future internet

MDPI

Article

Distributed Big Data Storage Infrastructure for Biomedical Research Featuring High-Performance and Rich-Features

Xingjian Xu *, Lijun Sun and Fanjun Meng

College of Computer Science and Technology, Inner Mongolia Normal University, Hohhot 010022, China
* Correspondence: xingjian@imnu.edu.cn

Abstract: The biomedical field entered the era of "big data" years ago, and a lot of software is being developed to tackle the analysis problems brought on by big data. However, very few programs focus on providing a solid foundation for file systems of biomedical big data. Since file systems are a key prerequisite for efficient big data utilization, the absence of specialized biomedical big data file systems makes it difficult to optimize storage, accelerate analysis, and enrich functionality, resulting in inefficiency. Here we present F3BFS, a functional, fundamental, and future-oriented distributed file system, specially designed for various kinds of biomedical data. F3BFS makes it possible to boost existing software's performance without modifying its main algorithms by transmitting raw datasets from generic file systems. Further, F3BFS has various built-in features to help researchers manage biology datasets more efficiently and productively, including metadata management, fuzzy search, automatic backup, transparent compression, etc.

Keywords: distributed file system; biomedical big data; big data storage

Citation: Xu, X.; Sun, L.; Meng, F. Distributed Big Data Storage Infrastructure for Biomedical Research Featuring High-Performance and Rich-Features. *Future Internet* **2022**, *14*, 273. https://doi.org/10.3390/fi14100273

Academic Editor: Davide Tosi

Received: 27 August 2022
Accepted: 22 September 2022
Published: 24 September 2022

1. Introduction

The advancements in high-throughput sequencing and many other related technologies are providing a faster and cheaper way to profile the biology system. Biomedical data accumulates at an unprecedented rate, even faster than Moore's Law [1], undoubtedly pushing biology into the era of big data. Biomedical big data is characterized by its huge volume size, diverse data types, and heterogeneity in its complex structure, challenging most of the data utilization procedures, such as data storage, transmission, and analysis. Many attempts [2–4] have been made to accelerate biology big data analysis, including developing new algorithms to adopt cloud-based program frameworks or utilizing deep learning technologies.

However, the file system, as well as the efficient analysis software, is crucial for maximizing the value of big data. Compared to the analysis research and their relevant tools development, the underlying file system on which the raw datasets and analysis results are stored has seldom been studied and customized to fulfill the special requirements of biology. Currently, researchers mainly use two kinds of file systems to manage their biology data [5,6]: local file systems and distributed file systems. For personal or small lab usage, a local file system constructed by several local storage devices is a convenient solution, such as Ext4 (https://ext4.wiki.kernel.org/ (accessed on 21 September 2022)), XFS (http://xfs.org/ (accessed on 21 September 2022)), etc. The major disadvantages of local file systems are their limitation of overall storage capacity and low scalability, since the number of usable disks is constrained by local computers and once the local file system is set up and formatted, it is not easy to add more disks to the original file system. For the demand of a robust file system with better performance for big data, a more popular solution is the generic distributed file system [7], such as GlusterFS (https://github.com/gluster/glusterfs (accessed on 21 September 2022)), CephFS (https://docs.ceph.com/ (accessed on 21 September 2022)), etc. A distributed file system is usually constructed by connecting

disks installed in multiple server nodes together, so the capacity and scalability is no longer a problem, however the network communication overhead decreases the speed and latency of read/write operations.

Moreover, besides the disadvantages above, since most existing file systems only provide general purpose storage platforms for generic datasets and are not specially designed for biomedical big data, they inevitably behave poorly in the following main aspects. Firstly, the generic file systems have not been optimized for the unique characteristics of biology big data, such as high repeatability, limited representation forms, and a relatively fixed file format, etc. Secondly, due to the complicated architecture of generic software, it is very hard to integrate more specialized features into the existing file systems [8]. Last but not least, they lack the built-in functionality to perform basic routine tasks related to biological data maintenance, such as sequence search, metadata management, and transparent compression. Some notable attempts have been made, such as openBIS [6], OMERO [5], relzfs [9], and FASTAFS [10], but the problems above are still unresolved: openBIS tends to provide a smart WEB GUI interface to collect and analysis data from lab experiment; OMERO is mainly focused on image data; relzfs is only a prototype of auto compression filesystem; FASTAFS is only suitable for data in FASTA format.

Here, we present F3BFS, an integrated file system dedicated to biology big data, featuring functional built-in utilities, fundamental infrastructure, and future-oriented architecture. F3BFS is suggested to be able to greatly reduce the tedious routine workload of the data repository maintainer and increase the efficiency of the researchers.

2. Methods

2.1. Optimization for Biomedical Big Data

In contrast to the general purpose distributed file system, F3BFS has been optimized for the management of biological big data in numerous ways. Since it is extremely difficult to improve all aspects of a file system, it should be mentioned that all these optimizations for biomedical big data come with a price. As an example, sequential reading of large datasets are designed to have a performance boost, while consuming more system resources and decreasing the performance of other IO types. Generally, F3BFS is recommended only for biomedical big data storage, and, when used for this purpose, it offers the best performance and usability.

The main optimization directions for this project are to improve the IO performance as well as the usability of the file system. According to related surveys [11,12], in most cases, biomedical data storage is typically characterized by the following commonalities and, accordingly, the specific measures for optimizations are:

1. The majority of IO operations are read—especially sequential read—not write. Therefore, when designing the F3BFS overall architecture and underlying data structure, we prioritize improving read performance while sacrificing some write performance. As a result, the read operations of the cluster are greatly accelerated, and the architecture of F3BFS is simplified as it does not require advanced data write features such as copy-on-write or write-ahead logging;
2. A biomedical data file typically contains more metadata, which is needed to be queried frequently, such as, e.g., the species it belongs to, the sequence technology, etc. As such, we automate the extraction of metadata from data files and save it to a centralized database when the data is uploaded to F3BFS;
3. As there are only a few types of common biomedical datasets, more targeted optimizations can be made. For example, we intentionally design an additional layer of a file system to integrate functional utilities, which we call the peripheral layer.

2.2. Content Aware Chunking

For most file systems, the whole content of a file is not stored continuously in a storage device; they are first spliced into much smaller pieces than have actually been saved. It is often used to refer to the small pieces of spliced data as chunks, and the chunking algorithm

determines how the whole dataset is to be spliced. Since general purpose file systems need to balance the IO performance of all data types, they usually tend to keep the chunk size at a fixed length.

F3BFS, however, does not benefit from chunks with a fixed length. For biomedical data, the most commonly seen file types are various kinds of sequence files, such as DNA, RNA, protein, and short reads produced by next-generation sequencing technology. Normally, a sequence file contains many short sequences that are separated by lines prefixed with some markers, such as ">". Many bioinformatics applications require reading at least one short sequence to function. Obviously, chunking the sequence file into short sequences as chunks is the simplest way to improve the read performance for bioinformatics applications.

In light of the above ideas and based on content-defined chunking (CDC) [13], we propose the content aware chunking (CAC) algorithm, which plays an important role in the read operation acceleration and provides high quality transparent compression. The major difference between CDC and CAC is that CAC not only takes care of rolling hashes of sibling sliding windows, but also considers the context of cut points, for example the natural gap between sequences, the mate-pair reads information, etc.

2.3. Transparent Compression

The compression encoder in F3BFS allows the user to store files in a compressed form, and the decompression procedure is totally transparent to applications that need to read the data in the cluster, which means the reader application will automatically get the uncompressed data without having to alter its code. The underlying compression algorithm is relative lempel-ziv (RLZ) [14], which is a high quality compression algorithm for biology sequence files and allows for fast access to arbitrary parts of the sequence file with a very slight space overhead. Since the original author of RLZ has not provided a reference code project, we have to implement RLZ with suffix array algorithm [15], inspired by the related work in Ref. [9].

2.4. Metadata as Extended Attributes

As described in the Section 1, biomedical datasets often have various inline metadata (that hold in file content), and without these data, the whole dataset is meaningless. In most cases, traditional storage solutions require opening the entire file in order to read inline metadata, which is rather time-consuming. Moreover, the metadata is frequently queried.

F3BFS dumps inline metadata from file content to a fast key-value database, RocksDB, to improve metadata querying performance. With this change, most metadata query operations are expected to be redirected to RocksDB, greatly reducing the IO workload of the storage cluster and improving query speed. Upon the mounted file system of the F3BFS, all metadata is exposed using the standard xattr kernel extension.

Furthermore, we implement metadata fuzzy search, which allows users to search for files with a specific metadata pattern. When researchers need to explore a large data repository stored in F3BFS, this feature is extremely helpful. This implementation relies primarily on an open-source library named fuzzy-matcher (https://github.com/lotabout/fuzzy-matcher (accessed on 21 September 2022)).

3. Implementation

The overall architecture of F3BFS is mainly composed of three layers (Figure 1): storage layer, access layer, and peripheral layer. Considering the high requirement of runtime performance, development efficiency and system robustness, the core components of F3BFS are mainly implemented in Rust, which is a static low-level programming language. Since it relies on some features provided by Linux kernel, currently F3BFS, it only supports Linux OS with kernel version $\geq$5.10. We plan to migrate F3BFS to UNIX (such as FreeBSD) in future. Additionally, F3BFS also has a simple Web GUI console and HTTP REST API binding for convenient usage.

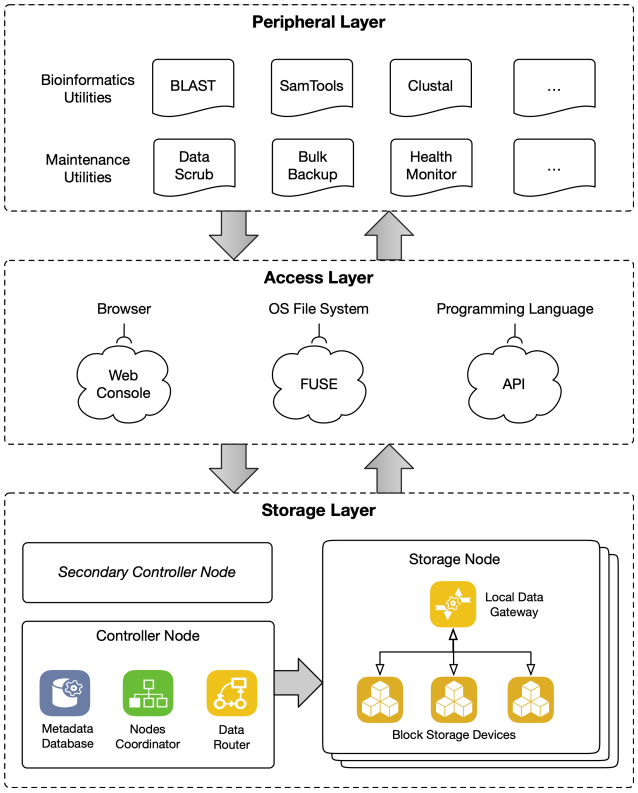


Figure 1. Three layer architecture overview of F3BFS.

3.1. Storage Layer

As part of the F3BFS architecture, the storage layer is deployed on the controller, secondary controller, and storage nodes, containing the core components of F3BFS. Unlike other nodes in the cluster, the controller node does not actually store the user data, but instead stores the cluster's metadata. As a redundant copy of the main controller, the secondary controller can be used for high availability purposes. If the main controller node goes down for whatever reason, the secondary controller will step in and continue to serve the cluster until the main controller is fixed.

The rest of the nodes in the cluster, apart from the controller and secondary controller, serve as the storage nodes, and are thus responsible for storing the chunked user data. To be noted, in order to keep the simplicity of the overall architecture, although F3BFS provides high availability of the controller node, it does not have a mechanism to guarantee the data safety in the storage node. As a consequence, it is recommended that users create RAID volumes and use those volumes as the datastore of the storage node.

3.1.1. Metadata Database

This metadata database contains the attributes of the files that are stored in the cluster. Basic metadata can be categorized into three main types: directory entry, inode, and chunk tag, for example, the file name, the file length, the creation date, the change date, the access date, the permission information, etc. In addition to the basic attributes described above, F3BFS also supports the setting of extended attributes (xattr) for files, for example, the file type, species or tissue type, sample time, etc.

A cluster's performance depends heavily on the performance of the metadata database since the file metadata is frequently accessed. In order to speed up the metadata database read/write operations and maintain the reliability at the same time, F3BFS use RocksDB (http://rocksdb.org/ (accessed on 21 September 2022)) as its metadata database back-end engine. Known for its high performance and optimization for fast storage (such as NAND storage), RocksDB is a popular NOSQL database that is widely used to store key-value pairs.

3.1.2. Nodes Coordinator

All nodes in a cluster, except the controller node, must maintain an active TCP connection with the nodes coordinator process and periodically report its internal state for the coordinator. The node internal state mainly includes the CPU and RAM utilization, block device health, and storage usage, etc.

By querying the nodes coordinator, F3BFS will be able to ensure that the workload is balanced between the nodes, kick out the node that has failed from the cluster at the earliest opportunity, and send a health warning to the cluster administrator. All data held by node coordinator is stored in RAM and is logged to the file system at a predetermined interval of time.

3.1.3. Data Router

It is expected that all data inflows and outflows will pass through the data router on the controller node. The data router transforms raw data, chunks it, and dispatches the chunks to different storage nodes. A diagram of the upload data flow in F3BFS is shown in Figure 2, and the download process is quite the opposite.

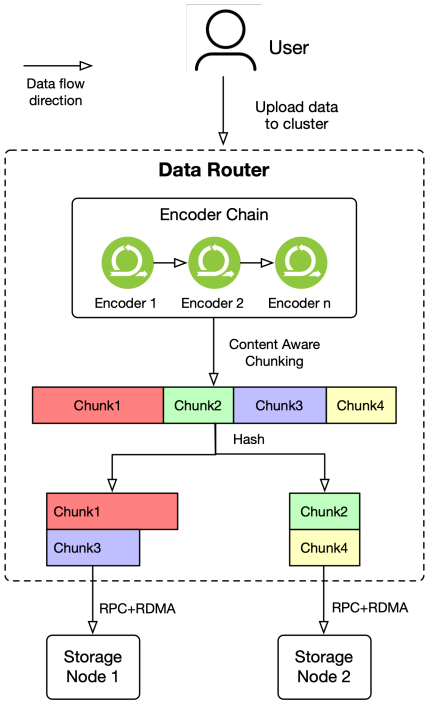


Figure 2. Data flow during the upload procedure.

When a user performs an upload operation, the uploaded data will first be processed by the encoder chain consisting of a series of encoders. Encoders take the output of previous encoders in a chain as their input, and convert the input data into their output, which is the input of the next encoder. A F3BFS encoder allows F3BFS to achieve a wide range of features, such as transparent compression, automatic extraction of sequence metadata, automatic creation of BLAST alignment databases, etc. Through the processing of the encoder chain, the raw uploaded data is finally transformed into the internal stored format.

The mechanism of the encoder and decoder makes it easier for researchers to add their own functions. F3BFS is designed to be extensible by mechanism. Since biomedical analysis is developing very fast, this mechanism is very important to prevent F3BFS from becoming obsolete. The encoders and decoders in chain are highly configurable. By default, F3BFS only enable the compression encoder. Other available encoders are shown in Table 1. Each encoder has a corresponding decoder, which has the opposite function and is responsible for retrieving the original data from the internal stored format.

Table 1. Available encoders/decoders provided by F3BFS.

Encoder/Decoder Name	Function
compression [9]	Transparent compression
seqmeta [10]	Sequence metadata extraction
blastdb [16]	BLAST database construction
jbrowse [17]	JBrowse database construction
clustal [18]	Clustal database construction

Chunking is a method of dividing large files into smaller files—called chunks. For the purpose of data deduplication, achieving better compression quality and improving the parallelism, the output data of the encoder chain will then be chunked into smaller chunks by using the content aware chunking (CAC) algorithm. In order to ensure that chunks can be properly recreated when the user requests to retrieve the file, the data router will send chunk information to the metadata database once it confirms that the uploaded data has been successfully stored in the cluster.

As a final step, the data router will utilize the hash function to determine which storage nodes are being used to store the chunks. Specifically, the hash function is used to resolve the target storage node to which the chunk will be sent to and stored on. In order to transmit data between the data router and the storage node more efficiently, F3BFS will make use of remote-direct-memory-access (RDMA) whenever it is possible to do so, given that the underlying network hardware supports it. Network communication within a data center can be enhanced through the use of RDMA, which provides low-latency network communication.

3.1.4. Local Data Gateway

It is necessary to install a local data gateway on each storage node in order to exchange data with the data router and to report the node state to the nodes coordinator on the controller node. When a local data gateway receives the chunk data from the data router, it will serialize the data into a block device. It is also the local data gateway's responsibility to read the chunk data from the block device and send it to the data router when it receives the chunk retrieval request.

Notably, the local data gateway does not have the ability to do data striping. It is possible for the user to create RAID volumes manually for their local data gateway in order to have data safety and to assure read/write performance.

3.2. Access Layer

Via the access layer, users can visit the data stored on F3BFS, and administrate the cluster itself. There are three ways to access your stored data in F3BFS: via the web console, through the FUSE mount point, and directly through the API.

3.2.1. Web Console

The most convenient way to access F3BFS is by using the web console, which is typically hosted in a web browser. Through the web console, users can manage stored data, upload new datasets, manage the F3BFS cluster, and perform routine cluster maintenance tasks.

There are, however, some limitations of the data accessing feature in the web console, restricted by the capabilities of the web browser. For example, users cannot upload a dataset larger than 500 MB or a download dataset larger than 5 GB to a cluster in a web console, and the IO speed is also limited to 1 MB per second. Therefore, we only recommend that users perform lightweight tasks via the web console.

3.2.2. FUSE Mount Point

FUSE, also known as file system in user space, is a software library for computer operating systems that allows non-privileged users to create their own file systems without

having to probe kernel modules. By supporting FUSE, F3BFS enables users to easily mount their cluster on an existing file system and access the stored data just like a normal file, whether that is on Linux, Windows, or macOS.

The access to data with a FUSE mount point is convenient, but in comparison to F3BFS APIs, the performance is a bit slower, since the FUSE operations need to be transcoded to use the F3BFS APIs. Essentially, the most significant role played by FUSE is that it enables the traditional applications to connect to the F3BFS cluster without altering their existing source code to adopt F3BFS API.

3.2.3. API

A set of POSIX-like file system APIs are provided for programmers by F3BFS as part of its product package. When it comes to programs developed by users of file systems, the most convenient and efficient way to communicate with F3BFS is through the bundled APIs. Currently, F3BFS provides two kinds of APIs (Figure 3): Rust language API and HTTP REST API.

For programs implemented with C or C++ language, they will be able to easily invoke Rust language API via Rust FFI module. However, for programs implemented with other languages, they will need to make use of HTTP REST API in order to access F3BFS, sacrificing some performance at runtime. We plan to offer more API bindings for other commonly used programming languages in the next release.

Compared to the web console and FUSE mount, API bindings are more efficient and generally have better runtime performance in some kinds of workloads, for example, small files read, random read, and file delete [19]. We recommend that programmers use these API bindings in their newly developed programs, rather than the traditional POSIX APIs, for optimal performance.

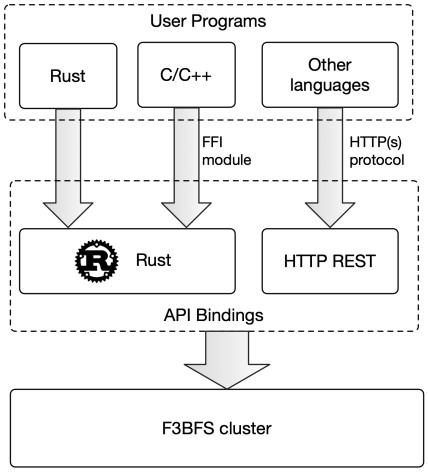


Figure 3. API bindings provided by F3BFS.

3.3. Peripheral Layer

As F3BFS is dedicated to managing biomedical datasets, it offers many advantages over general-purpose distributed file systems, including the fact that it integrates a wide range of commonly used functional utilities in its peripheral layer. In addition, it should be noted that the majority of utilities (Table 2) in the peripheral layer are ported from mature bioinformatics tools.

F3BFS provides a very convenient way for third-party utilities to integrate with it. There are a few steps that users have to follow if they want to integrate a third-party utility into F3BFS: (1) rewrite the IO operations with F3BFS API; (2) rename the compiled binary with the fixed prefix "f3u-", and put all required files of this utility into the particular directory; (3) write a configuration file to tell F3BFS the basic information of this utility and how to launch it.

Table 2. Some commonly used utilities bundled in the peripheral layer.

Utility	Function
BLAST [16]	Basic local alignment search tool
Clustal Omega [18]	Multiple sequence alignment program
SamTools [20]	Tools for high-throughput sequencing data
Bedtool [21]	Genomics analysis toolkit
BamView [22]	Display alignments in BAM files
FastQC [23]	Quality control tool
TagCleaner [24]	Detect and remove tag sequences
FastUniq [25]	de novo duplicates removal tool
Bowtie2 [26]	Sequencing reads alignment
HISAT2 [27]	Sequencing reads mapping tool
SNVer [28]	SNP calling

There are several useful cluster maintenance tools that are available as part of the peripheral layer in addition to the bioinformatics utilities, such as, bulk backup tool, cluster health monitor, data scrub tool, etc. As far as security issues are concerned, these tools are only available to cluster administrators.

4. Evaluation

In this section, the IO performance of F3BFS at runtime and its functional utilities are evaluated. The comparison of the performance, including the latency and throughput, is carried out with Ceph (version 15.2) and XFS (version 5.18), two general-purpose file systems. Ceph is a widely used distributed file system, while XFS is a local file system. Since local file systems do not need to communicate with the network, unlike distributed file systems, they generally have better performance when the data stored is not too large, so we use XFS as the baseline standard in performance evaluation.

4.1. Experimental Setup

There are 18 nodes in the experimental cluster, all of which have the same configuration of hardware: Intel(R) Xeon(R) CPU E5-2640 v4, 128GB RAM, and 6 $\times$ 2 TB solid state disks (SSD). Note that all disks are run in raw disk mode and not in RAID mode. All the nodes are connected by the bonding interface composed of two 10 GB network connections. The client node is connected to the cluster controller node by a 10 GB network connection. The operating system for all nodes is Arch Linux with kernel version 5.18.9.

The Ceph and F3BFS clusters in our experiment are all composed of six nodes and one client node each. For XFS, only one node is allocated and only one 4 GB HDD is used. The storage engine of Ceph in this case is BlueStore, and the messenger type is async + posix. Since the Ceph file system does not not support RDMA, it has to be configured to use the TCP/IP network stack.

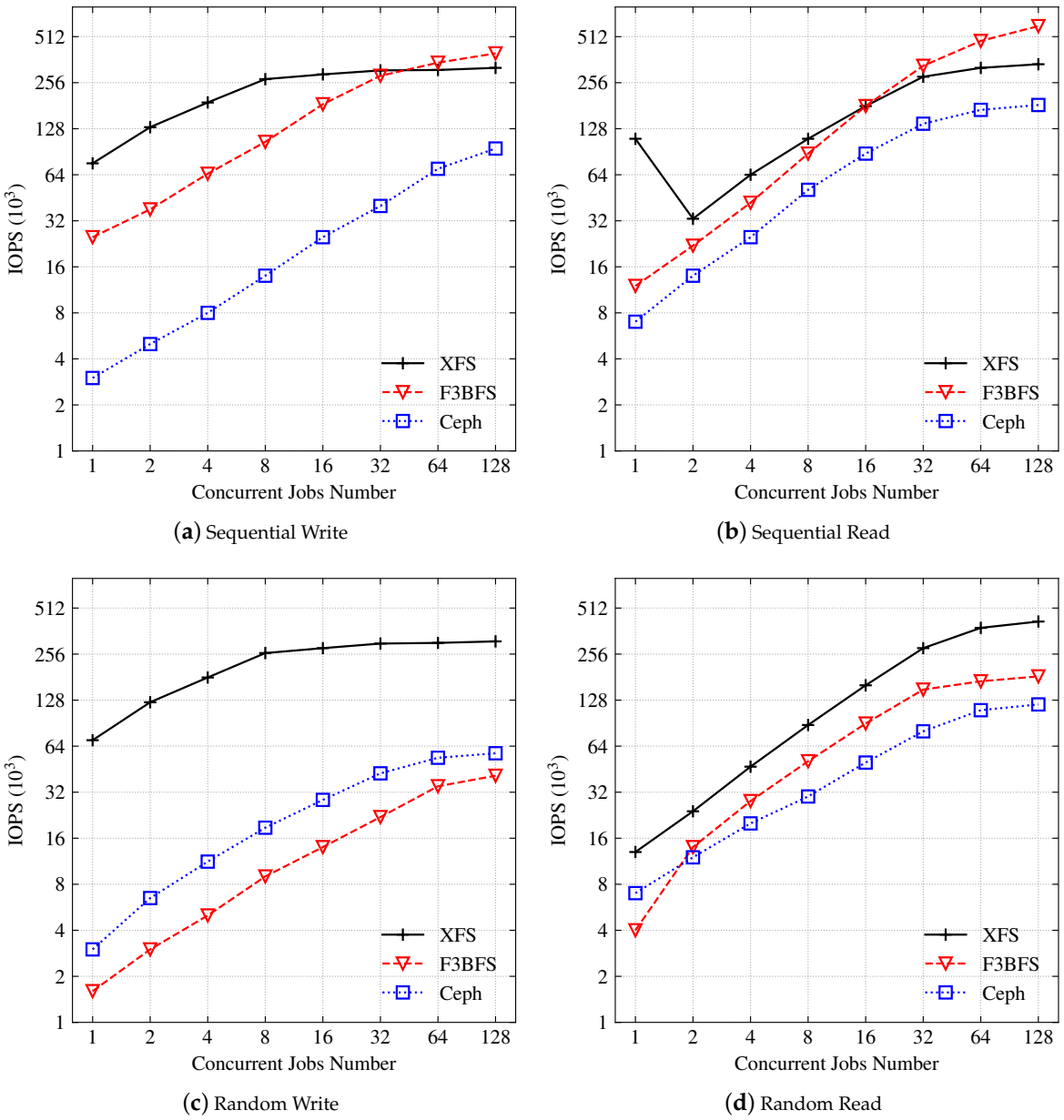
We take the general data workload generation method as PolarFS [29]. The variety of workloads in the experiment are generated using FIO (https://github.com/axboe/fio (accessed on 21 September 2022)), with different I/O request sizes and parallelism levels. Prior to evaluating the performance, the test files are created by FIO and then extended to the 10G. For biomedical sequence workload, we use the dataset (the dataset accession numbers in Genome Warehouse database are GWHANVS00000000, GCA_000966675.1, GCA_000411955.5, GWHAOTW00000000, and GWHANRF00000000) downloaded from Genome Warehouse [30].

4.2. Throughput

Throughput performance is measured by input/output operations per second (IOPS) and input/output sequences per second (IOSPS), where IOPS measures general data storage performance and IOSPS measures biomedical sequence storage performance.

4.2.1. IOPS

Figure 4 shows the IOPS of three file systems with a different number of concurrent tasks being executed. With a small number of concurrent jobs, the local file system XFS always performs the best, since it only requires one disk and thus its main performance bottleneck is the local disk IOPS. The sequential reading IOPS of XFS drops dramatically with the number of clients increasing to 2, most likely due to the built-in DRAM buffer of SSD and its prefetch cache technique. If the firmware receives a sequential read workload request, it will prefetch the subsequent data blocks into its DRAM as cache. This evaluation result is also reported by other work [29].



(**a**) Sequential Write (**b**) Sequential Read

(**c**) Random Write (**d**) Random Read

Figure 4. IOPS evaluation for sequential and random IO operations, respectively.

In general, for sequential read/write, they are scalable until they reach their limit due to the I/O bottleneck, but CephFS' bottleneck is its software, as it cannot saturate the disk I/O bandwidth. F3BFS has some advantages in this regard, particularly in cases where there is a high concurrent workload on the file system. For random read/write, both distributed file systems have certain disadvantages compared to the local file system XFS.

This is due to the fact that the data is chunked and stored on multiple devices, requiring more time for the chunk addressing and network communications. Nevertheless, we can still observe that F3BFS performs slightly better than Ceph in terms of random read speeds.

For the requirement of increasing data read performance, F3BFS is intentionally designed for having poor random write performance, as shown in Figure 4d. However, this issue is not real a major flaw, because for biomedical big data analyses, most kinds of raw data are collected from biology experiment and are generally not modified after being written, thus often making the reading speed of the data more important than its writing speed. For this reason, we believe it is worthwhile to sacrifice a small amount of random write performance in exchange for a significant increase in read performance.

4.2.2. IOPS Latency

Latency is the time between a user issuing a request and of the system receiving a response. From the perspective of a file system, latency can be used to measure the time it takes to complete a single I/O request. In this section, we evaluate I/O latencies for reading or writing 64 KB transfer sizes. Since the local file system XFS does not require a network connection to transfer data, it naturally has the lowest latency performance. From the results shown in Figure 5, it is proven that F3BFS has a lower and more stable I/O latency than Ceph.

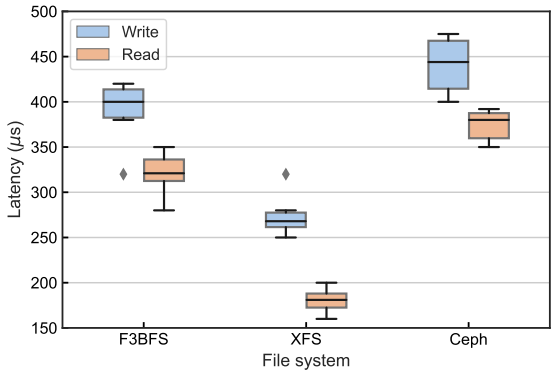
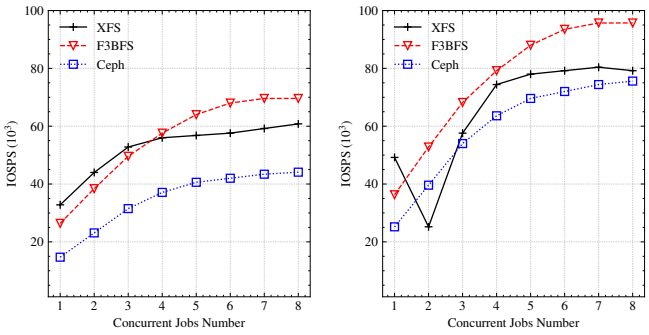


Figure 5. IOPS read/write latency in F3BFS, XFS, and Ceph.

4.2.3. Sequence Throughput

Sequence throughput can be measured by the total number of input/output sequence per seconds, denoted as IOSPS. By evaluating the IOSPS, we can examine the more realistic performance of each file system for biological big data processing. When preparing the feeding data for test, sequences that are too long or too short will be filtered out, and only sequences of appropriate length (from 5 kilobase to 10 kilobase) will be kept to ensure the fairness of testing. Since the sequence will be treated by F3BFS as a logical whole when it is stored, there will no longer be a noteworthy distinction between random and sequential read/write in this test.

As shown in Figure 6, the speed of both reading and writing scales as well as the number of concurrent jobs increases. F3BFS, even with its network transmission overload, is much faster than local disk file system XFS when it comes to sequence read speed. By comparing Figure 4c with Figure 6a, we can see that the random write performance is low, but it does not affect the sequence write performance, which is more common for biomedical data storage.



(**a**) Write (**b**) Read

Figure 6. IOSPS evaluation for sequences IO.

4.3. Scalability

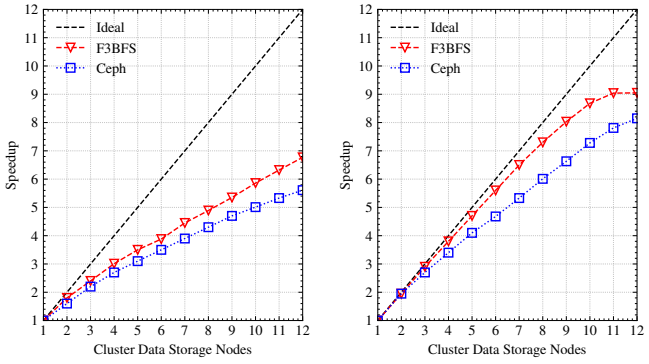
In parallel computing, scalability refers to a parallel system's ability to adjust its performance and costs according to changes in application and system requirements. The speedup [31] of an system is often used as a measure of its scalability. In this section, F3BFS and Ceph are compared for their IO throughput speedups, which can be roughly calculated as the following equation:

$$S = \frac{T_n}{T_1} \tag{1}$$

where S is the I/O throughput speedup, T_1 is the I/O throughput when there is only one data storage and T_n is the I/O throughput when the data storage node numbers increase to n.

It is theoretically possible for the speedup of the system to increase linearly with the number of nodes if it is designed very carefully, which is called linear speedup. For most parallel systems, their ultimate optimization goal is to make its speedup curve converge to the linear speedup curve.

Figure 7 shows the sequential read/write throughput speedup of F3BFS and Ceph, in which the dashed line represents the linear speedup curve. The feeding dataset is biology sequences, which is the same as the dataset used in the throughput evaluation. For both read and write throughput, F3BFS generally performs better than Ceph. Especially noteworthy, as shown in Figure 7b, is that the speedup is converged to about 920 MB after the number of storage nodes reaches 11 because the read throughput reaches the upper limit of the network bandwidth (10 GB) at this point. At this time, the speedup is currently restricted by the network rather than F3BFS itself.



(**a**) Write (**b**) Read

Figure 7. Data throughput speedups with different cluster nodes.

5. Conclusions

F3BFS provides a high performance and feature-rich distributed file system dedicated for biomedical big data. It is recommended for biomedical database administrators, labs that have the need for data storage, and related bioinformatic developers. With the use of F3BFS, researchers will be able to access biomedical datasets and exploit them in a more efficient way. Additionally, the function of F3BFS is open to contributions from other developers, and it can be easily extended through various mechanisms, such as customization of encoders/decoders, extension points of peripheral layers, etc.

There are the two main directions in which we plan to extend F3BFS in future work: (1) Provide the built-in data redundancy mechanism. Currently, it is still required that users deploy storage nodes of F3BFS on RAID volumes. (2) Validate the reliability and performance on a wider range of hardware and network configurations. The data safety and reliability is extremely important for file systems and we have to admit that F3BFS is only validated in server-types of different hardwares. Users are encouraged to report any problems at the issue tracker of the F3BFS project.

Author Contributions: Conceptualization, X.X.; Funding acquisition, X.X. and F.M.; Project administration, X.X.; Software, X.X. and L.S.; Validation, L.S. and F.M.; Writing—original draft, X.X. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by grants from the Fundamental Research Funds for Inner Mongolia Normal University (2022JBQN105), Inner Mongolia Education Department Sociology and Philosophy Special Project (ZSZX21088, ZSZX21092), Inner Mongolia Education Department Science and Technology Funds (NJZY20020), and Talent Project of Inner Mongolia University (No. 2017YJRC020).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets and code used in this paper to produce the experimental results are publicly available at GitHub (https://github.com/pxlab/f3bfs (accessed on 21 September 2022)). The project code of biolitNER is also open sourced and accessible at GitHub under the GPLv3 license.

Acknowledgments: We thank Su for providing sufficient computing resources and helping us set up the relevant network hardware.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IOPS	Input/Output Operations Per Second
IOSPS	Input/Output Sequences Per Second

References

1. Stephens, Z.D.; Lee, S.Y.; Faghri, F.; Campbell, R.H.; Zhai, C.; Efron, M.J.; Iyer, R.; Schatz, M.C.; Sinha, S.; Robinson, G.E. Big data: Astronomical or genomical? *PLoS Biol.* **2015**, *13*, e1002195. [CrossRef] [PubMed]
2. Amin, S.; Uddin, M.I.; AlSaeed, D.H.; Khan, A.; Adnan, M. Early detection of seasonal outbreaks from twitter data using machine learning approaches. *Complexity* **2021**, *2021*, 5520366. [CrossRef]
3. Thakur, N.; Han, C.Y. An Exploratory Study of Tweets about the SARS-CoV-2 Omicron Variant: Insights from Sentiment Analysis, Language Interpretation, Source Tracking, Type Classification, and Embedded URL Detection. *COVID* **2022**, *2*, 1026–1049. [CrossRef]
4. Tang, B.; Pan, Z.; Yin, K.; Khateeb, A. Recent advances of deep learning in bioinformatics and computational biology. *Front. Genet.* **2019**, *10*, 214. [CrossRef]
5. Allan, C.; Burel, J.M.; Moore, J.; Blackburn, C.; Linkert, M.; Loynton, S.; MacDonald, D.; Moore, W.J.; Neves, C.; Patterson, A.; et al. OMERO: Flexible, model-driven data management for experimental biology. *Nat. Methods* **2012**, *9*, 245–253. [CrossRef]
6. Bauch, A.; Adamczyk, I.; Buczek, P.; Elmer, F.J.; Enimanev, K.; Glyzewski, P.; Kohler, M.; Pylak, T.; Quandt, A.; Ramakrishnan, C.; et al. openBIS: A flexible framework for managing and analyzing complex data in biology research. *BMC Bioinform.* **2011**, *12*, 468. [CrossRef]

7. Vashist, S.; Gupta, A. A Review on Distributed File System and Its Applications. *Int. J. Adv. Res. Comput. Sci.* **2014**, *5*, 235–237.
8. Pillai, T.S.; Chidambaram, V.; Alagappan, R.; Al-Kiswany, S.; Arpaci-Dusseau, A.C.; Arpaci-Dusseau, R.H. All File Systems Are Not Created Equal: On the Complexity of Crafting {Crash-Consistent} Applications. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), Broomfield, CO, USA, 6–8 October 2014; pp. 433–448.
9. Navarro, G.; Sepúlveda, V.; Marín, M.; González, S. Compressed filesystem for managing large genome collections. *Bioinformatics* **2019**, *35*, 4120–4128. [CrossRef]
10. Hoogstrate, Y.; Jenster, G.W.; van de Werken, H.J. FASTAFS: File system virtualisation of random access compressed FASTA files. *BMC Bioinform.* **2021**, *22*, 535. [CrossRef]
11. Ison, J.; Ménager, H.; Brancotte, B.; Jaaniso, E.; Salumets, A.; Raček, T.; Lamprecht, A.L.; Palmblad, M.; Kalaš, M.; Chmura, P.; et al. Community curation of bioinformatics software and data resources. *Brief. Bioinform.* **2020**, *21*, 1697–1705. [CrossRef]
12. Liu, B.; Madduri, R.K.; Sotomayor, B.; Chard, K.; Lacinski, L.; Dave, U.J.; Li, J.; Liu, C.; Foster, I.T. Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses. *J. Biomed. Inform.* **2014**, *49*, 119–133. [CrossRef] [PubMed]
13. Xia, W.; Zou, X.; Jiang, H.; Zhou, Y.; Liu, C.; Feng, D.; Hua, Y.; Hu, Y.; Zhang, Y. The design of fast content-defined chunking for data deduplication based storage systems. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 2017–2031. [CrossRef]
14. Kuruppu, S.; Puglisi, S.J.; Zobel, J. Optimized relative Lempel-Ziv compression of genomes. In Proceedings of the Thirty-Fourth Australasian Computer Science Conference, Perth, Australia, 17–20 January 2011; Volume 113, pp. 91–98.
15. Manber, U.; Myers, G. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.* **1993**, *22*, 935–948. [CrossRef]
16. Ye, J.; McGinnis, S.; Madden, T.L. BLAST: Improvements for better sequence analysis. *Nucleic Acids Res.* **2006**, *34*, W6–W9. [CrossRef]
17. Buels, R.; Yao, E.; Diesh, C.M.; Hayes, R.D.; Munoz-Torres, M.; Helt, G.; Goodstein, D.M.; Elsik, C.G.; Lewis, S.E.; Stein, L.; et al. JBrowse: A dynamic web platform for genome visualization and analysis. *Genome Biol.* **2016**, *17*, 66. [CrossRef]
18. Sievers, F.; Higgins, D.G. Clustal omega. *Curr. Protoc. Bioinform.* **2014**, *48*, 3–13. [CrossRef]
19. Vangoor, B.K.R.; Tarasov, V.; Zadok, E. To FUSE or Not to FUSE: Performance of User-Space File Systems. In Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST 17), Santa Clara, CA, USA, 27 February–2 March 2017; pp. 59–72.
20. Li, H.; Handsaker, B.; Wysoker, A.; Fennell, T.; Ruan, J.; Homer, N.; Marth, G.; Abecasis, G.; Durbin, R. The sequence alignment/map format and SAMtools. *Bioinformatics* **2009**, *25*, 2078–2079. [CrossRef]
21. Quinlan, A.R. BEDTools: The Swiss-army tool for genome feature analysis. *Curr. Protoc. Bioinform.* **2014**, *47*, 11–12. [CrossRef]
22. Carver, T.; Böhme, U.; Otto, T.D.; Parkhill, J.; Berriman, M. BamView: Viewing mapped read alignment data in the context of the reference sequence. *Bioinformatics* **2010**, *26*, 676–677. [CrossRef]
23. Brown, J.; Pirrung, M.; McCue, L.A. FQC Dashboard: Integrates FastQC results into a web-based, interactive, and extensible FASTQ quality control tool. *Bioinformatics* **2017**, *33*, 3137–3139. [CrossRef]
24. Schmieder, R.; Lim, Y.W.; Rohwer, F.; Edwards, R. TagCleaner: Identification and removal of tag sequences from genomic and metagenomic datasets. *BMC Bioinform.* **2010**, *11*, 341. [CrossRef] [PubMed]
25. Xu, H.; Luo, X.; Qian, J.; Pang, X.; Song, J.; Qian, G.; Chen, J.; Chen, S. FastUniq: A fast de novo duplicates removal tool for paired short reads. *PLoS ONE* **2012**, *7*, e52249. [CrossRef] [PubMed]
26. de Ruijter, A.; Guldenmund, F. The bowtie method: A review. *Saf. Sci.* **2016**, *88*, 211–218. [CrossRef]
27. Kim, D.; Paggi, J.M.; Park, C.; Bennett, C.; Salzberg, S.L. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nat. Biotechnol.* **2019**, *37*, 907–915. [CrossRef]
28. Wei, Z.; Wang, W.; Hu, P.; Lyon, G.J.; Hakonarson, H. SNVer: A statistical tool for variant calling in analysis of pooled or individual next-generation sequencing data. *Nucleic Acids Res.* **2011**, *39*, e132. [CrossRef]
29. Cao, W.; Liu, Z.; Wang, P.; Chen, S.; Zhu, C.; Zheng, S.; Wang, Y.; Ma, G. PolarFS: An ultra-low latency and failure resilient distributed file system for shared storage cloud database. *Proc. VLDB Endow.* **2018**, *11*, 1849–1862. [CrossRef]
30. CNCB-NGDC Members and Partners. Database Resources of the National Genomics Data Center, China National Center for Bioinformation in 2022. *Nucleic Acids Res.* **2021**, *50*, D27–D38.
31. Eager, D.L.; Zahorjan, J.; Lazowska, E.D. Speedup versus efficiency in parallel systems. *IEEE Trans. Comput.* **1989**, *38*, 408–423. [CrossRef]