nimblestorage

Mapping the Demands of Real-World Apps – One IO at a Time

By David N. Adamson, Ph.D.
Principal Data Scientist, Nimble Storage

Anyone who has purchased hardware for business applications knows that determining the needs of a deployment can be a complex process. While different applications are known to have different needs, even deployments of the same application can differ from one another, making the standardization of benchmarks difficult. There are more benchmarking tools and guidelines continue today than ever before, some of them directly contradictory.

At Nimble Storage, we're able to provide some objective perspective, thanks to InfoSight™. The InfoSight platform has collected hundreds of terabytes of customer telemetry, from thousands of customer applications, in just the past year alone. In 2015, we used this platform to analyze in ensemble the IO requests of the applications running in our installed base and presented our results. This research report extends that analysis so that we not only show how common applications differ from one another in aggregate, but also to what degree IO requests from the same application can vary from deployment to deployment.

Table of Contents

Illustrations

1 Introduction

With the ability to see the full ensemble of an application's deployments at once, we can answer questions about what behaviors are common or uncommon, which we could not address when looking at aggregates alone. To provide the context for those questions we begin by revisiting some performance fundamentals.

1.1 Performance Fundamentals

There are many important variables that influence performance, including but not limited to IO size, fraction of reads vs. writes, and the extent to which specific operations occur sequentially. Fundamentally, we can think of performance either in terms of "transaction efficiency" or "transfer efficiency":

- When transaction efficiency is the priority, we measure performance by the number of transactions that can be completed in a period of time (e.g. in IO/sec), and the latency of those individual transactions (e.g. in milliseconds).
- In contrast, when transfer efficiency is the priority, we measure performance in throughput (e.g. in MB/sec).

Increasing transaction efficiency requires a trade-off with transfer efficiency and vice versa (Fig 1). As an example, more transactions are possible when the transaction payload is smaller (i.e. IO/sec is higher when IO size is smaller) while, in contrast, aggregating data into fewer transactions improves data throughput (i.e. MB/sec is higher when IO size is larger).

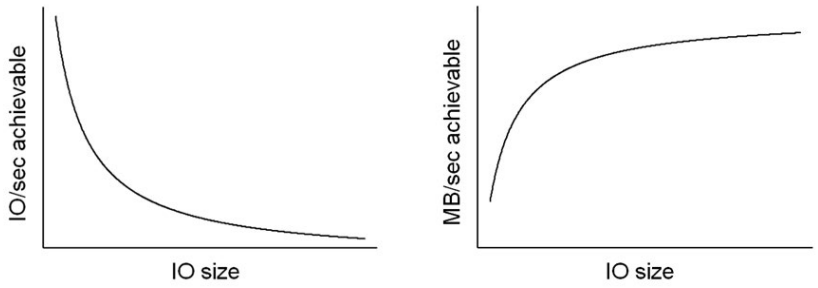


Figure 1: Two Limiting Cases for Performance.
Left: Transaction efficiency (IO/sec) improves as IO size decreases.
Right: Transfer efficiency (MB/sec) improves as IO size increases.

1.2 "Split the Difference" vs. "Divide and Conquer"

When designing applications, software engineers have to make decisions on how to manage these efficiency tradeoffs. How are applications designed to optimize both transactional and data-transfer efficiency? Do applications perform IO at intermediate request sizes and effectively "split the difference" between transaction-centric and data-transfer centric performance, or do they instead "divide and conquer" splitting their interactions with the storage into two categories that are separately optimized for either transaction performance or for data transfer performance?

To answer this question, we cannot assume that all deployments of the same application will behave similarly. Just as we can expect that different applications will place different emphasis on the importance of transactions vs. data transfer, we should also expect that differences in how distinct businesses use the same application can also affect this balance. In the analysis below, we begin with an aggregate picture of Nimble's user base – and extend our analysis from there to examine deployment-to-deployment variation for each of several common application categories.

2 Aggregate Insights

The key insight in aggregate is that real-world IOs concentrate in distinct **transaction-optimized** and **transfer-optimized** regimes.

We begin our analysis by first looking at the IO of the Nimble install base collectively, without separating out individual applications. In this high level view we can already see some hints as to whether applications "split the difference" (trading off transaction and transfer performance by performing mostly intermediately-sized IO) or "divide and conquer" (performing a mixture of small transaction-optimized IO in concert with larger transfer-optimized IO).

Indeed, when we count the proportion of Nimble user IO that falls into the size intervals below, we see a multimodal distribution (i.e. a distribution with multiple distinct peaks) (Figure 2). A solid majority of the IO (59%) appears in <= 8 KB or "transaction-optimized" range. Simultaneously, a smaller but still significant proportion of the IO (24%) appears as its own distinct peak in the >= 64 KB or "transfer-optimized" range. Importantly, while operations in the "transaction-optimized" range represented the majority of operations performed, it isn't too surprising that the operations in the "transfer-optimized" range moved the majority of the data (81%). In contrast, the intervening range of IO sizes (i.e. greater than 8 KB and less than 64 KB) accounts for only 17% of the IOs performed and 12% of the data transferred.

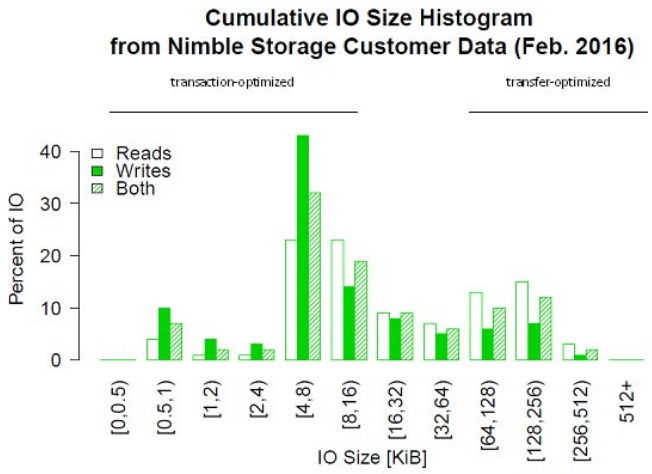


Figure 2: Cumulative Distribution of IO Sizes from Real-World Applications.
A month's worth of user IOs from >7500 Nimble customers is sorted and counted by block size. We can see that the majority of the IO performed fell into the less-than-16 KB "transaction-optimized" range (52% of all read operations, 74% of all write operations). A second smaller population of IO fell into the >=64 KB "transfer-optimized" range (31% of read operations, 14% of write operations), but it should be noted that those large-block operations carried the vast majority of the data (84% of all data read and 72% of all data written). Note that brackets [] indicate inclusive interval boundaries while parentheses () indicate exclusive interval boundaries; as an example: the interval [4-8) KB includes all IO sized greater than or equal to 4 KB and strictly less than 8 KB.

2.1 Average IO Size Presents an Inverted Picture of Real-World IO

The absence of much transaction or data transfer activity in the "greater than 8 KB and less than 64 KB" range is interesting because the average IO size falls solidly in that range. This is true not only for the cumulative average size of all of the IO we observed (average read size 45 KB, average write size 24 KB, overall 36 KB) but also when we find the average IO size of deployments taken individually (Figure 3).

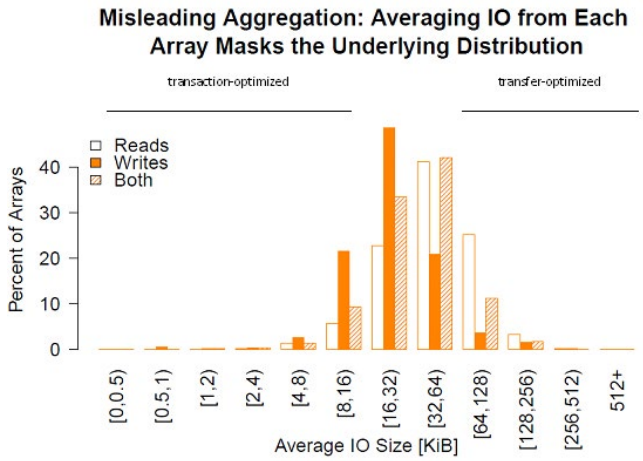


Figure 3: Histogram of Average IO Sizes per Nimble Array.
The same telemetry used to generate Figure 2 is repurposed to show the proportion of arrays with an average IO size in a given interval. By performing this average the detailed behavior is lost and consolidated into a single number per system. With every array doing some mixture of small and large IO, this average smears together the small with the large and gives a value somewhere in between. A distribution of these per-array averages is easy to misinterpret. On its own, it might be taken to suggest that there isn't much activity in the [4-8) KB range but we know from Figure 2 that this just isn't true; 23% of reads and 43% of writes are performed in that interval. Note that brackets [] indicate inclusive interval boundaries while parentheses () indicate exclusive interval boundaries; as an example: the interval [4-8) KB includes all IO sized greater than or equal to 4 KB and strictly less than 8 KB.

If we were to mistakenly assume that real-world IO conformed to this distribution of averages we would get a very different answer to our question: the majority of both the count of operations (76%) and the amount of data transferred (64%) would seem to fall within the intermediate (greater than 8 KB and less than 64 KB) range, reflecting a world where applications were not following the "divide and conquer" strategy, but were "splitting the difference" as we described above.

How do these two distributions, derived from the same dataset give such radically different results? As an analogy, one can imagine a city block with 10 single-story buildings and a single 100-story skyscraper. The average height of a building on the block is 10 stories tall – even though none of the buildings on the block are 10 stories tall – not even close. Rather than give the average, it would be more informative to say that >90% of buildings are one story tall but >90% of the floors are in buildings with at least 100 stories.

The complete disparity between these results highlights the importance of tabulating the sizes of IOs and not simply measuring their averages. Only by looking at a true histogram of IO by operation size (which takes 24 sensors) is the actual picture of how applications operate apparent. Simply measuring the number of IO performed and the data transmitted in aggregate (which only takes 4 sensors) is wholly insufficient.

We are not done yet though. Just because the aggregate picture shows one trend doesn't mean that individual applications or deployments won't exhibit different behaviors. We cannot determine from these histograms alone what proportion of individual deployments follow the "divide and conquer" vs. the "split the difference" pattern of IO requests, if either. To answer that question we look at the distributions of application IO requests at the individual deployment level.

2.2 Deployment-to-Deployment Variability

Having examined what the IO distribution of the Nimble install base looks like in aggregate – we turn our attention to how different deployments vary from one to the next. In order to summarize this information concisely, we use a combination of density maps and contours in addition to histograms (Figure 4).

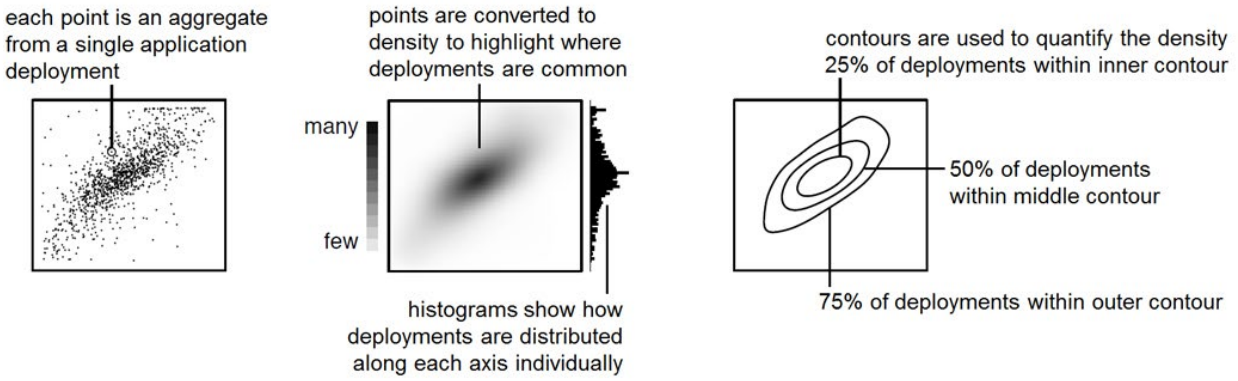


Figure 4: How toInterpret Density Maps: Summarizing Deployment-to-Deployment Variability.
Left: An individual point indicates the aggregate x-axis and y-axis values from a single application deployment. In this view, points can cluster on top of one another and make it difficult to determine where deployments concentrate most. Center: By converting the leftmost plot to a density map, the ambiguity resulting from overlaying points is removed and darker regions show where deployments are concentrated. Histograms on the top and right side of the panel show how deployments are distributed relative to the x- and y-axis dimensions respectively. Right: to allow the density map to be interpreted quantitatively, we add three contour lines to each density map; the innermost contour contains 25% of the observed deployments, the middle contour contains 50%, and the outermost contains 75%. We note that for some applications, one or more contours may be split across multiple regions of the density map; because the contours are drawn at locations of equal density – this splitting occurs when the deployments for that application cluster into distinct regions.

Having outlined the method for visually representing individual deployments – we now turn to specific metrics. For each deployment we are interested in knowing to what degree an individual deployment is "transaction optimized" or "transfer optimized"; we can quantify this by measuring the proportion of operations with IO sizes less than 8 KB (transaction optimized) and, separately, the proportion of data transmitted with by IO sizes larger than 64 KB (transfer optimized). We then plot the degree to which a system is transaction optimized on the x-axis and the degree to which a deployment is transfer optimized on the y-axis (Figure 5.1). We perform this analysis separately for read and write IO. In both cases, real-world deployments concentrate into three of the four regions: transfer specialized, transaction specialized, or the "divide and conquer" scenario.

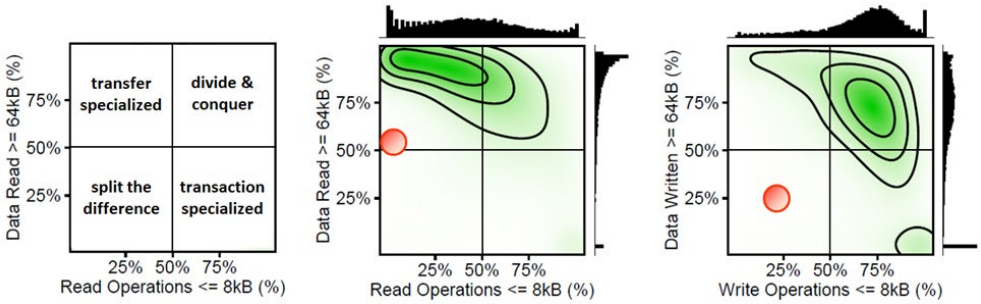


Figure 5.1: Density Maps of Deployments
(Transaction- vs. Data Transfer-Efficiency Optimization; Application Agnostic).
X-axes: the proportion of all operations performed at sizes less than 8 KB. Y-axes: the proportion of all operations performed at sizes larger than 64 KB. Left: indication of the significance of each quadrant. Center & Left: read & write data respectively. Green density & contours indicate the proportion of deployments observed with the corresponding X- and Y- axis values as described in Figure 4. Orange circles indicate where a deployment would appear if it were to exhibit an IO distribution equal to that shown in Figure 3.

2.3 Real-World Deployments Look Nothing Like the Average

Importantly, this analysis illustrates how very rare it is for even an individual deployment to exhibit IO patterns corresponding to a "split the difference" compromise between transaction-focused and data-transmission-focused efficiency. Additionally, by superimposing on this density map the location where a deployment would appear if it were to exhibit the IO distribution in Figure 3, we also have a very concrete representation of just how poor the average IO size really is at conveying information about how applications actually use storage.

To get another angle on this IO block size issue, we estimated the median read and median write size for each deployment observed. The distinction between read and write behavior observed in Figure 5.2 is apparent here as well – while the median read size spans a range of IO sizes from deployment-to-deployment, the median write size is fairly consistently 4 KB (meaning that for each of those deployments more than 50% of their operations are 4 KB or less. This image is consistent with a general observation that writes are typically much more transactional in nature than reads: which makes sense if we think of the writes being small incremental "updates".

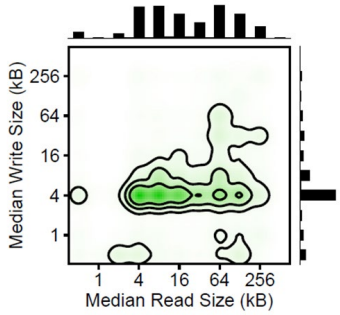


Figure 5.2: Density Maps of Deployments (Median IO Sizes; Application Agnostic). X-axis: median read size. Y-axis: median write size. With reasonable consistency, most deployments have a median write size of 4 KB (meaning that at least 50% of writes are 4 KB or smaller in those deployments). Reads, however, have median sizes that vary much more widely.

2.4 The Read vs. Write Balance

Given that in both Figure 5.1 and Figure 5.2 we have distinguished between read and write activity for individual workloads, it seems important as well to quantify the proportion of the total IO that is reads. We perform that analysis in conjunction with an estimate of logical sequentiality provided by our operating system to show the correlation between higher proportions of sequentiality and higher proportions of reads. As can be seen the deployment-to-deployment variability in the proportion of IO performed as reads is very wide and close to uniform on the 0-100% interval with some enrichment towards 100% writes (far left) and towards 50% reads, 50% writes (center).

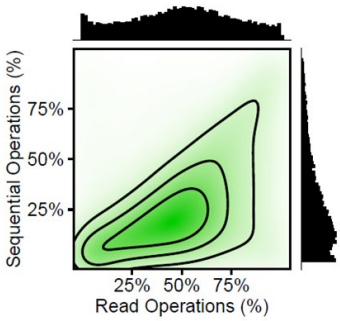


Figure 5.3: Density Maps of Deployments (Read Proportion & Sequentiality; Application Agnostic). X-axis: the proportion of operations for each deployment that are reads. Y-axis: the proportion of operations for each deployment that are sequential (in logical block address space as identified by the Nimble OS).

3 Application-Specific Insights

Taking this analysis further, we next perform this same deployment-by-deployment breakdown on specific applications. Within each of those application categories – we see some interesting trends. The pattern shown here in the aggregate view, however, holds true for each application as well: it is very rare for any application deployment to perform the bulk of its IO requests and the bulk of its data transfer at IO sizes between 8 KB and 64 KB in size.

In order to get the most out of our analysis of individual applications we discuss related applications together and have added two additional density maps: one highlighting the median read and write sizes for each application, and another illustrating the percentage of IO that is performed sequentially (out of all IO) and the proportion of IO that is read (out of all IO).

3.1 Virtual Servers and Virtual Desktops

Virtual Servers and Virtual Desktops Infrastructure (VDI) exhibit similar cumulative IO histograms (Figure 6.1). Both applications perform a significant proportion of their IO in the 4 KB bin with Virtual Servers exhibiting a second large-block >=64 KB IO mode that in VDI is not as prominent. This pattern, where Virtual Servers resemble VDI but with some extra "transfer-optimized workload" added on continues through to the density map analysis (Figure 6.2). Virtual Desktops and Virtual Servers both very consistently show median write sizes at 4 KB (Figure 6.1 Far Left) with additional deployment-to-deployment variation in the median read size; Virtual Servers, however, tend to reach much higher median read sizes than Virtual Desktops. This difference is consistent with Virtual Servers tending towards more sequentiality than VDI (Figure 6.2 Far Right) since we expect the read operations to also be more sequential when optimizing for data transfer speeds. In short, Virtual Desktops are like Virtual Servers but with a greater proportion of transaction-optimized reads.

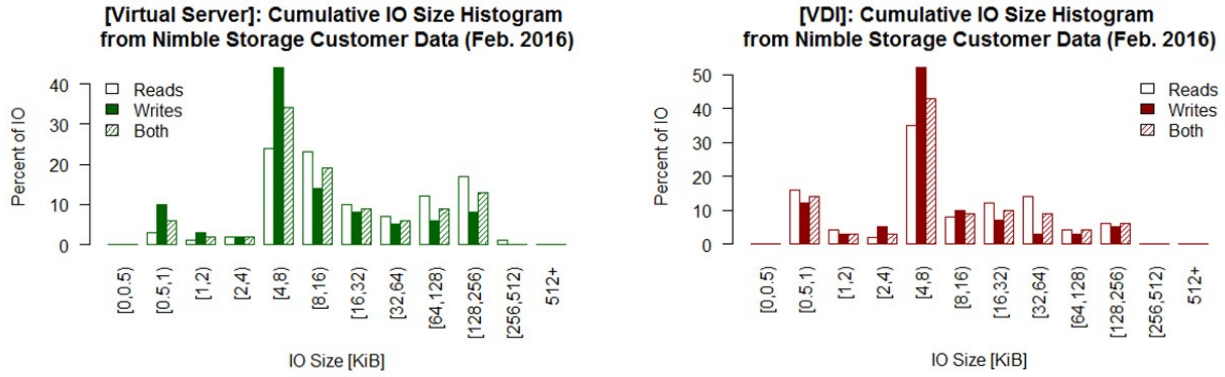


Figure 6.1: Cumulative IO Size Histograms for Virtual Server and Virtual Desktop Infrastructure.
Left: Virtual Server (Green). Right: Virtual Desktop Infrastructure (Red).

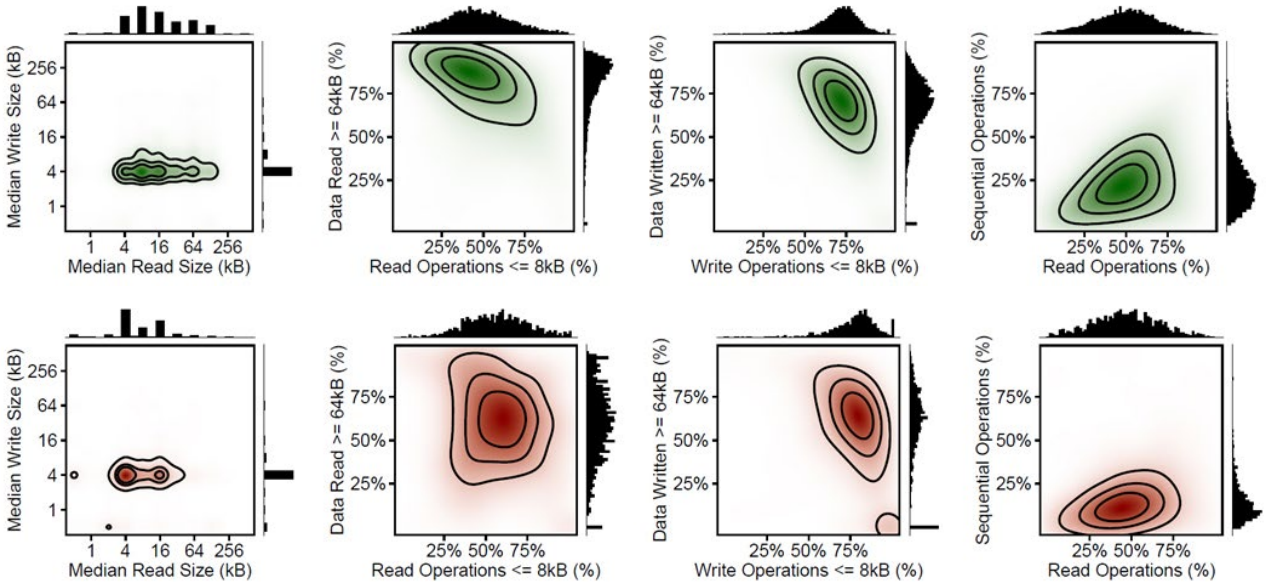


Figure 6.2: Density Maps for Virtual Server and Virtual Desktop Infrastructure Deployments.
Top: Virtual Server (Green). Bottom: Virtual Desktop Infrastructure (Red). Far Left: median write size vs. median read size. Center Left: percentage all read operations <= 8 KB in size vs. percentage of all data read by operations >= 64 KB. Center Right: percentage all write operations <= 8 KB in size vs. percentage of all data written by operations >= 64 KB. Far Right: percent of total operations performed sequentially as determined by Nimble OS vs. percent of total operations that were reads. For details on how to interpret these density maps, see Figure 4.

3.2 Oracle and Microsoft SQL Server Databases

In contrast to the virtualized environments we examined above, Oracle and SQL Server databases have their "transactional IO" peak in the 8 KB bin (Figure 7.1), rather than the 4 KB bin (Figure 6.1) ostensibly trading off some transaction performance for larger transaction payloads. When contrasting Oracle against SQL Server, SQL Server performs more than 50% of its operations in either the 8 KB or 64 KB bins showing strong bimodality (Figure 7.1 Right), while Oracle has its IO sizes distributed more broadly but still with its own transaction-optimized (8 KB bin) and transmission-optimized (128 KB bin) modes (Figure 7.1 Left). Examining deployment-to-deployment variability for Oracle and SQL Server shows two and three high-density clusters for the median IO size respectively (Figure 7.2 Far Left); Oracle deployments tend to cluster around [8 KB]/[8 KB] and [128 KB]/[8 KB] ([median read size]/[median write size]) while SQL Server deployments cluster around [8 KB]/[64 KB], [64 KB]/[64 KB] and [8 KB/[8 KB]. These results are consistent with a scenario where each database deployment individually has a bimodal IO size distribution – but the ratio of transactional vs. transmission-optimized IO changes from deployment to deployment (e.g. a bias towards OLTP vs. OLAP). Lastly, while both databases show transaction-optimized reads to varying degrees, SQL Server seems to more frequently exhibit transaction-optimized write activity (Figure 7.2 Center Right).

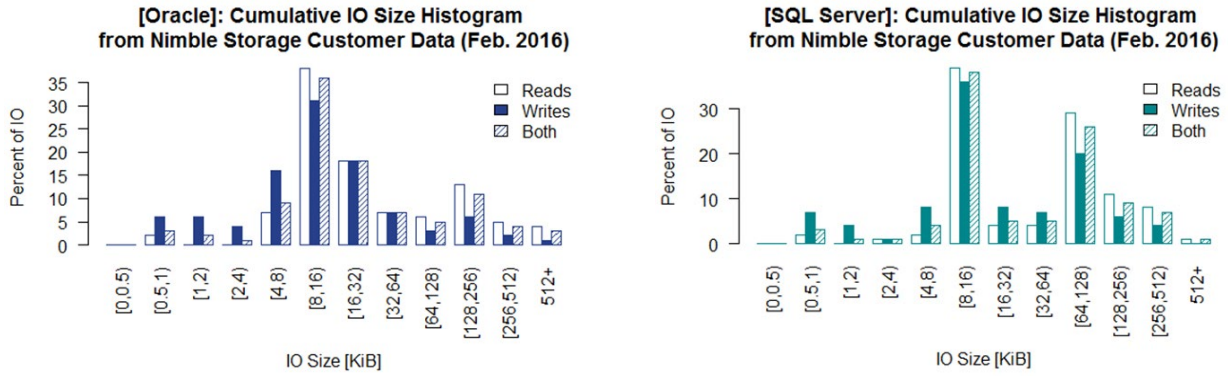


Figure 7.1: Cumulative IO Size Histograms for Oracle and Microsoft SQL Server Databases.
Left: Oracle (Blue). Right: Microsoft SQL Server (Teal).

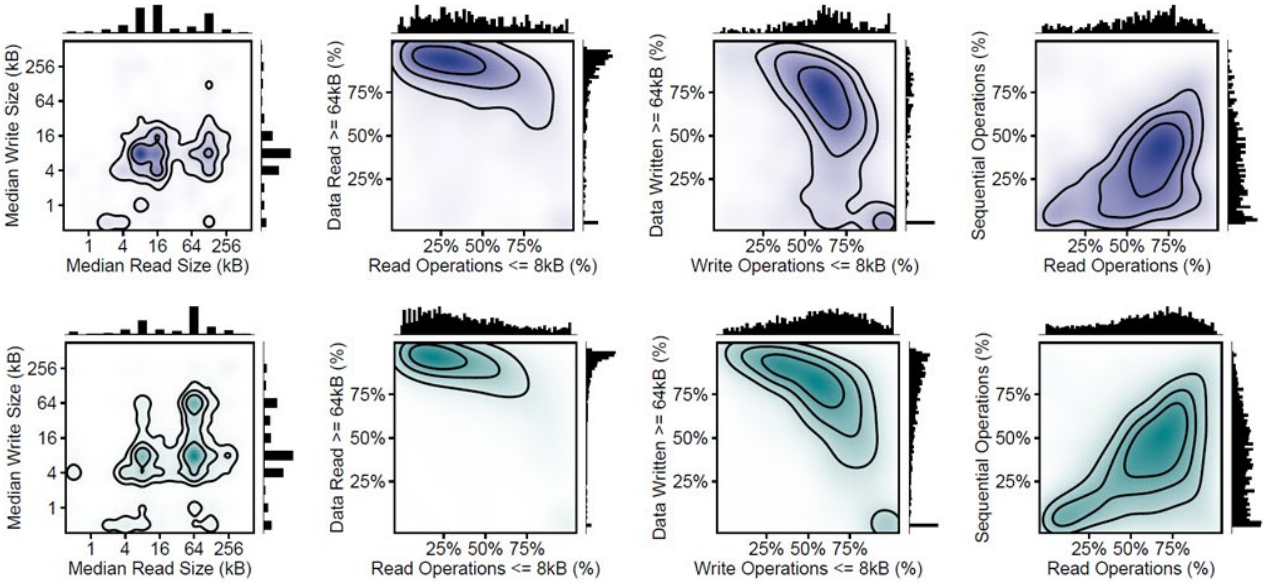


Figure 7.2: Density Maps for Oracle and Microsoft SQL Server Database Deployments.
Top: Oracle (Blue). Bottom: Microsoft SQL Server (Teal). Far Left, Center Left, Center Right, Far Right: Same as Figure 6.2.

3.3 File Servers and Microsoft Sharepoint

Comparing File Servers and Sharepoint is interesting because while both applications are mechanisms for institutions to share files, Sharepoint is optimized for collaboration and can enable small, highly interactive updates to small files (like power point presentations and text documents), while File Servers can handle both small files and very large files alike – albeit in a less collaboratively updated manner. While both applications can be used for similar files in a similar way – they can also be used very differently: Sharepoint much more interactively and File Servers with much larger files. This distinction manifests itself in the IO histograms, where Sharepoint's "collaborative updating" causes it to show one of the higher proportions of 512 KB writes (Figure 8.1 Right); this is comparable, for example, with VDI (Figure 6.1 Right). File Servers, in contrast, perform comparably few writes below 4 KB (Figure 8.1 Left). Also in keeping with File Server's ability to store larger files than Sharepoint, we see a much higher proportion of File Server deployments with their write activity in the transmit-specialized quadrant while Sharepoint instances more frequently have their write activity occupying the transaction-specialized quadrant (Figure 8.2 Center Right). This is also reflected in the larger population of File Server deployments exhibiting a large proportion of sequential IO (Figure 8.2 Far Right). The heterogeneity of the use cases for File Servers is something not well captured by the Cumulative IO size histogram alone (Figure 8.1 Left) because the "huge file" large-write-IO use case for File Servers is still only a subset of the overall File Server population. This differentiation, however, is much more visible in the density map views (Figure 8.2 Far Left).

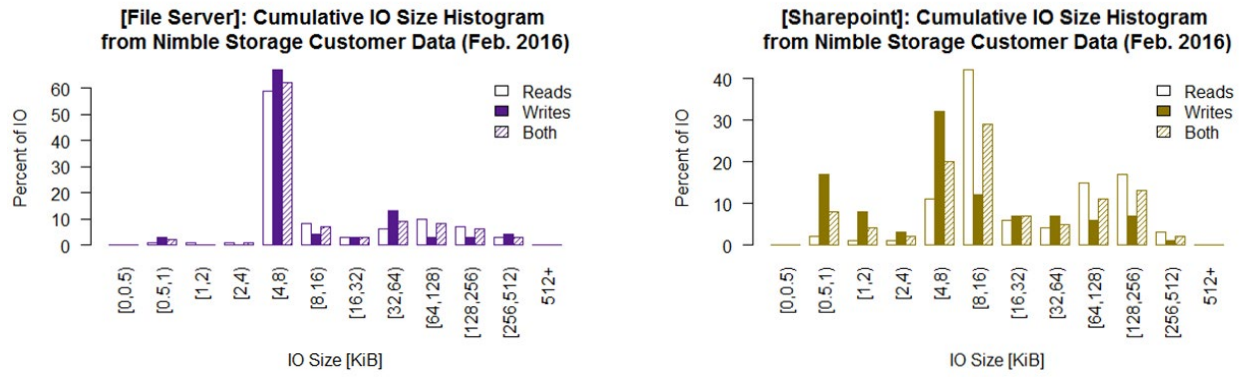


Figure 8.1: Cumulative IO Size Histograms for File Server and Microsoft Sharepoint. Left: File Server (Purple). Right: Microsoft Sharepoint (Gold).

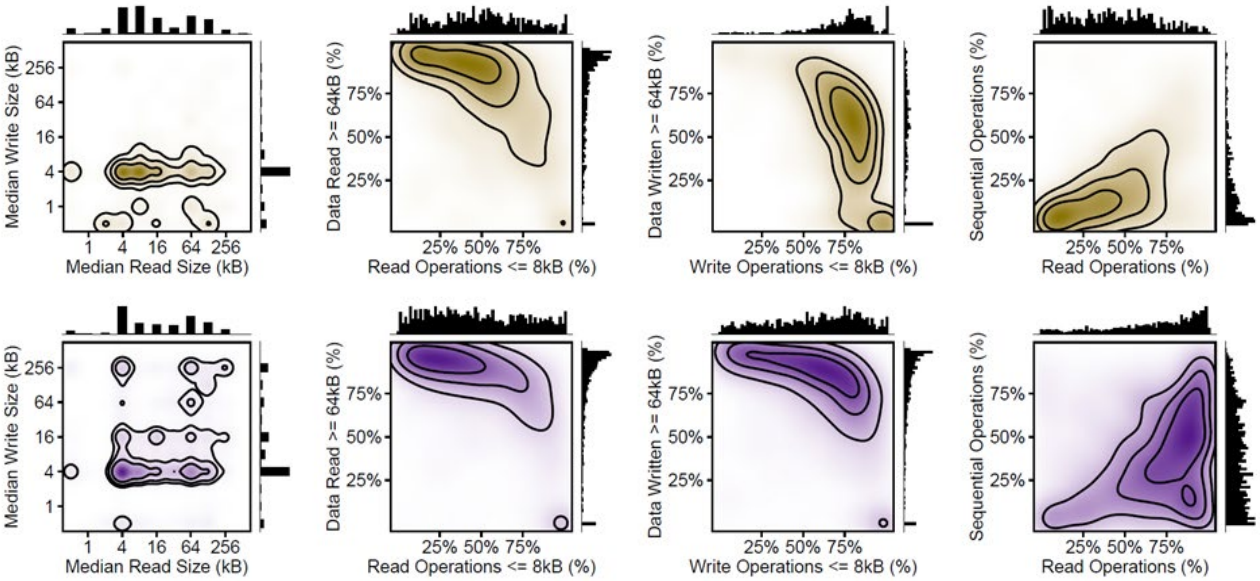


Figure 8.2: Density Maps for File Server and Microsoft Sharepoint Deployments.
Top: File Server (Purple). Bottom: Microsoft Sharepoint (Gold). Far Left, Center Left, Center Right, Far Right: Same as Figure 6.2.

3.4 Microsoft Exchange (2003, 2007, 2010, and 2013)

Microsoft Exchange is particularly interesting because the application has evolved significantly from 2003 to 2013 in terms of how it interacts with its underlying storage. While Microsoft Exchange versions 2003, 2007, 2010 and 2013 all behave at least somewhat differently – the biggest change from an IO perspective came between 2007 and 2010. The evolution of Exchange is in many ways the conversion of the application from a transaction-centric to a data-transmission-centric paradigm as email mailboxes grew in size and as disks grew in capacity without proportionally scaling IO/sec capability. Both Exchange 2003 and 2007 perform an extremely large proportion of their IO at 4 KB and 8 KB respectively (Figure 9.1).

The increase in IO size from 4 KB to 8 KB represents the first incremental move from a transaction-centric paradigm. In both cases most IO is non-sequential, but the proportion of sequential reads increases generally from Exchange 2003 to Exchange 2007. In the same way that the IO in the aggregate histogram concentrates highly at one IO size, the deployment-to-deployment variability of IO sizes are the smallest observed amongst the applications discussed here (Figure 9.2).

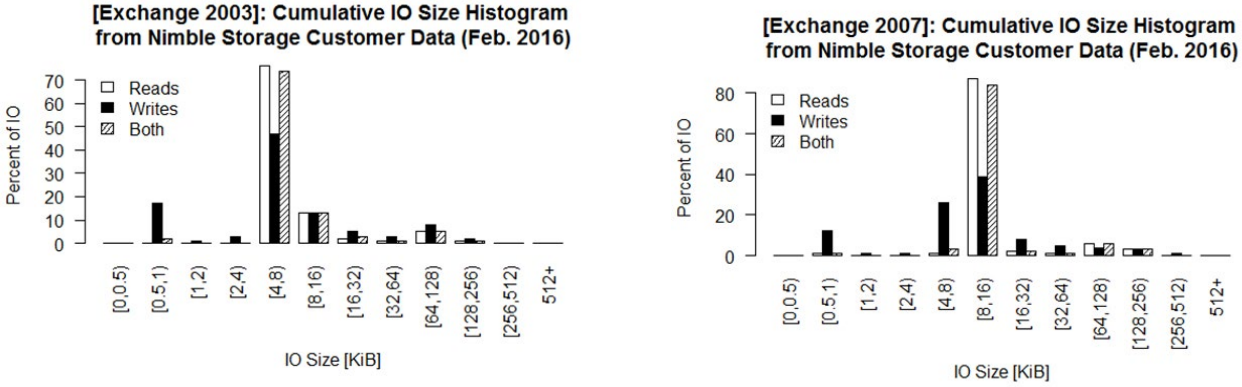


Figure 9.1: Cumulative IO Size Histograms for Microsoft Exchange 2003 & 2007.
Left: Exchange 2003 (Grayscale). Right: Exchange 2007 (Grayscale).

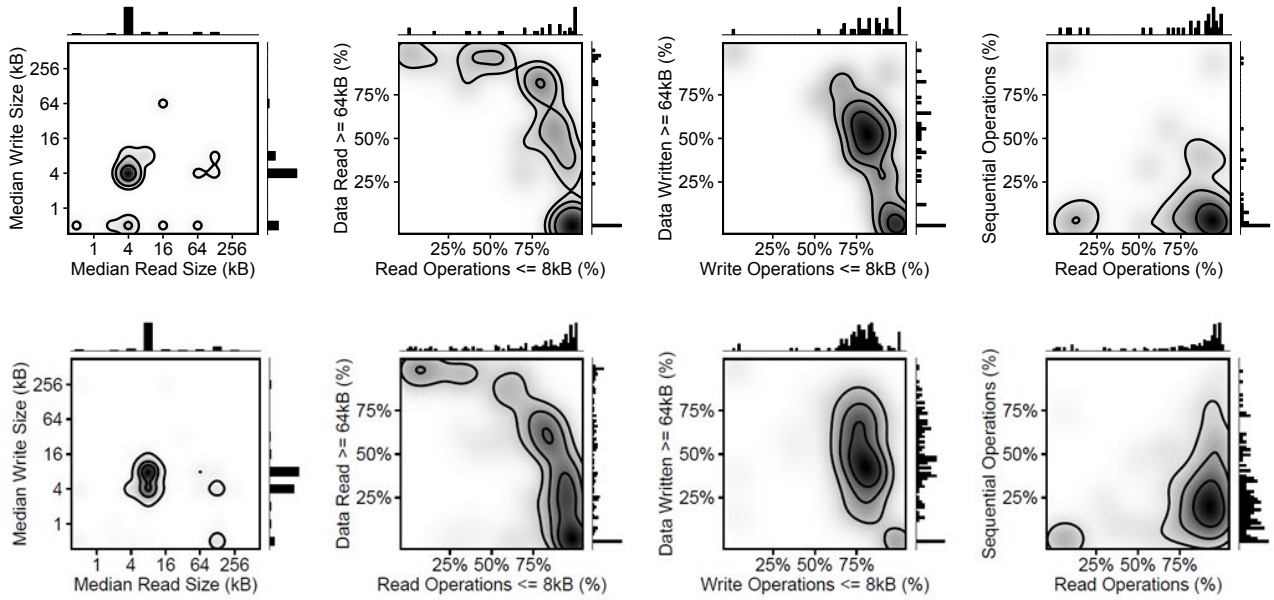


Figure 9.2: Density Maps for Microsoft Exchange 2003 & 2007 Deployments.
Top: Exchange 2003 (Grayscale). Bottom: Exchange 2007 (Grayscale). Far Left, Center Left, Center Right, Far Right: Same as Figure 6.2.

The transition from Exchange 2007 (Figure 9.1 Right) to Exchange 2010 (Figure 10.1 Left) resulted in a seismic shift in the application's IO profile. A significant proportion of the read operations and to a lesser extent the write operations were aggregated into much larger operations, reflecting an optimization towards the growing transfer-centric needs of the application. As is consistent with a large scale shift towards transfer-optimized operations, this aggregation was accompanied by a significant increase of the sequentiality of the requests at the storage layer (Figure 9.2 Center Right vs. Figure 10.2 Center Right). Interestingly, at the same time that most Exchange operations were being aggregated into large and more sequential requests, the 8 KB transactional IO peak in the Exchange 2007 (Figure 9.1 Right) histogram was shifted back to 4 KB in Exchange 2010 (Figure 10.1 Left). In other words – the optimizations made in going from Exchange 2007 to Exchange 2010 did not represent a monotonic shift from a transaction-centric IO profile to a more transfer-centric IO distribution; it actually represented two separate optimizations: some operations were optimized for better transactional performance while, simultaneously, others were optimized for data transmission efficiency. By comparison, the differences between Exchange 2010 and Exchange 2013 were not as large in terms of the IO profile sent to the storage. The bimodality that was introduced in transitioning from Exchange 2007 to Exchange 2013 was retained (Figure 10.1).

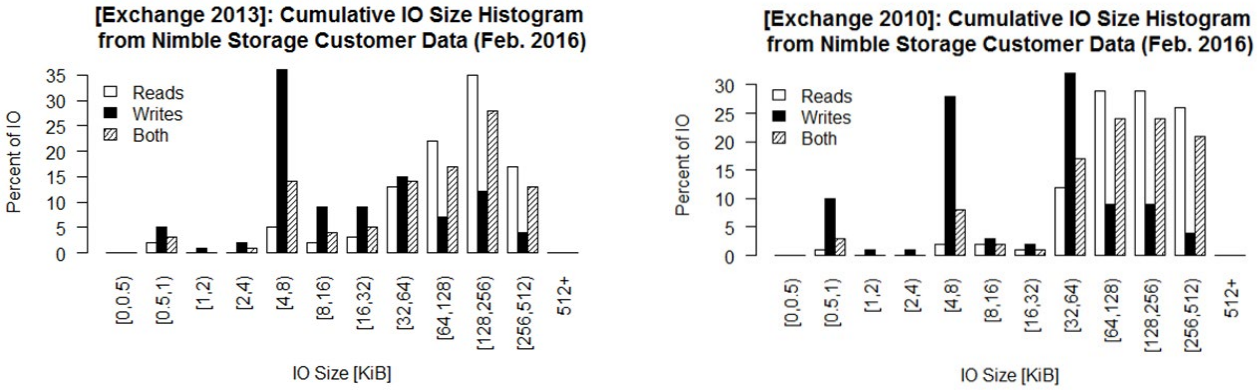


Figure 10.1: Cumulative IO Size Histograms for Microsoft Exchange 2010 & 2013.
Left: Exchange 2010 (Grayscale). Right: Exchange 2013 (Grayscale).

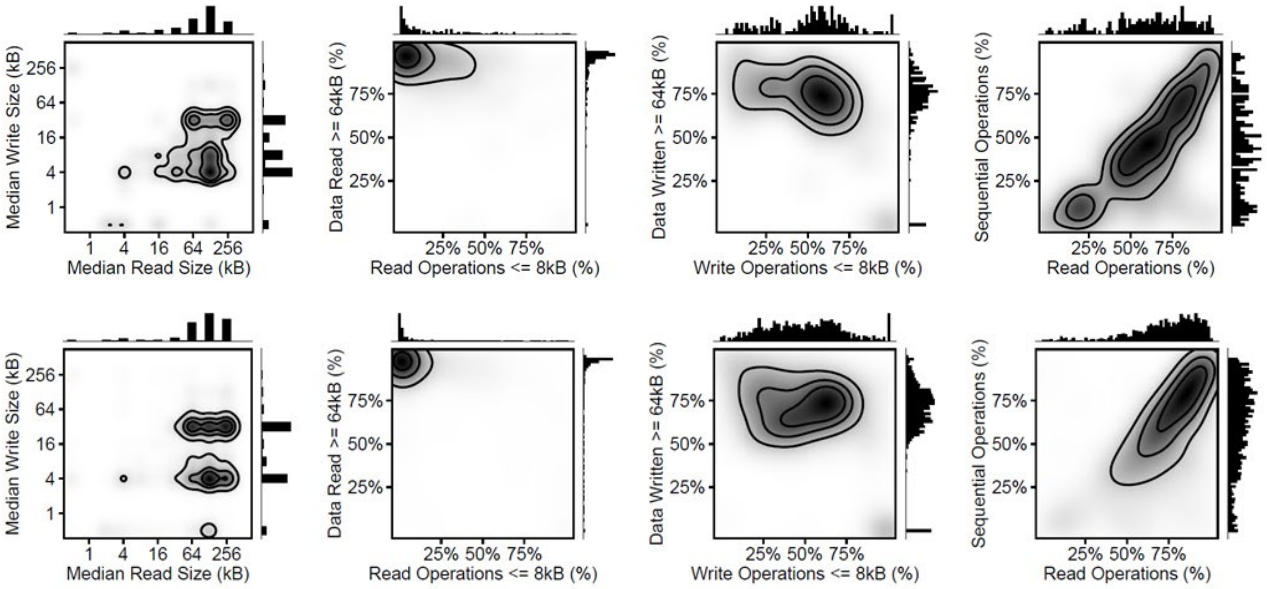


Figure 10.2: Density Maps for Microsoft Exchange 2010 & 2013 Deployments.
Top: Exchange 2010 (Grayscale). Bottom: Exchange 2013 (Grayscale). Far Left, Center Left, Center Right, Far Right: Same as Figure 6.2.

3.5 Splunk

Splunk, as the recipient of large quantities of machine-generated telemetry, is the most consistently write-heavy application observed in this study (Figure 11.2 Far Right). Those writes are largely randomly located (Figure 11.2 Far Right) and largely 4 KB (Figure 11.1, Figure 11.2 Far Right) and other than the proportion of IO dedicated to write activity, Splunk resembles other Virtual Server environments with regards to its IO distribution and deployment-to-deployment distributions (Figure 11.2, Figure 6.2).

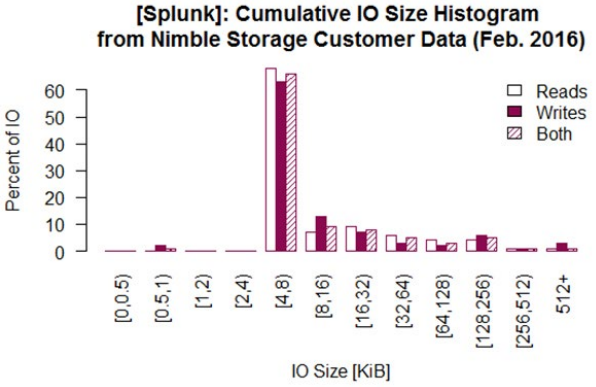


Figure 11.1: Cumulative IO Size Histograms for Splunk.
Left: Splunk (Magenta).

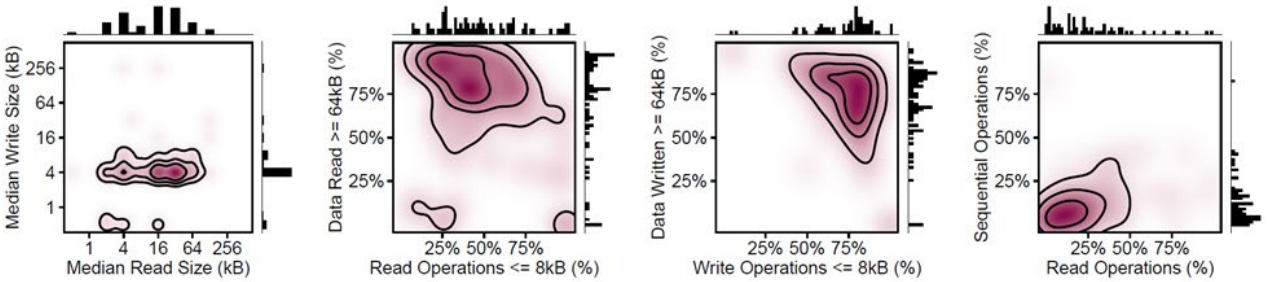


Figure 11.2: Density Maps for Splunk Deployments.
Single Row: Splunk (Magenta). Far Left, Center Left, Center Right, Far Right: Same as Figure 6.2.

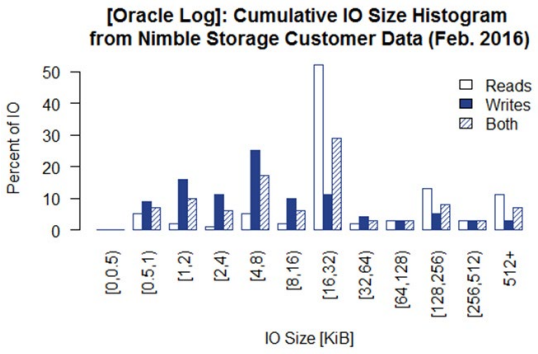
4 Conclusion

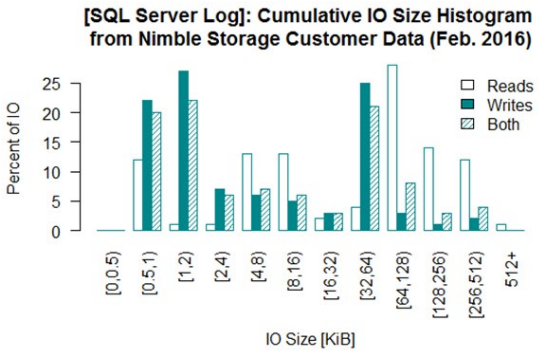
At this point we have shown how the IO profiles of numerous applications can be understood in terms of the proportion of their IO that is optimized for transactions vs. data transfer. While different applications will generate different distributions of IO requests, a common theme runs throughout: while the relative importance of transactions vs. data transfer fluctuates from application-to-application and from deployment-to-deployment, when applications need to optimize for some combination of both – that compromise is not achieved by "splitting the difference". Rather than issuing requests of intermediate sizes – the overall distribution of requests is split. A few very large operations handle data movement while many tiny transactions maintain the applications currency and responsiveness. This result seems very sensible – but simultaneously proves that performing IOPS or throughput (MBPS) benchmarking at intermediate IO sizes (16 KB to 64 KB) will not be particularly informative for testing application performance. Instead – IOPS measurements should be performed at small (8 KB or lesser) IO sizes and throughput (MBPS) measurements should be taken at large (64 KB or greater) IO sizes. Those are the ranges where business applications do their work.

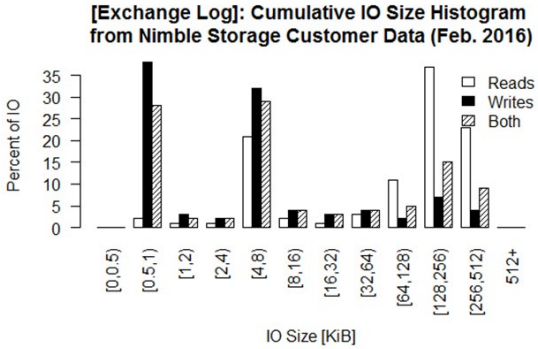
4.1 Methodology

Nimble Storage arrays retain a histogram of the sizes of incoming read and write requests. Each IO request the array receives is recorded in this histogram. While the histogram collects the proportion of IO observed within set intervals (24 independent sensors incrementing over time, one for each histogram bin for both reads and writes independently) – additional sensors identify the proportion of IO that aligns with the inclusive bin boundaries. When we measure for each array the proportion of IO that is a multiple of 4 KB in size, the median value is 87%. Because a large proportion of the IO sizes are known exactly, we can have good confidence that assuming a uniform distribution of IO sizes within each bin (on a log scale) will allow us to generate an accurate estimate of the median IO size for each array. Since the proportions of IO larger than or smaller than multiples of 4 KB are explicitly captured in these histograms, no estimation heuristics are required for determining the proportion of IO in any interval closed on the left and open on the right – those values are measured explicitly.

5 Appendix: Cumulative Histograms of IO Sizes (Application Logs)







NIMBLE STORAGE
211 River Oaks Parkway, San Jose, CA 95134
Phone: 408-432-9600; 877-364-6253
Email: info@nimblestorage.com
www.nimblestorage.com