US 20210334284A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2021/0334284 A1**

Bansal et al. (43) **Pub. Date: Oct. 28, 2021**

(54) **SYSTEM AND METHOD OF QUERYING OBJECTS ON DEMAND**

(71) Applicant: **Nutanix, Inc.**, San Jose, CA (US)

(72) Inventors: **Anirudh Kumar Bansal**, Delhi (IN); **Divya Harish Saglani**, Pune (IN); **Manik Taneja**, Bangalore (IN); **Naveen Reddy Gundlagutta**, Bangalore (IN); **Nikhil Mundra**, Udaipur (IN)

(73) Assignee: **Nutanix, Inc.**, San Jose, CA (US)

(21) Appl. No.: **16/904,502**

(22) Filed: **Jun. 17, 2020**

(30) **Foreign Application Priority Data**

Apr. 28, 2020 (IN) 202041018166

Publication Classification

(51) **Int. Cl.**

G06F 16/2455	(2006.01)
G06F 16/242	(2006.01)
G06F 16/28	(2006.01)
G06F 16/2457	(2006.01)

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

CPC .. ***G06F 16/24554*** (2019.01); ***G06F 16/24573*** (2019.01); ***G06F 16/284*** (2019.01); ***G06F 16/244*** (2019.01)

(57) **ABSTRACT**

An illustrative embodiment disclosed herein is an apparatus including a processor having programmed instructions that receive a structured query language (SQL) query, identify a bucket, identify metadata relationships specified in the SQL query, and execute the SQL query to generate a list of objects included in the bucket and having metadata satisfying the metadata relationships.

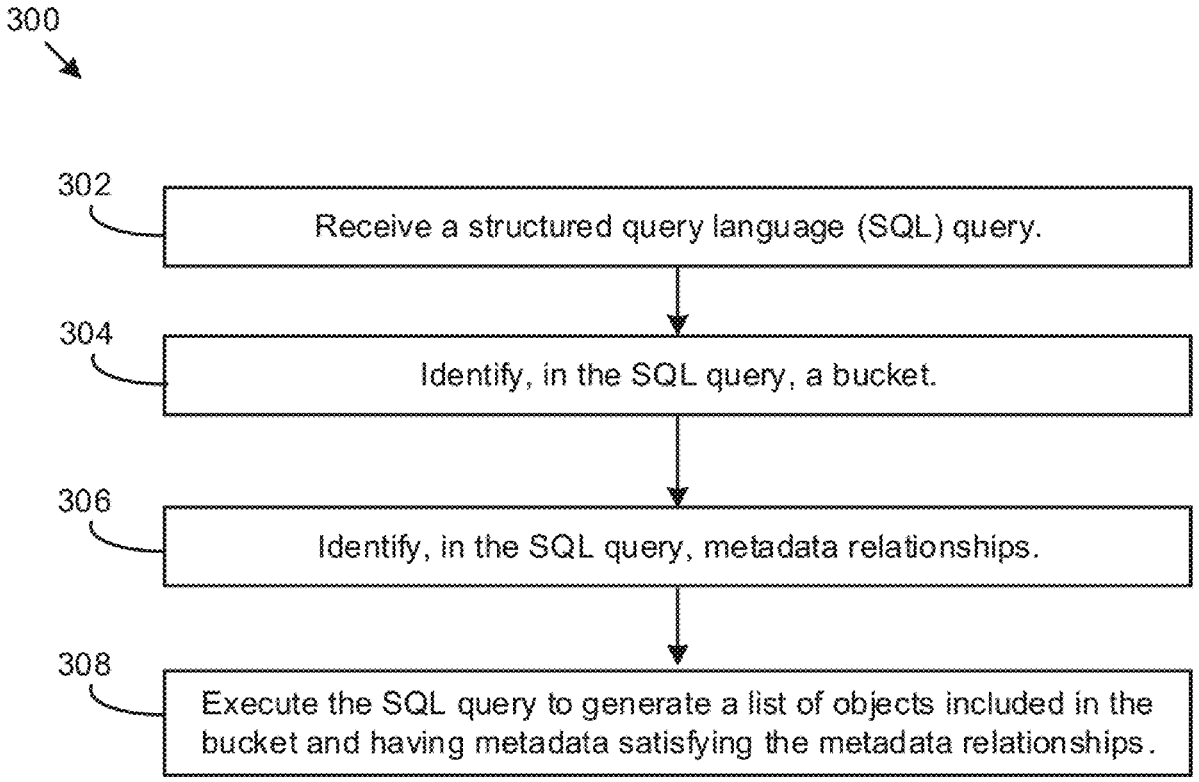
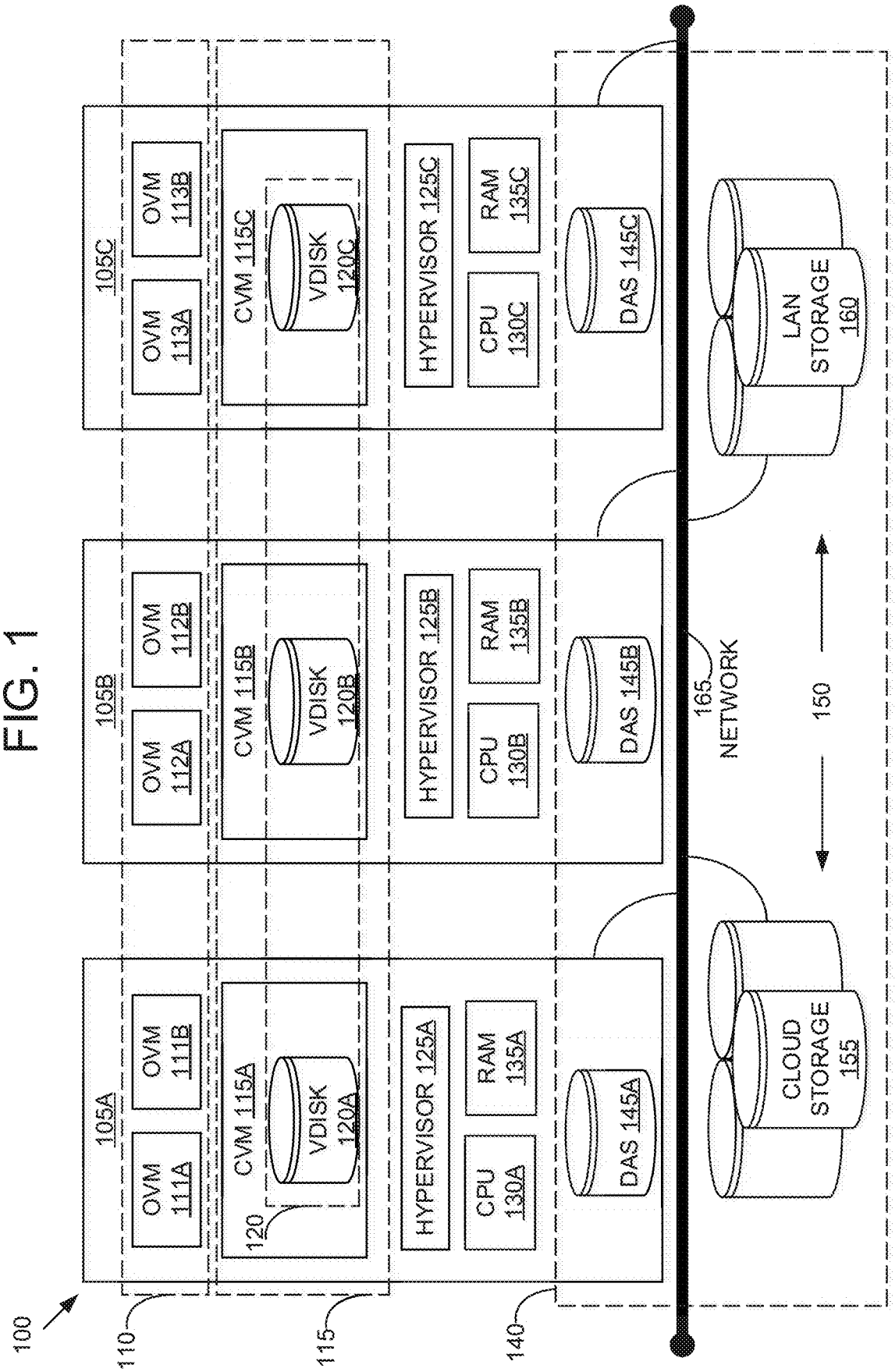


FIG. 1


100
110
115
140
105A
OVM 111A
OVM 111B
CVM 115A
VDISK 120A
120
HYPERVISOR 125A
CPU 130A
RAM 135A
DAS 145A
105B
OVM 112A
OVM 112B
CVM 115B
VDISK 120B
HYPERVISOR 125B
CPU 130B
RAM 135B
DAS 145B
105C
OVM 113A
OVM 113B
CVM 115C
VDISK 120C
HYPERVISOR 125C
CPU 130C
RAM 135C
DAS 145C
165
NETWORK
150
LAN STORAGE 160
CLOUD STORAGE 155

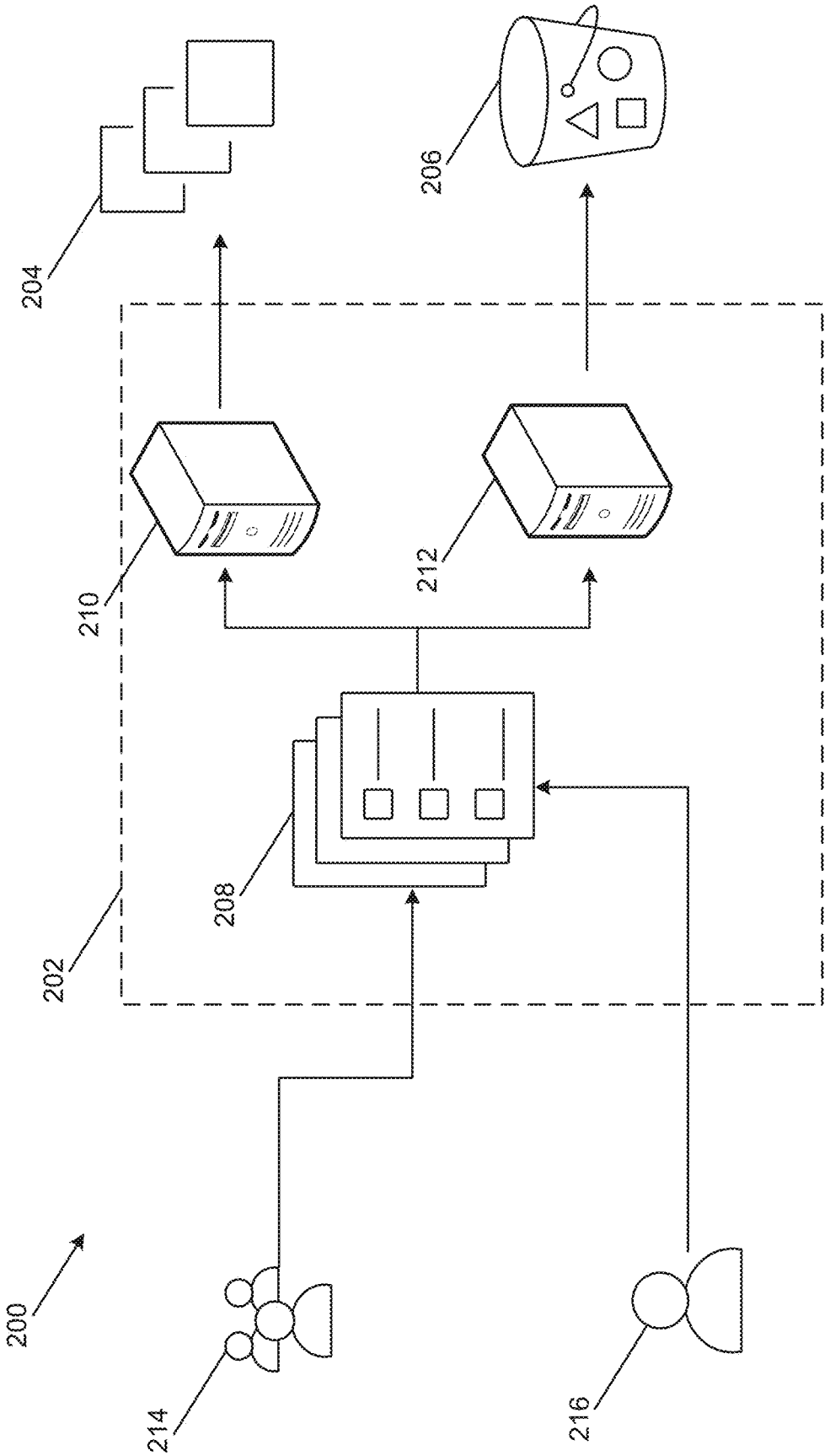


FIG. 2

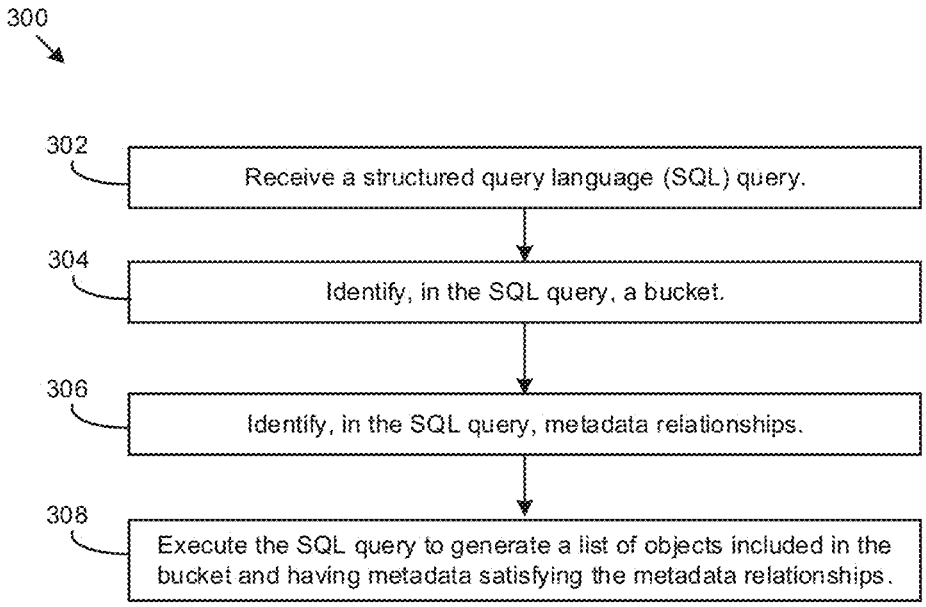

300
302
Receive a structured query language (SQL) query.
304
Identify, in the SQL query, a bucket.
306
Identify, in the SQL query, metadata relationships.
308
Execute the SQL query to generate a list of objects included in the bucket and having metadata satisfying the metadata relationships.

FIG. 3

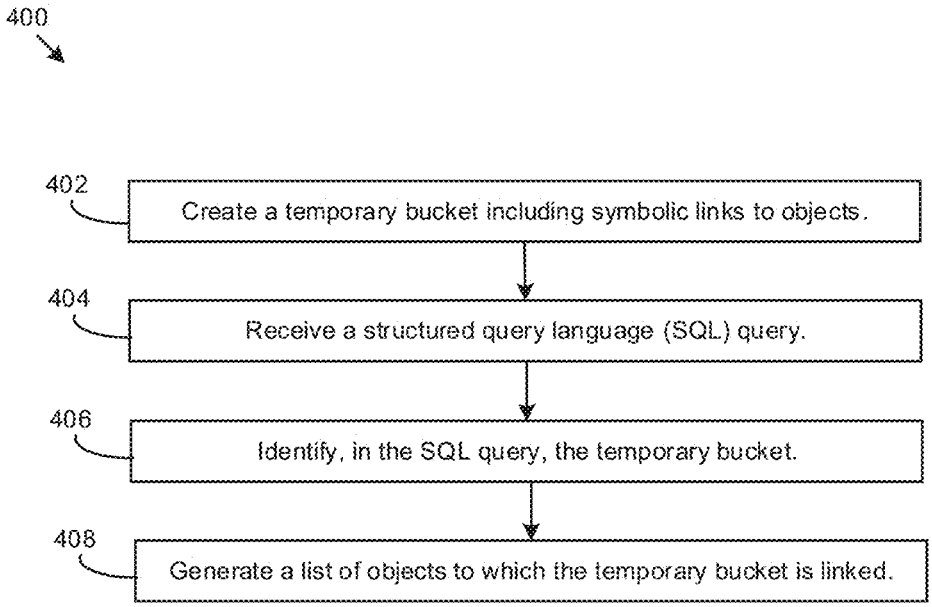

400
402
Create a temporary bucket including symbolic links to objects.
404
Receive a structured query language (SQL) query.
406
Identify, in the SQL query, the temporary bucket.
408
Generate a list of objects to which the temporary bucket is linked.

FIG. 4

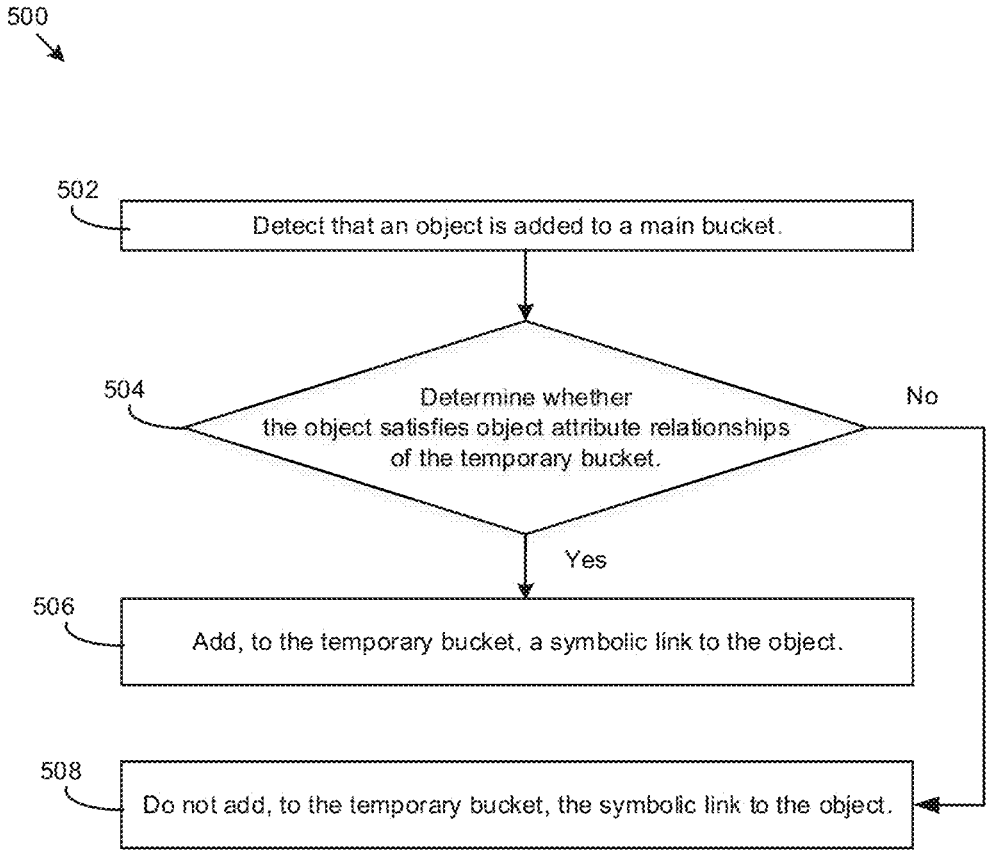

500
502
Detect that an object is added to a main bucket.
504
Determine whether
the object satisfies object attribute relationships
of the temporary bucket.
No
Yes
506
Add, to the temporary bucket, a symbolic link to the object.
508
Do not add, to the temporary bucket, the symbolic link to the object.

FIG. 5

SYSTEM AND METHOD OF QUERYING OBJECTS ON DEMAND

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application is related to and claims priority under 35 U.S. § 119(e) from Indian Provisional Application No. 202041018166, filed Apr. 28, 2020, titled "SYSTEM AND METHOD OF QUERYING OBJECTS ON DEMAND," the entire contents of which are incorporated herein by reference for all purposes.

BACKGROUND

[0002] The following description is provided to assist the understanding of the reader. None of the information provided or references cited is admitted to be prior art.

[0003] Virtual computing systems are widely used in a variety of applications. Virtual computing systems include one or more host machines running one or more virtual machines concurrently. The virtual machines utilize the hardware resources of the underlying host machines. Each virtual machine may be configured to run an instance of an operating system. Modern virtual computing systems allow several operating systems and several software applications to be safely run at the same time on the virtual machines of a single host machine, thereby increasing resource utilization and performance efficiency. However, the present-day virtual computing systems have limitations due to their configuration and the way they operate.

SUMMARY

[0004] Aspects of the present disclosure relate generally to a virtualization environment, and more particularly to a system and method for querying objects on demand.

[0005] An illustrative embodiment disclosed herein is an apparatus including a processor having programmed instructions that receive a structured query language (SQL) query, identify a bucket, identify metadata relationships specified in the SQL query, and execute the SQL query to generate a list of objects included in the bucket and having metadata satisfying the metadata relationships.

[0006] Another illustrative embodiment disclosed herein is a non-transitory computer readable storage medium having instructions stored thereon that, upon execution by a processor, causes the processor to perform operations including receiving a structured query language (SQL) query, identifying a bucket, identifying metadata relationships specified in the SQL query, and executing the SQL query to generate a list of objects included in the bucket and having metadata satisfying the metadata relationships.

[0007] Another illustrative embodiment disclosed herein is a computer-implemented method including receiving, by a processor, a structured query language (SQL) query, identifying, by the processor, a bucket, identifying, by the processor, metadata relationships specified in the SQL query, and executing, by the processor, the SQL query to generate a list of objects included in the bucket and having metadata satisfying the metadata relationships.

[0008] Further details of aspects, objects, and advantages of the invention are described below in the detailed description, drawings, and claims. Both the foregoing general description and the following detailed description are exemplary and explanatory, and are not intended to be limiting as to the scope of the invention. Particular embodiments may include all, some, or none of the components, elements, features, functions, operations, or steps of the embodiments disclosed above. The subject matter which can be claimed comprises not only the combinations of features as set out in the attached claims but also any other combination of features in the claims, wherein each feature mentioned in the claims can be combined with any other feature or combination of other features in the claims. Furthermore, any of the embodiments and features described or depicted herein can be claimed in a separate claim and/or in any combination with any embodiment or feature described or depicted herein or with any of the features of the attached claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] FIG. **1** is an example block diagram of a virtual computing system, in accordance with some embodiments of the present disclosure.

[0010] FIG. **2** illustrates an example diagram of an object storage service (OSS), in accordance with some embodiments of the present disclosure.

[0011] FIG. **3** illustrates an example method for executing SQL queries, in accordance with some embodiments of the present disclosure.

[0012] FIG. **4** illustrates an example method for executing SQL queries, in accordance with some embodiments of the present disclosure.

[0013] FIG. **5** illustrates an example method for updating temporary buckets, in accordance with some embodiments of the present disclosure.

[0014] The foregoing and other features of the present disclosure will become apparent from the following description and appended claims, taken in conjunction with the accompanying drawings. Understanding that these drawings depict only several embodiments in accordance with the disclosure and are, therefore, not to be considered limiting of its scope, the disclosure will be described with additional specificity and detail through use of the accompanying drawings.

DETAILED DESCRIPTION

[0015] In the following detailed description, reference is made to the accompanying drawings, which form a part hereof. In the drawings, similar symbols typically identify similar components, unless context dictates otherwise. The illustrative embodiments described in the detailed description, drawings, and claims are not meant to be limiting. Other embodiments may be utilized, and other changes may be made, without departing from the spirit or scope of the subject matter presented here. It will be readily understood that the aspects of the present disclosure, as generally described herein, and illustrated in the figures, can be arranged, substituted, combined, and designed in a wide variety of different configurations, all of which are explicitly contemplated and make part of this disclosure.

[0016] Object storage is the fastest growing form of distributed storage. Object stores are distributed key-value stores that are used to store massive amounts of unstructured data. Object storage provides simplified data-access APIs to GET/PUT objects within a bucket. However, storage admins have little insight into the data being stored in an object storage. Thus, existing APIs have proved incapable in enabling deep storage analytics.

[0017] Companies and customers have had to resort to building custom solutions that are unwieldy and expensive. Some existing solutions allow a customer to run an SQL query, but it can only run against a single object. This limits its usefulness to only specific use-cases.

[0018] Some embodiments of the disclosure herein, provides a unified SQL (structured query language) embedded within an object store deployment that extends this API to enable queries not just on a single object, but also on all the objects that are contained within a bucket, as well as on the metadata of the objects. The metadata can store information such as who created the object, when was it created, what is the size of the object, what is the content type. Moreover, some implementations of the disclosure herein allow object attribute-based or metadata-based temporary buckets/indices to be created on demand. Leveraging object attribute-based or metadata-based indices can result in reduced latency associated with serving queries on, for example, historical datasets, such as in online analytical processing workloads. Further, some embodiments of the disclosure herein update the object attribute-based or metadata-based indices, which can be useful for workloads that have dynamic data, such as online transaction processing workloads.

[0019] Some aspects disclosed herein bring the power of no-SQL databases directly to traditional object stores. Some embodiments are tightly coupled solutions that can scale horizontally by sharding object data and/or deploying additional storage nodes. Some embodiments expand the use and scope of object storage solutions to many data processing applications that can now directly query the object store deployment without resorting to custom and expensive solutions. Some aspects simplify information technology (IT) administration. Not only would an object store deployment be capable of storing large amounts of data, it becomes capable of providing SQL query functionality that is a hallmark of typical database system. This may pave the way for an object store deployment to be one-stop storage solution, thereby freeing a customer from resorting to custom, third party solutions which would result in huge amount of savings in terms of cost and IT resources.

Object Virtualization Technology and Environment

[0020] Referring now to FIG. **1**, a virtual computing system **100** is shown, in accordance with some embodiments of the present disclosure. The virtual computing system **100** includes a plurality of nodes, such as a first node **105**A, a second node **105**B, and a third node **105**C. The nodes may be collectively referred to herein as "nodes **105**." Each of the nodes **105** may also be referred to as a "host" or "host machine." The first node **105**A includes an object virtual machine ("OVMs") **111**A and **111**B (collectively referred to herein as "OVMs **111**"), a controller virtual machine ("CVM") **115**A, and a hypervisor **125**A. Similarly, the second node **105**B includes OVMs **112**A and **112**B (collectively referred to herein as "OVMs **112**"), a CVM **115**B, and a hypervisor **125**B, and the third node **105**C includes OVMs **113**A and **113**B (collectively referred to herein as "OVMs **113**"), a CVM **115**C, and a hypervisor **125**C. The OVMs **111**, **112**, and **113** may be collectively referred to herein as "OVMs **110**." The CVMs **115**A, **115**B, and **115**C may be collectively referred to herein as "CVMs **115**." The nodes **105** are connected to a network **165**.

[0021] The virtual computing system **100** also includes a storage pool **140**. The storage pool **140** may include network-attached storage (NAS) **150** and direct-attached storage (DAS) **145**A, **145**B, and **145**C (collectively referred to herein as DAS **145**). The NAS **150** is accessible via the network **165** and, in some embodiments, may include cloud storage **155**, as well as local area network ("LAN") storage **160**. In contrast to the NAS **150**, which is accessible via the network **165**, each of the DAS **145**A, the DAS **145**B, and the DAS **145**C includes storage components that are provided internally within the first node **105**A, the second node **105**B, and the third node **105**C, respectively, such that each of the first, second, and third nodes may access its respective DAS without having to access the network **165**.

[0022] The CVM **115**A may include one or more virtual disks ("vdisks") **120**A, the CVM **115**B may include one or more vdisks **120**B, and the CVM **115**C may include one or more vdisks **120**C. The vdisks **120**A, the vdisks **120**B, and the vdisks **120**C are collectively referred to herein as "vdisks **120**." The vdisks **120** may be a logical representation of storage space allocated from the storage pool **140**. Each of the vdisks **120** may be located in a memory of a respective one of the CVMs **115**. The memory of each of the CVMs **115** may be a virtualized instance of underlying hardware, such as the RAMs **135** and/or the storage pool **140**. The virtualization of the underlying hardware is described below.

[0023] In some embodiments, the CVMs **115** may be configured to run a distributed operating system in that each of the CVMs **115** run a subset of the distributed operating system. In some such embodiments, the CVMs **115** form one or more Nutanix Operating System ("NOS") cluster. In some embodiments, the one or more NOS clusters include greater than or fewer than the CVMs **115**. In some embodiments, each of the CVMs **115** run a separate, independent instance of an operating system. In some embodiments, the one or more NOS clusters may be referred to as a storage layer.

[0024] In some embodiments, the OVMs **110** form an OVM cluster. OVMs of an OVM cluster may be configured to share resources with each other. The OVMs in the OVM cluster may be configured to access storage from the NOS cluster using one or more of the vdisks **120** as a storage unit. In some embodiments, the OVM cluster include greater than or fewer than the OVMs **110**.

[0025] Some or all of the OVMs **110** in the OVM cluster may be configured to run software-defined object storage service, such as Nutanix Buckets™. As part of the object storage service (OSS), the OVMs **110** may be configured to deploy (e.g., create) a collection of buckets. A bucket is a virtual representation of, and is created on (e.g., on top of), a virtual disk (e.g., the virtual disk **120**A in FIG. **1**), or other data store. A bucket is like a folder except that a bucket has a hierarchy flat, whereas a folder has recursion (e.g., subfolders). The OVMs **110** store/add one or more objects in/to one or more of the buckets (by storing the one or more objects in one or more virtual disks **120** backing the one or more buckets), and manage the buckets and objects. An object can be anything: a file, a document, a spreadsheet, a video, a data, metadata, etc. When buckets are created, they are assigned (e.g., given) endpoints through which the OVMs **110**, external users or applications interfacing with the OVMs **110**, can access them. Examples of endpoints are uniform resource locators (URLs). After a bucket is created, objects can be added.

[0026] Multiple OVM clusters and/or multiple NOS clusters may exist within a given virtual computing system (e.g., the virtual computing system **100**). The one or more OVM clusters may be referred to as a client layer or object layer. The OVM clusters may be configured to access storage from multiple NOS clusters. Each of the OVM clusters may be configured to access storage from a same NOS cluster. A central management system, such as Prism Central, may manage a configuration of the multiple OVM clusters and/or multiple NOS clusters. The configuration may include a list of OVM clusters, a mapping of each OVM cluster to a list of NOS clusters from which the OVM cluster may access storage, and/or a mapping of each OVM cluster to a list of vdisks that the OVM cluster owns or has access to.

[0027] Each of the OVMs **110** and the CVMs **115** is a software-based implementation of a computing machine in the virtual computing system **100**. The OVMs **110** and the CVMs **115** emulate the functionality of a physical computer. Specifically, the hardware resources, such as CPU, memory, storage, etc., of a single physical server computer (e.g., the first node **105**A, the second node **105**B, or the third node **105**C) are virtualized or transformed by the respective hypervisor (e.g. the hypervisor **125**A, the hypervisor **125**B, and the hypervisor **125**C), into the underlying support for each of the OVMs **110** and the CVMs **115** that may run its own operating system, a distributed operating system, and/or applications on the underlying physical resources just like a real computer. By encapsulating an entire machine, including CPU, memory, operating system, storage devices, and network devices, the OVMs **110** and the CVMs **115** are compatible with most standard operating systems (e.g. Windows, Linux, etc.), applications, and device drivers. Thus, each of the hypervisors **125** is a virtual machine monitor that allows the single physical server computer to run multiple instances of the OVMs **110** (e.g. the OVM **111**) and at least one instance of a CVM **115** (e.g. the CVM **115**A), with each of the OVM instances and the CVM instance sharing the resources of that one physical server computer, potentially across multiple environments. By running the multiple instances of the OVMs **110** on a node of the nodes **105**, multiple workloads and multiple operating systems may be run on the single piece of underlying hardware computer to increase resource utilization and manage workflow.

[0028] The hypervisors **125** of the respective nodes **105** may be configured to run virtualization software, such as, ESXi from VMWare, AHV from Nutanix, Inc., XenServer from Citrix Systems, Inc., etc. The virtualization software on the hypervisors **125** may be configured for managing the interactions between the respective OVMs **110** (and/or the CVMs **115**) and the underlying hardware of the respective nodes **105**. Each of the CVMs **115** and the hypervisors **125** may be configured as suitable for use within the virtual computing system **100**.

[0029] In some embodiments, each of the nodes **105** may be a hardware device, such as a server. For example, in some embodiments, one or more of the nodes **105** may be an NX-1000 server, NX-3000 server, NX-5000 server, NX-6000 server, NX-8000 server, etc. provided by Nutanix, Inc. or server computers from Dell, Inc., Lenovo Group Ltd. or Lenovo PC International, Cisco Systems, Inc., etc. In other embodiments, one or more of the nodes **105** may be another type of hardware device, such as a personal computer, an input/output or peripheral unit such as a printer, or any type of device that is suitable for use as a node within the virtual computing system **100**. In some embodiments, the virtual computing system **100** may be part of a data center.

[0030] The first node **105**A may include one or more central processing units ("CPUs") **130**A, the second node **105**B may include one or more CPUs **130**B, and the third node **105**C may include one or more CPUs **130**C. The CPUs **130**A, **130**B, and **130**C are collectively referred to herein as the CPUs **130**. The CPUs **130** may be configured to execute instructions. The instructions may be carried out by a special purpose computer, logic circuits, or hardware circuits of the first node **105**A, the second node **105**B, and the third node **105**C. The CPUs **130** may be implemented in hardware, firmware, software, or any combination thereof. The term "execution" is, for example, the process of running an application or the carrying out of the operation called for by an instruction. The instructions may be written using one or more programming language, scripting language, assembly language, etc. The CPUs **130**, thus, execute an instruction, meaning that they perform the operations called for by that instruction.

[0031] The first node **105**A may include one or more random access memory units ("RAM") **135**A, the second node **105**B may include one or more RAM **135**B, and the third node **105**C may include one or more RAM **135**C. The RAMs **135**A, **135**B, and **135**C are collectively referred to herein as the RAMs **135**. The CPUs **130** may be operably coupled to the respective one of the RAMs **135**, the storage pool **140**, as well as with other elements of the respective ones of the nodes **105** to receive, send, and process information, and to control the operations of the respective underlying node. Each of the CPUs **130** may retrieve a set of instructions from the storage pool **140**, such as, from a permanent memory device like a read only memory ("ROM") device and copy the instructions in an executable form to a temporary memory device that is generally some form of random access memory ("RAM"), such as a respective one of the RAMs **135**. One of or both of the ROM and RAM be part of the storage pool **140**, or in some embodiments, may be separately provisioned from the storage pool. The RAM may be stand-alone hardware such as RAM chips or modules. Further, each of the CPUs **130** may include a single stand-alone CPU, or a plurality of CPUs that use the same or different processing technology.

[0032] Each of the DAS **145** may include a variety of types of memory devices. For example, in some embodiments, one or more of the DAS **145** may include, but is not limited to, any type of RAM, ROM, flash memory, magnetic storage devices (e.g., hard disk, floppy disk, magnetic strips, etc.), optical disks (e.g., compact disk ("CD"), digital versatile disk ("DVD"), etc.), smart cards, solid state devices, etc. Likewise, the NAS **150** may include any of a variety of network accessible storage (e.g., the cloud storage **155**, the LAN storage **160**, etc.) that is suitable for use within the virtual computing system **100** and accessible via the network **165**. The storage pool **140**, including the NAS **150** and the DAS **145**, together form a distributed storage system configured to be accessed by each of the nodes **105** via the network **165**, one or more of the OVMs **110**, one or more of the CVMs **115**, and/or one or more of the hypervisors **125**.

[0033] Each of the nodes **105** may be configured to communicate and share resources with each other via the network **165**, including the respective one of the CPUs **130**, the respective one of the RAMs **135**, and the respective one

of the DAS **145**. For example, in some embodiments, the nodes **105** may communicate and share resources with each other via one or more of the OVMs **110**, one or more of the CVMs **115**, and/or one or more of the hypervisors **125**. One or more of the nodes **105** may be organized in a variety of network topologies.

[0034] The network **165** may include any of a variety of wired or wireless network channels that may be suitable for use within the virtual computing system **100**. For example, in some embodiments, the network **165** may include wired connections, such as an Ethernet connection, one or more twisted pair wires, coaxial cables, fiber optic cables, etc. In other embodiments, the network **165** may include wireless connections, such as microwaves, infrared waves, radio waves, spread spectrum technologies, satellites, etc. The network **165** may also be configured to communicate with another device using cellular networks, local area networks, wide area networks, the Internet, etc. In some embodiments, the network **165** may include a combination of wired and wireless communications.

[0035] Although three of the plurality of nodes (e.g., the first node **105**A, the second node **105**B, and the third node **105**C) are shown in the virtual computing system **100**, in other embodiments, greater than or fewer than three nodes may be used. Likewise, although only two of the OVMs are shown on each of the first node **105**A (e.g. the OVMs **111**), the second node **105**B, and the third node **105**C, in other embodiments, greater than or fewer than two OVMs may reside on some or all of the nodes **105**.

[0036] It is to be understood again that only certain components and features of the virtual computing system **100** are shown and described herein. Nevertheless, other components and features that may be needed or desired to perform the functions described herein are contemplated and considered within the scope of the present disclosure. It is also to be understood that the configuration of the various components of the virtual computing system **100** described above is only an example and is not intended to be limiting in any way. Rather, the configuration of those components may vary to perform the functions described herein.

Querying Objects on Demand

[0037] FIG. **2** illustrates an example diagram of an object storage service (OSS) **200**, in accordance with some embodiments. The OSS **200** includes a compute layer **202**, a metadata server **204**, and a bucket **206**. The compute layer **202** includes a user interface (UI) service **208**, a metadata service **210** in communication with the metadata server **204**, and an object controller (OC) service **212** in communication with the bucket **206**. Users **214** and IT administrator **216** are in communication with the UI service **208**.

[0038] The UI service **208** includes a processing having programmed instructions (herein, the UI service **208** includes programmed instructions) to receive a query, such as an SQL (structured query language) query, from the users **214** or the IT administrator **216**. In some embodiments, the SQL query is received as an application programming interface (API) call, such as an HTTP-based web API call. In some embodiments, the SQL query is HTTP 1.1 compliant. In some embodiments, the SQL query includes one or more parameters (e.g., query target, bucket name, bucket endpoint, prefix, object attribute relationships, metadata relationships, metadata parameters, a prefix, etc.) specified/indicated by the users **214** or the IT administrator **216**.

[0039] In some embodiments, the UI service **208** includes programmed instructions to determine (e.g., identify) the query target to be a metadata server **204** or a bucket **206**. In some embodiments, if the UI service **208** determines that the query target is a metadata server **204**, the UI service **208** includes programmed instructions to direct the SQL query to the metadata service **210**, where the query is executed on metadata in the metadata server **204**. In some embodiments, if the UI service **208** determines that the query target is a bucket **206**, the UI service **208** includes programmed instructions to direct the SQL query to the OC service **212**, where the query is executed on the bucket **206**. In some embodiments, directing the SQL query includes forwarding the API call. In some embodiments, directing the SQL query includes parsing the API call to identify one or more parameters and sending the one or more parameters. In some embodiments, sending the one or more parameters includes encapsulating the one or more parameters in a message, a frame, a packet, a function, another API call, another SQL query, etc. In some embodiments, the UI service **208** has programmed instructions to execute (e.g., run, evaluate) the SQL query.

[0040] The OC service **212** includes a processing having programmed instructions (herein, the OC service **212** includes programmed instructions) to execute a SQL query on/against one or more objects in the bucket **206**. In some embodiments, the OC service **212** includes programmed instructions to scan the one or more objects in the bucket **206** and generate a list of objects from the one or more objects in the bucket **206**.

[0041] An example SQL query executed by the OC service **212** is shown below:

```
POST /?select&select-type=2/ HTTP/1.1
Host: examplebucket.nutanix.com
Date: Tue, 17 Oct 2017 01:49:52 GMT
Authorization: authorization string
Content-Length: content length
<?xml version="1.0" encoding="UTF-8"?>
<SelectRequest>
    <Expression>Select * from S3Bucket object_inventory
</Expression>
    <ExpressionType>SQL</ExpressionType>
```

[0042] In the example, the word "S3Bucket" indicates that the query target is a bucket (as opposed to a metadata server). The bucket name is "object_inventory." The endpoint (e.g., name, identifier, URL) of the bucket **206** in the example is "examplebucket.nutanix.com." When executing, the OC service **212** executes the query on objects from the bucket **206** having the endpoint "examplebucket.nutanix.com," and the name "object_inventory," as shown based on the command "Select * from S3Bucket object_inventory." In some embodiments, the OC service **212** generates a list of objects from the bucket "object_inventory."

[0043] The OC service **212** can receive the SQL query from the UI service **208**. In some embodiments, the OC service **212** includes programmed instructions to determine the query target to be a bucket. In some embodiments, the OC service **212** includes programmed instructions to execute SQL query on/against one or more objects in the bucket **206** responsive to either receiving the SQL query from the UI service **208** or determining the query target to be a bucket.

[0044] In some embodiments, the OC service **212** includes programmed instructions to identify a predicate/condition) (e.g., a prefix, an object attribute relationship, a metadata relationship) specified in the SQL query. Based on identifying a prefix specified in the SQL query, in some embodiments, the OC service **212** includes programmed instructions to determine which of the one or more objects in the bucket **206** have a prefix matching the prefix in the SQL query. In some embodiments, the list of objects include the objects identified by the OC service **212** as having the matching prefix.

[0045] An example of an SQL query (modifying the above example) to specify a prefix is shown below:

[0046] POST /?select&select-type=2?prefix=megastore/ HTTP/1.1

[0047] In the example, The OC service **212** selects (e.g., filters, prunes, identifies, narrows down the scope of the query to only run on) objects having a prefix matching the prefix "megastore," as shown based on the prefix "prefix=megastore." In some embodiments, the OC service **212** generates a list of the selected objects.

[0048] In some embodiments, the OC service **212** includes programmed instructions to identify object-related/object-oriented attribute relationships (e.g., object attribute relationships) specified in the SQL query. Based on identifying the object attribute relationships in the SQL query, in some embodiments, the OC service **212** includes programmed instructions to determine which of the one or more objects in the bucket **206** have one or more object attributes satisfying (e.g., all of) the object attribute relationships in the SQL query. In some embodiments, the OC service **212** includes programmed instructions to scan the objects of the one or more objects in the bucket **206**. In some embodiments, the OC service **212** includes programmed instructions to generate a list of objects the list of objects including the one or more objects identified by the OC service **212** as having the one or more object attributes satisfying the object attribute relationships in the SQL query. The object attribute relationship may be a relationship (e.g., greater than, less than, equal to, or a combination thereof) between an object attribute (e.g., parameter) and a number (e.g., predetermined threshold, dynamic threshold, function, etc.), a relationship between two object attributes, two or more such relationships (e.g., a range), etc. The object attributes can be object related (e.g., object-oriented). For example, the object can be an "order_inventory" and the object attributes can include "item_type" and "price."

[0049] An example of an SQL query to specify an object attribute is shown below:

```
<Expression>Select * from S3Bucket 'order_inventory' where 'item_type' = 'book' and 'genre' = 'sci_fi'; </Expression>
```

[0050] In the example, The OC service **212** selects objects having "'item_type'='book' and 'genre'='sci_fi'." In some embodiments, the OC service **212** generates a list of the selected objects.

[0051] The metadata service **210** includes a processing having programmed instructions (herein, the metadata service **210** includes programmed instructions) to execute/run a SQL query on/against metadata of the one or more objects in the bucket **206**. In some embodiments, the metadata service **210** includes programmed instructions to determine one or more metadata relationships specified in the SQL query. In some embodiments, the metadata service **210** includes programmed instructions to scan the metadata of the one or more objects in the bucket **206** and generating a list of objects from the one or more objects satisfying the one or more metadata relationships. The metadata relationship may be a relationship between a metadata parameter and a number (e.g., predetermined threshold, dynamic threshold, function, etc.), a relationship between two metadata parameters, two or more such relationships (e.g., a range), etc. Metadata parameters can include last time an object was written to (e.g., updated), last time an object was read, a type (e.g., filetype) of the object, a size of the object, etc.

[0052] An example SQL query executed by the metadata service **210** is shown below:

```
<Expression>Select * from S3Meta where modified >
'current_time( )-
300' and fileType = 'csv' and size >= '2048'; </Expression>
```

[0053] In the example, the word "S3Meta" indicates that the query target is a metadata server. As in the previous example, the endpoint of the bucket **206** in the example is "examplebucket.nutanix.com." When executing, the metadata service **210** executes the query on metadata of objects from the bucket **206** having the endpoint "examplebucket.nutanix.com," as shown based on the command "Select * from S3Meta." Additionally, or alternatively, the metadata service **210** identifies a bucket name specified in the SQL query and executes the query from the bucket **206** having the bucket name. The metadata service **210** selects objects having metadata satisfying the following relationships: "modified>'current_time( )–300' and fileType='csv' and size>='2048'." In this example, the metadata service **210** returns the list of objects in the bucket **206** that have been added in the last 5 minutes, have a file type csv, and are greater than or equal to 2048 bytes.

[0054] Referring now to FIG. **3**, an example method **300** for executing SQL queries is shown, in accordance with some embodiments of the present disclosure. The method **300** may be implemented using, or performed by, the OSS **200**, one or more components of the OSS **200**, or a processor associated with the OSS **200** or one or more components of the OSS **200**, which is detailed herein with respect to FIGS. **1-2**. Additional, fewer, or different operations may be performed in the method **300** depending on the embodiment. Operations of method **300** can be combined with operations of methods **400-500**.

[0055] A processor, such as the processor (e.g., the CPU **130**A) associated with one or more components of the OSS **200**, receives a structured query language (SQL) query (**302**). The processor receives the SQL query from a user (e.g., one of the Users **214**) or an admin (e.g., the IT administrator **216**). In some embodiments, in the SQL query, the user or admin specifies at least one of a bucket name, a bucket endpoint, a query target, or metadata relationships. The processor identifies, in the SQL query, a bucket (**304**). In some embodiments, the bucket is identified based on at least one of the bucket name or the bucket endpoint. In some embodiments, the bucket includes one or more objects. In some embodiments, the processor determines, in the SQL query, a query target to be a metadata server (e.g., the metadata server **204**) associated with the bucket (e.g., the

one or more objects in the bucket). In some embodiments, the processor determines the query target to be a bucket (e.g., the bucket **206**).

[0056] The processor identifies, in the SQL query, metadata relationships (**306**). In some embodiments, the metadata relationships include one or more relationships between a metadata parameter and a number/threshold/limit. In some embodiments, the metadata relationships include one or more relationships between a metadata parameter, a lower number, and a higher number (e.g., a range). The processor executes (e.g. runs, evaluates, processes) the SQL query to generate a list of objects included in (e.g., belonging to, assigned to, etc.) the bucket and having metadata satisfying the metadata relationships (**308**).

[0057] In some embodiments, the processor identifies object attribute relationships. In some embodiment, the processor executes the SQL query to generate a list of objects belonging to the bucket and having object attributes satisfying the object attribute relationships.

[0058] Referring back to FIG. **2**, in some embodiments, one of the components of the compute layer **202** (e.g., the metadata service **210** or the OC service **212**) have programmed instructions to create a temporary bucket (e.g., notational bucket, virtual index). In some embodiments, the temporary bucket has a reduced number of the one or more objects (e.g., a reduced dataset) as compared to the bucket **206**. The temporary bucket includes symbolic links (symlinks) to the objects (e.g., the reduced number of objects) in the bucket **206**. In some embodiments, the temporary bucket does not copy the objects from the bucket **206**, such that the virtual disk backing the bucket **206** has no knowledge of the temporary bucket. The temporary bucket can either be stored in shared memory or as a special object within the bucket **206**. In some embodiments, one of the components of the compute layer **202** has programmed instructions to store the predicate that was used to filter the objects that are contained in the temporary bucket The predicate can be stored in the metadata server **204** as metadata of the temporary bucket.

[0059] An example of creating an object attribute-based temporary bucket is shown below:

```
create data_index 'sci-fi_index' using select object from S3Bucket
‘order_inventory‘ where ‘item_type‘ = ‘book‘ and ‘genre‘ = ‘sci_fi‘
```

[0060] In the example, a temporary bucket “sci-fi_index” is created based on an SQL query executed by the OC Service **212** on objects satisfying the relationship “‘item_type’=‘book’ and ‘genre’=‘sci_fi’.”

[0061] An example of creating a metadata-based temporary bucket is shown below:

```
create index ‘orders_last_24hrs’ using select object from S3Meta
where bucket=‘order_history’ and modified > current_time( ) - 86400
and fileType = “csv”
```

[0062] In the example, a temporary bucket “order_last_24 hrs” is created based on an SQL query executed by the metadata service **210** on objects satisfying the relationship “bucket=‘order_history’ and modified>current_time( )−86400 and fileType=“csv”.” In some embodiments, a keyword identify (e.g., “index”) is used to identify the creation of a temporary bucket.

[0063] In some embodiments, the OC service **212** includes programmed instructions to execute a SQL query against the temporary bucket. Whenever an object (e.g., entry) in the list of the temporary bucket is requested, the request is redirected to the object that is stored in the bucket **206**.

[0064] An example of executing an SQL query on a temporary bucket is shown below:

```
select city, sum(order_total) as city_total from
‘orders_last_24_hrs' group by city order by city_total desc limit 10;
```

[0065] In the example, the SQL query is run against the metadata-based temporary bucket “order_last_24 hrs.” In some embodiments, the OC service **212** includes programmed instructions to generate a list of objects from the temporary bucket. In some embodiments, the OC service **212** includes programmed instructions to filter (e.g., by including further conditions), sort, or filter and sort, the objects in the temporary bucket and generate a list of objects from the filtered/sorted objects. In some embodiments, one of the components of the compute layer **202** deletes the temporary bucket. In some embodiments, one of the components of the compute layer **202** deletes the temporary bucket by executing (e.g., issuing) a “drop table” SQL query. In some embodiments, the SQL query is run against an object attribute-based temporary bucket

[0066] Referring now to FIG. **4**, an example method **400** for executing SQL queries is shown, in accordance with some embodiments of the present disclosure. The method **400** may be implemented using, or performed by, the OSS **200**, one or more components of the OSS **200**, or a processor associated with the OSS **200** or one or more components of the OSS **200**, which is detailed herein with respect to FIGS. **1-2**. Additional, fewer, or different operations may be performed in the method **400** depending on the embodiment. Operations of method **400** can be combined with operations of methods **300** and **500**.

[0067] A processor, such as the processor (e.g., the CPU **130**A) associated with one or more components of the OSS **200**, creates a temporary bucket (**402**). The temporary bucket includes symbolic links to objects belonging to a bucket (e.g., the bucket of method **300**). In some embodiments, the processor creates the temporary bucket by executing an SQL query (e.g., a first SQL query, an SQL query similar to the SQL query of method **300**, a command, a request, a call). In some embodiments, the linked objects have metadata satisfying a predicate (e.g., the metadata relationships of method **300**) specified in the (first) SQL query. In some embodiments, the predicate includes a metadata relationship for when the object was created. In some embodiments, the linked objects are static (e.g., the object attributes thereof are not changing above a predetermined frequency threshold). In some embodiments, the linked objects have object attributes satisfying a predicate (e.g., object attribute relationships) specified in the (first) SQL query.

[0068] The processor receives an SQL query (e.g., a second SQL query) (**404**). The processor identifies, in the (second) SQL query, the temporary bucket (**406**). The processor generates a list of objects to which the temporary bucket is linked (**408**). In some embodiments, the temporary bucket in method **400** is used for OLAP (online analytical processing) workloads, where the second query is run

against a historical dataset in order to, in some embodiments, generate reports for recent time periods (e.g., last 24 hours, last 7 days).

[0069] Referring back to FIG. **2**, in some embodiments, one of the components of the compute layer **202** has programmed instructions to update a temporary bucket as objects are uploaded to the bucket **206**, updated (e.g., object attributes or metadata are updated), or deleted from the bucket **206**. In some embodiments, one of the components of the compute layer **202** has programmed instructions to create an event handler. In some embodiments, the event handler gets invoked whenever an object is either uploaded from the bucket **206**, updated, or deleted (e.g., removed) from the bucket **206**. For example, when an object is updated or a new object is uploaded, and the object is in the bucket **206** that contains objects to which a temporary bucket is linked (e.g., has symlinks), the event handler evaluates (e.g., re-evaluates) the object against the predicate used in creating the temporary bucket. In some embodiments, the object attributes or the metadata of the object is compared to a predicate (e.g., metadata relationships or object attribute relationships) of the temporary bucket. If the object matches (e.g., has metadata/object attributes that match/satisfies) the predicate, then one of the components of the compute layer **202** has programmed instructions to add the object (e.g., a symlink thereof) to the temporary bucket if it is not in the temporary bucket, or retain the object (e.g., a symlink thereof) if it is in the temporary bucket. If the object does not match the predicate, then one of the components of the compute layer **202** has programmed instructions to not add the object (e.g., a symlink thereof) to the temporary bucket if it is not in the temporary bucket, or remove the object (e.g., a symlink thereof) from the temporary bucket if it is in the temporary bucket. In some embodiments, when an object is deleted from the bucket **206**, and one of the components of the compute layer **202** determines that the object is in the temporary bucket, one of the components of the compute layer **202** has programmed instructions to delete the object from the temporary bucket. In some embodiments, removing the object can be deleted at the time that the object needs to be fetched (e.g., during query evaluation). In some embodiments, the temporary buckets are used for OLTP (Online transaction processing) workloads.

[0070] In some embodiments, the OC Service **212** has programmed instructions to select a temporary bucket from multiple temporary buckets against which the one of the OC Service **212** would run a query that is presented to the OC Service **212**. In some embodiments, the OC Service **212** has programmed instructions to lookup the list of temporary buckets in the system and try and run the query against the most relevant temporary bucket. In some embodiments, the OC Service **212** has programmed instructions to determine the most relevant temporary bucket by matching the most relevant temporary bucket (e.g., the name thereof) to the temporary bucket (e.g., the name thereof) specified in the query.

[0071] Referring now to FIG. **5**, an example method **500** for updating temporary buckets is shown, in accordance with some embodiments of the present disclosure. The method **500** may be implemented using, or performed by, the OSS **200**, one or more components of the OSS **200**, or a processor associated with the OSS **200** or one or more components of the OSS **200**, which is detailed herein with respect to FIGS. **1-2**. Additional, fewer, or different operations may be performed in the method **500** depending on the embodiment. Operations of method **500** can be combined with operations of methods **300-400**.

[0072] A processor, such as the processor (e.g., the CPU **130**A) associated with one or more components of the OSS **200**, detects that an object is added to (e.g., uploaded to, updated on) a main bucket (e.g., the bucket **206**) including one or more objects the temporary bucket is linked to (**502**). In some embodiments, the processor invokes (e.g., executes) an event handler (e.g., instructions, method, function, routine) in response to the object being added to the main bucket. The processor (e.g., via the event handler) determines whether the object (e.g., object attributes of the object) or metadata of the object satisfies a predicate (e.g., object attribute relationships or metadata relationships specified/used in creating the temporary bucket) of the temporary bucket (**504**). If the processor determines that the object or metadata satisfies the predicate, the processor adds, to the temporary bucket, the object (e.g., a symbolic link to the object) (**506**). If the processor determines that the object or metadata does not satisfy the predicate, the processor does not add, to the temporary bucket, the object (e.g., a symbolic link to the object) (**508**).

[0073] In some embodiments, the processor detects that an object (e.g., the object attributes or metadata thereof) in the main bucket is being updated. In some embodiments, the processor invoked the event handler in response to the object being updated. In some embodiments, if the processor determines that the object attribute/metadata satisfies the predicate, and the temporary bucket has a symlink to the object, the symlink is retained. In some embodiments, if the processor determines that the object attribute/metadata satisfies the predicate, and the temporary bucket does not have a symlink to the object, the symlink is added. In some embodiments, if the processor determines that the object attribute/metadata does not satisfy the predicate, and the temporary bucket has a symlink to the object, the symlink is removed. In some embodiments, the symlink is marked for removal and is removed at the next execution of an SQL query using the temporary bucket. In some embodiments, if the processor determines that the object attribute/metadata does not satisfy the predicate, and the temporary bucket does not have a symlink to the object, the symlink is not added.

[0074] Each of the elements/entities/components of the virtual computing system **100** and the OSS **200** (e.g., the metadata server **204**, the bucket **206**, the UI service **208**, the metadata service **210**, and the OC service **212**), is implemented using hardware, software, or a combination of hardware or software, in one or more embodiments. For instance, some of the elements or entities of the virtual computing system **100** and the OSS **200** may be implemented as an apparatus comprising custom VLSI circuits or gate arrays, off-the-shelf semiconductors such as logic chips, transistors, or other discrete components. Some of the elements or entities of the virtual computing system **100** and the OSS **200** may be implemented as an apparatus comprising programmable hardware devices such as field programmable gate arrays, programmable array logic, programmable logic devices, or the like. Some of the elements or entities of the virtual computing system **100** and the OSS **200** can include any application, program, library, script, task, service, process or any type and form of executable instructions executed by one or more processors (e.g. the CPU **130**A), in one or more embodiments. Each of the one or more proces-

sors is hardware. The instructions may be stored on one or more computer readable and/or executable storage media including non-transitory storage media such as non-transitory storage media in the storage pool **140** with respect to FIG. **1**.

[0075] In some embodiments, one or more of the components of the OSS **200** are run on (e.g., included in) at least one of one or more of the OVMs **110** or one or more CVMs **115**. In some embodiments, at least of the bucket **206** or the metadata server **204** is backed by one or more vdisks **120** of FIG. **1** and/or a component of the storage pool **140**. In some embodiments, the OSS **200** can include more than one of at least one of the metadata server **204**, the bucket **206**, the UI service **208**, the metadata service **210**, or the OC service **212**).

[0076] It is to be understood that any examples used herein are simply for purposes of explanation and are not intended to be limiting in any way.

[0077] The herein described subject matter sometimes illustrates different components contained within, or connected with, different other components. It is to be understood that such depicted architectures are merely exemplary, and that in fact many other architectures can be implemented which achieve the same functionality. In a conceptual sense, any arrangement of components to achieve the same functionality is effectively "associated" such that the desired functionality is achieved. Hence, any two components herein combined to achieve a particular functionality can be seen as "associated with" each other such that the desired functionality is achieved, irrespective of architectures or intermedial components. Likewise, any two components so associated can also be viewed as being "operably connected," or "operably coupled," to each other to achieve the desired functionality, and any two components capable of being so associated can also be viewed as being "operably couplable," to each other to achieve the desired functionality. Specific examples of operably couplable include but are not limited to physically mateable and/or physically interacting components and/or wirelessly interactable and/or wirelessly interacting components and/or logically interacting and/or logically interactable components.

[0078] With respect to the use of substantially any plural and/or singular terms herein, those having skill in the art can translate from the plural to the singular and/or from the singular to the plural as is appropriate to the context and/or application. The various singular/plural permutations may be expressly set forth herein for sake of clarity.

[0079] It will be understood by those within the art that, in general, terms used herein, and especially in the appended claims (e.g., bodies of the appended claims) are generally intended as "open" terms (e.g., the term "including" should be interpreted as "including but not limited to," the term "having" should be interpreted as "having at least," the term "includes" should be interpreted as "includes but is not limited to," etc.). It will be further understood by those within the art that if a specific number of an introduced claim recitation is intended, such an intent will be explicitly recited in the claim, and in the absence of such recitation no such intent is present. For example, as an aid to understanding, the following appended claims may contain usage of the introductory phrases "at least one" and "one or more" to introduce claim recitations. However, the use of such phrases should not be construed to imply that the introduction of a claim recitation by the indefinite articles "a" or "an" limits any particular claim containing such introduced claim recitation to inventions containing only one such recitation, even when the same claim includes the introductory phrases "one or more" or "at least one" and indefinite articles such as "a" or "an" (e.g., "a" and/or "an" should typically be interpreted to mean "at least one" or "one or more"); the same holds true for the use of definite articles used to introduce claim recitations. In addition, even if a specific number of an introduced claim recitation is explicitly recited, those skilled in the art will recognize that such recitation should typically be interpreted to mean at least the recited number (e.g., the bare recitation of "two recitations," without other modifiers, typically means at least two recitations, or two or more recitations). Furthermore, in those instances where a convention analogous to "at least one of A, B, and C, etc." is used, in general such a construction is intended in the sense one having skill in the art would understand the convention (e.g., "a system having at least one of A, B, and C" would include but not be limited to systems that have A alone, B alone, C alone, A and B together, A and C together, B and C together, and/or A, B, and C together, etc.). In those instances where a convention analogous to "at least one of A, B, or C, etc." is used, in general such a construction is intended in the sense one having skill in the art would understand the convention (e.g., "a system having at least one of A, B, or C" would include but not be limited to systems that have A alone, B alone, C alone, A and B together, A and C together, B and C together, and/or A, B, and C together, etc.). It will be further understood by those within the art that virtually any disjunctive word and/or phrase presenting two or more alternative terms, whether in the description, claims, or drawings, should be understood to contemplate the possibilities of including one of the terms, either of the terms, or both terms. For example, the phrase "A or B" will be understood to include the possibilities of "A" or "B" or "A and B." Further, unless otherwise noted, the use of the words "approximate," "about," "around," "substantially," etc., mean plus or minus ten percent.

[0080] The foregoing description of illustrative embodiments has been presented for purposes of illustration and of description. It is not intended to be exhaustive or limiting with respect to the precise form disclosed, and modifications and variations are possible in light of the above teachings or may be acquired from practice of the disclosed embodiments. It is intended that the scope of the invention be defined by the claims appended hereto and their equivalents.

1. An apparatus comprising a processor having programmed instructions that:

receive a structured query language (SQL) query as an S3 application programming interface (API) call;

identify a bucket of unstructured data residing in an S3-compliant object storage system;

identify metadata relationships specified in the SQL query; and

execute the SQL query to generate a list of objects from the unstructured data included in the bucket, wherein the list of objects have metadata satisfying the metadata relationships.

2. The apparatus of claim **1**, the processor having programmed instructions that:

identify a prefix specified in the SQL query; and

execute the SQL query to generate a second list of objects having an object prefix matching the specified prefix.

3. The apparatus of claim 1, the processor having programmed instructions that:

identify a temporary bucket specified in the SQL query and having links to one or more objects in the bucket; and

execute the SQL query to generate a third list of objects to which the temporary bucket is linked.

4. The apparatus of claim 3, the processor having programmed instructions that create the temporary bucket based upon a second SQL query specifying the metadata relationships.

5. The apparatus of claim 3, wherein the temporary bucket is stored as an object within the bucket.

6. The apparatus of claim 1, wherein the bucket is specified in the SQL query.

7. The apparatus of claim 1, wherein metadata relationships include a range corresponding to a metadata parameter.

8. A non-transitory computer readable storage medium having instructions stored thereon that, upon execution by a processor, causes the processor to perform operations comprising:

receiving a structured query language (SQL) query as an S3 application programming interface (API) call;

identifying a bucket of unstructured data residing in an S3-compliant object storage system;

identifying metadata relationships specified in the SQL query; and

executing the SQL query to generate a list of objects from the unstructured data included in the bucket, wherein the list of objects have metadata satisfying the metadata relationships.

9. The medium of claim 8, the operations further comprising:

identify a prefix specified in the SQL query; and

execute the SQL query to generate a second list of objects having an object prefix matching the specified prefix.

10. The medium of claim 8, the operations further comprising:

identifying a temporary bucket specified in the SQL query and having links to one or more objects in the bucket; and

executing the SQL query to generate a third list of objects to which the temporary bucket is linked.

11. The medium of claim 10, the operations further comprising creating the temporary bucket based upon a second SQL query specifying the metadata relationships.

12. The medium of claim 10, wherein the temporary bucket is stored as an object within the bucket.

13. The medium of claim 8, wherein the bucket is specified in the SQL query.

14. The medium of claim 8, wherein metadata relationships include a range corresponding to a metadata parameter.

15. A computer-implemented method comprising:

receiving, by a processor, a structured query language (SQL) query as an S3 application programming interface (API) call;

identifying, by the processor, a bucket of unstructured data residing in an S3-compliant object storage system;

identifying, by the processor, metadata relationships specified in the SQL query; and

executing, by the processor, the SQL query to generate a list of objects from the unstructured data included in the bucket, wherein the list of objects have metadata satisfying the metadata relationships.

16. The method of claim 15, further comprising:

identify a prefix specified in the SQL query; and

execute the SQL query to generate a second list of objects having an object prefix matching the specified prefix.

17. The method of claim 15, further comprising:

identifying a temporary bucket specified in the SQL query and having links to one or more objects in the bucket; and

executing the SQL query to generate a third list of objects to which the temporary bucket is linked.

18. The method of claim 17, further comprising creating the temporary bucket based upon a second SQL query specifying the metadata relationships.

19. The method of claim 17, wherein the temporary bucket is stored as an object within the bucket.

20. The method of claim 15, wherein the bucket is specified in the SQL query.

* * * * *