Valmar: High-Bandwidth Real-Time Streaming Data Management

David Bigelow
University of California, Santa Cruz
dbigelow@cs.ucsc.edu

Scott Brandt
University of California, Santa Cruz
scott@cs.ucsc.edu

John Bent[1]
EMC
John.Bent@emc.com

HB Chen
Los Alamos National Laboratory
hbchen@lanl.gov

***Abstract*—In applications ranging from radio telescopes to Internet traffic monitoring, our ability to generate data has outpaced our ability to effectively capture, mine, and manage it. These ultra-high-bandwidth data streams typically contain little useful information and most of the data can be safely discarded. Periodically, however, an event of interest is observed and a large segment of the data must be preserved, including data preceding detection of the event. Doing so requires guaranteed data capture at source rates, line speed filtering to detect events and data points of interest, and TiVo-like ability to save past data once an event has been detected. We present Valmar, a system for guaranteed capture, indexing, and storage of ultra-high-bandwidth data streams. Our results show that Valmar performs at nearly full disk bandwidth, up to several orders of magnitude faster than flat file and database systems, works well with both small and large data elements, and allows concurrent read and search access without compromising data capture guarantees.**

I. Introduction

In an information-driven world, the ability to capture and store data in real time is of the utmost importance. The scope and intent of such data capture, however, varies widely. Individuals record television programs for later viewing, governments maintain vast sensor networks to warn against calamity, scientists conduct experiments requiring immense data collection, and automated monitoring tools supervise a host of processes which human hands rarely touch. All such tasks have the same basic requirements – guaranteed capture of streaming real-time data – but with greatly differing parameters of size and scope. Our ability to process and interpret data has grown faster than our ability to store and manage it, which has led to the curious condition of being able to recognize the importance of data without being able to store it, and hence unable to later profit by it.

Data at large scales is very difficult to store indefinitely: the Large Hadron Collider (LHC) generates continuous data at a rate of around 300 MB/s [7], the Long Wavelength Array (LWA) is designed for an initial data rate of 3.75 GB/s [9], and Internet routers can handle traffic ranging from a few MB/s to as much as 5 GB/s in a single core router. At 100 PB/year for the LWA and potentially more for Internet traffic, the greater portion of this data must be *defined* as "unimportant" when judged against the cost of its long-term storage. Though continuously collected, data at this magnitude can only be stored for a short time before overwriting it with new, retaining permanently only that which is most interesting. No storage system has yet been designed to effectively manage this class of data: massive amounts continuously written but rarely read.

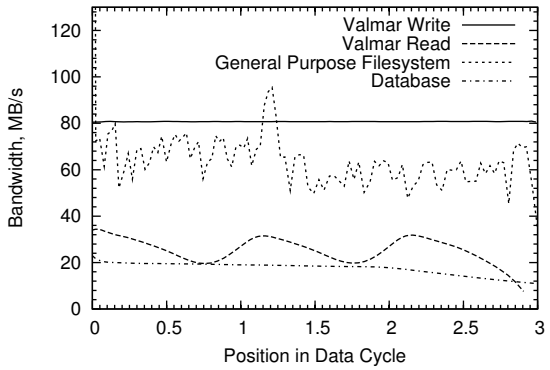

Fig. 1. Comparison of Valmar with a general-purpose filesystem and a database system. Only Valmar has bandwidth available for reads; the other two cannot maintain the requested write volume.

Addressing this problem requires us to combine certain aspects of a database with other aspects of a general-purpose filesystem, neither of which is well-suited to the problem on its own. Indexing many small elements (as with Internet traffic) and executing arbitrary queries upon them is best handled, in theory, by a database. In practice, no database is capable of keeping up with a never-ending stream of IP packets in the magnitude that we target, and nor can it execute timely queries while continuing to keep up with the collection of new real-time data. In contrast, general-purpose filesystems perform well when handling large amounts of data, but are less impressive when the individual data elements become small, and less impressive still if one attempts a generalized search over them. Neither system is capable of making performance guarantees and upholding quality of service deadlines on bulk-storage rotational disk drives operating at near-full capacity, as this problem requires.

We have developed methods to manage high-bandwidth real-time data while making performance guarantees about the rate of capture. Data is automatically indexed such that it can be quickly located via queries, and old data expires automatically to make room for new, unless specifically marked as "interesting." This system can operate at nearly the full

978-1-4673-1747-4/12/$31.00

[1]Formerly of Los Alamos National Laboratory, at the time this research was performed.

rate of the underlying disk, without the significant bandwidth loss from metadata management or cataloging present in many other systems. Parameters have been developed in consultation with astronomers and network experts at several institutions, including individuals associated with the LWA and LSST projects, and cybersecurity experts working with network intrusion detection systems.

Figure 1 shows a comparison of data recording ability between our prototype system (Valmar), a general-purpose filesystem using flat files, and a database approach. Valmar shows consistent performance, necessary when making performance guarantees, and a higher bandwidth availability than the other two systems.

II. Overview and System Design

"Write-once, read-maybe" data is not typical in storage systems. Data may sometimes be stored with no intent of reading it again, except where needed for failure recovery, as in checkpointing or backup systems. In contrast, we now focus on a problem area which requires the erasure of old data in order to store new, and will often read data back as part of its normal operation — but each individual piece of data will only "maybe" be read, and the majority will expire unseen.

Our methods are designed to work on individual commodity disk drives utilizing commodity hardware and infrastructure. This highly decentralized approach allows us to use a heterogeneous environment and to configure individual drive loads based on individual capabilities. Instances communicate with each other in order to provide reliability services, but we focus here on the individual instances and their mode of operation, rather than their inter-process activities.

A. Ring Buffer Model

The operational state of this type of storage system can be described thusly:

1) Current data is preserved for a limited time only. The extent of that time depends on system capacity and policy determination.
2) Current data can be specifically marked for preservation, with the understanding that such preservation will reduce the total available storage capacity.
3) Old data is automatically overwritten in a first-in, first-out manner as new data enters the system.
4) New data has priority at all times with respect to storage system capability allocation; it is being generated in real time and there are no second chances for collection.

This manner of data collection describes a system commonly known as a "ring buffer," though the standard model does not usually incorporate in-place preservation of existing data.

B. Disk Performance and Data Chunking

Rotational disk drives are often regarded as unreliable for real-time purposes due to their mechanical nature and the opacity of their internal data arrangement, yet they are among the most efficient devices available for bulk online data storage. Latency may differ by orders of magnitude based on the order in which one accesses data [8], and the available bandwidth changes significantly based on different arrangements of physical data location on the spinning platter. Such real-time systems that have made use of mechanical disks have usually assumed absolute worst-case conditions in order to guarantee their deadlines, and have thus been able to utilize only a fraction of the drive's peak capabilities. Although it is necessary to adhere to worst-case limitations to make a real-time guarantee, it is possible to establish tighter bounds on those worst-case figures and use a higher percentage of a drive's capabilities.

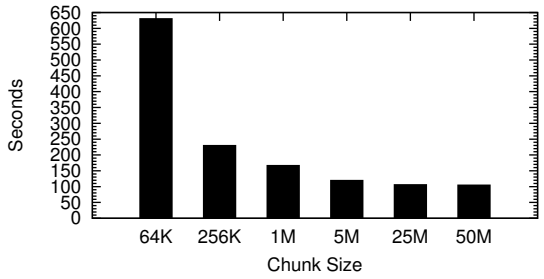

Fig. 2. Comparison of write times with varying chunk sizes, for 10 GB of data.

We now define two terms:

- A **Data Element** is a single piece of data that should be treated as an indivisible unit. For example, a data element might be a single snapshot from an optical telescope, or a single IP packet intercepted from network monitoring.
- A **Data Chunk** is a unit that the storage system interacts with. It may contain a single large data element or many smaller ones. It may also contain additional metadata-like information from outside the standard datastream.

In order to maximally exploit a drive's performance, two main factors must be taken into account: the bandwidth limitations of the spinning platter, and the actual physical layout of logically-consecutive data on the disk. The bandwidth of the spinning platter is a physical limitation and cannot be entirely mitigated, but awareness of that constraint allows one to understand the bandwidth curve and regulate demand accordingly [2]. However, the required time for a series of operations cannot be determined merely by studying the bandwidth curve; physically repositioning the disk hardware to read two non-consecutive regions may add a significant amount to the total time. This overhead is always present, but can be more or less relevant depending on the size of the surrounding operations. If I/O operations are small and disk repositioning frequent, observed bandwidth is significantly less than if I/O operations are very large, and disk repositioning infrequent.

The problem is most acute when the drive is nearly full and free space is only available in small non-contiguous regions. Since we expect each drive to have a constant utilization rate of near-maximum, data fragmentation would quickly become a major problem in any standard filesystem. In order to eliminate this fragmentation, we must carefully choose and restrict data layout, size, and placement. As a example of this principle,

Figure 2 shows actual results from a disk drive for a non-worst-case scenario. It measures the time required to write 10 GB of data to the drive with several different chunk sizes. In each case, the chunks are proportionally written throughout the entire space of the disk, from outermost to innermost track and many places in between. Even a 64-KB chunk size (the smallest shown on the graph) is considerably larger than the minimum for most filesystems.

It is obvious that we cannot achieve reasonable efficiency over a continuous stream of I/O operations unless we constrain the minimum I/O size to be on the order of several megabytes, and several tens of megabytes is better still. Therefore, we define a minimum chunk size and allow no writes or data placement smaller than this size. Read operations may be smaller where desired, since reads do not change the structure on disk. This minimum chunk size prevents small and inefficient I/O operations and has the simultaneous effect of drastically reducing data fragmentation in comparison to regular file systems. Chunk size is customizable based on the exact nature of the data, but a general rule of thumb is that "bigger is better" from the perspective of absolute performance on the disk, up to several tens of MB in size.

C. Indexing and Searching

The ability to store data in real time is wasted if one cannot later *find* that data. This problem is particularly acute in a low-lifetime system since the inability to locate the data quickly may translate to an inability to *ever* locate it, as it will soon no longer exist. When data is lost in this manner, it is as if it had never been captured at all. Therefore, not only must the data storage be subject to real-time deadlines and performance guarantees, but the categorization and indexing of data must also be done in real time, as fast as the raw data itself arrives.

Indexing may be managed at several different scales. If data elements are large and require little indexing data per element, there are few problems. If data elements are small and need to be categorized based on multiple aspects at once, the indexing information may amount to a sizable fraction of the full dataset. As the indexing becomes more complex and involves larger amounts of the original data, searching becomes problematic from any perspective: a full search approaches the equivalent of reading the entire data set. A small index may be kept in main memory alone and this will simplify all problems related to queries, but we here consider the case where index size requires that at least part of it be kept on disk.

Indexing information is best treated in the same manner as the data itself: gather indexing elements together into chunks which can be treated as a single I/O unit. Indexing chunks can then be stored throughout the drive alongside data chunks. If the size of the indexing chunks can be computed in advance (for example, if IP packets are a constant 1500 bytes at all times), then the size of the required indexing information can be pre-computed, and dedicated regions of the disk may be set aside for them. This could allow indexing chunks to be stored in the highest bandwidth region of the drive, for example, speeding up queries. However, when data element sizes are variable, indexing chunks should be treated as data chunks and placed on the disk drive by the same placement methods and ordering layout.

Indexing chunks "expire" when they no longer reference current data, and thus maintain the same lifecycle pattern as data chunks. If certain data chunks are preserved, partially-expired indexing chunks must also be preserved to properly reference the data, which can introduce a storage overhead as unnecessary portions of the index are needlessly preserved. An engineering refinement is to use spare bandwidth to periodically consolidate partially-expired indexing chunks so as to keep wasted storage space at a minimum.

Meanwhile, the search process has similarities to both standard file systems and databases, but with different techniques and stricter deadlines. The quick cycle of data precludes a gradual buildup of a full index, and the most likely outcome is that no search will ever be performed on a given portion of data as it quickly expires. When a search *is* performed, it is almost certainly a one-time occurrence to preserve a region of data, and will likely not be repeated. This means that search results should not be preemptive nor cached. Such behavior would not only confer no advantage, but would in fact hinder those searches which *are* required.

In a system where data lifetime is measured in hours, it is safe to assume that a search has only one purpose: to find data that is "interesting," and by implication, data that should be marked for preservation (or alternately, to find data that is no longer interesting for the purpose of "unpreserving" it). Therefore, we define a real-time search as follows:

1) A real-time search is conducted over all data present in the system at the time the query is initiated.
2) A real-time search is considered successful if it locates all specified data in the system, and that data is still present in the system by the time the results are returned.
3) A real-time search is considered at least partially unsuccessful (or, as having missed its deadline) when it is unable to consider data because of expiration, or if it locates data which is no longer present in the system by the time the results are returned.

We impose no absolute time requirements on such a search in the primary definition, though a faster search is naturally superior to a slower one.

D. Prototype System

Valmar is implemented as a multithreaded process that runs in userspace and accesses disk drives as raw devices. Multiple processes may run on the same node, as many independent processes as disks, and additional components may be configured to accept multiple data streams and configure them for RAID-like reliability purposes. All accesses to any given disk must take place through the associated process in order to properly manage bandwidth. Each process/disk is governed by configuration settings that specify, among other information, the chunk size, the element size (or size range), and the aspect(s) upon which each element is indexed.

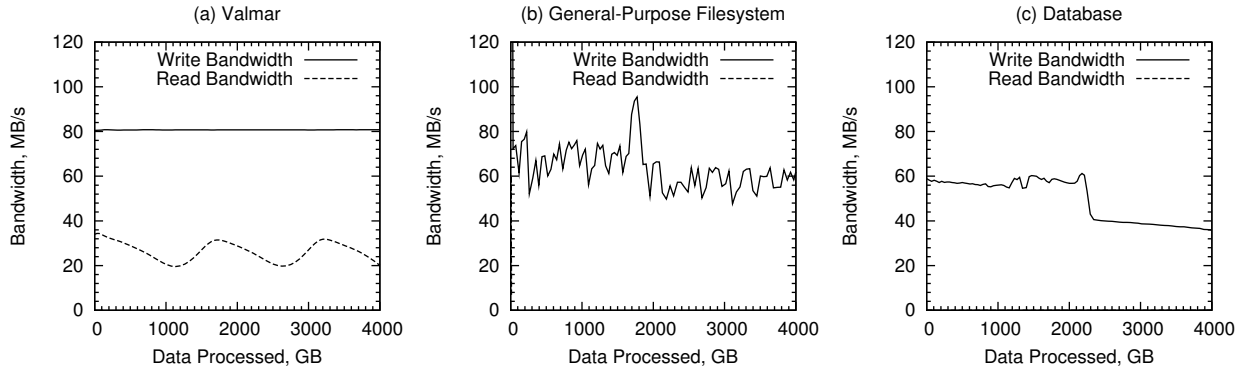

Fig. 3. Performance of Valmar, a general-purpose filesystem, and a database, storing large elements. Read bandwidth in graph (b) is so low as to be indistinguishable from zero, and graph (c) has a read performance of exactly zero.

III. Testing and Evaluation

Since no existing systems are designed to handle this type of workload, we constructed two comparison systems to measure against our own prototype, Valmar. The first comparison system is based around a general-purpose filesystem, which performs best with large data elements and minimal indexing. The second system is based around a database, which is appropriate for complex queries on small data elements, though it too can handle larger data elements requiring minimal indexing. Here, we evaluate the performance of Valmar and the two comparison systems.

A. Testing Procedure

Our testing took place on a variety of hard drives on different sets of hardware, and we obtained similar relative results for each setup. All experiments presented in this section are run on the same hardware configuration for accurate comparisons between all results. The disk shown in these results is a Hitachi Deskstar HD32000 IDK/7K 7200 RPM SATA drive of 2 decimal TB capacity. All results in this section should be interpreted as using the binary conventions of bytes.

We used ext2 as our comparison general-purpose file system, and again as the underlying file system for our mysql-based database comparison system. We chose ext2 because it has superior performance compared to several other file systems, including ext3 and XFS, most likely due to their journaling nature. Both comparison systems were constructed such that they prioritized data capture over data reading, rather than relying on an unwanted "fair" allocation of resources. Furthermore, rather than relying on system metadata to determine the order of data expiration, we included an external index structure in both comparison systems, similar to what Valmar uses. Without this external indexing system, the performance of both comparison systems drops precipitously when expiring data, since it takes a great deal of time to even *find* the data due to be expired, before actually overwriting it. We also adopted a policy of in-place element replacement in an attempt to prevent excessive fragmentation, greatly improving the performance of both comparison systems at the expense of lowering their total capacity.

Data cycles were constrained to about 1.5 TB, out of 1.8 available in order to avoid the innermost portion of the disk platter, which has considerably lower performance for the small amount of additional capacity that it offers. Valmar was physically restricted to the outermost portions of the platter, while the comparison systems were allowed to allocate their data structures over the entire drive, which gave them a greater flexibility and higher performance than they could reach with comparable restrictions. We set a desired write bandwidth of 80 MB/s over the entire course of the disk, which was near-maximum performance for the worst-performing regions at the innermost tracks. In Valmar, data chunk sizes were also set at 80 MB, a sufficiently high amount to gain advantage from the chunking method, and conveniently corresponding to exactly one second worth of data.

B. Large Elements

We first tested the system using large (5 MB) fixed-size elements as a baseline. These elements were indexed only according to time, with no complex data categorization scheme. The system was allowed to run for multiple data cycles, randomly preserving data chunks, reading them to return to an outside process, and unpreserving the chunks again. The results of these tests can be seen in Figure 3.

Part (a) shows performance achieved by Valmar. The write bandwidth stays steady at 80 MB/s bandwidth, and the read bandwidth follows a sinusoidal/sawtooth pattern in accordance with the disk head's movement from the outermost to the innermost regions of the disk platter. The system can continue this mode of operation indefinitely, precisely as specified in the problem description.

Part (b) shows performance achieved by the general-purpose filesystem. Although the raw disk ability can easily support 80 MB/s of write bandwidth, the filesystem is unable to keep up. Instead, it has an extremely jagged performance curve, usually spiking between 60 and 70 MB/s bandwidth, and once spiking above 80 MB/s in a particularly favorable region, but never stabilizing at the rate it needs. Furthermore, as the second cycle of data enters the system (thus requiring old data to be deleted/overwritten), average performance drops below the first data cycle. During the course of the test, the general-

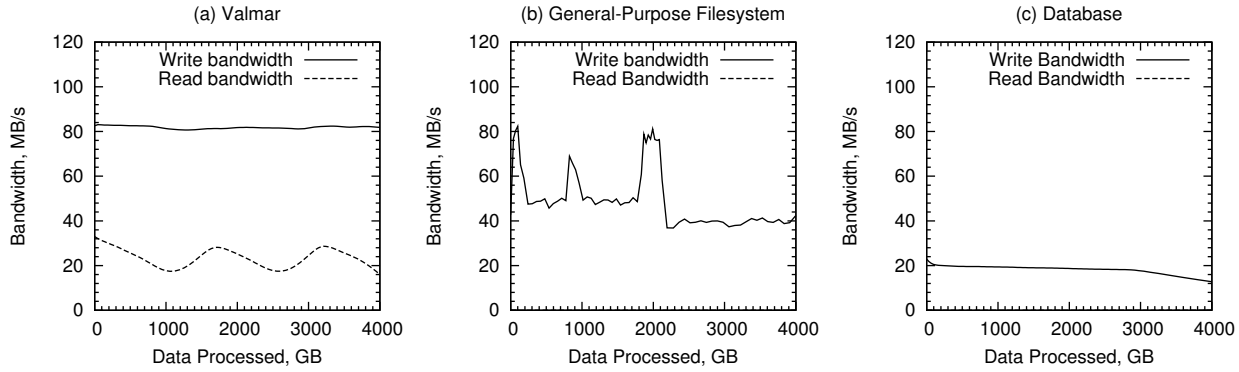

Fig. 4. Performance of Valmar, a general-purpose filesystem, and a database, storing small elements. Read bandwidths in graphs (b) and (c) are zero

purpose filesystem lost about 26.1% of the total data. Since writes were prioritized, reads were extremely rare, and read bandwidth remained zero for nearly the entire test (only 650 MB were read in total at various points during the course of the test).

Part (c) shows performance achieved by the database. Although databases are not normally used to store large amounts of relatively uncomplicated binary data, it is included here for comparison purposes. Its performance is comparable to that of a flat file system for just over 2 TB of processed data, and is much more stable. However, during the second data cycle, performance drops rapidly from about 60 MB/s to 40 MB/s, and continues to decrease from that point. As with the general-purpose filesystem, the database system does not achieve full data capture and thus has zero read bandwidth. During the course of the test, the database lost about 40.7% of all data.

C. Small Elements

After we established the baseline patterns with large data elements, we moved on to the harder-to-manage small elements with variable indexing requirements. IP packets are a practical real-world example, due to their diverse nature and highly-structured format. It would have been impractical to use an actual network trace due to the size and rate of the data (a single test requires multiple terabytes of data streaming into a single point source over the duration of the experiment), so we generated our own simulated IP traffic instead. We used 10,000 source addresses communicating with 1,000,000 destination addresses, at about 80 MB/s (the exact amount often being a little bit greater to account for variable packet sizes).

IP packets were primarily indexed according to timestamp, source address, destination address, and a metadata-like classification number based on the contents of the packet. Each packet varied between 20 and 1500 bytes in size, uniformly distributed in that range. Although real Internet traffic is weighted towards larger-size packets, smaller packets place a greater strain in the indexing system, which is part of what we wished to test. The results of IP-packet testing can be seen in Figure 4, again with Valmar and the two comparison systems.

Part (a) again shows performance by Valmar. Write bandwidth is slightly greater than 80 MB/s on average, due to the slight overhead of the indexing system, and the additional storage of indexing chunks. This adds a small jitter to the write bandwidth. However, this is an expected effect from this type of data and does not result in any lost packets. Read bandwidth again follows a sinusoidal/sawtooth pattern in progression with the disk head position on the disk platter. Overall, even with data elements with an average size of less than 0.02% the size of the previously-tested large elements, performance is steady. Again, the system can continue in this mode of operation indefinitely, losing no packets, precisely as required.

Part (b) shows performance by the general-purpose filesystem. This test was performed with a 128 KB element size (each element holding many packets), since general-purpose file systems generally do not react well when attempting to store hundreds of millions of 20-1500 byte files in a constantly rotating data cycle. Even with an element size over a hundred times larger than what Valmar was using, performance was extremely erratic. Write bandwidth quickly drops to around 50 MB/s during the first cycle of data, and thereafter drops to around 40 MB/s of write bandwidth after several dips and spikes. Again, read performance was zero due to the inability to keep up with the writing data cycle. During the course of the test, 51.3% of the data was lost.

Part (c) shows database performance. Although a database is well-suited to indexing and querying over a large number of small elements (or "rows" in database terminology), these results show that it is not well suited to record that data in real time. Performance is limited to no more than a 20 MB/s write speed, and it drops further as the system continues into further data cycles. There is no opportunity to read data due to write priorities, leading to a constant read bandwidth of zero. During the course of this test, 79.4% of the data was lost.

IV. Related Work

Ring buffers are not a new concept in this type of data management and are already used in some commercial production systems. DataTurbine [17] use this model, though it (and similar systems) are focused on high-bandwidth data and do not emphasize the storage system. Rajasekar et al. has proposed a virtual object ring buffer framework [14] designed to manage sensor data, but again does not focus on storage.

Network monitoring tools like Argus [1], NetFlow [5], and other commercial products are designed to capture network

traffic for status reporting, anomaly detection, and status reporting. None of these systems are designed to track and store past data, only focusing on the future when something is detected. "Time Machine" [10] *is* designed to consider storage, but only in the sense that it is aware of storage limitations and prioritizes accordingly.

Due to the difficulties in making quality of service guarantees on unreliable mechanical devices, existing work often guarantees only one aspect of performance, or makes only statistical guarantees. RT-Mach [11] make real-time guarantees through extreme worst-case usage pattern predictions while Lottery Scheduling [18] and others focus on statistical guarantees through isolating workloads.

Horizon [13] also uses a two-tier approach to manage QoS in distributed storage, including disk-level tools to move data on and off the hardware. DROPS [15] makes reservations based on throughput while Fahrrad [12] guarantees disk head *time* rather than bandwidth or latency. Large storage systems like Ceph [19] can often guarantee a certain service level, but only to the degree of categorizing traffic to maintain an appropriately "fair" level of service.

Some systems meet real-time I/O deadlines by requiring certain preset conditions and workloads. Clockwise [3] and other multimedia storage servers use a predictable file layout and workload specification to meet predefined real-time deadlines in these circumstances. Semantic File Systems [6] has some applicability in this problem area, where files can be classified and arranged on data content. This work was extended in Connections [16], and the Damasc [4] project goes further with configurable layers added on top of the file system.

V. Conclusion

Though current research has increasingly applied real-time deadlines and quality of service considerations to storage systems in recent years, it has not yet focused on the problem space of high-bandwidth, low-lifetime data. We have developed methods for managing this type of data, from the initial collection to its expiration, including its temporary storage on mechanical disk drives. Additionally, we have established methods by which this data can be indexed and queried in real time without interrupting or hindering the ongoing collection of new data. These techniques can be applied to both large and small data elements, can work with arbitrary indexing patterns, and can be refined for practical improvements on a given data pattern.

We have shown experimental results from our prototype system, Valmar, that demonstrate its ability to make real-time performance guarantees on mechanical disk drives, and practically index hundreds of millions of small elements at once, as in the case of IP packets. Neither general-purpose filesystems nor databases can match or even approach our performance. In the future, we will continue to extend this work to show its practicality when used over a large distributed system, maintaining the performance we have demonstrated here.

References

[1] "ARGUS FAQ," http://www.qosient.com/argus/faq.shtml, 2011. [Online]. Available: http://www.qosient.com/argus/faq.shtml

[2] D. Bigelow, S. Brandt, J. Bent, and H. Chen, "Mahanaxar: Quality of service guarantees in high-bandwidth, real-time streaming data storage," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, may 2010, pp. 1 –11.

[3] P. Bosch, S. Mullender, and P. Jansen, "Clockwise: a mixed-media file system," in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 2, July 1999, pp. 277 –281 vol.2.

[4] S. Brandt, C. Maltzahn, N. Polyzotis, and W.-C. Tan, "Fusing data management services with file systems," in *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, ser. PDSW '09. New York, NY, USA: ACM, 2009, pp. 42–46. [Online]. Available: http://doi.acm.org.oca.ucsc.edu/10.1145/1713072.1713085

[5] Cisco Systems, "Introduction to Cisco IOS NetFlow - A Technical Overview," Cisco Systems, Tech. Rep. C17-408326-01, October 2007.

[6] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole, Jr., "Semantic file systems," in *Proceedings of the thirteenth ACM symposium on operating systems principles*, ser. SOSP '91. New York, NY, USA: ACM, 1991, pp. 16–25. [Online]. Available: http://doi.acm.org.oca.ucsc.edu/10.1145/121132.121138

[7] L. C. Grid, "Gridbriefings: Grid computing in five minutes," August 2008.

[8] W. W. Hsu, A. J. Smith, and H. C. Young, "The automatic improvement of locality in storage systems," *ACM Trans. Comput. Syst.*, vol. 23, no. 4, pp. 424–473, 2005.

[9] "http://www.phys.unm.edu/~lwa/index.html."

[10] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer, "Building a time machine for efficient recording and retrieval of high-volume network traffic," in *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. Berkeley, CA, USA: USENIX Association, 2005, pp. 23–23.

[11] A. Molano, K. Juvva, and R. Rajkumar, "Real-time filesystems. Guaranteeing timing constraints for disk accesses in RT-Mach," in *The 18th IEEE Real-Time Systems Symposium*, December 1997, pp. 155–165.

[12] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. M. Wong, and C. Maltzahn, "Efficient guaranteed disk request scheduling with fahrrad," in *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. New York, NY, USA: ACM, 2008, pp. 13–25.

[13] A. Povzner, D. Sawyer, and S. Brandt, "Horizon: efficient deadline-driven disk i/o management for distributed storage systems," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 1–12. [Online]. Available: http://doi.acm.org.oca.ucsc.edu/10.1145/1851476.1851478

[14] A. Rajasekar, S. Lu, R. Moore, F. Vernon, J. Orcutt, and K. Lindquist, "Accessing sensor data using meta data: a virtual object ring buffer framework," in *DMSN '05: Proceedings of the 2nd international workshop on data management for sensor networks*. New York, NY, USA: ACM, 2005, pp. 35–42.

[15] L. Reuther and M. Pohlack, "Rotational-position-aware real-time disk scheduling using a dynamic active subset (DAS)," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, dec. 2003, pp. 374 – 385.

[16] C. A. N. Soules and G. R. Ganger, "Connections: using context to enhance file search," in *Proceedings of the twentieth ACM symposium on operating systems principles*, ser. SOSP '05. New York, NY, USA: ACM, 2005, pp. 119–132. [Online]. Available: http://doi.acm.org.oca.ucsc.edu/10.1145/1095810.1095822

[17] S. Tilak, P. Hubbard, M. Miller, and T. Fountain, "The Ring Buffer Network Bus (RBNB) DataTurbine Streaming Data Middleware for Environmental Observing Systems," in *e-Science*, Bangalore, India, 10/12/2007.

[18] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proceedings of the First Symposium on Operating Systems Design and Implementation*, November 1994.

[19] J. Wu and S. Brandt, "Providing quality of service support in object-based file system," in *24th IEEE Conference on Mass Storage Systems and Technologies*, September 2007, pp. 157–170.