US 20250258796A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2025/0258796 A1**

Akidau et al. (43) **Pub. Date: Aug. 14, 2025**

(54) **CONTINUOUS INGESTION OF CUSTOM FILE FORMATS**

(71) Applicant: **Snowflake Inc.**, Bozeman, MT (US)

(72) Inventors: **Tyler Arthur Akidau**, Seattle, WA (US); **Thierry Cruanes**, San Mateo, CA (US); **Benoit Dageville**, San Mateo, CA (US); **Ganeshan Ramachandran Iyer**, Redmond, WA (US); **Subramanian Muralidhar**, Mercer Island, WA (US)

(21) Appl. No.: **19/193,450**

(22) Filed: **Apr. 29, 2025**

Related U.S. Application Data

(63) Continuation of application No. 18/050,122, filed on Oct. 27, 2022, now Pat. No. 12,314,225.

Publication Classification

(51) **Int. Cl.**

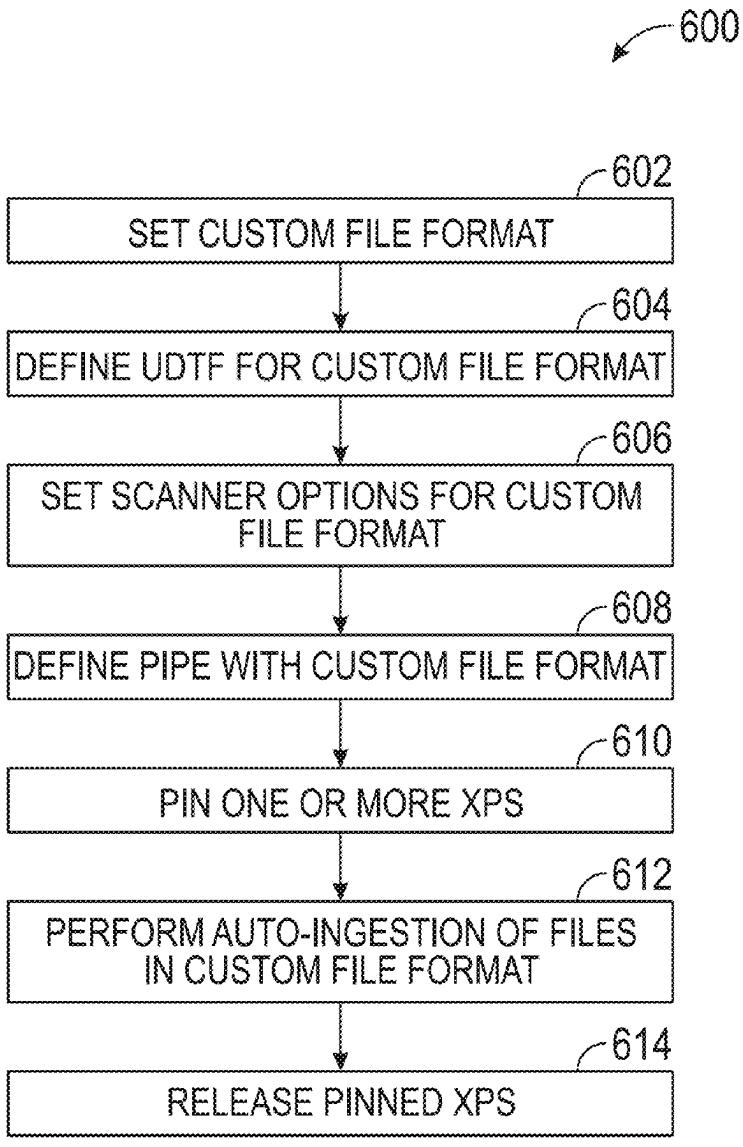
G06F 16/14	(2019.01)
G06F 9/50	(2006.01)
G06F 16/11	(2019.01)

(52) **U.S. Cl.**

CPC ***G06F 16/148*** (2019.01); ***G06F 9/5022*** (2013.01); ***G06F 16/116*** (2019.01)

(57) **ABSTRACT**

Techniques for continuous ingestion of files using custom file formats are described. A custom file format may include formats not natively supported by a data system. Unstructured files (e.g., images) may also be considered custom file formats. A custom file format may be set using a user defined table function and scanner options.



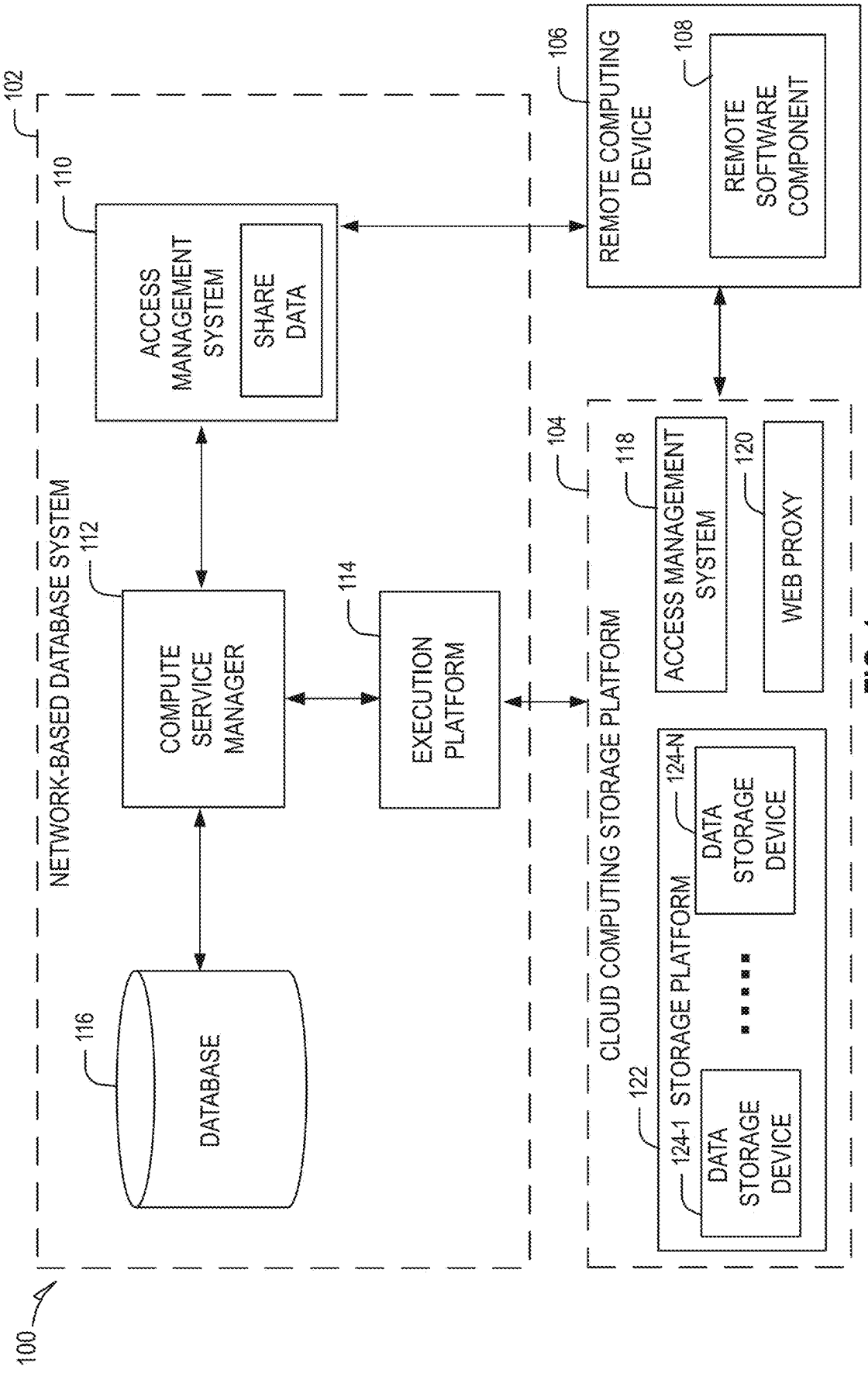

100
102
NETWORK-BASED DATABASE SYSTEM
110
ACCESS MANAGEMENT SYSTEM
SHARE DATA
106
REMOTE COMPUTING DEVICE
108
REMOTE SOFTWARE COMPONENT
112
COMPUTE SERVICE MANAGER
114
EXECUTION PLATFORM
104
CLOUD COMPUTING STORAGE PLATFORM
118
ACCESS MANAGEMENT SYSTEM
120
WEB PROXY
116
DATABASE
122
124-1 STORAGE PLATFORM
DATA STORAGE DEVICE
124-N
DATA STORAGE DEVICE

FIG. 1

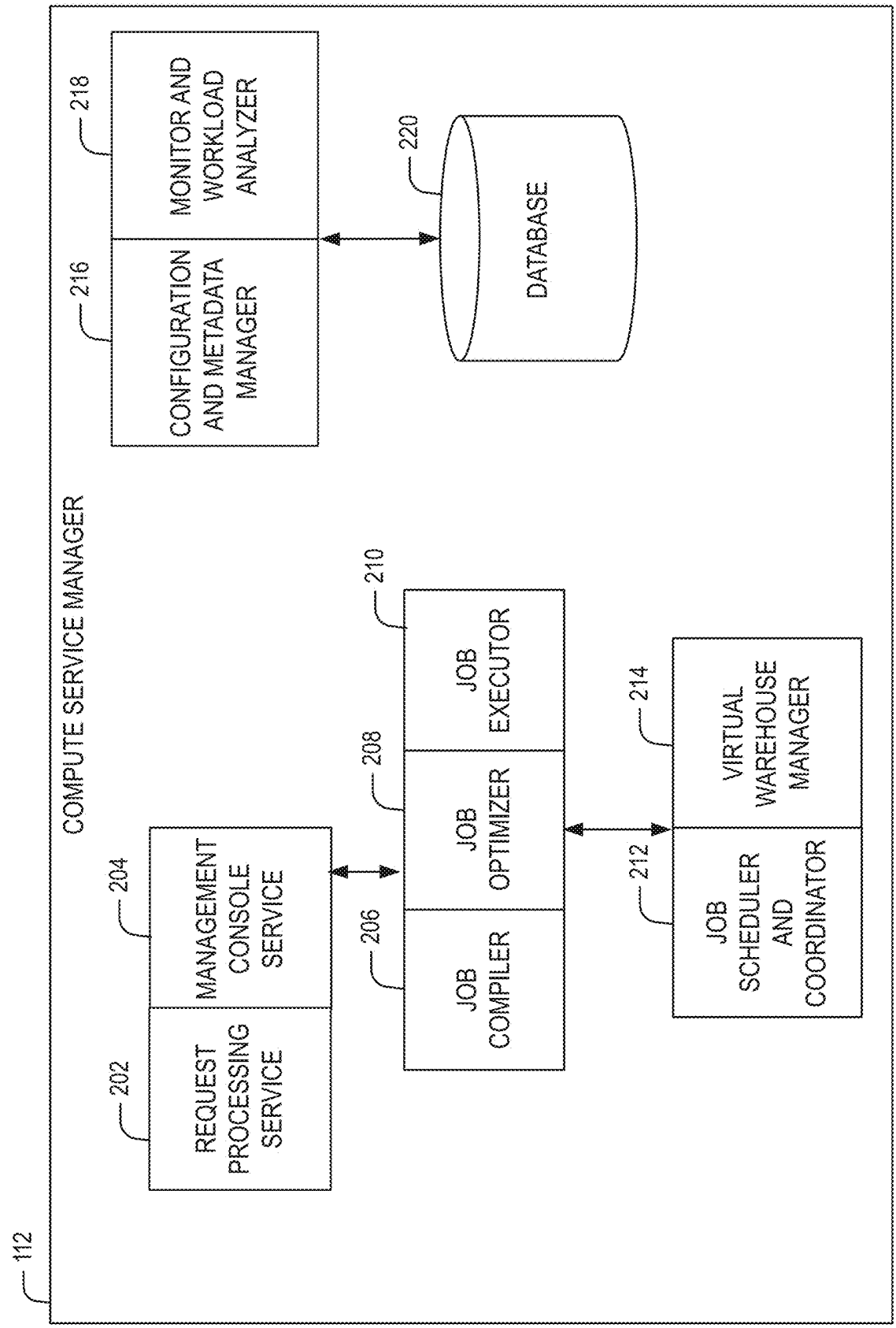

112
COMPUTE SERVICE MANAGER
202
REQUEST PROCESSING SERVICE
204
MANAGEMENT CONSOLE SERVICE
206
JOB COMPILER
208
JOB OPTIMIZER
210
JOB EXECUTOR
212
JOB SCHEDULER AND COORDINATOR
214
VIRTUAL WAREHOUSE MANAGER
216
CONFIGURATION AND METADATA MANAGER
218
MONITOR AND WORKLOAD ANALYZER
220
DATABASE

FIG. 2

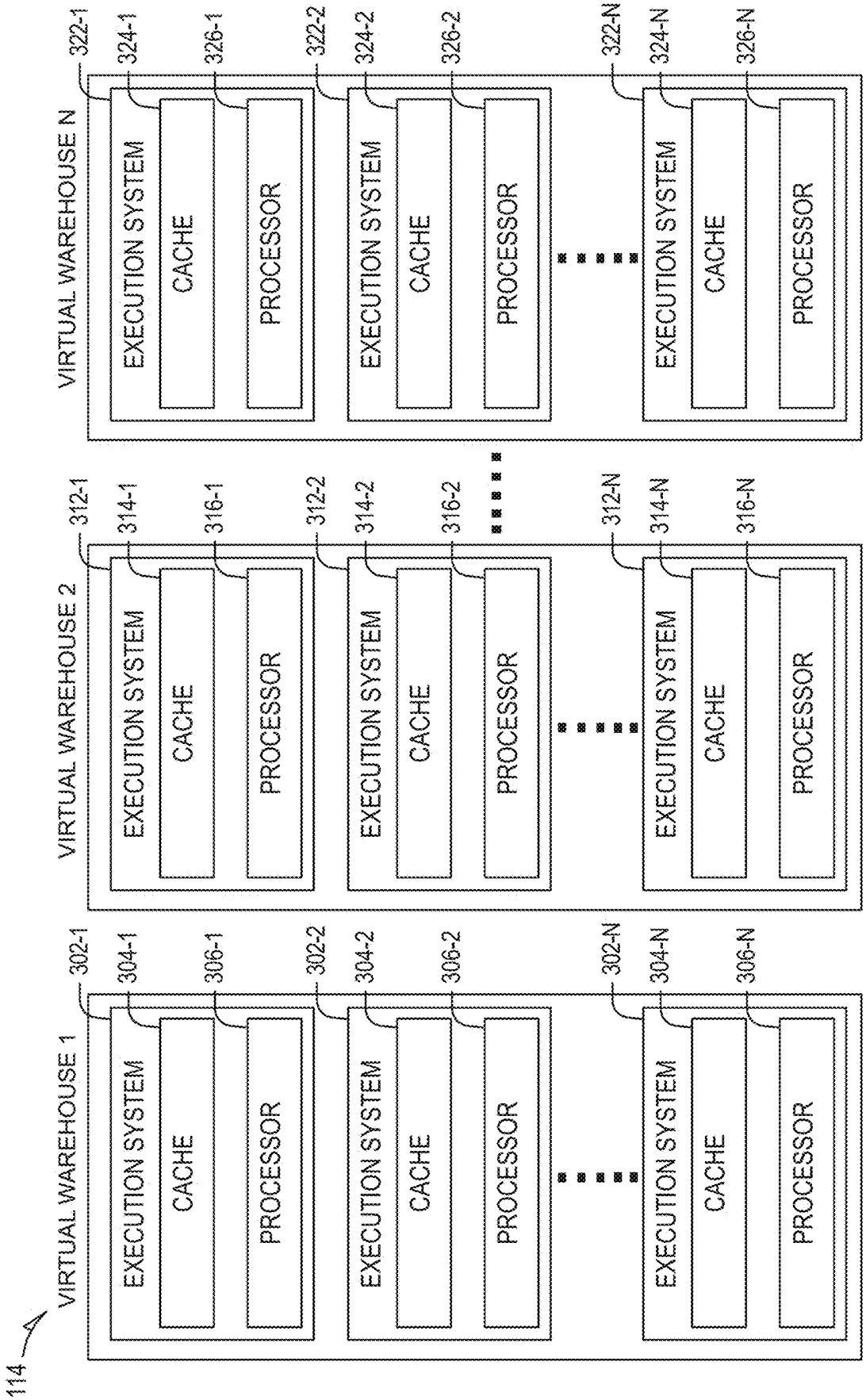

114
VIRTUAL WAREHOUSE 1
VIRTUAL WAREHOUSE 2
VIRTUAL WAREHOUSE N
EXECUTION SYSTEM
CACHE
PROCESSOR
302-1
304-1
306-1
302-2
304-2
306-2
302-N
304-N
306-N
312-1
314-1
316-1
312-2
314-2
316-2
312-N
314-N
316-N
322-1
324-1
326-1
322-2
324-2
326-2
322-N
324-N
326-N

FIG. 3

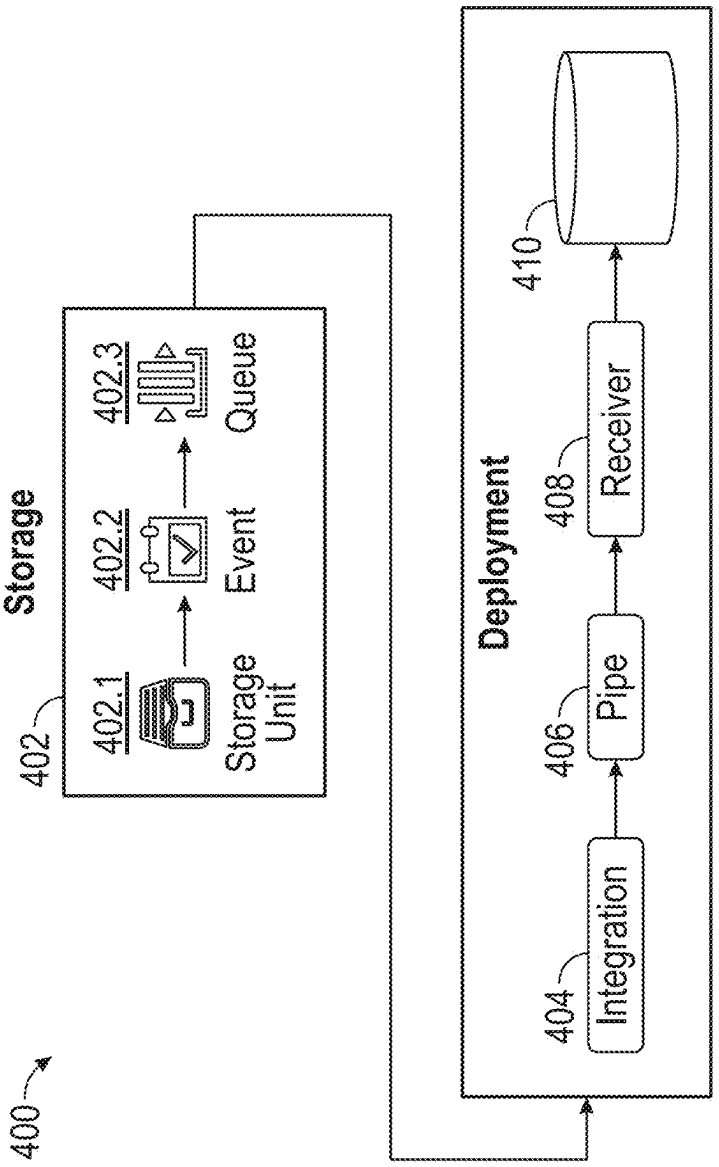

400
Storage
402
402.1
Storage Unit
402.2
Event
402.3
Queue
Deployment
404
Integration
406
Pipe
408
Receiver
410

FIG. 4

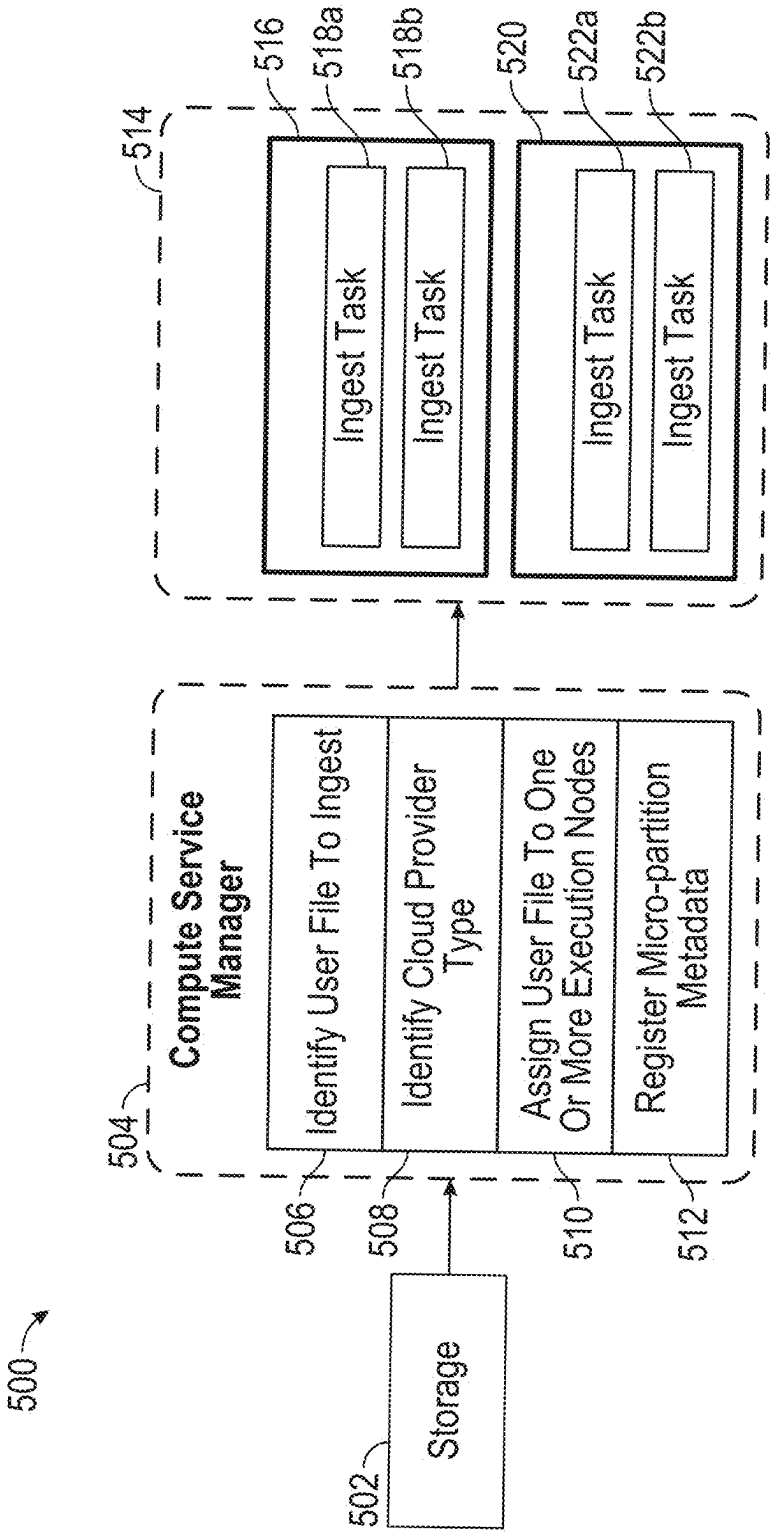

500
Storage
502
Compute Service Manager
504
Identify User File To Ingest
Identify Cloud Provider Type
Assign User File To One Or More Execution Nodes
Register Micro-partition Metadata
506
508
510
512
514
Ingest Task
Ingest Task
Ingest Task
Ingest Task
516
518a
518b
520
522a
522b

FIG. 5

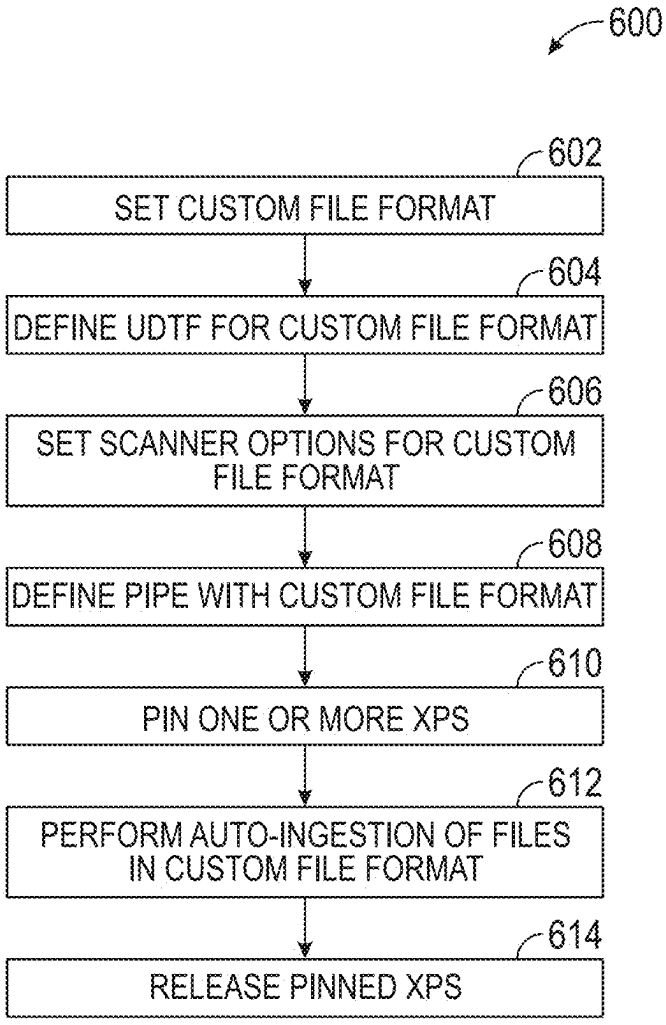

600
602
SET CUSTOM FILE FORMAT
604
DEFINE UDTF FOR CUSTOM FILE FORMAT
606
SET SCANNER OPTIONS FOR CUSTOM FILE FORMAT
608
DEFINE PIPE WITH CUSTOM FILE FORMAT
610
PIN ONE OR MORE XPS
612
PERFORM AUTO-INGESTION OF FILES IN CUSTOM FILE FORMAT
614
RELEASE PINNED XPS

FIG. 6

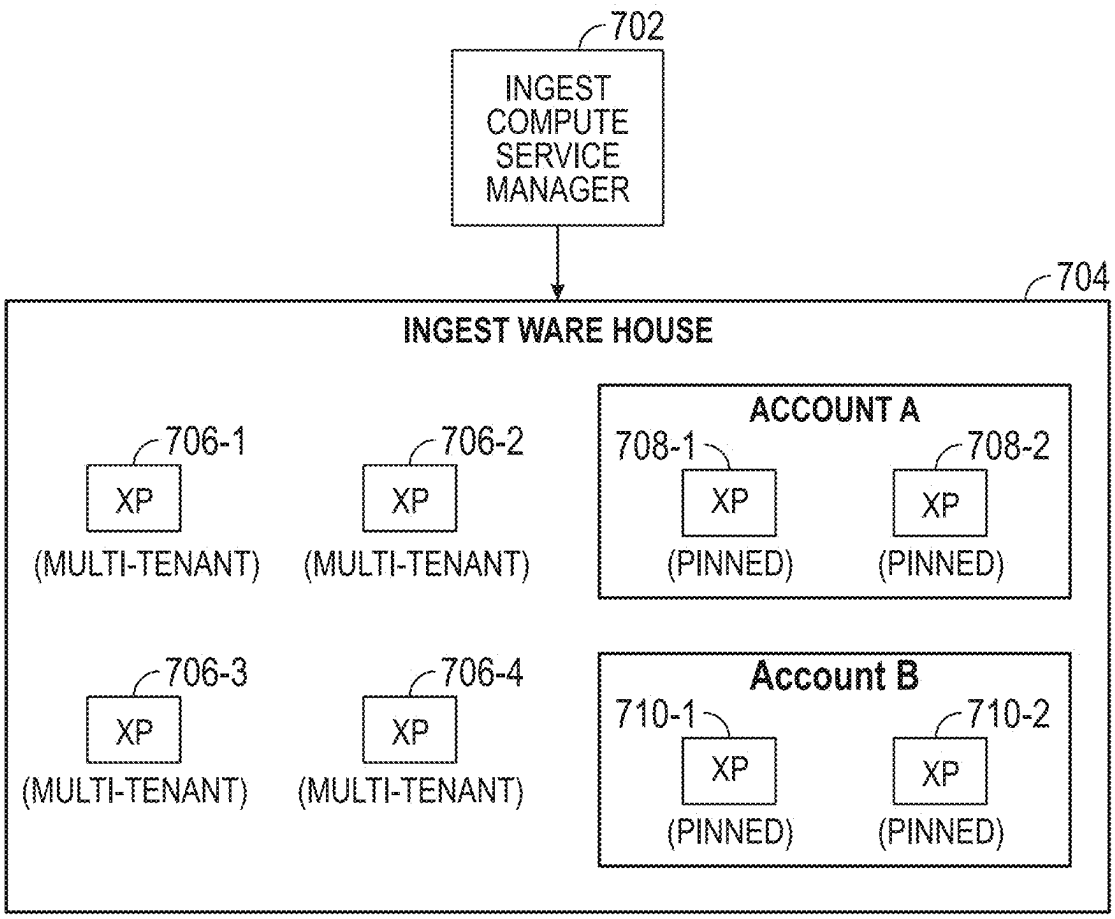

702
INGEST COMPUTE SERVICE MANAGER
704
INGEST WARE HOUSE
706-1
XP
(MULTI-TENANT)
706-2
XP
(MULTI-TENANT)
ACCOUNT A
708-1
XP
(PINNED)
708-2
XP
(PINNED)
706-3
XP
(MULTI-TENANT)
706-4
XP
(MULTI-TENANT)
Account B
710-1
XP
(PINNED)
710-2
XP
(PINNED)

FIG. 7

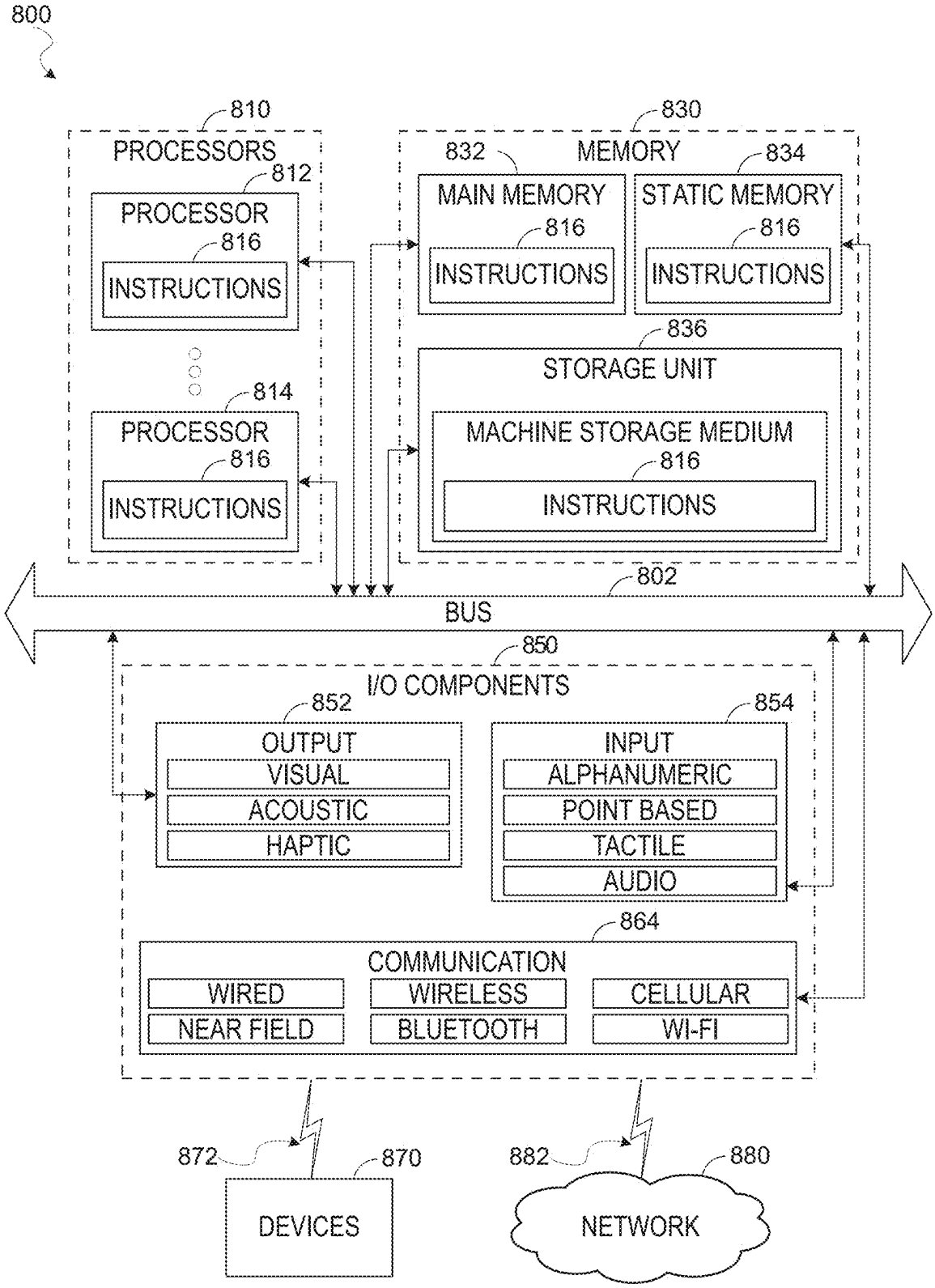

800
810
PROCESSORS
812
PROCESSOR
816
INSTRUCTIONS
814
PROCESSOR
816
INSTRUCTIONS
830
MEMORY
832
MAIN MEMORY
816
INSTRUCTIONS
834
STATIC MEMORY
816
INSTRUCTIONS
836
STORAGE UNIT
MACHINE STORAGE MEDIUM
816
INSTRUCTIONS
802
BUS
850
I/O COMPONENTS
852
OUTPUT
VISUAL
ACOUSTIC
HAPTIC
854
INPUT
ALPHANUMERIC
POINT BASED
TACTILE
AUDIO
864
COMMUNICATION
WIRED
WIRELESS
CELLULAR
NEAR FIELD
BLUETOOTH
WI-FI
872
870
DEVICES
882
880
NETWORK

FIG. 8

CONTINUOUS INGESTION OF CUSTOM FILE FORMATS

PRIORITY CLAIM

[0001] This application is a Continuation of U.S. patent application Ser. No. 18/050,122, filed Oct. 27, 2022, the content of which is hereby incorporated by reference in its entirety.

TECHNICAL FIELD

[0002] The present disclosure generally relates to data systems, such as data systems, and, more specifically, to data ingestion techniques for different file formats.

BACKGROUND

[0003] Data systems, such as database systems, may be provided through a cloud platform, which allows organizations and users to store, manage, and retrieve data from the cloud. A variety of techniques can be employed for uploading and storing data in a database or table in a cloud platform. Uploading techniques typically cannot account for different file formats.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] Various ones of the appended drawings merely illustrate example embodiments of the present disclosure and should not be considered as limiting its scope.

[0005] FIG. **1** illustrates an example computing environment with a cloud database system, according to some example embodiments.

[0006] FIG. **2** is a block diagram illustrating components of a compute service manager, according to some example embodiments.

[0007] FIG. **3** is a block diagram illustrating components of an execution platform, according to some example embodiments.

[0008] FIG. **4** is a simplified block diagram of a system for automated data ingestion, according to some example embodiments.

[0009] FIG. **5** is a schematic block diagram of a process of ingesting data into a database, according to some example embodiments.

[0010] FIG. **6** is a flow diagram of a method for auto-ingestion of custom file formats, according to some example embodiments.

[0011] FIG. **7** illustrates a data system framework for ingestion of files with custom file formats, according to some example embodiments.

[0012] FIG. **8** illustrates a diagrammatic representation of a machine in the form of a computer system within which a set of instructions may be executed for causing the machine to perform any one or more of the methodologies discussed herein, in accordance with some embodiments of the present disclosure.

DETAILED DESCRIPTION

[0013] The description that follows includes systems, methods, techniques, instruction sequences, and computing machine program products that embody illustrative embodiments of the disclosure. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide an understanding of various embodiments of the inventive subject matter. It will be evident, however, to those skilled in the art, that embodiments of the inventive subject matter may be practiced without these specific details. In general, well-known instruction instances, protocols, structures, and techniques are not necessarily shown in detail.

[0014] Techniques for continuous ingestion of files using custom file formats are described. A custom file format may include formats not natively supported by a data system. Unstructured files (e.g., images) may also be considered custom file formats.

[0015] FIG. **1** illustrates an example shared data processing platform **100**. To avoid obscuring the inventive subject matter with unnecessary detail, various functional components that are not germane to conveying an understanding of the inventive subject matter have been omitted from the figures. However, a skilled artisan will readily recognize that various additional functional components may be included as part of the shared data processing platform **100** to facilitate additional functionality that is not specifically described herein.

[0016] As shown, the shared data processing platform **100** comprises the network-based database system **102**, a cloud computing storage platform **104** (e.g., a storage platform, an AWS® service, Microsoft Azure®, or Google Cloud Services®), and a remote computing device **106**. The network-based database system **102** is a cloud database system used for storing and accessing data (e.g., internally storing data, accessing external remotely located data) in an integrated manner, and reporting and analysis of the integrated data from the one or more disparate sources (e.g., the cloud computing storage platform **104**). The cloud computing storage platform **104** comprises a plurality of computing machines and provides on-demand computer system resources such as data storage and computing power to the network-based database system **102**. While in the embodiment illustrated in FIG. **1**, a data warehouse is depicted, other embodiments may include other types of databases or other data processing systems.

[0017] The remote computing device **106** (e.g., a user device such as a laptop computer) comprises one or more computing machines (e.g., a user device such as a laptop computer) that execute a remote software component **108** (e.g., browser accessed cloud service) to provide additional functionality to users of the network-based database system **102**. The remote software component **108** comprises a set of machine-readable instructions (e.g., code) that, when executed by the remote computing device **106**, cause the remote computing device **106** to provide certain functionality. The remote software component **108** may operate on input data and generates result data based on processing, analyzing, or otherwise transforming the input data. As an example, the remote software component **108** can be a data provider or data consumer that enables database tracking procedures.

[0018] The network-based database system **102** comprises an access management system **110**, a compute service manager **112**, an execution platform **114**, and a database **116**. The access management system **110** enables administrative users to manage access to resources and services provided by the network-based database system **102**. Administrative users can create and manage users, roles, and groups, and use permissions to allow or deny access to resources and services. The access management system **110** can store shared

data that securely manages shared access to the storage resources of the cloud computing storage platform **104** amongst different users of the network-based database system **102**, as discussed in further detail below.

[0019] The compute service manager **112** coordinates and manages operations of the network-based database system **102**. The compute service manager **112** also performs query optimization and compilation as well as managing clusters of computing services that provide compute resources (e.g., virtual warehouses, virtual machines, EC2 clusters). The compute service manager **112** can support any number of client accounts such as end users providing data storage and retrieval requests, system administrators managing the systems and methods described herein, and other components/devices that interact with compute service manager **112**.

[0020] The compute service manager **112** is also coupled to database **116**, which is associated with the entirety of data stored on the shared data processing platform **100**. The database **116** stores data pertaining to various functions and aspects associated with the network-based database system **102** and its users.

[0021] In some embodiments, database **116** includes a summary of data stored in remote data storage systems as well as data available from one or more local caches. Additionally, database **116** may include information regarding how data is organized in the remote data storage systems and the local caches. Database **116** allows systems and services to determine whether a piece of data needs to be accessed without loading or accessing the actual data from a storage device. The compute service manager **112** is further coupled to an execution platform **114**, which provides multiple computing resources (e.g., virtual warehouses) that execute various data storage and data retrieval, as discussed in greater detail below.

[0022] Execution platform **114** is coupled to multiple data storage devices **124-1** to **124**-N that are part of a cloud computing storage platform **104**. In some embodiments, data storage devices **124-1** to **124**-N are cloud-based storage devices located in one or more geographic locations. For example, data storage devices **124-1** to **124**-N may be part of a public cloud infrastructure or a private cloud infrastructure. Data storage devices **124-1** to **124**-N may be hard disk drives (HDDs), solid state drives (SSDs), storage clusters, Amazon S3 storage systems or any other data storage technology. Additionally, cloud computing storage platform **104** may include distributed file systems (such as Hadoop Distributed File Systems (HDFS)), object storage systems, and the like.

[0023] The execution platform **114** comprises a plurality of compute nodes (e.g., virtual warehouses). A set of processes on a compute node executes a query plan compiled by the compute service manager **112**. The set of processes can include: a first process to execute the query plan; a second process to monitor and delete micro-partition files using a least recently used (LRU) policy, and implement an out of memory (OOM) error mitigation process; a third process that extracts health information from process logs and status information to send back to the compute service manager **112**; a fourth process to establish communication with the compute service manager **112** after a system boot; and a fifth process to handle all communication with a compute cluster for a given job provided by the compute service manager **112** and to communicate information back to the compute service manager **112** and other compute nodes of the execution platform **114**.

[0024] The cloud computing storage platform **104** also comprises an access management system **118** and a web proxy **120**. As with the access management system **110**, the access management system **118** allows users to create and manage users, roles, and groups, and use permissions to allow or deny access to cloud services and resources. The access management system **110** of the network-based database system **102** and the access management system **118** of the cloud computing storage platform **104** can communicate and share information so as to enable access and management of resources and services shared by users of both the network-based database system **102** and the cloud computing storage platform **104**. The web proxy **120** handles tasks involved in accepting and processing concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management. The web proxy **120** provides HTTP proxy service for creating, publishing, maintaining, securing, and monitoring APIs (e.g., REST APIs).

[0025] In some embodiments, communication links between elements of the shared data processing platform **100** are implemented via one or more data communication networks. These data communication networks may utilize any communication protocol and any type of communication medium. In some embodiments, the data communication networks are a combination of two or more data communication networks (or sub-Networks) coupled to one another. In alternative embodiments, these communication links are implemented using any type of communication medium and any communication protocol.

[0026] As shown in FIG. **1**, data storage devices **124-1** to **124**-N are decoupled from the computing resources associated with the execution platform **114**. That is, new virtual warehouses can be created and terminated in the execution platform **114** and additional data storage devices can be created and terminated on the cloud computing storage platform **104** in an independent manner. This architecture supports dynamic changes to the network-based database system **102** based on the changing data storage/retrieval needs as well as the changing needs of the users and systems accessing the shared data processing platform **100**. The support of dynamic changes allows network-based database system **102** to scale quickly in response to changing demands on the systems and components within network-based database system **102**. The decoupling of the computing resources from the data storage devices **124-1** to **124**-N supports the storage of large amounts of data without requiring a corresponding large amount of computing resources. Similarly, this decoupling of resources supports a significant increase in the computing resources utilized at a particular time without requiring a corresponding increase in the available data storage resources. Additionally, the decoupling of resources enables different accounts to handle creating additional compute resources to process data shared by other users without affecting the other users' systems. For instance, a data provider may have three compute resources and share data with a data consumer, and the data consumer may generate new compute resources to execute queries against the shared data, where the new compute resources are managed by the data consumer and do not affect or interact with the compute resources of the data provider.

[0027] Compute service manager **112**, database **116**, execution platform **114**, cloud computing storage platform **104**, and remote computing device **106** are shown in FIG. **1** as individual components. However, each of compute service manager **112**, database **116**, execution platform **114**, cloud computing storage platform **104**, and remote computing environment may be implemented as a distributed system (e.g., distributed across multiple systems/platforms at multiple geographic locations) connected by APIs and access information (e.g., tokens, login data). Additionally, each of compute service manager **112**, database **116**, execution platform **114**, and cloud computing storage platform **104** can be scaled up or down (independently of one another) depending on changes to the requests received and the changing needs of shared data processing platform **100**. Thus, in the described embodiments, the network-based database system **102** is dynamic and supports regular changes to meet the current data processing needs.

[0028] During typical operation, the network-based database system **102** processes multiple jobs (e.g., queries) determined by the compute service manager **112**. These jobs are scheduled and managed by the compute service manager **112** to determine when and how to execute the job. For example, the compute service manager **112** may divide the job into multiple discrete tasks and may determine what data is needed to execute each of the multiple discrete tasks. The compute service manager **112** may assign each of the multiple discrete tasks to one or more nodes of the execution platform **114** to process the task. The compute service manager **112** may determine what data is needed to process a task and further determine which nodes within the execution platform **114** are best suited to process the task. Some nodes may have already cached the data needed to process the task (due to the nodes having recently downloaded the data from the cloud computing storage platform **104** for a previous job) and, therefore, be a good candidate for processing the task. Metadata stored in the database **116** assists the compute service manager **112** in determining which nodes in the execution platform **114** have already cached at least a portion of the data needed to process the task. One or more nodes in the execution platform **114** process the task using data cached by the nodes and, if necessary, data retrieved from the cloud computing storage platform **104**. It is desirable to retrieve as much data as possible from caches within the execution platform **114** because the retrieval speed is typically much faster than retrieving data from the cloud computing storage platform **104**.

[0029] As shown in FIG. **1**, the shared data processing platform **100** separates the execution platform **114** from the cloud computing storage platform **104**. In this arrangement, the processing resources and cache resources in the execution platform **114** operate independently of the data storage devices **124-1** to **124**-N in the cloud computing storage platform **104**. Thus, the computing resources and cache resources are not restricted to specific data storage devices **124-1** to **124**-N. Instead, all computing resources and all cache resources may retrieve data from, and store data to, any of the data storage resources in the cloud computing storage platform **104**.

[0030] FIG. **2** is a block diagram illustrating components of the compute service manager **112**, in accordance with some embodiments of the present disclosure. As shown in FIG. **2**, a request processing service **202** manages received data storage requests and data retrieval requests (e.g., jobs to be performed on database data). For example, the request processing service **202** may determine the data necessary to process a received query (e.g., a data storage request or data retrieval request). The data may be stored in a cache within the execution platform **114** or in a data storage device in cloud computing storage platform **104**. A management console service **204** supports access to various systems and processes by administrators and other system managers. Additionally, the management console service **204** may receive a request to execute a job and monitor the workload on the system.

[0031] The compute service manager **112** also includes a job compiler **206**, a job optimizer **208**, and a job executor **210**. The job compiler **206** parses a job into multiple discrete tasks and generates the execution code for each of the multiple discrete tasks. The job optimizer **208** determines the best method to execute the multiple discrete tasks based on the data that needs to be processed. The job optimizer **208** also handles various data pruning operations and other data optimization techniques to improve the speed and efficiency of executing the job. The job executor **210** executes the execution code for jobs received from a queue or determined by the compute service manager **112**.

[0032] A job scheduler and coordinator **212** sends received jobs to the appropriate services or systems for compilation, optimization, and dispatch to the execution platform **114**. For example, jobs may be prioritized and processed in that prioritized order. In an embodiment, the job scheduler and coordinator **212** determines a priority for internal jobs that are scheduled by the compute service manager **112** with other "outside" jobs such as user queries that may be scheduled by other systems in the database but may utilize the same processing resources in the execution platform **114**. In some embodiments, the job scheduler and coordinator **212** identifies or assigns particular nodes in the execution platform **114** to process particular tasks. A virtual warehouse manager **214** manages the operation of multiple virtual warehouses implemented in the execution platform **114**. As discussed below, each virtual warehouse includes multiple execution nodes that each include a cache and a processor (e.g., a virtual machine, an operating system level container execution environment).

[0033] Additionally, the compute service manager **112** includes a configuration and metadata manager **216**, which manages the information related to the data stored in the remote data storage devices and in the local caches (i.e., the caches in execution platform **114**). The configuration and metadata manager **216** uses the metadata to determine which data micro-partitions need to be accessed to retrieve data for processing a particular task or job. A monitor and workload analyzer **218** oversees processes performed by the compute service manager **112** and manages the distribution of tasks (e.g., workload) across the virtual warehouses and execution nodes in the execution platform **114**. The monitor and workload analyzer **218** also redistributes tasks, as needed, based on changing workloads throughout the network-based database system **102** and may further redistribute tasks based on a user (e.g., "external") query workload that may also be processed by the execution platform **114**. The configuration and metadata manager **216** and the monitor and workload analyzer **218** are coupled to a data storage device **220**. Data storage device **220** in FIG. **2** represent any data storage device within the network-based database system **102**. For example, data storage device **220** may repre-

sent caches in execution platform **114**, storage devices in cloud computing storage platform **104**, or any other storage device.

[0034] FIG. **3** is a block diagram illustrating components of the execution platform **114**, in accordance with some embodiments of the present disclosure. As shown in FIG. **3**, execution platform **114** includes multiple virtual warehouses, which are elastic clusters of compute instances, such as virtual machines. In the example illustrated, the virtual warehouses include virtual warehouse 1, virtual warehouse 2, and virtual warehouse n. Each virtual warehouse (e.g., EC2 cluster) includes multiple execution nodes (e.g., virtual machines) that each include a data cache and a processor. The virtual warehouses can execute multiple tasks in parallel by using the multiple execution nodes. As discussed herein, execution platform **114** can add new virtual warehouses and drop existing virtual warehouses in real time based on the current processing needs of the systems and users. This flexibility allows the execution platform **114** to quickly deploy large amounts of computing resources when needed without being forced to continue paying for those computing resources when they are no longer needed. All virtual warehouses can access data from any data storage device (e.g., any storage device in cloud computing storage platform **104**).

[0035] Although each virtual warehouse shown in FIG. **3** includes three execution nodes, a particular virtual warehouse may include any number of execution nodes. Further, the number of execution nodes in a virtual warehouse is dynamic, such that new execution nodes are created when additional demand is present, and existing execution nodes are deleted when they are no longer necessary (e.g., upon a query or job completion).

[0036] Each virtual warehouse is capable of accessing any of the data storage devices **124-1** to **124**-N shown in FIG. **1**. Thus, the virtual warehouses are not necessarily assigned to a specific data storage device **124-1** to **124**-N and, instead, can access data from any of the data storage devices **124-1** to **124**-N within the cloud computing storage platform **104**. Similarly, each of the execution nodes shown in FIG. **3** can access data from any of the data storage devices **124-1** to **124**-N. For instance, the storage device **124-1** of a first user (e.g., provider account user) may be shared with a worker node in a virtual warehouse of another user (e.g., consumer account user), such that the other user can create a database (e.g., read-only database) and use the data in storage device **124-1** directly without needing to copy the data (e.g., copy it to a new disk managed by the consumer account user). In some embodiments, a particular virtual warehouse or a particular execution node may be temporarily assigned to a specific data storage device, but the virtual warehouse or execution node may later access data from any other data storage device.

[0037] In the example of FIG. **3**, virtual warehouse 1 includes three execution nodes **302-1**, **302-2**, and **302**-N. Execution node **302-1** includes a cache **304-1** and a processor **306-1**. Execution node **302-2** includes a cache **304-2** and a processor **306-2**. Execution node **302**-N includes a cache **304**-N and a processor **306**-N. Each execution node **302-1**, **302-2**, and **302**-N is associated with processing one or more data storage and/or data retrieval tasks. For example, a virtual warehouse may handle data storage and data retrieval tasks associated with an internal service, such as a clustering service, a materialized view refresh service, a file compaction service, a storage procedure service, or a file upgrade service. In other implementations, a particular virtual warehouse may handle data storage and data retrieval tasks associated with a particular data storage system or a particular category of data.

[0038] Similar to virtual warehouse 1 discussed above, virtual warehouse 2 includes three execution nodes **312-1**, **312-2**, and **312**-N. Execution node **312-1** includes a cache **314-1** and a processor **316-1**. Execution node **312-2** includes a cache **314-2** and a processor **316-2**. Execution node **312**-N includes a cache **314**-N and a processor **316**-N. Additionally, virtual warehouse 3 includes three execution nodes **322-1**, **322-2**, and **322**-N. Execution node **322-1** includes a cache **324-1** and a processor **326-1**. Execution node **322-2** includes a cache **324-2** and a processor **326-2**. Execution node **322**-N includes a cache **324**-N and a processor **326**-N.

[0039] In some embodiments, the execution nodes shown in FIG. **3** are stateless with respect to the data the execution nodes are caching. For example, these execution nodes do not store or otherwise maintain state information about the execution node, or the data being cached by a particular execution node. Thus, in the event of an execution node failure, the failed node can be transparently replaced by another node. Since there is no state information associated with the failed execution node, the new (replacement) execution node can easily replace the failed node without concern for recreating a particular state.

[0040] Although the execution nodes shown in FIG. **3** each include one data cache and one processor, alternative embodiments may include execution nodes containing any number of processors and any number of caches. Additionally, the caches may vary in size among the different execution nodes. The caches shown in FIG. **3** store, in the local execution node (e.g., local disk), data that was retrieved from one or more data storage devices in cloud computing storage platform **104** (e.g., S3 objects recently accessed by the given node). In some example embodiments, the cache stores file headers and individual columns of files as a query downloads only columns necessary for that query.

[0041] To improve cache hits and avoid overlapping redundant data stored in the node caches, the job optimizer **208** assigns input file sets to the nodes using a consistent hashing scheme to hash over table file names of the data accessed (e.g., data in database **116** or database **122**). Subsequent or concurrent queries accessing the same table file will therefore be performed on the same node, according to some example embodiments.

[0042] As discussed, the nodes and virtual warehouses may change dynamically in response to environmental conditions (e.g., disaster scenarios), hardware/software issues (e.g., malfunctions), or administrative changes (e.g., changing from a large cluster to smaller cluster to lower costs). In some example embodiments, when the set of nodes changes, no data is reshuffled immediately. Instead, the least recently used replacement policy is implemented to eventually replace the lost cache contents over multiple jobs. Thus, the caches reduce or eliminate the bottleneck problems occurring in platforms that consistently retrieve data from remote storage systems. Instead of repeatedly accessing data from the remote storage devices, the systems and methods described herein access data from the caches in the execution nodes, which is significantly faster and avoids the bottleneck problem discussed above. In some embodiments,

the caches are implemented using high-speed memory devices that provide fast access to the cached data. Each cache can store data from any of the storage devices in the cloud computing storage platform **104**.

[0043] Further, the cache resources and computing resources may vary between different execution nodes. For example, one execution node may contain significant computing resources and minimal cache resources, making the execution node useful for tasks that require significant computing resources. Another execution node may contain significant cache resources and minimal computing resources, making this execution node useful for tasks that require caching of large amounts of data. Yet another execution node may contain cache resources providing faster input-output operations, useful for tasks that require fast scanning of large amounts of data. In some embodiments, the execution platform **114** implements skew handling to distribute work amongst the cache resources and computing resources associated with a particular execution, where the distribution may be further based on the expected tasks to be performed by the execution nodes. For example, an execution node may be assigned more processing resources if the tasks performed by the execution node become more processor-intensive. Similarly, an execution node may be assigned more cache resources if the tasks performed by the execution node require a larger cache capacity. Further, some nodes may be executing much slower than others due to various issues (e.g., virtualization issues, network overhead). In some example embodiments, the imbalances are addressed at the scan level using a file stealing scheme. In particular, whenever a node process completes scanning its set of input files, it requests additional files from other nodes. If the one of the other nodes receives such a request, the node analyzes its own set (e.g., how many files are left in the input file set when the request is received), and then transfers ownership of one or more of the remaining files for the duration of the current job (e.g., query). The requesting node (e.g., the file stealing node) then receives the data (e.g., header data) and downloads the files from the cloud computing storage platform **104** (e.g., from data storage device **124-1**), and does not download the files from the transferring node. In this way, lagging nodes can transfer files via file stealing in a way that does not worsen the load on the lagging nodes.

[0044] Although virtual warehouses 1, 2, and n are associated with the same execution platform **114**, the virtual warehouses may be implemented using multiple computing systems at multiple geographic locations. For example, virtual warehouse 1 can be implemented by a computing system at a first geographic location, while virtual warehouses 2 and n are implemented by another computing system at a second geographic location. In some embodiments, these different computing systems are cloud-based computing systems maintained by one or more different entities.

[0045] Additionally, each virtual warehouse is shown in FIG. **3** as having multiple execution nodes. The multiple execution nodes associated with each virtual warehouse may be implemented using multiple computing systems at multiple geographic locations. For example, an instance of virtual warehouse 1 implements execution nodes **302-1** and **302-2** on one computing platform at a geographic location and implements execution node **302-N** at a different computing platform at another geographic location. Selecting particular computing systems to implement an execution node may depend on various factors, such as the level of resources needed for a particular execution node (e.g., processing resource requirements and cache requirements), the resources available at particular computing systems, communication capabilities of networks within a geographic location or between geographic locations, and which computing systems are already implementing other execution nodes in the virtual warehouse.

[0046] Execution platform **114** is also fault tolerant. For example, if one virtual warehouse fails, that virtual warehouse is quickly replaced with a different virtual warehouse at a different geographic location.

[0047] A particular execution platform **114** may include any number of virtual warehouses. Additionally, the number of virtual warehouses in a particular execution platform is dynamic, such that new virtual warehouses are created when additional processing and/or caching resources are needed. Similarly, existing virtual warehouses may be deleted when the resources associated with the virtual warehouse are no longer necessary.

[0048] In some embodiments, the virtual warehouses may operate on the same data in cloud computing storage platform **104**, but each virtual warehouse has its own execution nodes with independent processing and caching resources. This configuration allows requests on different virtual warehouses to be processed independently and with no interference between the requests. This independent processing, combined with the ability to dynamically add and remove virtual warehouses, supports the addition of new processing capacity for new users without impacting the performance observed by the existing users.

[0049] As mentioned above, data from a client storage can be uploaded to the data warehouse. Some techniques can use a "copy" command for this transfer. The "copy" command is typically manually performed or performed based on a set schedule (say, every 15 minutes). However, the use of such "copy" commands can add latency.

[0050] Consequently, latency can be improved by implementing auto-ingestion techniques, as described in further detail below. FIG. **4** is a simplified block diagram of system **400** for automated data ingestion, according to some example embodiments. The system may include a storage **402**, which may be provided as cloud storage (e.g., Amazon S3 storage, Azure storage, GCP storage, etc.). The storage **402** may include client data to upload to the data warehouse.

[0051] The storage **402** may store files (or data) to be ingested into a database **410**. In some embodiments, the storage **402** may include a storage unit **402.1**, an event block **402.2**, and a queue **402.3**. The system may also include a deployment to ingest data in the database **410**. A deployment may include multiple components such as a metadata store/DB, a front-end layer, a load balancing layer, a data warehouse, etc., as discussed above with respect to FIGS. **1-3**. The deployments may be provided as public or private deployments. A public deployment may be implemented as a multi-tenant environment, where each tenant or account shares processing and/or storage resources. For example, in a public deployment, multiple accounts may share a metadata store, a front-end layer, a load balancing layer, a data warehouse, etc. A private deployment, on the other hand, may be implemented as a dedicated, isolated environment, where processing and/or storage resources may be dedicated.

[0052] The deployment may be communicatively coupled to the queue **402.3**, and may include an integration **404**, a pipe **406**, and a receiver **408**. Integration **404** may be configured to receive a notification when new data becomes available in queue **402.3**. For example, the queue may include a pool of Simple Queue Service™ (SQS) queues as part of an Amazon Web Services™ S3 bucket. The pool of SQS queues may be provided to client accounts to add user files to a bucket. A notification may be automatically generated when one or more user files are added to a client account data bucket. A plurality of customer data buckets may be provided for each client account. The automatically generated notification may be received by the integration **404**.

[0053] For example, the integration **404** may provide information relating to an occurrence of an event in the queue **402.3**. Events may include creation of new data, update of old data, and deletion of old data. The integration **404** may also provide identification information for a resource associated with the event, e.g., the user file that has been created, updated, or deleted. The integration **404** may communicate with the queue **402.3** because the integration **404** may be provided with credentials for the queue **402.3**, for example by an administrator and/or user. In an embodiment, the integration **404** may poll the queue **402.3** for notifications.

[0054] The integration **404** may deliver the notification to the pipe **406**, which may be provided as a single pipe or multiple pipes. The pipe **406** may store information relating to what data and the location of the data for automatic data ingestion related to the queue **402.3**.

[0055] The receiver **408** may perform the automated data ingestion, and then store the ingested data in the database **410**. Data ingestion may be performed using the techniques described in U.S. patent application Ser. No. 16/201,854, entitled "Batch Data Ingestion in Database Systems," filed on Nov. 27, 2018, which is incorporated herein by reference in its entirety, including but not limited to those portions that specifically appear hereinafter, the incorporation by reference being made with the following exception: In the event that any portion of the above-referenced application is inconsistent with this application, this application supersedes the above-referenced application.

[0056] FIG. **5** is a schematic block diagram of a process **500** of ingesting data into a database, according to some example embodiments. The process **500** begins and a storage **502** sends an ingest request, such as a notification. The storage **502** may directly or indirectly communicate with the database system to send in the ingest request. In some embodiments, the ingest request is a notification provided by a third-party vendor storage account, or the ingest request may arise from a compute service manager polling a data lake associated with the client account to determine whether any user files have been added to the client account that have not yet been ingested into the database. The notification includes a list of files to insert into a table of the database. The files are persisted in a queue specific to the receiving table of the database.

[0057] The ingest request is received by a compute service manager **504**. The compute service manager **504** identifies at step **506** a user file to ingest. At step **508**, the compute service manager identifies a cloud provider type associated with the client account. At step **510**, the compute service manager **504** may assign the user file to one or more execution nodes, based at least in part on the detected cloud provider type, and registers at step **512** micro-partition metadata associated with a database table after the file is ingested into a micro-partition of the database table. The compute service manager **504** provisions one or more execution nodes **516**, **520** of an execution platform **514** to perform one or more tasks associated with ingesting the user file. Such ingest tasks **518***a*, **518***b*, **522***a*, **522***b* include, for example, cutting a file into one or more sections, generating a new micro-partition based on the user file, and/or inserting the new micro-partition in a table of the database.

[0058] The process **500** begins an ingest task that is executed by a warehouse. The ingest task may pull user files from the queue for a database table until it is told to stop doing so. The ingest task may periodically cut a new user file and add it to the database table. In one embodiment, the ingest process is "serverless" in that it is an integrated service provided by the database or compute service manager **504**. That is, a user associated with the client account need not provision its own warehouse or a third-party warehouse in order to perform the ingestion process. For example, the database or database provided (e.g., via instances of the compute service manager **504**) may maintain the ingest warehouse that then services one or more or all accounts/customers of the database provider.

[0059] In some embodiments, there may be more than one ingest task pulling from a queue for a given table, and this might be necessary to keep up with the rate of incoming data. In some embodiments, the ingest task may decide the time to cut a new file to increase the chances of getting an ideal sized file and avoid "odd sized" files that would result if the file size was lined up with one or more user files. This may come at the cost of added complexity as the track line number of the files consumed must be tracked.

[0060] Users may have files in custom file formats that may not be amenable to ingestion. For example, a data system may support ingestion of native formats, such csv, json, avro, parquet, orc, xml. However, users may have data in other formats. For example, users may have data in structured file formats, such as syslogs, HL7 messages, VCF, EBDIC, KDF files, dicom images, and complex file formats, such as images, pdf, video, audio, etc., that are not typically supported by data systems. Custom file formats may include formats not natively supported by the data system. Unstructured files (e.g., images) may also be considered custom file formats. Some data systems may have workarounds for custom file formats such as using tasks that have INSERT statements to insert data in the custom file format into a table. However, this workaround suffers from issues. Tasks typically do not have overlapping executions so work will be queued until the previous execution is completed, which adds to latency. Scaling to large volume of continuous events cannot be performed.

[0061] Techniques for continuous ingestion of custom file formats are described below. FIG. **6** is a flow diagram of a method **600** for auto-ingestion of custom file formats, according to some example embodiments. Method **600** may be performed using the auto ingestion techniques described above with reference to FIGS. **4** and **5**. That is, a compute service manager, as described herein (e.g., compute service manager **504**), may receive notifications of files in a custom file format to be ingested and create a query plan for ingesting those files. The compute service manager may

assign ingest tasks to execution nodes of one or more execution platforms (XPs) as described above.

[0062] At operation **602**, a custom file format may be created for the data system. For example, a "file format" command, which is used to set natively supported file formats, can be extended to include a "custom" option (e.g., "TYPE=CUSTOM").

[0063] At operation **604**, a user defined table function (UDTF, also referred to as user defined function (UDF)) for the custom file format may be defined. The UDTF can be used to parse the custom file format and return a table. For example, the UDTF can return a table with one or more rows and a variant column. The UDTF may function as a scanner for auto-ingestion, as described herein. The UDTF may include a first argument, which is related to the file to be ingested, and a second argument, which is related to scanner options. The first argument may include a reference to a stage from which the file to be ingested can be read, such as a stage file URL or a scoped URL. The stage may be a cloud storage location (e.g., S3 storage system). From the stage, a stream of the file to be ingested may be obtained.

[0064] The UDTF can parse the stream using parsing code. The parsing code may read bytes from the stream of the file to be ingested. The parsing code may be written in a different programming language, such as java, python, etc., that what is used by the data system. The parsing code may be specific to the custom file format (e.g., HL7). In some examples, the parsing code can use third party libraries (e.g., File Reference) or users can write their own code. The parsing code may decrypt the file and decompress it, if needed. The parsing code returns a stream of row set with at least one variant column.

[0065] The UDTF may be provided in a different programming language (e.g, .java, python) that what is used in the data system. The UDTF therefore may be treated as untrusted code by the data system and the UDTF may be executed in a sandbox. In computer security, a sandbox (e.g., sandbox environment) is a security mechanism for separating running programs, usually to prevent system failures or prevent exploitation of software vulnerabilities. A sandbox can be used to execute untested or untrusted programs or code, possibly from unverified or untrusted third parties, suppliers, users or websites, without risking harm to the host machine or operating system. A sandbox can provide a tightly controlled set of resources for guest programs to run in, such as storage and memory scratch space. Network access, the ability to inspect the host system or read from input devices can be disallowed or restricted. UDTFs typically can run in a sandbox environment.

[0066] A sandbox process, in an example, is a program that reduces the risk of security breaches by restricting the running environment of untrusted applications using security mechanisms such as namespaces and secure computing modes (e.g., using a system call filter to an executing process and all its descendants, thus reducing the attack surface of the kernel of a given operating system). Moreover, in an example, the sandbox process is a lightweight process in comparison to an execution node process and is optimized (e.g., closely coupled to security mechanisms of a given operating system kernel) to process a database query in a secure manner within the sandbox environment. In some embodiments, the UDTF is executed using UDTF server, which is restricted from accessing certain files an file systems. For example, the UDTF server and a worker process handing other operations for the continuous auto-ingestion may be provided as different processors on the same machine.

[0067] In some embodiments, the sandbox process can utilize a virtual network connection in order to communicate with other components within the subject system. A specific set of rules can be configured for the virtual network connection with respect to other components of the subject system. For example, such rules for the virtual network connection can be configured for a particular UDTF to restrict the locations (e.g., particular sites on the Internet or components that the UDTF can communicate) that are accessible by operations performed by the UDTF. Thus, in this example, the UDTF can be denied access to particular network locations or sites on the Internet.

[0068] The sandbox process can be understood as providing a constrained computing environment for a process (or processes) within the sandbox, where these constrained processes can be controlled and restricted to limit access to certain computing resources.

[0069] Examples of security mechanisms can include the implementation of namespaces in which each respective group of processes executing within the sandbox environment has access to respective computing resources (e.g., process IDs, hostnames, user IDs, file names, names associated with network access, and inter-process communication) that are not accessible to another group of processes (which may have access to a different group of resources not accessible by the former group of processes), other container implementations, and the like. By having the sandbox process execute as a sub-process to the execution node process, in some embodiments, latency in processing a given database query can be substantially reduced (e.g., a reduction in latency by a factor of 10× in some instances) in comparison with other techniques that may utilize a virtual machine solution by itself.

[0070] The sandbox process can utilize a sandbox policy to enforce a given security policy. The sandbox policy can be a file with information related to a configuration of the sandbox process and details regarding restrictions, if any, and permissions for accessing and utilizing system resources. Example restrictions can include restrictions to network access, or file system access (e.g., remapping file system to place files in different locations that may not be accessible, other files can be mounted in different locations, and the like). The sandbox process restricts the memory and processor (e.g., CPU) usage of the user code runtime, ensuring that other operations on the same execution node can execute without running out of resources.

[0071] As mentioned above, the sandbox process is a sub-process (or separate process) from the execution node process, which in practice means that the sandbox process resides in a separate memory space than the execution node process. In an occurrence of a security breach in connection with the sandbox process (e.g., by errant or malicious code from a given UDTF), if arbitrary memory is accessed by a malicious actor, the data or information stored by the execution node process is protected.

[0072] At operation **606**, scanner options for the custom file format may be set. Scanner options may relate to how to handle various properties of the data in the file to be ingested, which is in the custom file format. The scanner options may include options related to compression, record delimiter, filed delimiter, file extension, skip reader, skipping

blank lines, date format, time format, timestamp format, binary format, null values, column count mismatch.

[0073] At operation **608**, pipe definition for one or more pipes may be set to include the custom file format. Moreover, notification of an ingest request for file(s) in the custom file format may be received, as described above. At operation **610**, one or more XPs may be pinned for the auto-ingestion of the file(s) to be ingested in the custom file format. As described in further detail below, because the UDTF may include untrusted code, designated or pinned XPs may be used for the auto-ingestion of the custom file format. The pinned one or more XPs may use the pipe(s) set for the custom file format for performing the continuous ingestion.

[0074] At operation **612**, the pinned XPs may perform the auto-ingestion of the files to be ingested in the custom file format. The stream output of rows with one or more variant columns of the UDTF may be buffered by the pipe. After the number of buffered rows exceed a threshold value, the pipe may ingest the buffered rows into the source table, as described above. The rows may be created into format files used by the data system and may be registered and committed to the source table. At operation **614**, after all files are ingested, the pinned XPs may be released so that they can be used for other tasks, including other ingest tasks.

[0075] FIG. **7** illustrates a data system framework for ingestion of files with custom file formats, according to some example embodiments. The framework may include an ingest compute service manager **702**, an ingest warehouse **704**, a set of multi-tenant XPs **706.1-706.4**, a first set of pinned XPs **708.1-708.2**, and a second set of pinned XPs **710.1-710.2**. The ingest compute service manager **702** may include the functionalities described above for compute service manager described herein (e.g., compute service manager **112**, compute service manager **504**). The ingest compute service manager **702** may receive notifications of ingest request from a plurality of accounts associated with the multi-tenant data system. The compute service manager **702** may assign ingest tasks to one or more XPs based on the ingest request notifications. The ingest requests may be for natively supported file formats and/or custom file formats, as described herein.

[0076] The ingest warehouse **704** may include a pool of XPs to perform the ingest tasks assigned by the compute service manager **702**. For files in natively supported formats, multi-tenant XPs **706.1-706.4** may be used to perform the ingest tasks. Multi-tenant XPs **706.1-706.4** may perform ingest tasks for multiple accounts at the same time for natively supported format files. However, as described above, for ingest tasks related to custom file formats, pinned XPs are used because the UDTFs defining the custom file format in the pipe definition may include untrusted code. The pinned XPs may be referred to as per-account XP instances for pipes that use custom file formats.

[0077] For example, pinned XPs **708.1-708.2** may be acquired from the pool and pinned by Account A to perform ingest tasks for Account A that are in a custom file format defined by Account A. Because the XPs **708.1-708.2** are pinned while performing ingest tasks for Account A, they cannot perform ingest tasks for other accounts at the same time. After the XPs **708.1-708.2** complete the ingest tasks associated with the ingest request from Account A for files in the custom file format, XPs **708.1-708.2** can be released and be used for other tasks.

[0078] Likewise, pinned XPs **710.1-710.2** may be acquired from the pool and pinned by Account B to perform ingest tasks for Account B that are in a custom file format defined by Account B. Because the XPs **710.1-710.2** are pinned while performing ingest tasks for Account B, they cannot perform ingest tasks for other accounts at the same time. After the XPs **710.1-710.2** complete the ingest tasks associated with the ingest request from Account B for files in a custom file format, XPs **710.1-710.2** can be released and can be used for other tasks. In some examples, smaller side XPs may be used for performing ingest tasks for files with custom file formats so that larger XPs may be reserved for handling multi-tenant ingest tasks.

[0079] FIG. **8** illustrates a diagrammatic representation of a machine **800** in the form of a computer system within which a set of instructions may be executed for causing the machine **800** to perform any one or more of the methodologies discussed herein, according to an example embodiment. Specifically, FIG. **8** shows a diagrammatic representation of the machine **800** in the example form of a computer system, within which instructions **816** (e.g., software, a program, an application, an applet, an app, or other executable code) for causing the machine **800** to perform any one or more of the methodologies discussed herein may be executed. For example, the instructions **816** may cause the machine **800** to execute any one or more operations of any one or more of the methods described herein. As another example, the instructions **816** may cause the machine **800** to implement portions of the data flows described herein. In this way, the instructions **816** transform a general, non-programmed machine into a particular machine **800** (e.g., the remote computing device **106**, the access management system **110**, the compute service manager **112**, the execution platform (XP) **114**, the access management system **118**, the Web proxy **120**) that is specially configured to carry out any one of the described and illustrated functions in the manner described herein.

[0080] In alternative embodiments, the machine **800** operates as a standalone device or may be coupled (e.g., networked) to other machines. In a networked deployment, the machine **800** may operate in the capacity of a server machine or a client machine in a server-client network environment, or as a peer machine in a peer-to-peer (or distributed) network environment. The machine **800** may comprise, but not be limited to, a server computer, a client computer, a personal computer (PC), a tablet computer, a laptop computer, a netbook, a smart phone, a mobile device, a network router, a network switch, a network bridge, or any machine capable of executing the instructions **816**, sequentially or otherwise, that specify actions to be taken by the machine **800**. Further, while only a single machine **800** is illustrated, the term "machine" shall also be taken to include a collection of machines **800** that individually or jointly execute the instructions **816** to perform any one or more of the methodologies discussed herein.

[0081] The machine **800** includes processors **810**, memory **830**, and input/output (I/O) components **850** configured to communicate with each other such as via a bus **802**. In an example embodiment, the processors **810** (e.g., a central processing unit (CPU), a reduced instruction set computing (RISC) processor, a complex instruction set computing (CISC) processor, a graphics processing unit (GPU), a digital signal processor (DSP), an application-specific integrated circuit (ASIC), a radio-frequency integrated circuit

(RFIC), another processor, or any suitable combination thereof) may include, for example, a processor **812** and a processor **814** that may execute the instructions **816**. The term "processor" is intended to include multi-core processors **810** that may comprise two or more independent processors (sometimes referred to as "cores") that may execute instructions **816** contemporaneously. Although FIG. **8** shows multiple processors **810**, the machine **800** may include a single processor with a single core, a single processor with multiple cores (e.g., a multi-core processor), multiple processors with a single core, multiple processors with multiple cores, or any combination thereof.

[0082] The memory **830** may include a main memory **832**, a static memory **834**, and a storage unit **836**, all accessible to the processors **810** such as via the bus **802**. The main memory **832**, the static memory **834**, and the storage unit **836** store the instructions **816** embodying any one or more of the methodologies or functions described herein. The instructions **816** may also reside, completely or partially, within the main memory **832**, within the static memory **834**, within the storage unit **836**, within at least one of the processors **810** (e.g., within the processor's cache memory), or any suitable combination thereof, during execution thereof by the machine **800**.

[0083] The I/O components **850** include components to receive input, provide output, produce output, transmit information, exchange information, capture measurements, and so on. The specific I/O components **850** that are included in a particular machine **800** will depend on the type of machine. For example, portable machines such as mobile phones will likely include a touch input device or other such input mechanisms, while a headless server machine will likely not include such a touch input device. It will be appreciated that the I/O components **850** may include many other components that are not shown in FIG. **8**. The I/O components **850** are grouped according to functionality merely for simplifying the following discussion and the grouping is in no way limiting. In various example embodiments, the I/O components **850** may include output components **852** and input components **854**. The output components **852** may include visual components (e.g., a display such as a plasma display panel (PDP), a light emitting diode (LED) display, a liquid crystal display (LCD), a projector, or a cathode ray tube (CRT)), acoustic components (e.g., speakers), other signal generators, and so forth. The input components **854** may include alphanumeric input components (e.g., a keyboard, a touch screen configured to receive alphanumeric input, a photo-optical keyboard, or other alphanumeric input components), point-based input components (e.g., a mouse, a touchpad, a trackball, a joystick, a motion sensor, or another pointing instrument), tactile input components (e.g., a physical button, a touch screen that provides location and/or force of touches or touch gestures, or other tactile input components), audio input components (e.g., a microphone), and the like.

[0084] Communication may be implemented using a wide variety of technologies. The I/O components **850** may include communication components **864** operable to couple the machine **800** to a network **880** or devices **870** via a coupling **882** and a coupling **872**, respectively. For example, the communication components **864** may include a network interface component or another suitable device to interface with the network **880**. In further examples, the communication components **864** may include wired communication components, wireless communication components, cellular communication components, and other communication components to provide communication via other modalities. The devices **870** may be another machine or any of a wide variety of peripheral devices (e.g., a peripheral device coupled via a universal serial bus (USB)). For example, as noted above, the machine **800** may correspond to any one of the remote computing device **106**, the access management system **118**, the compute service manager **112**, the execution platform **114**, the Web proxy **120**, and the devices **870** may include any other of these systems and devices.

[0085] The various memories (e.g., **830**, **832**, **834**, and/or memory of the processor(s) **810** and/or the storage unit **836**) may store one or more sets of instructions **816** and data structures (e.g., software) embodying or utilized by any one or more of the methodologies or functions described herein. These instructions **816**, when executed by the processor(s) **810**, cause various operations to implement the disclosed embodiments.

[0086] As used herein, the terms "machine-storage medium," "device-storage medium," and "computer-storage medium" mean the same thing and may be used interchangeably in this disclosure. The terms refer to a single or multiple storage devices and/or media (e.g., a centralized or distributed database, and/or associated caches and servers) that store executable instructions and/or data. The terms shall accordingly be taken to include, but not be limited to, solid-state memories, and optical and magnetic media, including memory internal or external to processors. Specific examples of machine-storage media, computer-storage media, and/or device-storage media include non-volatile memory, including by way of example semiconductor memory devices, e.g., erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), field-programmable gate arrays (FPGAs), and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The terms "machine-storage media," "computer-storage media," and "device-storage media" specifically exclude carrier waves, modulated data signals, and other such media, at least some of which are covered under the term "signal medium" discussed below.

[0087] In various example embodiments, one or more portions of the network **880** may be an ad hoc network, an intranet, an extranet, a virtual private network (VPN), a local-area network (LAN), a wireless LAN (WLAN), a wide-area network (WAN), a wireless WAN (WWAN), a metropolitan-area network (MAN), the Internet, a portion of the Internet, a portion of the public switched telephone network (PSTN), a plain old telephone service (POTS) network, a cellular telephone network, a wireless network, a Wi-Fi® network, another type of network, or a combination of two or more such networks. For example, the network **880** or a portion of the network **880** may include a wireless or cellular network, and the coupling **882** may be a Code Division Multiple Access (CDMA) connection, a Global System for Mobile communications (GSM) connection, or another type of cellular or wireless coupling. In this example, the coupling **882** may implement any of a variety of types of data transfer technology, such as Single Carrier Radio Transmission Technology (1×RTT), Evolution-Data Optimized (EVDO) technology, General Packet Radio Service (GPRS) technology, Enhanced Data rates for GSM

Evolution (EDGE) technology, third Generation Partnership Project (3GPP) including 3G, fourth generation wireless (4G) networks, Universal Mobile Telecommunications System (UMTS), High-Speed Packet Access (HSPA), Worldwide Interoperability for Microwave Access (WiMAX), Long Term Evolution (LTE) standard, others defined by various standard-setting organizations, other long-range protocols, or other data transfer technology.

[0088] The instructions **816** may be transmitted or received over the network **880** using a transmission medium via a network interface device (e.g., a network interface component included in the communication components **864**) and utilizing any one of a number of well-known transfer protocols (e.g., hypertext transfer protocol (HTTP)). Similarly, the instructions **816** may be transmitted or received using a transmission medium via the coupling **872** (e.g., a peer-to-peer coupling) to the devices **870**. The terms "transmission medium" and "signal medium" mean the same thing and may be used interchangeably in this disclosure. The terms "transmission medium" and "signal medium" shall be taken to include any intangible medium that is capable of storing, encoding, or carrying the instructions **816** for execution by the machine **800**, and include digital or analog communications signals or other intangible media to facilitate communication of such software. Hence, the terms "transmission medium" and "signal medium" shall be taken to include any form of modulated data signal, carrier wave, and so forth. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal.

[0089] The terms "machine-readable medium," "computer-readable medium," and "device-readable medium" mean the same thing and may be used interchangeably in this disclosure. The terms are defined to include both machine-storage media and transmission media. Thus, the terms include both storage devices/media and carrier waves/modulated data signals.

[0090] The various operations of example methods described herein may be performed, at least partially, by one or more processors that are temporarily configured (e.g., by software) or permanently configured to perform the relevant operations. Similarly, the methods described herein may be at least partially processor-implemented. For example, at least some of the operations of the methods described herein may be performed by one or more processors. The performance of certain of the operations may be distributed among the one or more processors, not only residing within a single machine, but also deployed across a number of machines. In some example embodiments, the processor or processors may be located in a single location (e.g., within a home environment, an office environment, or a server farm), while in other embodiments the processors may be distributed across a number of locations.

[0091] Although the embodiments of the present disclosure have been described with reference to specific example embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader scope of the inventive subject matter. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense. The accompanying drawings that form a part hereof show, by way of illustration, and not of limitation, specific embodiments in which the subject matter may be practiced. The embodiments illustrated are described in sufficient detail to enable those skilled in the art to practice the teachings disclosed herein. Other embodiments may be used and derived therefrom, such that structural and logical substitutions and changes may be made without departing from the scope of this disclosure. This Detailed Description, therefore, is not to be taken in a limiting sense, and the scope of various embodiments is defined only by the appended claims, along with the full range of equivalents to which such claims are entitled.

[0092] Such embodiments of the inventive subject matter may be referred to herein, individually and/or collectively, by the term "invention" merely for convenience and without intending to voluntarily limit the scope of this application to any single invention or inventive concept if more than one is in fact disclosed. Thus, although specific embodiments have been illustrated and described herein, it should be appreciated that any arrangement calculated to achieve the same purpose may be substituted for the specific embodiments shown. This disclosure is intended to cover any and all adaptations or variations of various embodiments. Combinations of the above embodiments, and other embodiments not specifically described herein, will be apparent, to those of skill in the art, upon reviewing the above description.

[0093] In this document, the terms "a" or "an" are used, as is common in patent documents, to include one or more than one, independent of any other instances or usages of "at least one" or "one or more." In this document, the term "or" is used to refer to a nonexclusive or, such that "A or B" includes "A but not B," "B but not A," and "A and B," unless otherwise indicated. In the appended claims, the terms "including" and "in which" are used as the plain-English equivalents of the respective terms "comprising" and "wherein." Also, in the following claims, the terms "including" and "comprising" are open-ended; that is, a system, device, article, or process that includes elements in addition to those listed after such a term in a claim is still deemed to fall within the scope of that claim.

[0094] Described implementations of the subject matter can include one or more features, alone or in combination as illustrated below by way of example.

[0095] Example 1. A method comprising: defining, by a data system comprising at least one hardware processor, a custom file format for use in auto-ingestion of files in the custom file format; setting a user defined function (UDF) to parse files in the custom file format and return a table with one or more rows and a variant column; receiving a notification of at least one file in the custom file format for auto-ingestion; performing auto-ingestion of the at least one file using a pipe set for the custom file format executing the UDF; and storing data from the at least one file in a source table.

[0096] Example 2. The method of example 1, wherein the custom file format is not natively supported by the data system.

[0097] Example 3. The method of any of examples 1-2, wherein the custom file format includes an unstructured file type.

[0098] Example 4. The method of any of examples 1-3, wherein the data system operates using a first programming language and the UDF is in a second programming language.

[0099] Example 5. The method of any of examples 1-4, wherein the UDF is executed in a sandbox environment.

[0100] Example 6. The method of any of examples 1-5, further comprising: acquiring one or more execution nodes; assigning one or more ingest tasks related to the at least one file; pinning the one or more execution nodes such that the one or more execution nodes cannot perform other ingestion tasks for files in a different format than the custom file format; and performing, by the one or more execution nodes, the one or more ingest tasks.

[0101] Example 7. The method of any of examples 1-6, further comprising: releasing the one or more execution nodes such that they can perform other ingestions tasks for files in different formats than the custom file format.

[0102] Example 8. A system comprising: one or more processors of a machine; and a memory storing instructions that, when executed by the one or more processors, cause the machine to perform operations implementing any one of example methods 1 to 7.

[0103] Example 9. A machine-readable storage device embodying instructions that, when executed by a machine, cause the machine to perform operations implementing any one of example methods 1 to 7.

What is claimed is:

1. A method for managing a pool of execution nodes in a multi-tenant data system the method comprising:

in response to identifying a first file of a natively supported format for ingestion, acquiring a first set of execution nodes from the pool of execution nodes;

performing, using the first set of execution nodes, ingestion of the first file, wherein the first set of execution nodes remains available for concurrent assignment of tasks for other accounts of the multi-tenant data system;

in response to identifying a second file of a custom format not being natively supported by the multi-tenant data system for ingestion, acquiring a second set of execution nodes from the pool of execution nodes; and

performing, using the second set of execution nodes, ingestion of the second file using a user-defined function (UDF) to parse the custom format, wherein the second set of execution nodes is pinned during said ingestion of the second file to prevent concurrent assignment of tasks for other accounts of the multi-tenant data system.

2. The method of claim **1**, further comprising:

storing data from the second file in one or more rows in a source table in a format used by the multi-tenant data system;

registering the one or more rows in the source table; and

committing the one or more rows in the source table.

3. The method of claim **2**, further comprising:

in response to committing the one or more rows, releasing the second set of execution nodes.

4. The method of claim **1**, wherein parsing the second file returns a table with one or more rows and a variant column.

5. The method of claim **1**, wherein the custom format includes an unstructured file type.

6. The method of claim **1**, wherein the multi-tenant data system operates using a first programming language and the UDF is in a second programming language.

7. The method of claim **6**, wherein the UDF is executed in a sandbox environment.

8. A system comprising:

at least one hardware processor; and

at least one memory storing instructions that, when executed by the at least one hardware processor, cause the at least one hardware processor to perform operations comprising:

in response to identifying a first file of a natively supported format for ingestion, acquiring a first set of execution nodes from a pool of execution nodes in a multi-tenant data system;

performing, using the first set of execution nodes, ingestion of the first file, wherein the first set of execution nodes remains available for concurrent assignment of tasks for other accounts of the multi-tenant data system;

in response to identifying a second file of a custom format not being natively supported by the multi-tenant data system for ingestion, acquiring a second set of execution nodes from the pool of execution nodes; and

performing, using the second set of execution nodes, ingestion of the second file using a user-defined function (UDF) to parse the custom format, wherein the second set of execution nodes is pinned during said ingestion of the second file to prevent concurrent assignment of tasks for other accounts of the multi-tenant data system.

9. The system of claim **8**, further comprising:

storing data from the second file in one or more rows in a source table in a format used by the multi-tenant data system;

registering the one or more rows in the source table; and

committing the one or more rows in the source table.

10. The system of claim **9**, further comprising:

in response to committing the one or more rows, releasing the second set of execution nodes.

11. The system of claim **8**, wherein parsing the second file returns a table with one or more rows and a variant column.

12. The system of claim **8**, wherein the custom format includes an unstructured file type.

13. The system of claim **8**, wherein the multi-tenant data system operates using a first programming language and the UDF is in a second programming language.

14. The system of claim **13**, wherein the UDF is executed in a sandbox environment.

15. A machine-storage medium embodying instructions that, when executed by a machine, cause the machine to perform operations comprising:

in response to identifying a first file of a natively supported format for ingestion, acquiring a first set of execution nodes from a pool of execution nodes in a multi-tenant data system;

performing, using the first set of execution nodes, ingestion of the first file, wherein the first set of execution nodes remains available for concurrent assignment of tasks for other accounts of the multi-tenant data system;

in response to identifying a second file of a custom format not being natively supported by the multi-tenant data system for ingestion, acquiring a second set of execution nodes from the pool of execution nodes; and

performing, using the second set of execution nodes, ingestion of the second file using a user-defined function (UDF) to parse the custom format, wherein the second set of execution nodes is pinned during said

ingestion of the second file to prevent concurrent assignment of tasks for other accounts of the multi-tenant data system.

16. The machine-storage medium of claim **15**, further comprising:

storing data from the second file in one or more rows in a source table in a format used by the multi-tenant data system;

registering the one or more rows in the source table; and

committing the one or more rows in the source table.

17. The machine-storage medium of claim **16**, further comprising:

in response to committing the one or more rows, releasing the second set of execution nodes.

18. The machine-storage medium of claim **15**, wherein parsing the second file returns a table with one or more rows and a variant column.

19. The machine-storage medium of claim **15**, wherein the custom format includes an unstructured file type.

20. The machine-storage medium of claim **15**, wherein the multi-tenant data system operates using a first programming language and the UDF is in a second programming language, wherein the UDF is executed in a sandbox environment.

* * * * *