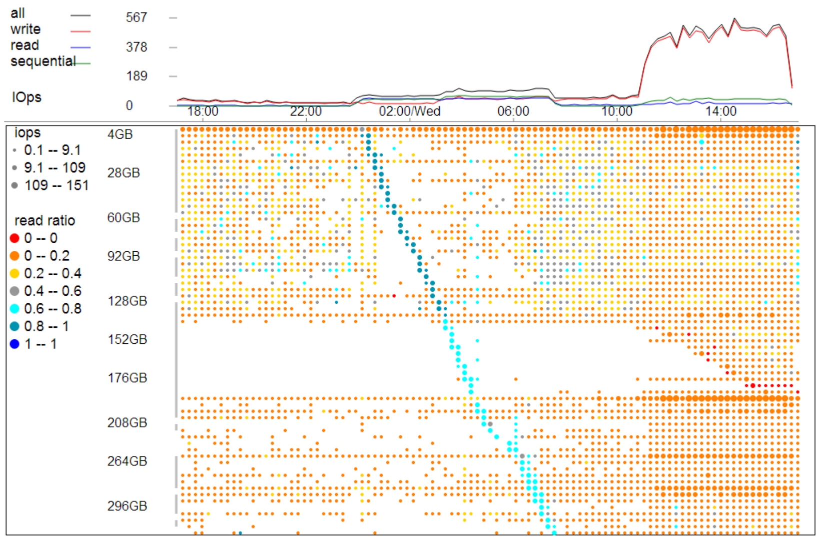


Figure 18: The read/write pattern of a typical Exchange volume

8 Summary

This article has described visualization techniques for block storage systems. They have been used to (1) solve customer performance issues on XIV storage systems, and (2) to gain insight into Linux filesystem behavior. We believe the techniques are generic and can be applied to other block storage systems.

This work is based on field experience gained during the last three years, of looking at traces, and solving customer problems. One of the main uses of this technology has been giving advice on SSD purchases.

9 Acknowledgments

We would like to thank Anthony Vattathil, Theodore Gregg, and Aviad Offer, for their help and support.

A number of commercial products are mentioned. *XIV* and *DB2* are trademarks of IBM. *SQL server* and *Exchange* are trademarks of Microsoft. *Sun* and *Solaris* are trademarks of Oracle Corporation. *ESX hypervisor* is a trademark of VMWare. *Pentium Core* is a trademark of Intel.

References

[1] FileBench. `http://sourceforge.net/projects/filebench`.

[2] A. D. Brunelle. `btt`, 2006.

[3] B. Dufrasne, I. K. Park, F. Perillo, H. Sautter, S. Solewin, and A. Vattathil. *Solid-State Drive Caching in the IBM XIV Storage System.* International Business Machines Corporation, USA, 2012.

[4] C. Mason. Seekwatcher, 2008.

[5] E. R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, USA, 1986.

[6] E. R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, CT, USA, 1990.

[7] E. R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative.* Graphics Press, Cheshire, CT, USA, 1997.

[8] G. Brendan. Visualizing system latency. *Communications of the ACM*, 53(7):48–54, July 2010.

[9] J. Axboe, A. D. Brunelle, and N. Scott. `blktrace`, 2006.

[10] O. Rodeh, J. Bacik, and C. Mason. BTRFS: The Linux B-tree Filesystem. *Transactions on Storage*, 9(3), August 2013.