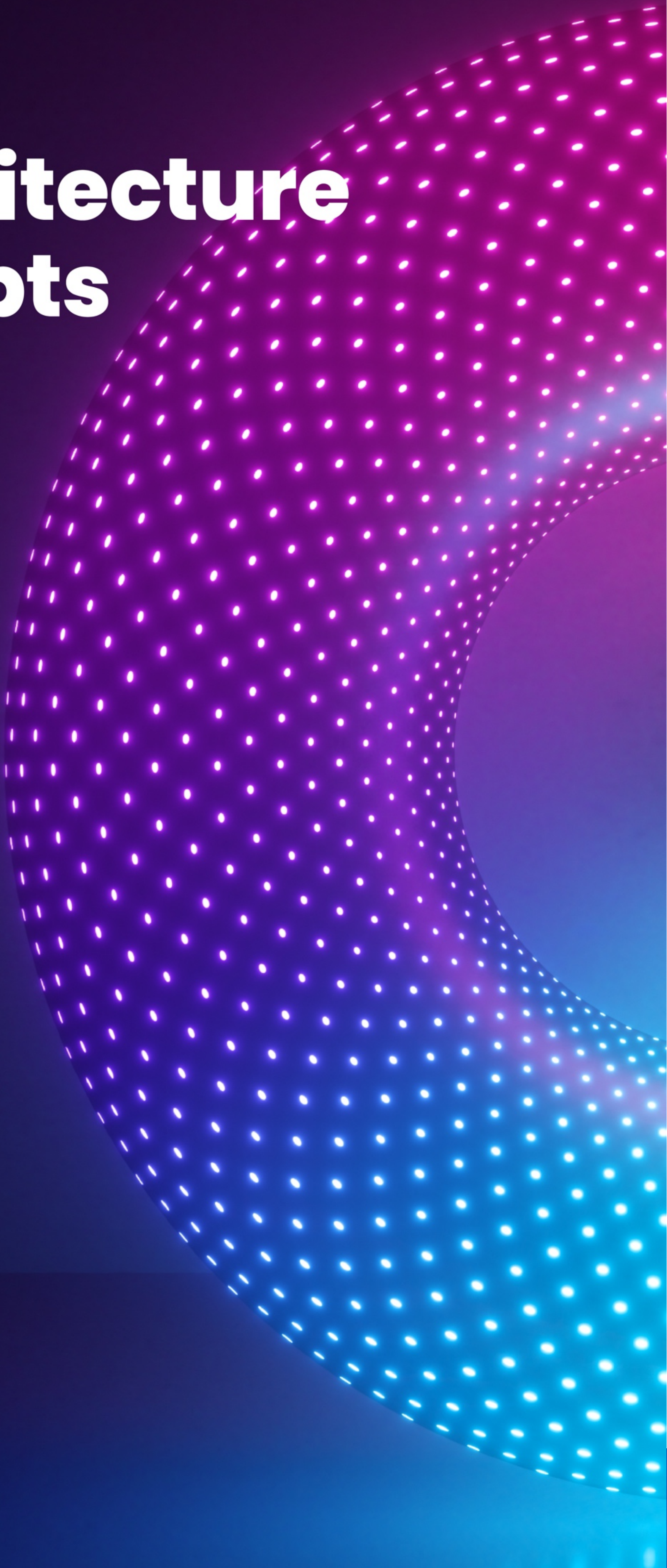
WEKA

WEKA Architecture Key Concepts

March 2022

Tech Brief

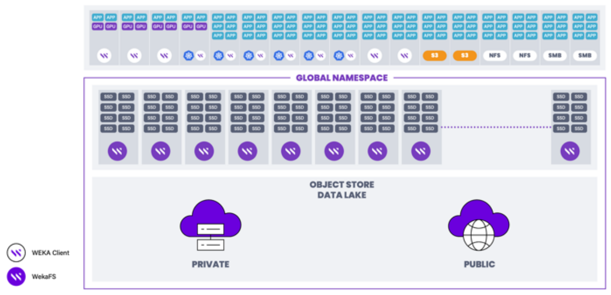


ABSTRACT

This white paper provides a technical summary of the architecture of the WEKA@ Data platform, with a focus on key architecture elements. For a more detailed look, see the WEKA FS Architecture Guide.

THE WEKA DATA PLATFORM

The WEKA Data Platform was written entirely from scratch to deliver the highest-performance file services by leveraging NVMe flash. The software includes integrated tiering that seamlessly expands the filesystem namespace to and from object storage, without the need for special data migration software or complex scripts; all data resides in a single namespace for easy access and management.

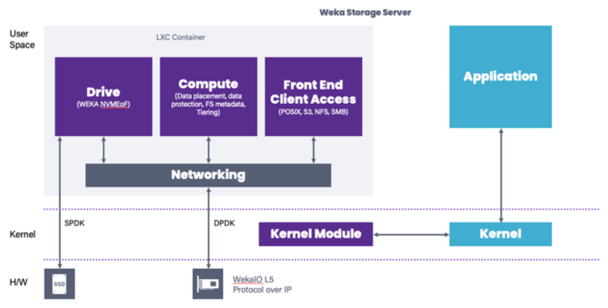


WEKA's unique architecture, as shown in Figure 1, is radically different from legacy storage systems, appliances, and hypervisor-based software-defined storage solutions because it not only overcomes traditional storage scaling and file sharing limitations but also allows parallel file access via POSIX, NFS, SMB, S3 and NVIDIA GPUDirect Storage (GDS). It provides a rich enterprise feature set, including local snapshots and remote snapshots to the cloud, clones, automated tiering, cloud-bursting, dynamic cluster rebalancing, private cloud multi-tenancy, backup, encryption, authentication, key management, user groups, quotas with advisory, soft and hard parameters and much more.

WekaFS ARCHITECTURE

At the core of the WEKA data platform is WekaFS, a distributed, parallel file system that eliminates the traditional block-volume layer managing underlying storage resources. This integrated architecture does not suffer the limitations of other shared storage solutions and delivers both scalability and performance effectively.

The image below provides an overview of the software architecture from the application layer all the way to the physical persistent media layer. The WEKA core components, including the WekaFS unified namespace and other functions such as virtual metadata servers (MDSs), execute in user space in a Linux container (LXC), effectively eliminating time-sharing and other kernel-specific dependencies. The notable exception is the WEKA Virtual File System (VFS) kernel driver, which provides the POSIX filesystem interface to applications. Using the kernel driver provides significantly higher performance than what can be achieved using a FUSE user-space driver, and it allows applications that require full POSIX compatibility to run on a shared storage system.


Weka Storage Server
User Space
LXC Container
Drive (WEKA NVMEoF)
Compute (Data placement, data protection, FS metadata, Tiering)
Front End Client Access (POSIX, S3, NFS, SMB)
Application
Networking
Kernel
SPDK
DPDK
Kernel Module
Kernel
H/W
SSD
WekaIO L5 Protocol over IP

WEKA supports all major Linux distributions and leverages virtualization and low-level Linux container techniques to run its own RTOS (Real-Time Operating System) in user space, alongside the original Linux kernel. WEKA manages its assigned resources (CPU cores, memory regions, network interface cards, and SSDs) to provide process scheduling, memory management, and to control the I/O and networking stacks. By not relying on the Linux kernel, WekaFS minimizes context switching, resulting in a shorter IO path and predictable low latencies. It also allows upgrading of the WekaFS backend storage services independently of Linux OS and WEKA client (front end) upgrades.

WekaFS functionality running in its RTOS is comprised of the following software components:

- File Services (Front-End) – manages multi-protocol connectivity
- File System Compute and Clustering (Back-End) – manages data distribution, data protection, and file system metadata services
- SSD Drive Agent – transforms the SSD into an efficient networked device
- Management Process – manages events, CLI, statistics, and call-home capability

WEKA core software runs inside LXC containers that have the benefit of improved isolation from other server processes. WEKA software, when deployed, is containerized as microservices: Multiple containers for SMB, NFS, S3, and other WekaFS data functions may exist per server. By spanning multiple LXC containers, WEKA enables even greater parallelism and the ability to use more CPU cores and RAM than a single LXC container. A WEKA VFS driver enables WekaFS to support full POSIX semantics and leverages lockless queues for I/O to achieve the best performance while enhancing interoperability. The WekaFS POSIX file system has the same runtime semantics as a local Linux file system (e.g., Ext4, XFS, and others), enabling applications that previously could not run on NFS shared storage because of POSIX locking requirements, MMAP files, performance limitations, or other reasons.

Bypassing the kernel means that WEKA's software stack is not only faster with lower latency, but is also portable across different bare-metal, VM, containerized, and cloud environments. WEKA also only uses the resources that are allocated to it inside its LXC containers, which means it can consume as little as one server core and a small amount of RAM in a shared environment (converged architecture- application and storage software sharing the same server) or as much as all the resources of the server (a dedicated appliance). The same software stack is utilized in either case.

FILE SYSTEM CLIENTS AND STORAGE SERVERS

Like other parallel file systems, we talk about WEKA in terms of storage servers and file system clients. Storage servers are all of the hosts that make up the file system. Filesystem clients are the HPC nodes that are accessing the file system though the native POSIX driver. The file system client is implemented as two loadable kernel modules and a userspace process called the frontend that ultimately communicates with software functions on the server side. Like other POSIX file systems, applications interact with the Linux VFS layer and not directly with specific file systems.

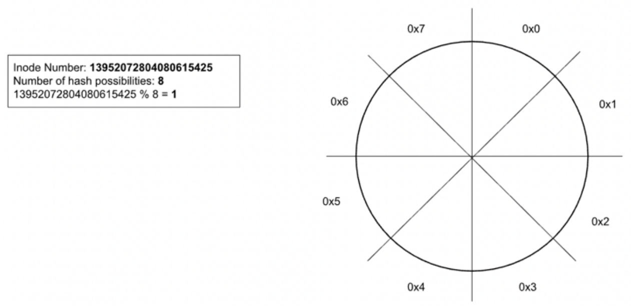
Each file system provides a kernel module that plugs into the VFS layer and provides the translations between standard POSIX calls and a specific file system. In the case of WEKA, there are two kernel modules. The first gateway module provides the direct VFS interaction and maintains the state for open filehandles. The second module splits and forwards the requests to the front-end userspace process. All of the communication with the storage servers is done by the front-end process.

Each storage server will have compute processes that run the data/metadata function of the file system. This is the core logic of the file system layer. It handles both metadata operations as well as data operations. In a production environment, there will be thousands of data/metadata functions. Each data/metadata function handles a slice of the file system determined by a distributed hashing algorithm.

Each back-end storage server also has multiple Drive processes to manage the NVME devices of the server. These not only process the data from the network but also manage the physical block mapping on the devices. Finally, each storage server also has a management process to handle the clustering of the servers. Five servers in the cluster are chosen to be part of a consensus algorithm. This allows a global state of what hardware is up and what is unreachable. The consensus protocol prevents the possibility of a “split brain” situation.

DISTRIBUTED HASHING

One key task a parallel file system must do is divide the work among multiple servers or even different processes on servers. WEKA does this by hashing certain elements of the operation to determine which server process will handle a particular operation. The following is an oversimplified example. Imagine a whole namespace. The goal of hashing is to divide up the namespace. One common way to do this is by taking a simple modulo of some value. In the example, the inode number is divided by the number of hash points we have. In this case it’s 8. We use the modulo or remainder as our answer. All values that have a modulo of 1 when divided by 8 will hash to the same value.



Inode Number: **13952072804080615425**
Number of hash possibilities: **8**
13952072804080615425 % 8 = **1**

This gives us a programmatic way to determine which server will handle this inode. Since everybody in the cluster knows the number of hash possibilities, they can all come up with the same answer. We can also use similar mechanisms to build keyed arrays as the keys can be hashed for efficient lookups. In practice, this is an oversimplified example. The file system does not hash on a simple modulo and more than the inode is hashed. However, distributed hashing is a key way WEKA distributes work across the cluster. It’s also used extensively in data structures to avoid scaling problems associated with traditional tree/b-tree methods.

MANAGING METADATA

With WEKA both POSIX and internal WEKA metadata is distributed. Each data/metadata (d/m) function manages it’s own slice of the file system. The data structure tracking all the metadata for a given d/m function is called the registry. For each d/m function there is a well-known superblock stored. The superblock contains a pointer to 512 L1 blocks. Each L1 block contains some number of L2 blocks. L2 blocks point at the various metadata blocks controlled by this d/m function. While the L1 blocks are allocated at cluster start, the L2 blocks are only created as needed.

There are several types of metadata blocks at the leaf level (all metadata blocks are 4k):

- File Inode - POSIX and WEKA proprietary file level information. C/M/A time, posix permissions, file length, etc.
- Directory Inode - POSIX information C/M/A time, posix permissions, file length, etc. Directory split level, (no entries are stored in a directory inode)
- Directory Slice - Dentries for a directory. May be spread across multiple slices
- Extent Descriptor Block - WEKA internal metadata describing where each 4k block of a 1MB section of
- a file is stored.

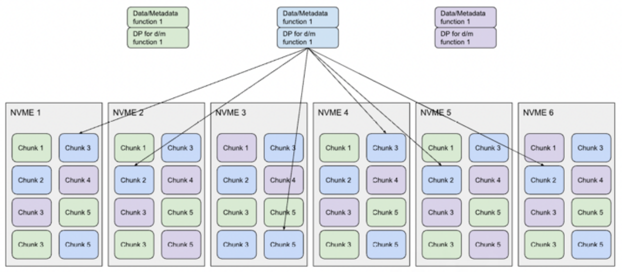
In general, the d/m function’s registry tree is stored, modified, and accessed in memory. There is a process to periodically write the tree out to disk as well. All of the metadata is protected at the same protection level as the data in the cluster. However, the superblock is replicated M+N times providing additional resiliency for this single

4k block. The super block also contains some additional data for failure recovery. It stores a bitmask of all used placements (covered later). It also stores the location of the journal tail.

DATA PROTECTION

WEKA data protection is based on failure domains not necessarily a single drive. The definition of a failure domain is configurable. However, the default failure domain is a storage server. A failure domain can be as small as a single drive or something larger like a multi-server enclosure or even a whole rack.

WEKA data protection level is enumerated M+N like a traditional raid. In this case, M is the number of data blocks that are protected by N parity blocks. For example, an 8+2 would provide fault tolerance for two failure domains with a storage capacity overhead of roughly 20%. M has a valid range of 4 to 16. N can be 2, 3, or 4.



Spares can also be configured. Spare space is not concentrated on a single device or failure domain. It's distributed across the cluster. This allows for faster rebuilds and allows the underlying hardware to contribute to the performance of the cluster. Keep in mind that one spare drive of capacity will be equal to the size of the largest failure domain. For example, if each storage server has 10 disks and one spare is configured, 10 disks worth of capacity will be reserved across the cluster. Up to 4 spares can be configured.

Each data/metadata function in WEKA stores the data blocks and metadata it's responsible for in its own dedicated allocations of storage. Each NVME device is carved up into 128MB allocations called "chunks". Each chunk is dedicated to a specific data/metadata function. Any given data/metadata function will likely own more than one chunk on a given NVMe drive..

SUMMARY

The WEKA Data Platform is based on WekaFS, a POSIX-compliant high-performance clustered, parallel file system that has been built from the ground up to run natively on NVMe based storage. It is an ideal solution for performance-intensive applications that demand high I/O and high concurrency to multiple clients. It is in widespread use across areas such as Life Sciences, Financial Analytics, GPU-based ML, DL and AI applications, EDA, and HPC applications. It excels in large bandwidth and small I/O-intensive applications that have relied on parallel file systems for best performance. WEKA reduces the cost and complexity of storage, requiring fewer hardware resources compared to traditional solutions. It also fully supports legacy protocols such as NFS and SMB and has a rich set of enterprise-class features.

To learn more visit www.weka.io.