ADA: An Application-Conscious Data Acquirer for Visual Molecular Dynamics

Abstract

Visual molecular dynamics (VMD) has been widely used by numerous molecular dynamics (MD) applications to animate and analyze the trajectory of an MD simulation. One challenge faced by domain scientists, however, is how to filter out inactive data (i.e., data irrelevant to the subject) from the enormous output of an MD simulation. To solve it, we propose ADA (application-conscious data acquirer), a light-weight file system middleware that can perform an application-conscious data pre-processing. It provides host CPUs with only data needed instead of an entire raw dataset. Next, we implement an ADA prototype, which is then integrated into three computing platforms: an SSD server, a nine-node OrangeFS storage cluster, and a fat-node server with 1 TB memory. Further, we evaluate ADA by running a computational biology application on the three platforms. Our experimental results show that compared to a traditional file system an ADA-assisted file system improves data processing turnaround time by up to 13.4x and reduces up to 2.5x memory usage for data rendering. Besides, ADA allows the 1TB memory server to render more than 2x VMD graphs while saving 3x energy consumption.

CCS Concepts • **Software and its engineering → File systems management**; **Secondary storage**.

Keywords File System Middleware, Data Layout, Memory Utilization, Energy Conservation, VMD

ACM Reference Format:
. 2021. ADA: An Application-Conscious Data Acquirer for Visual Molecular Dynamics. In ,. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 Introduction

Visual molecular dynamics (VMD) is a popular molecular graphics program designed for modeling, visualization, and analysis of biological systems such as proteins, nucleic acids, and lipid bilayer assemblies [11]. It has been widely adopted by numerous molecular dynamics (MD) applications as a graphical front end for rendering and animating molecule undergoing simulations on a remote computer. MD applications span a wide range of scientific domains, from VASP for chemical materials [14], XcrySDen for crystallines [13], to NAMD for computing and structural biology [17]. These applications normally generate a huge amount of simulation data for a visualization tool like VMD to visualize and analyze. For example, data collected at the European Molecular Biology Laboratory demand a storage capacity of 160 PB as of late 2018 [4]. In addition to holding raw data for CPUs to solve various computational problems, extra storage is needed to accommodate processed data and intermediate data from separate computational steps [3].

The huge amount of data generated by various MD applications plus their demands for a high-performance and energy-efficient storage system put a tremendous pressure on traditional rotation-based hard disk drives (HDDs). Since non-volatile memory (NVM) technologies (e.g., flash memory) can offer a much better I/O performance and energy-efficiency [7], NVM-based solid state drives (SSDs) started to replace HDDs in a wide spectrum of data-intensive applications [31]. Although SSDs noticeably improve the performance of these applications, they are still relatively expensive compared with HDDs. Besides, we observed that in many MD applications only a portion of a raw dataset are interested in or relevant to the study subject of a domain scientist. These data are called active data as they need to be frequently accessed, and then, analyzed by host CPUs. The rest part of the raw dataset are called inactive data, which are either seldom visited or simply abandoned. This observation plus the higher price of SSD motivate us to employ a cost-effective hybrid storage system with both HDDs and SSDs for MD applications. In such a hybrid storage system, relatively expensive SSDs are employed to store active data, whereas cheaper HDDs are used to hold inactive data. Inactive data is also called MISC data in this paper (see Section 2.1).

In addition to the need of data layout optimization, we found that many MD applications also require a graphical data pre-processing procedure so that active data can be efficiently retrieved before they are analyzed by host CPUs. For example, GROningen MAchine for Chemical Simulations (GROMACS) [25], an MD application designed for simulation of proteins, uses VMD to read the trajectories of atoms and then render them into a 3D animation. All raw data (i.e., trajectories of atoms and molecules) are compressed in order to

ICPP '21, ICPP 21,

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

save storage space. In this MD application, domain scientists are only interested in protein data (i.e., active data) as their goal is to investigate the behavior of proteins. However, the volume of protein data is less than 50% of that of the entire raw dataset as the rest part of the raw dataset is composed of liquid and ligands data (i.e., inactive data) [25]. And yet, the entire raw dataset needs to be decompressed in memory first, and then, the inactive data can be filtered out. The fact that a noticeable percentage of data in a raw dataset are inactive data can also be inferred from other applications [22] [20] [30] [15]. How to separate active data from noticeably large inactive data in the first place so that only the former can be provided to host CPUs for a further analysis becomes a challenge faced by the domain scientists. We found that this challenge exists in a wide range of MD applications such as VASP [14], XCrySDen [13], and LAMMPS [18] when they have visualization needs. The existing approach to extracting active data from a raw dataset degrades the performance of a compute node as it causes three issues: (1) each time a raw dataset is acquired a sequence of data pre-processing steps need to be performed in order to generate active data, which is a time-consuming repeated effort; (2) a large memory space is needed to store the original compressed raw data, the decompressed raw data, and the active data, which wastes the memory of a compute node because inactive data should not be fetched into memory in the first place; (3) precious CPU time of a compute node is wasted as it is used to perform a very simple data pre-processing procedure instead of sophisticated operations such as convolutions or high-dimensional matrix multiplications (see Fig. 3a).

To solve this common challenge of MD applications, in this paper we propose a light-weight file system middleware called ADA (application-conscious data acquirer) dedicated to VMD. ADA sits between VMD and an existing file system to conduct an application-conscious data pre-processing on a storage node. In particular, based on the I/O access pattern of an MD application, ADA is able to categorize its raw data into multiple groups with each having a distinct tag. Thus, ADA can provide host CPUs with only data needed instead of an entire raw dataset. As a result, only active data will be transferred from a storage node to a compute node so that all three issues mentioned above can be avoided (see Fig. 3). Further, we implement an ADA prototype, which is then integrated into three computing platforms: an SSD server, a nine-node OrangeFS storage cluster, and a fat-node server with 1 TB memory. Finally, we use a biology MD application called GPCR (G-Protein Coupled Receptor) [10] running on the three platforms to evaluate the efficacy of ADA. Our experimental results show that compared to a traditional file system an ADA-assisted file system improves data processing turnaround time (see Section 2.1) by up to 13.4x and reduces up to 2.5x memory usage for data rendering on the SSD server. Besides, ADA allows the fat-node server with 1 TB memory to render more than 2x VMD graphs while saving more than 3x energy consumption. Although ADA is built for VMD, its framework can be extended to support other computational science applications where a similar data pre-processing challenge exists. As long as an application can provide the structure of its raw data in a file format, ADA can acquire an understanding of this structure through analyzing the structure file. Essentially, ADA is an application-conscious data pre-processing middleware that can be integrated into an existing file system.

Table 1. Data Components of Three .xtc Files

Number of frames	Compressed file size (MB)		Protein data fraction (%)
	Complete data	Protein data	
626	100	44	44
1,251	200	98	49
5,006	800	348	43.5

Main contributions of this paper include (1) a light-weight file system middleware called ADA is proposed to largely improve the performance, memory utilization, and energy-efficiency of MD applications; (2) an ADA prototype is implemented, and then, integrated into three real-world computing platforms; (3) a comprehensive experimental study is provided to fully evaluate the efficacy of the ADA prototype.

The rest of the paper is organized as follows. Section 2 provides the background and motivation of this research. The design and implementation details of ADA are presented in Section 3, which is followed by an evaluation of ADA shown in Section 4. Section 5 summarizes the related work. Finally, Section 6 concludes this paper by pointing out a future direction of this research.

2 Background

In this section, we first briefly introduce VMD processes in visualizing data generated from a biology MD application called GPCR (G-Protein Coupled Receptor) [10]. Next, we explain how an ADA-assisted approach performs VMD data pre-processing differently from a traditional way.

2.1 VMD in the GPCR MD Application

VMD can act as a graphical front end for an external MD application by displaying and animating a molecule undergoing simulation on a remote computer. The challenge of operating VMD visualization on an ordinary cluster with limited memory capacity is that it needs to handle a large amount of data. Recently retrieved frames should be evacuated from the limited memory to make room for subsequent phases of frames. Frequent data swapping operations cause a low data hit rate under random frames accesses (e.g., replaying the frames back and forth), which further leads to a non-fluent VMD animation playback.

The goal of the GPCR MD application is to facilitate high-resolution structure-function studies on medically important proteins known as G-protein coupled receptors (GPCRs) by

making all data publicly available. In 2016, Hua *et al.* determined and analyzed the high-resolution atomic structure of human cannabinoid receptor 1 (CB1), which is also known as the marijuana receptor. These new findings provide insightful clues to understand why some drugs that interact with this receptor have had unexpectedly complex and sometimes harmful effects, while the utility of the crystal structure may provide inspiration for drug design toward refining efficacy and avoiding adverse effects [10].

Two major types of files employed by VMD in the GPCR MD application are .xtc (i.e., XTrkCAD) files and .pdb (i.e., protein data bank) file. While a .xtc file contains compressed trajectories of atoms and molecules, a .pdb file includes the structure of a protein. The data components of three sample trajectory files used in the GPCR MD application [10] are summarized in Table 1. From this table, one can see that the percentage of protein data in these three trajectory files varies from 43.5% to 49%. The implication is that more than half of the data (i.e., MISC data) stored in a .xtc file need to be first decompressed and then abandoned. Note that one .pdb file may contain the structure of one protein or structures of multiple proteins. One .xtc file is guided by a corresponding .pdb file. Besides, one .pdb file can guide multiple .xtc files, which represent different atom motion phases. Fig. 1 shows a frame of 3D graph of the entire raw data, protein data, and MISC data of eight .xtc files, respectively. Fig. 1b is modified (or "cleaned") from Fig. 1a so that only protein data are displayed. Fig. 1c shows the liquid that surrounds the protein.

Now we explain how VMD processes these two types of files to generate 3D animations for proteins. Fig. 2 illustrates a data processing procedure embedded in the source code of VMD. The procedure consists of two phases: data pre-processing and data rendering (i.e., data replaying). In phase one, VMD first checks a protein data bank file to retrieve the protein structure. Guided by the protein structure, VMD is then able to retrieve compressed trajectories from a corresponding .xtc file. Once the .xtc file is loaded into memory, VMD requires an additional memory space to accommodate uncompressed trajectories, which are interpreted as an array of frames. Finally, VMD rebuilds the 3D animation replay based on these frames. The total time taken between the retrieval of files (i.e., .xtc and .pdb) from storage and the completion of 3D graphics rendering is defined as data processing turnaround time in this paper. It is the sum of data pre-processing time and data rendering time (see Fig. 2). This time-consuming decompression and MISC data filtering procedure becomes a constant burden when biologists repeatedly study the behaviors of proteins.

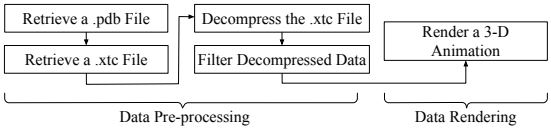
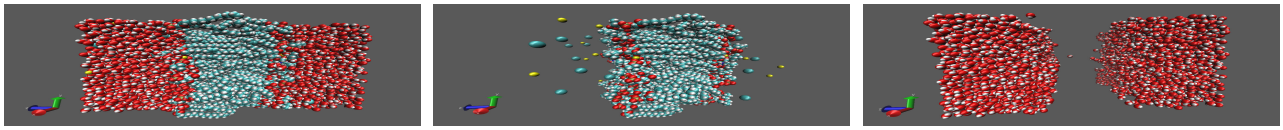


Figure 2. VMD data processing procedure.

2.2 VMD Data Processing Workflow

In this section, we first explain how a traditional VMD data processing procedure works on a cluster with both storage nodes and compute nodes. Next, we present our ADA-assisted VMD data processing workflow.

Fig. 3a shows a traditional VMD data processing workflow. A parallel file system like PVFS [8] is running on top of each storage node. While some storage nodes employ HDDs, others utilize SSDs. The entire compressed raw data are transferred into a group of compute nodes through a high-performance network architecture like InfiniBand. After receiving the raw data, the compute nodes first need to perform a data pre-processing including decompressing raw data and scanning for active data. Finally, all active data are fed into VMD to generate 3D animations. Note that the raw data are compressed and then transferred to compute nodes. Thus, raw data transferring is not a performance bottleneck. As we mentioned before, data pre-processing in compute nodes becomes a huge burden as it wastes precious CPU time and memory space of a compute node. Unlike a traditional VMD data processing scheme, an ADA-assisted VMD exploits the computing resources of storage nodes to perform data pre-processing. Fig. 3b illustrates how it works. First, storage nodes are logically divided into two groups: one PVFS file system manages all storage nodes with HDDs and another PVFS file system operates on all storage nodes with SSDs. A prototype of ADA is implemented as a middleware, which directly communicates with the two PVFS file systems. Once a raw dataset is passed to ADA, it starts to perform a data pre-processing procedure, after which only decompressed active data will be transferred to compute nodes. Consequently, compute nodes can concentrate on the



(a) Original raw data **(b)** Protein dataset **(c)** MISC dataset

Figure 1. VMD molecular structure rendered in one 3D graph frame.

most important task, which is rendering active data into a 3D animation.

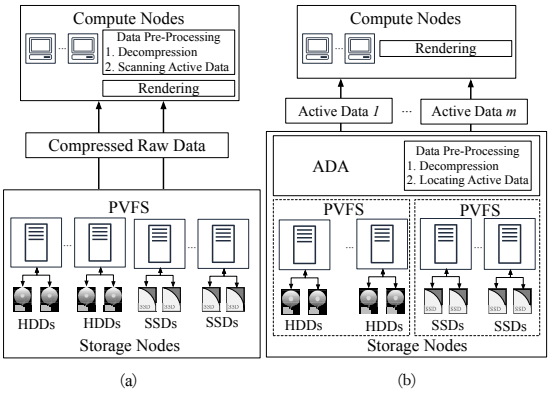


Figure 3. (a) Traditional VMD; (b) ADA-assisted VMD.

3 Design and Implementation of ADA

In this section, we present the design and implementation details of ADA. We first offer an overview of the ADA architecture, which is followed by an introduction to its two major components. Finally, we show how we implement it using VMD as a target application.

3.1 Architecture of ADA

Fig. 4 shows the architecture of ADA. A user API layer including either high-level I/O libraries or POSIX API is running on top of ADA. By employing a complex POSIX API, more architectures and richer functionality can be supported. Below the user API layer is ADA, which consists of two major components: a data pre-processor and an I/O determinator.

The data pre-processor categories a raw dataset, divides it into multiple groups, labels each group, and assigns a target file system to each group based on its label. The data decompressor will be invoked if the original data is compressed to save space. The I/O request pattern of an application guides the division of a raw dataset. For example, a scientific raw dataset representing different levels of precision will be divided into a few groups: one group representing short precision, one group standing for a moderate degree of precision, and several other groups for a high degree of precision. Since requirements of data accuracy vary, the number of groups of a raw dataset changes accordingly. The I/O determinator serves as the primary storage interface, which redirects all I/O calls to the underlying parallel file systems (e.g., ext4 or PVFS) and retrieves data from the target file systems upon requests. At the bottom of Fig. 4 are file systems, which serve as interfaces for storing selected groups of data. This layer can be considered as a physical disk interface layer. Take PVFS for example, it replaces the API on individual storage devices with an interface friendlier to the containers of objects used in the higher level of layers.

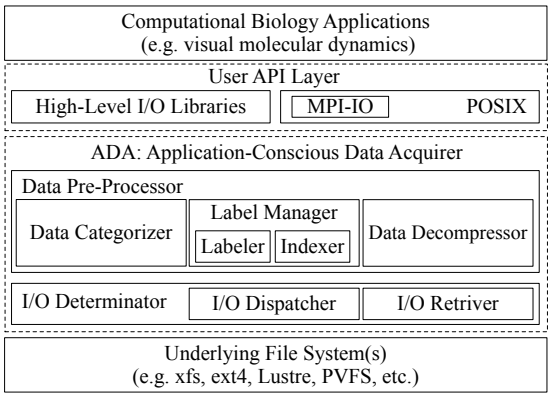


Figure 4. ADA architecture.

3.2 Data Pre-Processor

The data pre-processor has three components: a data categorizer, a label manager, and a data decompressor. The main purpose of the data pre-processor is to perform data pre-processing operations in storage nodes rather than compute nodes so that the latter can concentrate on data rendering. Essentially, these data pre-processing operations are intended to offer functionalities such as data rearrangement and data filtering such that only active data will be finally transferred to compute nodes. As a result, data replaying can be quickly executed on compute nodes, which is the most important task for biologists. Fig. 5 illustrates the data workflow of ADA. Once a dataset of an application arrives, the data decompressor first decompresses it into raw data and then the data categorizer divides the raw data into a group of data subsets based on the access behavior of the application. The labeler then assigns a tag to each data subset and stores its path on the underlying file system for later use. Note that the labeler manages tag information separately from data subsets. In other words, no additional information is injected to any of data subsets so that tags will not affect the representation of any data subset. When users send data queries for certain groups of datasets, the indexer uses tags from the queries to look for paths of datasets on the underlying file systems and passes them to the I/O retriever. The I/O retriever then raises I/O requests to the underlying file systems and then obtains the requested data.

The idea of data pre-processing is not new. However, offloading data re-organization operations from compute nodes to storage nodes has been proved to be effective in terms of reducing the total time of data manipulation [16]. This is because fewer participants are involved in the communication patterns. In Section 3.4, we explain how the data pre-processor is applied to VMD I/O optimization in details.

3.3 I/O Determinator

The core idea of the I/O determinator is to provide a way to judiciously manage the I/O load of an application in storage

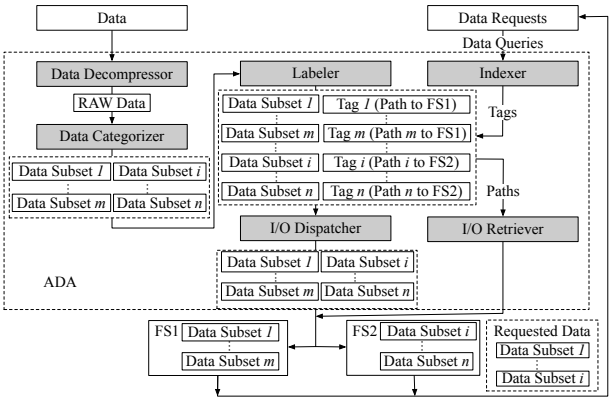


Figure 5. ADA data workflow

nodes. Coupled with the tags and target storage path passed from the data pre-processor, the I/O dispatcher sends each data subset to an underlying file system. The I/O retriever obtains the requested datasets by triggering file read via the datasets paths that are passed by the indexer (see Fig. 5). The I/O dispatcher is developed based on PLFS [2], a parallel log-structure file system. Since PLFS is transparent to an underlying parallel file system, a file system processes an assigned data subset as independent files without noticing that the contents of the files have been altered from the original data subset. Since PLFS supports multiple backends, the I/O dispatcher modifies this feature to distribute sub datasets with diverse target storage information to their right destinations. It achieves this goal by redirecting sub datasets to their intended backends. The underlying parallel file system performs actual data writing/reading operations. Fig. 6 presents an example of I/O dispatching. An application creates a file called *bar*, causing PLFS to create two container structures on the underlying file systems. Each of the container consists of a top-level directory called *bar/mnt** and several sub-directories to store data subsets, where *mnt** indicates mount points of underlying file systems (*mnt1* and *mnt2* in the example). The destination of each data subset is determined by the labler.

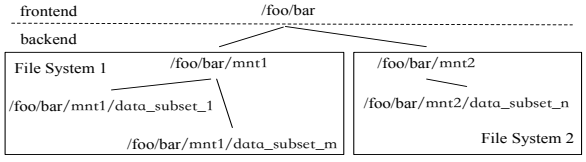


Figure 6. An I/O dispatching example.

3.4 Implementation of ADA

We implement a prototype of ADA, which employs VMD as a target application. First, ADA checks if a file to be written is generated by a target application (i.e., VMD in our case). If so, the file will be pre-processed by ADA. Otherwise, it will not be trapped by ADA. The data pre-processor then analyzes the atom information from a .pdb file. Next, it categorizes the molecules and then stores them by classes. The pseudo-code of the data pre-processor is shown in Algo. 1. After a pdb

Algorithm 1 ADA data pre-processor module

Require: File Pathname (.pdb file)
Ensure: Data Subset Ranges, Tags of each Subset

```
pdb_file ← open(Pathname)
/*initial standard C++ map containers to store label elements*/
labeler ← map()
offset, begin, end ← 0, 0, 0
prev_tag ← None
for atom in pdb_file do
    /*Categorizer Module*/
    tag ← GetType(atom)
    /*GetType reads atom type info from the pdb_file*/
    if prev_tag is None then
        prev_tag ← tag
    end if
    if tag is prev_tag then
        end ← end + 1
    else
        if tag not in labeler then
            labeler[tag] ← list([begin, end])
        else
            labeler[tag].extend([begin, end])
        end if
        /*Labeler Module*/
        prev_tag ← tag
        begin, end ← offset, offset + 1
    end if
    offset ← offset + 1
end for
/*Store the labeler to a file named label_file for later I/O reference */
label_file ← write(labeler)
```

analysis file is generated, the data pre-processor then divides a .xtc file according to the subset tags. If a biologist focuses on examining the behavior of a protein via VMD, then ADA will process data as follows: when the .pdb and .xtc files are sent to ADA for permanent storage, with the help of the .pdb file ADA data pre-processor categorizes the raw dataset into two groups: a protein dataset and a MISC dataset. The .xtc file is then decompressed and divided into protein trajectories and MISC trajectories with "**p**" and "**m**" label, respectively. ADA data pre-processor selects an SSD-based storage system for the "**p**" dataset and an HDD-based storage system for the "**m**" dataset. ADA I/O dispatcher then finalizes data dispatching operation to the underlying storage systems.

We modify and use the VMD command-line option for molecule data loading. $ `mol new foo.pdb` will start VMD and load a molecule from the file *foo.pdb* prior to $ `mol addfile /mnt/bar.xtc`, which loads the file *bar.xtc* from the frontend directory */mnt/*. We modify the `addfile` function to pass tag parameters to ADA. Hence $ `mol addfile /mnt/bar.xtc tag p` can only load

the subset labeled with "*p*". ADA uses the tag to determine the corresponding dropping data on the backend and calls POSIX `read` to retrieve the data.

With the help of ADA, the original dataset (see Fig. 1a) is divided into a protein subset (see Fig. 1b) and a MISC subset (see Fig. 1c), each of which is handled independently by one of the two separate parallel file systems.

4 Evaluation of ADA

In this section, we first evaluate ADA on an SSD server. Next, we test it on a small cluster with both SSDs and HDDs. Lastly, we measure its performance on a fat-node server with 1 TB memory. Major metrics employed include raw data retrieval time (i.e., time spent on retrieving .xtc and .pdb files from storage to memory), data processing turnaround time (see Section 2.1), and memory usage. Energy consumption is only measured in the experiments conducted on the fat-node server (see Section 4.3). Since it is almost impossible for us to augment the memory space of a compute node of the cluster to 1 TB, we use the fat-node server to "virtually represent" a compute node with a 1 TB memory space. The fat-node server testing environment can be viewed as a workaround for a cluster whose compute node has a large memory space. For simplicity, when we compare the performance of an ADA-assisted file system with that of an existing file system (e.g., ext4 or PVFS) we simply call it ADA hereafter.

4.1 Evaluation on an SSD Server

In this section, we evaluate ADA on an SSD server that equips with an Intel®Xeon®CPU E5-2603 v4 @1.70GHz, 16GB DRAM, and two 256GB NVMe SSDs. The CentOS release 6.10 (Final) and ext4 are employed as the operating system and the file system, respectively.

Since ADA divides and then dispatches an uncompressed dataset to two separate locations, it only needs to provide the protein trajectories subset for VMD visualization. The real file size loaded by ADA is smaller than that of ext4 even though the number of frames keeps the same. We summarize the differences in loaded data size between ext4 and ADA in Table 2. Note that ADA only transfers the de-compressed protein data subset that VMD can use while ext4 transfers the entire compressed dataset.

Raw Data Retrieval Time: Fig. 7a presents comparisons in time spent on retrieving (i.e., reading) raw data (i.e., .xtc and .pdb files) between an existing file system (i.e., ext4) and ADA. Notations used in Fig. 7 are summarized in Table 3. For example, *D-ADA (protein)* stands for the scenario that ADA transfers a decompressed protein data subset. Fig. 7a shows that *D-ADA (all)* delivers a performance similar to that of *D-ext4*. This is mainly because both transfer the same amount of data. Still, *D-ADA (all)* requires a slightly longer data transfer time compared with *D-ext4* because ADA needs to launch Indexer to search tags. *C-ext4* consistently performs the best

Table 2. Data Size Comparisons (ext4 vs. ADA)

Number of Frames	Loaded Data (MB)		Raw Data (MB)
	ext4 (Compressed)	ADA (De-compressed, protein)	
626	100	139	327
1,251	200	277	653
1,877	300	416	980
2,503	400	555	1,306
3,129	500	693	1,632
3,754	600	832	1,959
4,380	700	970	2,285
5,006	800	1,108	2,612

because it only needs to transfer the compressed data whose size is 1/3 of that of the raw data. On the other hand, *D-ADA (protein)* has to transfer 40% of the decompressed raw data.

Data Processing Turnaround Time: In terms of data processing turnaround time *D-ADA(all)* performs the same as *D-ext4* (see Fig. 7b). *D-ADA(protein)* performs even better than *D-ext4* (see Fig. 7b). This is because *D-ADA(protein)* transfers less data. As the number of frames increases, *D-ADA(protein)* delivers a much better performance than that of *C-ext4* (e.g., up to 13.4x). The reason is that the data decompression time becomes significantly longer than the data transfer time as the number of frames increases. Results shown in Fig. 7a and Fig. 7b imply that a time-consuming data decompression process may nullify the benefits of a shorter data transfer time gained by using a faster storage device like an SSD. In other words, the performance bottleneck of VMD data processing lies in the repetitive data pre-processing (e.g., data decompression) rather than a low data transfer rate. The results shown in this section demonstrate that simply replacing slower HDDs with faster SSDs cannot solve the problem of inefficient data pre-processing.

Table 3. Notations of Fig. 7

Notes	Description
C	VMD loads a compressed XTC file
D	VMD loads a raw XTC file w/o compression
ADA (all)	ADA transfers the entire raw data
ADA (protein)	ADA transfers the protein data

Memory Usage: Comparisons in memory usage between ext4 and ADA are presented in Fig. 7c. One can observe that the memory usage of ext4 is over 2.5x of that of ADA when the number of frames reaches 5,006. One can safely presume that within the same memory capacity ADA can load 2x frames compared with ext4. The trend in Fig. 7c indicates that an existing file system reaches the memory limit much earlier than ADA does. One can further presume that as the swapping mechanism is involved to extend the memory usage the loading time along with the latency of ext4 will be even more significant than that of ADA. The reason is that ADA only loads a small portion of each frame (i.e., protein

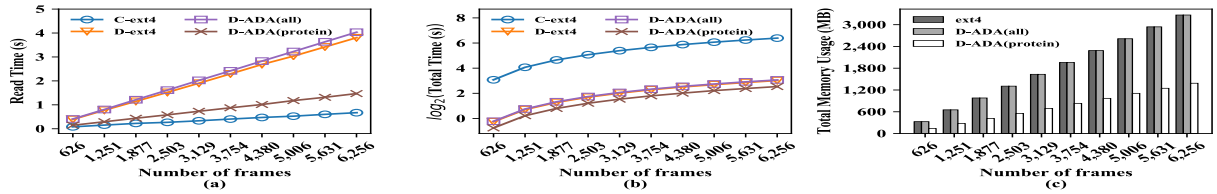


Figure 7. (a) Raw data retrieval time; (b) data processing turnaround time; (c) memory usage.

data). Besides, it has no need to allocate a large memory space to accommodate the compressed data like ext4 does.

CPU Workload: By comparing Fig. 7a with Fig. 7b, one can see that the data decompression time dominates the data pre-processing time in ext4. We use a profiler to examine the burst of CPU and then visualize the results via Flame Graph [1] in Fig. 8. One can observe from Fig. 8 that the data decompression weights more than 50% of the CPU burst time for VMD to build 3D graphics in ext4. Besides, CPU burst is involved whenever a .xtc file is referenced. The conclusion is that high-end CPUs spend more than half of its computing power on duplication of labor.

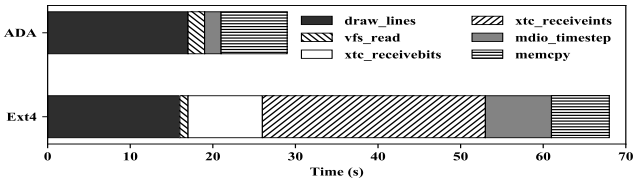


Figure 8. A comparison in CPU burst time.

An extra advantage of ADA is that it is able to assist VMD to view a dataset in a more fine-grained fashion. For example, a user can type in the command of $ `mol addfile /mnt/bar.xtc tag p` to open a .xtc file so that only the protein data in it are fetched. Thus, ADA can help an application better utilize the I/O bandwidth and memory space of a computing platform.

4.2 Evaluation on a Small Cluster

The ADA prototype is implemented on a nine-node cluster, where three nodes serve as the compute nodes while the other six nodes serve as storage nodes. Each of the compute nodes is equipped with a six-core Intel®Xeon®CPU E5-2603 v4 @1.70GHz CPU. Instead of managing the storage nodes with a single file system, we implement the ADA prototype on two independent PVFS (OrangFS) file systems with the help of PLFS. While one file system manages three HDD storage nodes with each consisting of two WD 1TB HDD drives, another file system manages three SSD storage nodes with each consisting of two Plextor 256GB SSD drives. Table 4 provides system parameters of the cluster. Both ADA and PVFS employ a hybrid storage system with three SSD storage nodes and three HDD storage nodes. Experimental results from PVFS are taken as the control groups.

Raw Data Retrieval Time: Fig. 9a presents comparisons in raw data retrieval time between PVFS and ADA. In this case, ADA only uses the underlying SSD storage nodes to transfer data, which is only 40% of a raw dataset. One can observe from Fig. 9a that as the number of frames increases the retrieval time (i.e., read time) of two ADA scenarios (i.e., *D-ADA (all)* and *D-ADA (protein)*) stay in between the best and the worst cases. ADA performs more than 2x better than PVFS (i.e., *D-ADA (all)* vs. *D-PVFS*) due to the better SSD read performance. *D-ADA (protein)* performs similarly to *C-PVFS* for the same reason discussed in Section 4.1.

Data Processing Turnaround Time: One can observe from Fig. 9b that the two scenarios of PVFS take a much longer data processing turnaround time as the number of frames increases. For example, when the number of frames is 6,256 the data processing turnaround time of D-PVFS is 9x of that of D-ADA(protein). Based on the trend shown in Fig. 9b, one can safely predict that the data processing turnaround time gap between compressed data (i.e., C-PVFS) and decompressed data (i.e., all rest three scenarios) will become wider as the number of frame numbers grows.

Memory Usage: Comparisons between PVFS and ADA in memory usage are presented in Fig. 9c. The overall trend shown in Fig. 9c is same as that of Fig. 7c. The reason is that in both scenarios the same groups of data are retrieved from storage to memory. The results shown in Fig. 7c and Fig. 9c confirm that D-ADA(protein) can noticeably save memory space for both a single server and a cluster.

4.3 Evaluation on a Fat-Node Server

We frequently heard an argument from domain scientists that the problem that we are addressing could be solved by employing a computing platform with larger memories. To verify whether this argument is true, and more importantly, whether ADA can effectively delay the time point when VMD runs out of memory, we evaluate an existing file system called XFS and ADA on a fat-node server with 1TB memory. Table 5 summaries the specifications of the fat-node server. Note that the server equips a RAID-50 HDD array, which guarantees a competitive I/O performance. Table 6 provides the details of data used in the experiments. In addition to the

Table 4. System Parameters

File Systems Spec.		Disk Systems Spec.		
CPU	Intel®Xeon®CPU E5-2603 v4 @1.70GHz	Disk Type	HDD	SSD
Operating System	CentOS 6.10 w/ 2.6.32-754 kernel	Brand	Western Digital	Plextor
File System	PVFS (OrangeFS 2.8.5)	Capacity	1TB	256GB
Node Quantity	9	Device Quantity	6	6
Node Arrangement	compute node *3 HDD node *3, SSD node *3	Data Transfer Rate	126MB/s (MAX)	Read: 3000MB/s (PEAK) Write: 1000MB/s (PEAK)
Average Power per Node	400W	Port Type	SATA	PCI-e

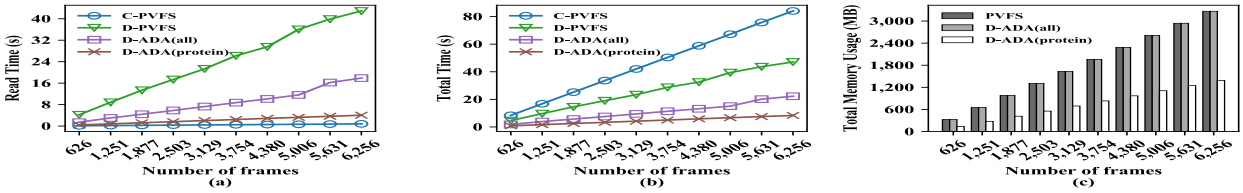


Figure 9. (a) Raw data retrieval time; (b) data processing turnaround time; (c) memory usage.

Table 5. Parameters of Fat-Node Server

CPU	Intel(R) Xeon E7-4820v3 @1.90GHz 40 cores (4 sockets), 2 threads per core
Main Memory	DDR-4 1,007GB
Operating System	CentOS 7.3 w/ 3.10 kernel
File System	XFS
Disk Array	WD HDD 1TB *10, RAID 50

three major metrics, energy consumption is also measured in this group of experiments.

Table 6. Data Size Comparisons (XFS vs. ADA)

Number of Frames	Loaded Data (GB)		Raw Data (GB)
	XFS (Compressed)	ADA (De-compressed, protein)	
62,560	10	13.9	32.7
187,680	30	41.6	98.0
312,800	50	69.3	163.3
437,920	70	97.0	228.6
625,600	100	138.6	326.6
938,400	150	207.9	489.9
1,251,200	200	277.2	653.2
1,564,000	250	346.5	816.5
1,876,800	300	415.8	979.8
2,502,400	400	554.4	1,306.4
3,440,800	550	762.3	1,796.3
4,379,200	700	970.2	2,286.2
5,004,800	800	1,108.8	2,612.8

Raw Data Retrieval Time and Data Processing Turnaround Time: Fig. 10a and Fig. 10b present a similar trend as that shown in Fig. 9a and Fig. 9b. The time difference between Fig. 10a and Fig. 10b is the time for data decompression and locating active data. One can see that as file size keeps growing the raw data retrieval time becomes increasingly insignificant in data processing turnaround time. The implication is that faster storage devices can provide a very limited contribution towards shrinking the data processing turnaround time. For example, it takes VMD around 400 minutes to retrieve and render 1,564,000 frames on the XFS system while the raw data retrieval time only weights less than 10% of the data processing turnaround time. How to quickly obtain the active data becomes a dominant factor in terms of improving data analysis efficiency in VMD. Besides, both XFS and ADA (all) are killed by the system due to memory shortage (see Fig. 10a and Fig. 10b) when VMD is trying to render 1,876,800 frames, which require 300 GB memory for compressed data and 979.8 GB memory for raw data. However, since ADA (protein) only needs to retrieve the protein data instead of the entire raw dataset from storage to memory, it will be aborted due to run-of-memory when VMD tries to render 5,004,800 frames, which contain more than 2x protein graphs.

Memory Usage: Fig. 10c presents comparisons of memory usage between XFS and ADA. One can see that the memory usage trend shown in Fig. 10c confirms our analysis in last paragraph that ADA can load more than 2x frames with the same memory capacity. Note that when the number of frames exceeds 1,876,800, only ADA (protein) still survives as both XFS and ADA(all) have been killed due to memory shortage (see Fig. 10c).

Energy Consumption: We use a power distribution unit that equips a Modbus-enabled power monitor to collect real-world server energy consumption for all experiments in this section. Fig. 10d presents the fat-node server's energy consumption within the data processing turnaround time window for each VMD process. Note that we physically isolate

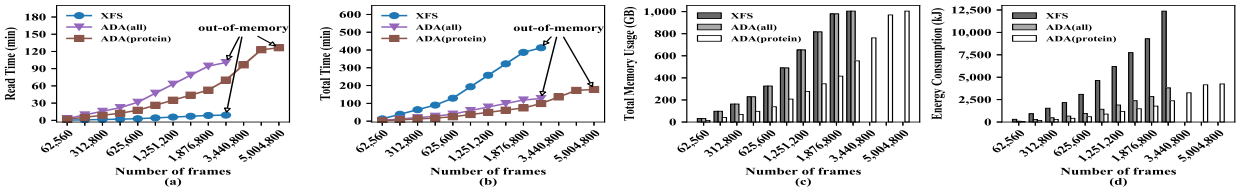


Figure 10. (a) Raw data retrieval time; (b) data processing turnaround time; (c) memory usage; (d) energy consumption.

the server during our experiments so that energy consumption values presented in Fig. 10d reflect the total energy consumed by the server for each VMD process. One can observe that XFS consumes more then 3x energy compared to ADA. Take the frame number 1,876,800 as an example, the server consumes more than 12,500 kilojoules energy if the data is organized by the XFS file system. This value drops to less than 5,000 kilojoules if ADA is employed. The value will be even less (i.e., 2,200 kilojoules) if ADA provides protein data only. This is a significant retrenchment in electricity bills if ADA is applied to a computing center.

5 Related Work

Persistent Storage Systems: NVM-based storage is quickly becoming a necessary component of future systems, driven by the projections of very limited DRAM main memory per node and plateauing I/O bandwidth [12]. Dulloor *et al.* implemented PMFS to enable low-overhead I/O access [6]. Kannan *et al.* presented pVM to exploit the capacity and storage advantages of NVM devices [23]. Xue *et al.* proposed AMF to enlarge address spaces without additional application modification [29]. Wang *et al.* designed an ephemeral burst buffer file system to support scalable and efficient aggregation of I/O bandwidth from NVM devices [26].

Middleware Layer: Studies have explored data staging and caching to buffer data temporarily for performance optimizations of data access in a near future [24]. Xu *et al.* proposed a large-scale object-based active storage platform for data analytics in IoT [28]. Xie *et al.* provided an active storage framework for an object-based storage platform to run data-intensive applications locally [27]. Sim *et al.* [19] presented TagIt that moves the procedure of generating metadata to storage nodes. Zheng *et al.* [32] presented a method that uses compute nodes' idle resources to analyze and index massive data generated by scientific APP in an in-situ way.

Computer Optimization for VMD: To analyze molecular dynamics, VMD is a commonly used tool for visualization. Stone *et al.* discusses the implementation of Embedded-system Graphics Library (EGL) support in VMD and outlines the benefits of the EGL approach for parallel rendering [21]. Decherchi *et al.* proposed NanoShaper that specifically aims at constructing and analyzing the molecular surface of nanoscopic systems [5]. Hilderbrand *et al.* argued the interactive visualization of MD trajectories will achieve better understanding, reliability, and re-usability of MD simulations [9].

6 Conclusions

In this paper, we identified a data processing challenge faced by many molecular dynamics applications for visualization and analysis. To solve it, we developed a prototype of a light-weight file system middleware called ADA dedicated to VMD, which is then integrated into a cluster and a fat-node server. Further, we applied ADA to optimize data visualization for an MD application named GPCR, which employs VMD to visualize protein structures. Our experimental results demonstrate that ADA can noticeably improve performance, memory usage, and energy consumption. Besides, it scales well when memory space is enlarged to 1 TB. In order to extend ADA to other computational biology applications, or more broadly, to other data-intensive applications, we plan to develop a dynamic data categorizing and labeling interface through which a user can describe the structure of his raw data in a configuration file.

References

[1] 2017. FlameGraph Visualization Tool. http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html.

[2] John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. 2009. PLFS: A Checkpoint Filesystem for Parallel Applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*. ACM, New York, NY, USA, Article 21, 12 pages. https://doi.org/10.1145/1654059.1654081

[3] Charles E Cook, Mary T Bergman, Guy Cochrane, Rolf Apweiler, and Ewan Birney. 2017. The European Bioinformatics Institute in 2017: data coordination and integration. *Nucleic Acids Research* 46, D1 (11 2017), D21–D29. https://doi.org/10.1093/nar/gkx1154 arXiv:http://oup.prod.sis.lan/nar/article-pdf/46/D1/D21/23162477/gkx1154.pdf

[4] Charles E Cook, Rodrigo Lopez, Oana Stroe, Guy Cochrane, Cath Brooksbank, Ewan Birney, and Rolf Apweiler. 2018. The European Bioinformatics Institute in 2018: tools, infrastructure and training. *Nucleic Acids Research* 47, D1 (11 2018), D15–D22. https://doi.org/10.1093/nar/gky1124 arXiv:http://oup.prod.sis.lan/nar/article-pdf/47/D1/D15/27437461/gky1124.pdf

[5] Sergio Decherchi, Andrea Spitaleri, John Stone, and Walter Rocchia. 2018. NanoShaper–VMD interface: computing and visualizing surfaces, pockets and channels in molecular systems. *Bioinformatics* 35, 7 (08 2018), 1241–1243.

[6] Subramanya R. Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. 2014. System

software for persistent memory. *Proc. Ninth Eur. Conf. Comput. Syst. - EuroSys '14* (2014), 1–15. https://doi.org/10.1145/2592798.2592814

[7] Devarshi Ghoshal and Lavanya Ramakrishnan. 2017. MaDaTS: Managing Data on Tiered Storage for Scientific Workflows. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '17)*. ACM, New York, NY, USA, 41–52. https://doi.org/10.1145/3078597.3078611

[8] Ibrahim F. Haddad. 2000. PVFS: A Parallel Virtual File System for Linux Clusters. *Linux J.* 2000, 80es, Article 5 (Nov. 2000).

[9] Peter W. Hildebrand, Alexander S. Rose, and Johanna K.S. Tiemann. 2019. Bringing Molecular Dynamics Simulation Data into View. *Trends in Biochemical Sciences* (2019). https://doi.org/10.1016/j.tibs.2019.06.004

[10] Tian Hua, Kiran Vemuri, Mengchen Pu, Lu Qu, Gye Won Han, Yiran Wu, Suwen Zhao, Wenqing Shui, Shanshan Li, Anisha Korde, Robert B. Laprairie, Edward L. Stahl, Jo Hao Ho, Nikolai Zvonok, Han Zhou, Irina Kufareva, Beili Wu, Qiang Zhao, Michael A. Hanson, Laura M. Bohn, Alexandros Makriyannis, Raymond C. Stevens, and Zhijie Liu. 2016. Crystal Structure of the Human Cannabinoid Receptor CB1. *Cell* 167, 3 (2016), 750–762.e14. https://doi.org/10.1016/j.cell.2016.10.004

[11] William Humphrey, Andrew Dalke, and Klaus Schulten. 1996. VMD: Visual molecular dynamics. *Journal of Molecular Graphics* 14, 1 (1996), 33 – 38. https://doi.org/10.1016/0263-7855(96)00018-5

[12] Jungwon Kim, Kittisak Sajjapongse, Seyong Lee, and Jeffery S. Vetter. 2017. Design and Implementation of Papyrus: Parallel Aggregate Persistent Storage. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1151–1162. https://doi.org/10.1109/IPDPS.2017.72

[13] Anton Kokalj. 2003. Computer graphics and graphical user interfaces as tools in simulations of matter at the atomic scale. *Computational Materials Science* 28, 2 (2003), 155 – 168. https://doi.org/10.1016/S0927-0256(03)00104-6 Proceedings of the Symposium on Software Development for Process and Materials Design.

[14] G. Kresse and J. Hafner. 1993. Ab initio molecular dynamics for liquid metals. *Phys. Rev. B* 47 (Jan 1993), 558–561. Issue 1. https://doi.org/10.1103/PhysRevB.47.558

[15] Xi Lin, Mingyue Li, Niandong Wang, Yiran Wu, Zhipu Luo, Shimeng Guo, Gye-Won Han, Shaobai Li, Yang Yue, Xiaohu Wei, Xin Xie, Yong Chen, Suwen Zhao, Jian Wu, Ming Lei, and Fei Xu. 2020. Structural basis of ligand recognition and self-activation of orphan GPR52. *Nature* 579, 7797 (2020). https://doi.org/10.1038/s41586-020-2019-0

[16] Jay Lofstead, Ivo Jimenez, Carlos Maltzahn, Quincy Koziol, John Bent, and Eric Barton. 2016. DAOS and Friends: A Proposal for an Exascale Storage System. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. 585–596. https://doi.org/10.1109/SC.2016.49

[17] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kalé, and Klaus Schulten. 2005. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry* 26, 16 (2005), 1781–1802. https://doi.org/10.1002/jcc.20289

[18] S. Plimpton. 1995. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J Comp Phys* 117 (1995), 1–19.

[19] Hyogi Sim, Youngjae Kim, Sudharshan S. Vazhkudai, Geoffroy R. Vallée, Seung-Hwan Lim, and Ali R. Butt. 2017. Tagit: An Integrated Indexing and Search Service for File Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. ACM, New York, NY, USA, Article 5, 12 pages. https://doi.org/10.1145/3126908.3126929

[20] Gaojie Song, Dehua Yang, Yuxia Wang, Chris de Graaf, Qingtong Zhou, Shanshan Jiang, Kaiwen Liu, Xiaoqing Cai, Antao Dai, Guangyao Lin, Dongsheng Liu, Fan Wu, Yiran Wu, Suwen Zhao, Li Ye, Gye Won Han, Jesper Lau, Beili Wu, Michael A. Hanson, Zhi-Jie Liu, Ming-Wei Wang, and Raymond C. Stevens. 2017. Human GLP-1 receptor transmembrane domain structure in complex with allosteric modulators. *Nature* 546, 7657 (2017), 312–315. https://doi.org/10.1038/nature22378

[21] J. E. Stone, P. Messmer, R. Sisneros, and K. Schulten. 2016. High Performance Molecular Visualization: In-Situ and Parallel Rendering with EGL. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1014–1023. https://doi.org/10.1109/IPDPSW.2016.127

[22] David Strugatsky, Reginald McNulty, Keith Munson, Chiung-Kuang Chen, S Michael Soltis, George Sachs, and Hartmut Luecke. 2013. Structure of the Proton-Gated Urea Channel From the Gastric Pathogen Helicobacter Pylori. *Nature* 493, 7431 (2013). https://doi.org/10.1038/nature11684

[23] Kannan Sudarsun, Gavrilovska Ada, and Schwan Karsten. 2016. pVM: Persistent Virtual Memory for Efficient Capacity Scaling and Object Storage. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16)*. ACM, New York, NY, USA, Article 13, 16 pages. https://doi.org/10.1145/2901318.2901325

[24] Yuan Tian, Scott Klasky, Weikuan Yu, Hasan Abbasi, Bin Wang, Norbert Podhorszki, Ray Grout, and Matthew Wolf. 2012. SMART-IO: SysteM-AwaRe Two-Level Data Organization for Efficient Scientific Analytics. In *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. 181–188. https://doi.org/10.1109/MASCOTS.2012.30

[25] David Van Der Spoel, Erik Lindahl, Berk Hess, Gerrit Groenhof, Alan E. Mark, and Herman J. C. Berendsen. 2005. GROMACS: Fast, flexible, and free. *Journal of Computational Chemistry* 26, 16 (2005), 1701–1718.

[26] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu. 2017. An Ephemeral Burst-Buffer File System for Scientific Applications. *Int. Conf. High Perform. Comput. Networking, Storage Anal. SC* (2017), 807–818. https://doi.org/10.1109/SC.2016.68

[27] Yulai Xie, Dan Feng, Yan Li, and Darrell D.E. Long. 2016. Oasis: An active storage framework for object storage platform. *Future Generation Computer Systems* 56, Supplement C (2016), 746 – 758. https://doi.org/10.1016/j.future.2015.08.011

[28] Quanqing Xu, Khin Mi Aung, Yongqing Zhu, and Khai Leong Yong. 2016. Building a large-scale object-based active storage platform for data analytics in the internet of things. *The Journal of Supercomputing* 72, 7 (01 Jul 2016), 2796–2814. https://doi.org/10.1007/s11227-016-1621-2

[29] D. Xue, C. Li, L. Huang, C. Wu, and T. Li. 2018. Adaptive Memory Fusion: Towards Transparent, Agile Integration of Persistent Memory. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 324–335. https://doi.org/10.1109/HPCA.2018.00036

[30] Shifan Yang, Yiran Wu, Ting-Hai Xu, Parker W. de Waal, Yuanzheng He, Mengchen Pu, Yuxiang Chen, Zachary J. DeBruine, Bingjie Zhang, Saheem A. Zaidi, Petr Popov, Yu Guo, Gye Won Han, Yang Lu, Kelly Suino-Powell, Shaowei Dong, Kaleeckal G. Harikumar, Laurence J. Miller, Vsevolod Katritch, H. Eric Xu, Wenqing Shui, Raymond C. Stevens, Karsten Melcher, Suwen Zhao, and Fei Xu. 2018. Crystal structure of the Frizzled 4 receptor in a ligand-free state. *Nature* 560, 7720 (2018). https://doi.org/10.1038/s41586-020-2019-0

[31] Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang. 2015. In-Memory Big Data Management and Processing: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (July 2015), 1920–1948. https://doi.org/10.1109/TKDE.2015.2427795

[32] Qing Zheng, Charles D. Cranor, Danhao Guo, Gregory R. Ganger, George Amvrosiadis, Garth A. Gibson, Bradley W. Settlemyer, Gary Grider, and Fan Guo. 2018. Scaling Embedded In-situ Indexing with deltaFS. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. Piscataway, NJ, USA, Article 3, 15 pages.