Feb 28, 2023

VAST Catalog: Treat Your File System Like a Database

Posted by Andy Pernsteiner

At VAST, we take great pride in listening to our customers and building features and functionality that solves real problems for them. Our customers deal with some of the world's largest data sets and they regularly push the boundaries of what's possible, pushing VAST to break new ground and deliver new and innovative ways to manage and extract value from data.

One of key benefits of VAST's DASE architecture is the ability to scale up to exabytes of capacity across billions of files and objects. But with this scale comes the challenge of managing all of that data. To tackle this, VAST has created a built-in metadata index that we call the VAST Catalog. Now you can search and find data easily – in a fraction of the time that traditional methods would take. What's even more interesting is that this technology allows your users and applications to start treating the file system like a database, enabling next generation AI and ML applications to use it as a self-referential feature store.

The VAST Catalog is the foundation for a Semantic Layer. This layer, composed on top of files and objects, eliminates the need to create or maintain separate systems. Now line of business and domain experts can quickly discover and access data using standard search terms.

Use Cases

Here are few examples of how customers can leverage the VAST Catalog:

AI/ML Feature Store

From a high level, AI and ML models need to train on data, and increasingly on significant amounts of it. When this data is processed and analyzed, features and attributes about each piece of data are generated. These features need to be stored somewhere that can be quickly and easily retrieved. Historically, data engineers and ML ops engineers have maintained separate databases or indexes where this information is kept. Oftentimes, these separate feature stores will have references or pointers back to

the original files or objects. This lineage is important to keep intact because one needs to be able to find the exact data used to generate the features, which in turn influence model development.

When the feature store and the data store are separate entities, it can be challenging to ensure that this lineage is preserved. It also means that there can be two “sources of truth,” which can lead to divergence, duplication of data, and a difficulty to “prove” that they are in sync with each other.

Using the VAST Catalog, Data and ML ops engineers can directly leverage Object and File System metadata to embed features and attributes directly on the object store by using S3 Tags and S3 Object Metadata, knowing that it will be indexed and ready for query. This also means their applications can treat it as a feature store.

Imagine if you could :

Upload a series of S3 objects, each which has metadata that look like so:

"Metadata": {

"city": "Pittsburgh",

"make": "Apple",

"model": "iPhone 14",

"state": "Pennsylvania",

"zip": "15218",

"processed": "false"

}

2. Once you have finished uploading your objects, you could run a query and find any file that has not been process as indicated by the tag_processed:

SELECT parent_path,name,size,user_tags

FROM "vast_big_catalog_table:/"

where user_tags_count > 0

AND element_at(user_tags, 'processed')='false';

parent_path | name | size | user_tags

----------+----------

/photos/ | andy-coffin.jpg | 414403 | {processed=false}

/vfs/processed/false/ | PXL_20230115_082215604.jpg | 3143258 | {processed=false}

/photos/ | PXL_20230215_222637434.MP.jpg | 5779131 | {processed=false}

File system housekeeping

Customers entrust VAST with many PBs (some in the 100s of PBs) of business-critical unstructured data. In most cases, customers are using VAST as a consolidation point for a number of previously disparate workloads, ranging from scratch to homedirs, from datalakes to backup repositories. Bringing all of this data together offered significant efficiencies in terms of both cost as well as reduced application and administrative burdens.

However, a challenge remains: on a system that houses PBs of data across billions of files and objects, how do users easily find what they are looking for? How can administrators understand how their capacity is being used, and ensure that they continue to serve their users effectively?

Sure, you could use a 3rd party product or open source application to crawl and index your file system, but that creates a new set of challenges:

- Scanning the file systems takes a long time – in some cases several days. This means your catalog is always out of sync with the actual current state of the file system.
- Scanning can impose load on the file system, which can impact performance for production applications and users.
- 3rd party products are additional infrastructure to manage and maintain; not to mention the additional costs for servers and software.

Now, imagine that you had a catalog that could tell you anything you wanted to know about what was stored on your file system and was always up to date? What could you do?

Find files

- Find all files older than 90 days larger than 10GB which exist in the /projects directory
- Find all files created since last week by a specific user
- Find all objects with the tag “processed” where the value = false

Capacity reporting

- Rank the users consuming the most capacity in specific folder/projects for example
- Understand the data types in your system by ranking capacity by file extension

How it works

Now that you understand how the VAST Catalog can be used, I’ll explain some of the details of what we built, and how it works.

VAST Catalog backing store

Any metadata catalog must rely on some kind of database in order to house all of the information about files and objects. A number of external systems in use today use open source technologies such as mysql, postgres, cassandra or elastic. We considered using such mechanisms but knew that they would eventually become chokepoints in terms of scale and performance. Additionally, we’d have to find a way to package and manage them, which adds to our own administrative burden. Therefore, we built our own database and like anything else on our platform, it’s designed for scale, performance, and ease of use. In the case of the VAST Catalog, it requires zero administration or setup; simply upgrade to the latest VAST SW version (4.6) and it’s ready to go.

Examining the database table used by the catalog illustrates the metadata attributes we are extracting:

- One record (row) for each file or object
- Each metadata attribute of interest is a separate column

- Complex metadata attributes (such as S3 tags) are stored in “map” columns (imagine a JSON doc in a single cell)
    - For commonly used tags and object metadata: indexed fields can be added on the fly.

Below is the table schema used by VAST Catalog:

Column	Type	Description
Phandle	row()	VAST internal identifier for each object/file (comparable to an inode number)
creation_time	timestamp	Element creation time (birthtime)
uid	integer	POSIX UID of file (owner)
owner_sid	varchar	Windows owner SID (owner)
owner_name	varchar	Owner user name
gid	integer	POSIX GID of file (group)
group_owner_sid	varchar	Windows group SID
group_owner_name	varchar	Group owner name
atime	timestamp	atime – last access time
mtime	timestamp	mtime – last content modification time
ctime	timestamp	ctime – last file system metadata change time
nlinks	integer	Number of associated hard links
element_type	integer	(eg: file, directory, named pipe, etc)
size	integer	Size of file/object
used	integer	Space used by file/object
name	varchar	File name

extension	varchar	File name extension (e.g. “gz” or “exe”)
parent_path	varchar	Parent path of file (dirname)
symlink_path	varchar	If object is a symlink, this is the target
major_device	integer	
minor_device	integer	
s3_locks_retention	row()	
nfs_mode_bits	integer	POSIX permissions mode (as integer)
name_aces_exist	Boolean	Flag to indicate extended ACEs (NTFS/NFS4/Posix) exist on this element
s3_locks_legal_hold	boolean	Legal hold lock on this object
user_tags_count	integer	Number of tags associated with this object
user_metadata	map()	Metadata items on object (x-amz-meta)
user_tags	map()	S3 Tags associated with this object

Note that the underlying VAST DB tables support schema evolution, which means that new columns can be added in the future. A potential future use case is to scrape headers for common file types and store them in the index.

Engine

Keeping this table up to date requires a scalable engine that can detect changes and insert them without compromising the performance of the system. In order to do this, we leveraged a core function which has been in use on VAST for over two years: our snapshot and replication engine. Based on a customer-definable schedule (that can be as frequently as every 15 seconds), VAST will take a snapshot at the root of the file system and build a changelist that is inserted into the table. This all happens without user intervention, and requires no additional management or monitoring.

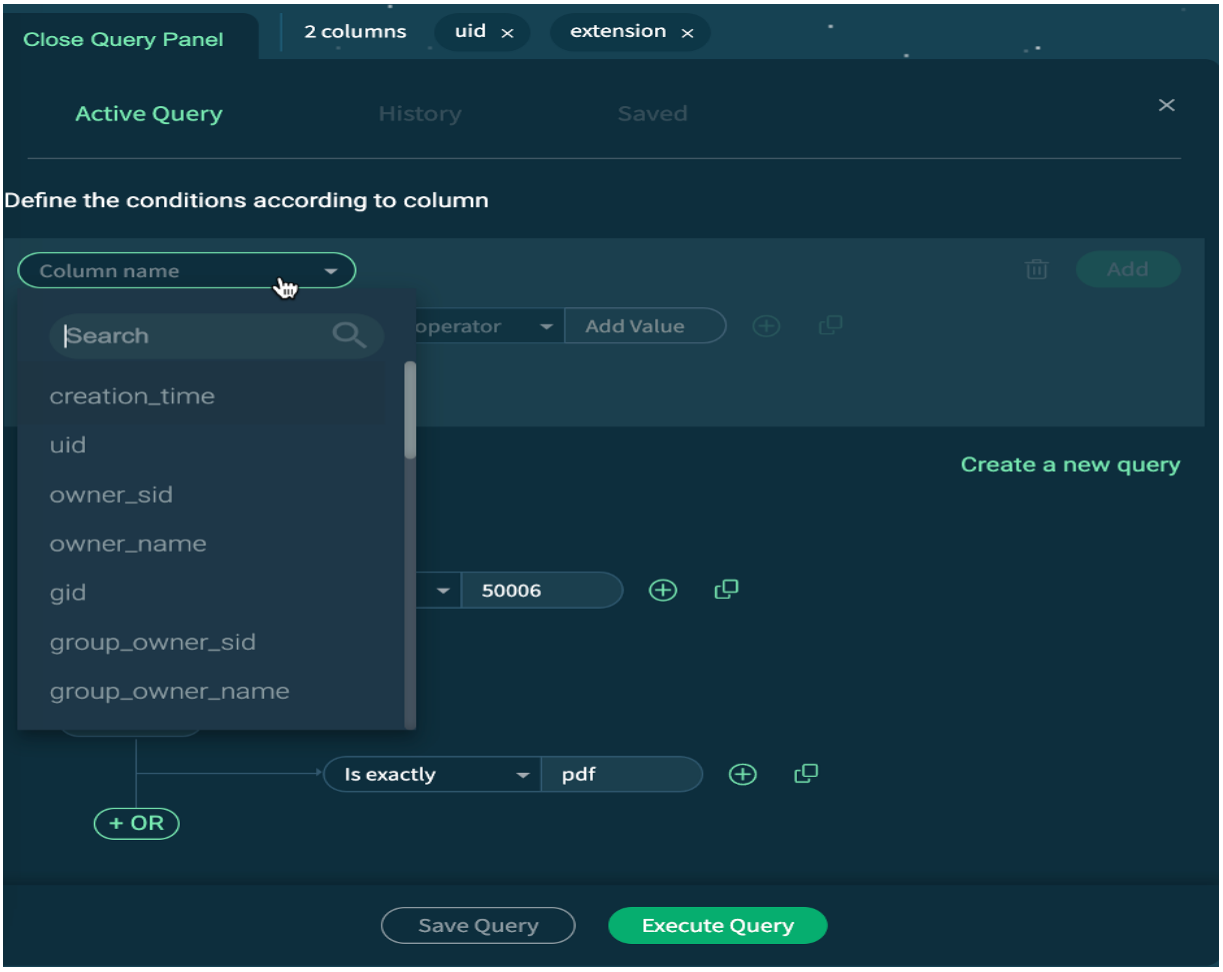
Additionally, because we are using our snapshot technology for this purpose, we are also taking snapshots of the catalog itself. This means that we have historical versions of the catalog which can be used for a number of use cases, such as querying the VAST Catalog at specific points in time for comparison.

Interfacing with VAST Catalog

Since we serve a wide variety of customers with different preferences for how they consume their data, we wanted to cover all the bases when it came to how they could interact with the VAST Catalog.

WebUI

VAST provides an intuitive, highly responsive user interface that allows administrators to easily perform searches and queries using any of the aforementioned attributes to find exactly what they are looking for. Faceting of results is simple and requires just a few short clicks. Results are displayed within seconds, even when the file system is populated with millions or even billions of files and objects.



CLI

Power users will appreciate the rich CLI that provides all the same functionality of the UI, while also allowing for pipelining of commands to allow for better sorting and aggregations. Consider the following command that sums the capacity consumed by a given user's files, faceted by minimum size and file extension:

```
big-catalog query \

--path /nametest \

--extension pdf \

--uid 50006 \

--size 52428800: \

-l1000\

|gawk -F\| '{print $4}' |gawk '{s+=$1} END {print s}'
```

API

VAST actually provides two APIs for interacting with the Catalog:

1- Our standard RESTful API, which is available for all administrative aspects, can also be used to perform VAST catalog queries
2- A new pythonSDK that allows for direct interaction with the underlying database that houses the VAST catalog. This can allow for more sophisticated applications to be built and incorporated into AI and ML pipelines.

Query Engines

VAST also exposes the catalog, as well as user-defined database tables, using some of the most popular open source query engines. Today, ApacheSpark as well as Trino are supported, through use of a storage connector deployed on those engines.

Use of a query engine offers some additional advantages:

- Use the SQL dialect for forming queries, which allows for significant power and flexibility
- Improved query performance when using a distributed query engine
- Better support for aggregation functions.

Here are a few examples:

```
SELECT parent_path,name,size,uid FROM

"vast_big_catalog_table:/"

WHERE uid = 50006

AND size > 52428800

AND extension = 'pdf';
```

An aggregation example:

```
SELECT sum(used)

FROM "vast_big_catalog_table:/nametest"

WHERE uid = 50006

AND size > 52428800 ;
```

Closing

The VAST Catalog is yet another concrete example of how VAST is a differentiated data platform. We can't wait to see how our customers leverage it in their workflows.