IoLens: Visual Analytics System for Exploring Storage I/O Tracking Process

Changmin Jeon[*†], Jiwon Ha[*†], Hyolim Hong[*†], Hyeon Jeon[‡], Hyeonsang Eom[†], Heonyoung Yeom[†], Jinwook Seo[†]

Seoul National University

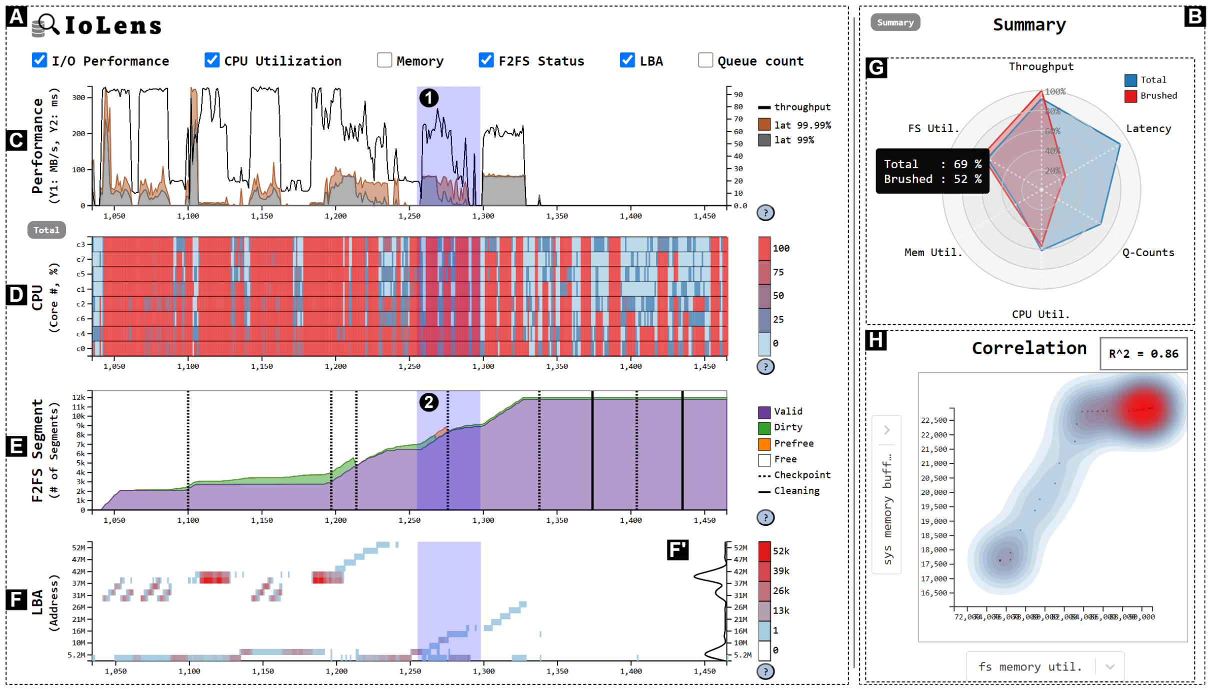


Figure 1: Overview of IoLens. (A) Metric view visualizes various system metrics, including performance information, acting as a basis for I/O performance analysis. (B) Summary view provides summarized and detailed content information from the left metric view. (C) Combination chart represents throughput as a line and latency as an area divided into two levels: 99% and 99.99%. (D) Heatmap is sorted and visualized in descending order by CPU utilization by core. (E) Cumulative stacked area chart visualizes the status of F2FS segments. (F) A heatmap visualizes Logical Block Address (LBA) access patterns. (G) Radar chart compares summarized system metrics for total data and brushed data, which can be switched to plain text using the toggle button. (H) Contour and scatter chart show correlation between user-selected system metrics. IoLens is available at `iolens.github.io/demo/`.

Abstract

As we enter the era of big data, a substantial amount of Input/Output (I/O) requests to storage devices are generated, making the maintenance of I/O performance important. Furthermore, I/O performance directly affects the overall user experience in edge devices. However, with the increasing complexity of systems, numerous factors influencing I/O performance have emerged, making it challenging to analyze and explore the overall I/O processing workflow. To address this issue, we introduce IoLens, a visual analytics tool that helps users explore system I/O performance from kernel I/O stack up to virtual file system and storage device drivers. Our tool helps users analyze I/O performance by identifying the overall workload of I/O requests and intuitively identifying anomalies. The effectiveness and applicability of IoLens have been validated through a usage scenario following a system engineer working on system kernels. A user study with four domain experts is conducted to further validate the usability of the tool.

[*]equal contribution
[†]e-mail: {salute, jwh0245, hyolim98, hseom, yeom, jseo}@snu.ac.kr
[‡]e-mail:hj@hcil.snu.ac.kr

Index Terms: Human-centered computing—Visualization—Visualization systems and tools; Computer systems organization

1 Introduction

The recent development of NAND flash made storage performance comparable to the performance of main memory devices. This made storage Input/Output (I/O) performance to significantly influence overall performance of the computer hardware system. Such importance has been reinforced as we entered the era of *big data*, where substantial data is saved in system storage (e.g., data center) and frequently accessed. However, analyzing I/O performance is challenging because it is influenced by various factors in the system rather than by specific factors, e.g., storage speed.

Therefore, several studies aiming to analyze and manage the I/O workload of storage devices have emerged [6, 15, 16]. The purpose of these studies is to reduce overall resource usage and energy consumption [6, 16] or improve the performance of supercomputer system [15] by optimizing storage device performance. Also, various benchmarks have been developed to measure system I/O performance, e.g., fio [1] and filebench [14]. These benchmarks are compatible across a variety of systems and provide key performance indicators such as throughput and latency for measurement results, allowing users to easily examine system performance. However, these benchmarks only provide summarized information about performance results despite their convenience.

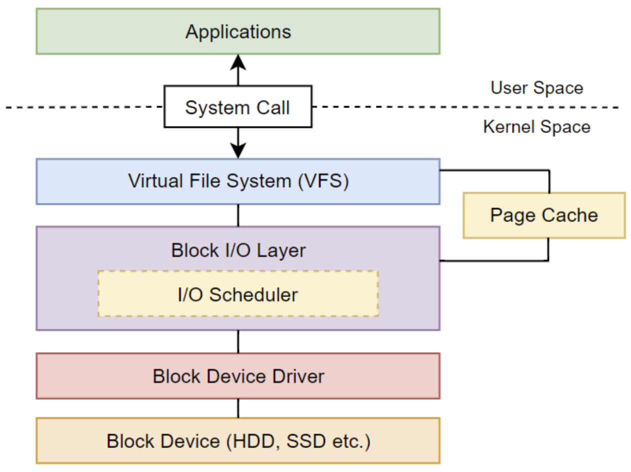


Figure 2: The components of the Linux kernel I/O stack, including the Virtual File System (VFS), Block Layer, Device Driver, I/O Scheduler, and Page Cache. The Linux kernel I/O stack is structured with multiple layers to efficiently handle I/O requests.

Moreover, the current practice of analyzing storage I/O performance has several limitations. For example, the existing tools that analyze the I/O stacks only focus on the block layer [3, 8]. Although the block layer is an important layer that acts as a buffer in the middle of the I/O stack, it is only one of the complex I/O stacks (Figure 2). I/O performance may fluctuate due to events in other layers of the I/O stacks, which makes it difficult to analyze performance accurately by examining only the block layer. Furthermore, in practice, a system analyst analyzes storage I/O through the system logs consisting of long texts. This method is complicated and burdensome, and it relies on the analyst's experience or skills. To resolve this issue, we have our research motivation to explain multiple layers within I/O stacks using appropriate visualizations, thus supporting users in performing I/O analysis effectively.

To this end, we present IoLens, a visual analytics tool that allows users to analyze storage I/O performance at a glance. IoLens explains the entire kernel I/O stack, from the device driver level (lowest) to the virtual file system layer (highest) (Figure 2). Our tool also visualizes CPU and memory utilization, which are crucial information in I/O performance analysis. Moreover, IoLens provide charts that correspond to different system metrics while linking them to support users in readily identifying anomalies related to performance degradation coupled with multiple metrics.

We demonstrate a usage scenario as a validation of the effectiveness and applicability of the proposed tool. In the scenario, we show that IoLens is capable of supporting users to understand I/O processing within the kernel and to find the fundamental cause of performance degradation. To this line, we also carried out a user study involving four domain experts, each possessing over six years of experience in dealing with system I/O performance. The experts agreed that IoLens is effective in identifying the characteristics of unfamiliar workloads, finding points of performance degradation, and analyzing the cause of the degradation.

2 Background and Related Work

2.1 The Understanding of Storage Device I/O

2.1.1 Kernel I/O stack

The kernel I/O stack's role is to process user space I/O requests passed to kernel space through system calls. The kernel I/O stack consists of a virtual file system (VFS), block layer, device driver, and various other components (Figure 2). While VFS provides core interfaces to support various file systems, the file system structurally manages data such as names, sizes, and addresses of files and directories. The block layer provides an interface to read and write data to block devices such as HDD and SSD. The device driver manages communication between the physical hardware and the operating system. Such a complicated I/O stack allows the kernel to handle I/O requests efficiently but also makes the analysis difficult. Therefore, it is analyzed only based on information from a specific layer, e.g., the block layer, in practice. Instead, we propose a visual analytics tool that can readily analyze entire I/O stacks within a single dashboard.

2.1.2 Log-structured file system

In Linux, the kernel supports various file systems through VFS. Recently, the Log-Structured File System (LFS) has been widely used due to its improved read and write performance based on sequential write policies [12]. LFS manages data by dividing the entire storage space on the file system into segments. The segment is again divided into a free segment that can record data, a valid segment where only valid data exists, and a dirty segment where unnecessary data exists. The file system obtains a free segment from the dirty segment through cleaning operation. Checkpoint is the process of storing metadata in the current file system to maintain data consistency in the file system. One widely used LFS is F2FS [7]. It was designed to perform flash storage devices more efficiently and provides a summary of the current file system status. In IoLens, we provided the status and internal events of the file system so that users can readily understand the behavior of the file system.

2.1.3 Meaning of I/O performance

I/O performance is represented by system metrics such as throughput and latency [5]. Throughput is bandwidth and IOPS (input/output operations per second), both important metrics in storage. Bandwidth is calculated as $IOPS * Chunksize$, usually in MB/s. Latency is the response time to a single I/O request, which is the time it takes for the i/O request to complete. While average latency is important, long or tail latency is critical in determining the user experience [4]. Queue depth is an index of how many I/O requests are being processed (in-flight I/O). If the queue depth is high, it is likely that the storage disk is not capable of handling many requests due to insufficient performance. In this case, the latency becomes longer because of the time waiting in the queue. Given that I/O performance is not solely determined by a single factor, it is imperative to comprehensively identify various pieces of information that can influence performance. In this context, IoLens supports integrated visual analytics based on diverse system metrics related to I/O performance.

2.2 Visual Analytics for System Performance

Recent studies provided insights for optimizing and monitoring system performances. For example, HPC2lusterScape [11] suggested a visual analytics system for enhancing resource utilization workload balance in high-performance computing clusters for big machine learning models. Shilpika et al. [13] utilized streaming functional data analysis for real-time hardware system monitoring. Meulder et al. also studied performance monitoring in cloud computing, focusing on the individual behavior of computing nodes [10] and suggested visualization methods to analyze the I/O traces in high-performance computing [9]. However, to the best of our knowledge, the visualization community lacks a comprehensive analytics system that specifically focuses on the storage I/O tracking process. Seekwatcher [8] is a tool that visualizes key performance metrics based on the output of *blktrace*. Nevertheless, these performance metrics alone are not sufficient to debug storage-related issues. The aim of our research is to fill this gap.

3 Tasks and Requirements

We conducted a preliminary semi-structured interview with three engineers with more than six years of experience in I/O performance optimization. The content of the interview included performance analysis procedures, improvements to the existing analysis process, and information that is easy to miss when analyzing performance. Throughout the interview, we mainly focus on revealing (1) the importance of considering various data in I/O performance analysis, and (2) the inconvenience of representing data using charts. Based on the results, we established the following tasks and requirements.

3.1 Tasks

(T1) Understanding overall I/O traces. When users analyze I/O performance, they should be able to identify the overall workload along with performance indexes. Users should also be able to obtain detailed information at specific time periods.

(T2) Detect anomalies related to performance. In system I/O performance analysis, scrutinizing anomalies enables the early detection of deviations from the normal operation of the system, facilitating the early identification of potential problems. Also, anomaly analysis aids in recognizing unexpected situations, such as performance degradation or system errors, and helps find appropriate improvements. Thus, users should be able to detect unexpected anomalies related to system I/O performance without much attention, ensuring that visualizations effectively distinguish anomalies from normal patterns.

(T3) Compare the indicators of the system. Users should be able to compare whether there is a correlation between system metrics. They also need to compare the data of the user-specified timeline with the overall data. This comparative analysis is essential to identify anomalies and resolve performance degradation issues.

3.2 Requirements

(R1) Provide an overview of I/O processing (T1). When analyzing a system, users tend to visualize only specific information they deem significant. However, users may overlook valuable details in this approach. Therefore, in order to facilitate users to gain unexpected insights, the system should comprehensively depict the overall system information. In addition to visualizing I/O performance data, the tool may offer an overview of I/O performance by combining related data to understand the overall context of I/O traces. The tool should also allow users to obtain detailed information about a specific time period.

(R2) Select the visual channels that maximize information delivery and legibility (T1, T2). Selecting the appropriate channel based on the data characteristic is crucial for effectively delivering information to users. In this tool, the application of suitable channels for visualizing information is determined through expert interviews, considering the nature and significance of each dataset. It is essential to contemplate whether the graph aims to illustrate temporal trends, distribution patterns, or other aspects. Additionally, visual legibility and the user's immediate comprehension must be taken into account. Ensuring that information is conveyed clearly and is easily interpretable enhances user understanding of the data and facilitates the identification of inherent patterns and anomalies. In instances where it enhances analysis, the tool should concurrently support multiple channels within a singular chart as required.

(R3) Visualize data with minimal loss (T1, T2). With tens of thousands of I/O requests occurring per second, a substantial volume of log data is generated. In order to extract meaningful information from this data, the tool should provide effective data visualization that takes into account both the quantity and complexity of the information. Furthermore, it is significantly important to ensure that all data, especially outlier data, is neither overlooked nor deleted. Outlier data is defined as values that deviate from the usual operational patterns, and this is often closely associated with anomalies. These outlier values can represent unexpected events, system issues, or other crucial information. Effectively identifying and analyzing outlier data allows users or administrators to pinpoint abnormal system behavior or exceptional situations in the system and to take appropriate actions. Therefore, safeguarding against the inadvertent exclusion or removal of any data, including outliers, is inevitable for a meticulous analysis.

(R4) Support users to compare system metrics (T2). The system should be structured to clearly distinguish anomalies related to system performance. For this purpose, anomalies need to be distinctly separated from usual patterns. To enhance the efficiency of analysis within the visualization, relevant information is consolidated, while unrelated information is presented independently. Additionally, representing all charts over time facilitates easy observation of patterns in the indicators.

(R5) Support comparison of partial and total data (T3). The tool should support users in comparing the part of the data with the entire data. The comparative feature enables users to gain a detailed understanding of the system's behavior at specific points in time or under particular conditions. The selection of partial data allows users to obtain profound insights into specific events or performance anomalies. Comparing this partial data with the entire data helps the analysis of the system's overall behavioral patterns. This functionality serves as an advantageous tool for users, allowing them to focus on specific aspects of the system while considering the broader context, facilitating in-depth performance analysis.

(R6) Support correlation between metrics of system (T3). The system may support functionality to visualize and quantify the correlation between the metrics of the system selected by the user. This feature is beneficial for understanding the interactions among various aspects of the system, allows users to visually confirm the correlations between specific items and provides a quantitative understanding, enabling users to comprehend the overall behavior of the system more effectively.

4 IoLens

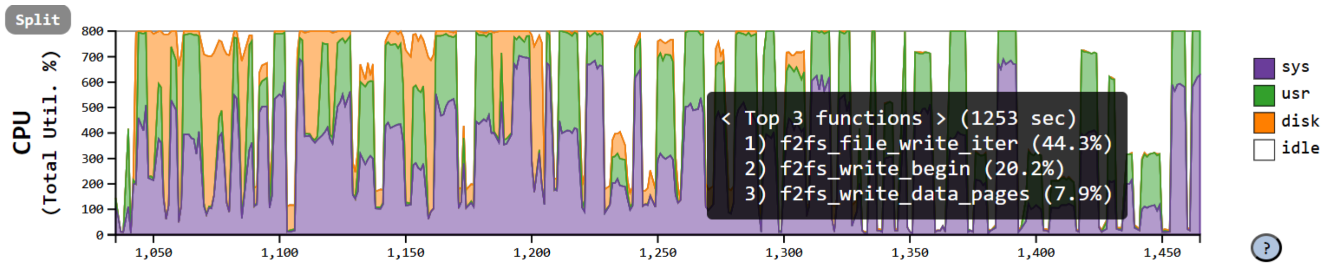
We present IoLens, a visual analytics tool to analyze storage I/O tracking processes. IoLens is designed as a dashboard served by a single webpage, where we used `D3` [2] and `React` for the implementation. We first explain how the data is preprocessed (Sect. 4.1). We then describe how two main views—metric view (Fig. 1A) and summary view (Fig. 1B)—are designed (Sect. 4.2, 4.3).

4.1 Preprocessing

We use the Linux 6.7.0 rc2 version of the kernel and collected system logs through several tools provided by the kernel. For example, I/O stack information is acquired through *f2fs status* and *ftrace*, and CPU and memory resource usage are recorded using *top*. We create a scenario to explore how high system resource usage can affect I/O performance. To create random scenarios, we use *stress* as a tool to load CPU and memory, and *fio* as an I/O benchmark tool. We set the conditions to random numbers to generate a random workload. Acquired data is parsed as a JSON file to be processed by web technology.

4.2 Metric View

The metric view is designed to support users in analyzing various system metrics (e.g., CPU utilization). The view acts as a grounded basis for overall I/O tracking process analysis. Charts corresponding to different metrics are provided within the view. We carefully designed the charts (e.g., heatmap for CPU and LBA; cumulative stacked area chart for F2FS Segment) to suit the characteristics of each metric. All charts are linked, sharing the same time axis. Users can check how other system metrics look at the specific timeline they want to analyze through the brushing. Moreover, users can use the checkboxes to toggle the charts they want to see. Note that we



```
< Top 3 functions > (1253 sec)
  1) f2fs_file_write_iter (44.3%)
  2) f2fs_write_begin (20.2%)
  3) f2fs_write_data_pages (7.9%)
```

Figure 3: CPU utilization with cumulative stacked chart, showing the system's CPU utilization over time. *sys* and *usr* refer to CPU resources used in kernel space and user space, *disk* refers to time spent waiting for I/O peripherals, and *idle* refers to time spent in idle operations. Hovering the mouse displays the top three file system functions with the highest CPU usage at that second. Clicking the split button switches to a heatmap chart showing usage by core.

placed the metric view on the left while the summary view is placed on the right. This is because, during our iterative design process, we found that users first became interested in the overall performance and then looked into summarized information.

4.2.1 Performance Chart

The performance chart (Figure 1-C) provides throughput and latency info, which are directly related to system I/O performance (Sect. 2.1.3). As the primary objective is I/O performance analysis, we positioned it at the top of the screen to grab the user's attention first. Among performance metrics, latency is the processing time for requests and may have abnormally long tail latency depending on the situation. When analyzing I/O performance, we need to focus on tail latency which directly affects the user experience. And we subdivided it into 99% and 99.99% to distinguish the level of tail latency. We expressed throughput as a line graph while using an area chart to express latencies. Since 99.99% of latency includes 99% latency, there is no part that is not expressed, even when drawn as an area graph. Contrast colors were used to clearly see the difference between 99% and 99.99% latency.

4.2.2 CPU Utilization Chart

The processing of I/O requests involves computation, and a shortage of computational power can result in delays in I/O processing. Indeed, when undertaking performance analysis, it is crucial to verify the availability of adequate CPU resources. CPU chart provides CPU utilization information in two different formats: heatmap (Figure 1-D) and cumulative stacked area chart (Figure 3), which users can toggle between by clicking the top-left toggle button. The cumulative stacked area chart shows different CPU utilization such as kernel space, user space, disk wait, and idle. The cumulative area chart is selected considering that the overall CPU utilization is a constant determined by the system, and the system used in the experiment consisted of 8 cores, so the maximum value is expressed as 800%. Additionally, when users hover over the stacked area chart, the tool shows which file system functions are active at a given point in time. If CPU and memory resources are insufficient due to file system operations, it is reasonable to suspect functions that require a significant amount of computation first. If the prediction is accurate, the root cause of the problem can be quickly identified and addressed. It provides information on the top 3 functions of file system with the highest CPU utilization at that specific moment. This cumulative stacked area chart allows users to understand how total CPU resources are distributed across these four areas. However, this approach shows overall CPU utilization, so it will end up showing average CPU usage both when certain cores are highly utilized while other cores are idle and when all CPU cores show overall average usage.

To address this issue, we also provide CPU usage information on a per-core basis in the form of a heatmap. It's possible to present a core-specific cumulative chart, but it's become too complicated. We chose the heatmap because the small-split cumulative chart didn't really show any meaningful information related to overall CPU utilization. In this heatmap, CPU core utilization is visually represented, with higher usage displayed in red and idle usage in blue for easy identification. Additionally, to allow users to easily determine which CPU cores were most used and most idle during the entire time frame, we sort and display core usage in descending order for the entire duration.

4.2.3 Memory and File system status Chart

The performance of I/O can be influenced by the system's condition. When there is insufficient memory, frequent memory eviction restricts caching capacity, and occasionally, additional I/O requests are generated internally during the file system cleaning process. In such instances, a user's I/O request may experience delays. Memory utilization data and the status of file system data are classified into specific categories within the same domain. Memory is divided into used memory, buffer/cache memory, and free memory (Figure 4), and the F2FS segment is divided into the valid segment, dirty segment, prefree segment, and free segment. When the data of all categories are added together, the sum always comes out to 100%, so it is expressed as a cumulative stacked area considering these characteristics. Contrast color maps were selected so that the stacked areas were clearly distinguished, and the free data was expressed in white to emphasize more important factors and to avoid confusion caused by using too many colors. File system status charts (Figure 1-E) show when segment cleaning events occur for garbage collection with solid lines and checkpoint events for file system consistency with dotted lines, so that the user can easily grasp the time when the segment changed.

4.2.4 LBA access and Queue counts Chart

When bottlenecks occur due to the concentration of I/O requests at a given moment, users can consider several ways to improve I/O performance. For example, users can improve overall I/O performance by switching to faster storage devices at additional cost, or by analyzing and optimizing workloads. To determine which factors to improve, users need to understand the storage's address access patterns and queue information. Initially, we attempted to represent all LBA access and queue count information using a scatter plot. However, since a general I/O trace is hundreds of thousands of data per second, using a scatter plot to visualize hundreds of seconds of data was difficult. Additionally, if multiple dots overlapped in a specific area on the scatter plot, it was hard to know how many dots were there. Therefore, to solve these problems, LBA access patterns and queue numbers are compressed and displayed as a heat map (Figure 1-F, 4). The heat map represents the aggregation of the frequency of LBA access and queue counts every second. Higher frequencies are depicted in red, while lower frequencies are shown

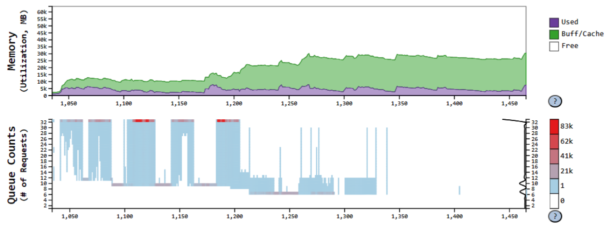


Figure 4: Charts showing memory usage and the number of I/Os waiting in the queue. The memory chart is classified into free, used, and cache according to memory properties, and the queue count chart expresses the concentration of waiting I/O as a heat map.

in blue. To minimize data loss, data with a value of 1 and data with a value of 0 are expressed in blue and white, respectively. The heatmap provides immediate insight into which values were accessed more frequently at specific timelines based on color. However, understanding the overall distribution of values is not straightforward. For example, in the heatmap of LBA access patterns (Figure 1-F), the address range between 37M and 43M appears to be the most accessed over the entire period. However, when examining the overall statistics, it becomes apparent that there was a significant amount of access near address 5.2M as well. Therefore, to allow users to easily identify the overall distribution, we provide a histogram on the right y-axis (Figure 1-F').

4.3 Summary View

After understanding the overall workload in the metric view on the left, when performing detailed analysis, it is necessary to check numerical values related to key information or compare various pieces of information. The summary view provides a radar chart that allows the user to compare the area of interest with the entire area, and hovering the label displays accurate numbers on the screen. Additionally, it provides a function to convert to plain text so that users can intuitively check numerical values. It also provides correlations to analyze correlations between metrics that are difficult to understand in a metric view.

4.3.1 Correlation Chart

IoLens allows users to check the correlation between performance-related metrics through a correlation chart (Figure 1-H). The correlation chart is implemented using the contour plot and the scatter plot together. The scatter plot intuitively shows the distribution of data, and the contour plot effectively expresses the density. The user can visualize the correlation between the two selected metrics by selecting the x and y-axis items of the correlation chart. Areas with high distribution density are shown in red, and areas with low distribution density in blue. In addition, since the correlation chart is linked to the charts on the left, it is updated in real-time to show the correlation within the selected data when the user uses the brushing interaction. In addition, intuitive quantitative values are provided together by calculating and showing the R^2 value in the upper right of the chart.

4.3.2 Summary Radar Chart

The summary radar chart (Figure 1-G) provides summarized information for six indicators of the system. Initially, it provides summarized data for the entire dataset (indicated in red). When the users brush on the left chart, additional summarized data for the brushed area is provided (shown in blue). It enables users to compare the total data and the brushed area's data. Users can also quantitatively compare total and brushed data by hovering over the name of the indicator. Additionally, if the user wants to compare the summarized information only in numbers, they can toggle the button at the top left to view the values of the entire data or brushed data only in numbers.

5 Usage Scenario

We provide a usage scenario with IoLens analyzing system I/O performance. We will now follow Susan, a system engineer working on system kernels. Here, Susan wants to find storage I/O performance degradation points and causes. For this purpose, she loads preprocessed data (Sect. 4.1) and starts an analysis.

At first, Susan looks at the performance chart (Figure 1-C). She finds that there is a gap between the latency 99.99% chart and the latency 99% chart at 1275 seconds, and the throughput decreases(Figure 1-1). The gap between latency 99% and latency 99.99 % means that the processing of some I/O requests has been delayed, resulting in a long latency. She thinks this is a point of performance degradation points and brushes the timeline to find the cause. The brushed timeline then displays the same time zone area of other charts so that the user can intuitively know the status of other indicators. While looking at various charts linked with brush, she finds an anomaly in the F2FS chart (Figure 1-E). An orange area and dashed line which is prefree segments and checkpoint event (Figure 1-2). At this point, dirty segments are converted to a free block through a prefree segment due to checkpoint. Through this, she concluded that the cause of the performance decrease was the occurrence of checkpoints.

As seen in the above scenario, IoLens can help users readily identify the cause of performance degradation through a brush linked to all charts. It is worth noting that this can hardly be done with the previous approach, which is to examine only the information of a specific layer at once. In this case, to find the cause of performance degradation as in the above scenario, users should look at the system log that has throughput and latency information and find points of performance degradation. Then, the expert may look at the part of the I/O-related information that is suspected to be the cause. At this time, if F2FS information is not provided, it is difficult to find the cause that Susan found in the above scenario. Therefore, if the workload is unfamiliar or the analyst is not sufficiently experienced, it may take a long time to determine the cause of performance degradation with the current practice.

5.1 User Study

5.1.1 Study Design

Participants. Four domain experts are recruited for the user study. Note that we recruited domain experts as IoLens is designed mainly for the experts. Two of them are Ph.D. students (E1, E2) majoring in computer science and engineering at a local university. The other two (S1, S2) are software engineers working for one of the leading chip manufacturers in the world. All participants have more than six years of experience in analyzing storage I/O.

Procedure. Every study was conducted in person. The detailed procedure is as follows: First, we briefly explained the purpose and the overall design of IoLens. We then demonstrate all the features of the tool. For example, we explained how all charts are linked through brush and what information each chart provides. Afterward, we asked participants to use IoLens freely to search for interesting findings or insights. The participants were guided to "think-aloud" their thoughts during the trial. After the trial, we conducted a post-hoc interview, asking for overall satisfaction and the possible improvements for IoLens.

5.1.2 Results and Discussions

Detecting the causes of performance degradation. At first, all participants mainly focused on the performance chart. Then brushed the timelines when there was a difference between the latency 99.99%

graph and the latency 99% graph in the performance chart (Figure 1C) or when the throughput got lower. After brushing the timelines, which is the point of performance degradation, participants compared the status of other charts at that time. Participants also checked the summary chart and correlation chart during the analysis. We observed that experts easily identified the cause of the performance degradation in Figure 1-1 by looking at an event (Figure 1-2) in which a dirty block changes to a free block due to a checkpoint in the brush-connected F2FS chart. Additionally (S1) said that fluctuation in the 1,100 second range seemed to be related to the checkpoint event in F2FS Segment. All participants said that the brushing feature linked with all charts allows the user to see various information at a glance, which is very helpful to quickly determine whether the system metric is related to performance degradation or not. (S2) said it was fascinating because this type of analysis had never been done before. Overall, participants commented that the cause would have been difficult to find without background knowledge that tail latency could be further delayed due to the internal operation of the file system, but the visualized information provided by IoLens was very intuitive and easy to identify.

Usefulness of the visualization design In F2FS segment chart (Figure 1E), E1 and S1 mentioned that using different colors in each status makes it easier to identify changes in segments. They said it was interesting to be able to see I/O performance and F2FS status together. E2 and S2 said that the histogram on the right (Figure 1F') of the LBA heatmap chart (Figure 1F) is useful as it provides information about the rate of accessing the data and the rate of accessing metadata. All participants acknowledged that the approach of providing which functions are performing the most calculations on the CPU at a given time through mouse hovering (Figure 3) can provide meaningful information for analysis. E2 said that the fact that CPU utilization information is provided not only overall, but also by core, makes it easier to identify when only a specific CPU is heavily used and the rest are underutilized.

Post-hoc Feedback All participants said that for familiar I/O workloads that they often analyze, a performance chart or summary information provided in plain text from benchmarks are sufficient to predict and identify the cause of the performance degradation. However, when new workloads are encountered with unknown characteristics, it is difficult to pinpoint the cause of performance degradation without newly analyzing the overall data. They mentioned that IoLens, by presenting various charts linked with a brush on a single screen, makes it easier to understand the characteristics of unfamiliar workloads and identify the reasons for performance degradation at specific points. Three experts (E1, S1, S2) acknowledged that IoLens is useful when deciding which layer to optimize to improve I/O performance. In particular. E1 explained the following scenario: *"If a user brushes when throughput is saturated, CPU and memory utilization may be low and the CPU's disk wait time and queue counts may be high. Then, the user can confirm that the CPU and memory do not limit performance, but the performance of the storage device may limit throughput. As a result, users may consider replacing the storage with higher performance storage to improve I/O throughput."* The participants also provided suggestions for further enhancement of our tool. For example, E1 and S2 suggested adding more options to the radar chart to allow users to select options. S1 was interested in the correlation chart(Figure 1H) and suggested that it would be useful if more various metrics were added to the chart.

6 Conclusion

In this study, we propose a visual analytics tool that incorporates various layers in the kernel I/O stack. Unlike current practice, which visualizes only a single layer, IoLens can help identify various events that occur during the processing of I/O requests. Moreover, IoLens helps users understand the overall workload on the system, examine when the anomalies happen, and determine how they impact system performance. IoLens thus provides an extended perspective towards the I/O performance analysis process. Our usage scenario and user study validate the effectiveness of the system.

We plan to extend the system coverage of IoLens. We would also like to improve it to a scalable chart by adding some functions. First, we automate the visual analysis process from data collection and extend it to a dynamic chart that processes real-time updated data. Also, we add the ability for users to navigate specific time frames. IoLens can be a more sustainable and widely available tool.

References

[1] J. Axboe. Flexible io tester (fio). *https://github.com/axboe/fio. Accessed*, 13:12–19, 2019.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[3] A. D. Brunelle. Block i/o layer tracing: blktrace. *HP, Gelato-Cupertino, CA, USA*, 57, 2006.

[4] J. Courville and F. Chen. Understanding storage i/o behaviors of mobile applications. In *2016 32nd Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–11, 2016. doi: 10.1109/MSST.2016.7897092

[5] B. Hou, F. Chen, Z. Ou, R. Wang, and M. Mesnier. Understanding i/o performance behaviors of cloud storage from a client's perspective. *ACM Trans. Storage*, 13(2), may 2017. doi: 10.1145/3078838

[6] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam. Hybridstore: A cost-efficient, high-performance storage system combining ssds and hdds. In *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 227–236, 2011. doi: 10.1109/MASCOTS.2011.64

[7] C. Lee, D. Sim, J. Hwang, and S. Cho. F2FS: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pp. 273–286. USENIX Association, Santa Clara, CA, Feb. 2015.

[8] C. Mason. Seekwatcher. *URL http://oss. oracle. com/~ mason/seekwatcher*, 2008.

[9] C. Muelder, C. Sigovan, K.-L. Ma, J. Cope, S. Lang, K. Iskra, P. Beckman, and R. Ross. Visual analysis of i/o system behavior for high-end computing. In *Proceedings of the Third International Workshop on Large-Scale System and Application Performance*, LSAP '11, p. 19–26, 2011. doi: 10.1145/1996029.1996036

[10] C. Muelder, B. Zhu, W. Chen, H. Zhang, and K.-L. Ma. Visual analysis of cloud computing performance using behavioral lines. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1694–1704, 2016. doi: 10.1109/TVCG.2016.2534558

[11] H. Park, A. Cho, H. Jeon, H. Lee, Y. Yang, S. Lee, H. Lee, and J. Choo. Hpcclusterscape: Increasing transparency and efficiency of shared high-performance computing clusters for large-scale ai models. *arXiv preprint arXiv:2310.02120*, 2023.

[12] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.*, 10(1):26–52, feb 1992. doi: 10.1145/146941.146943

[13] Shilpika, T. Fujiwara, N. Sakamoto, J. Nonaka, and K.-L. Ma. A visual analytics approach for hardware system monitoring with streaming functional data analysis. *IEEE Transactions on Visualization and Computer Graphics*, 28(6):2338–2349, 2022. doi: 10.1109/TVCG.2022.3165348

[14] V. Tarasov, E. Zadok, and S. Shepler. Filebench: A flexible framework for file system benchmarking. *login Usenix Mag.*, 41(1), 2016.

[15] K. Vijayakumar, F. Mueller, X. Ma, and P. C. Roth. Scalable i/o tracing and analysis. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, p. 26–31, 2009.

[16] Z. Yang, M. Awasthi, M. Ghosh, J. Bhimani, and N. Mi. I/o workload management for all-flash datacenter storage systems based on total cost of ownership. *IEEE Transactions on Big Data*, 8(2):332–345, 2022. doi: 10.1109/TBDATA.2018.2871114