

# **REPLICATION LAG-CONSTRAINED DELETION OF DATA IN A LARGE-SCALE DISTRIBUTED DATA STORAGE SYSTEM**

## **TECHNICAL FIELD**

**[0001]** The present invention relates to distributed data storage systems. More particularly, the present invention relates to replication lag-constrained deletion of data in a large-scale distributed data storage system.

## **BACKGROUND**

**[0002]** Today, many online services, including many Internet services used by users around the globe, are implemented as complex, large-scale distributed computing systems. These online services are often constructed from collections of software applications developed by different software development teams, often in different software programming languages. The collection of software applications may span hundreds or thousands of computing machines, across multiple data center facilities.

**[0003]** Because of this complexity, the architecture of an online service is typically structured in “tiers” with each tier composed of many computing machines. The tiers are conceptually stacked on top of one another from the perspective of processing network requests received over a data communications network (e.g., the Internet) from end-user devices and generating network responses to the network requests that are sent back over the data communications network to the end-user devices.

**[0004]** One of the tiers is typically composed of a large-scale distributed data storage system for persisting and retrieving data used by applications in an “application tier” of the online service. The application tier conceptually sits on top of the data storage system tier and may implement much of the end-user facing functionality of the online service. The “application” data used by the applications may include, for example, information provided by end-users, metadata about such information or any other information used by the applications as a part of providing the online service to end-users.

**[0004a]** Aspects of the invention are defined in the independent claims. Some optional features of the invention are defined in the dependent claims.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

**[0005]** FIG. 1 shows a large-scale distributed data storage system, per an example of the present techniques.

**[0006]** FIG. 2 shows a database server engine of a distributed data storage system cluster, per an example of the present techniques.

**[0007]** FIG. 3 shows a local replication process, per an example of the present techniques.

**[0008]** FIG. 4 shows a geographic replication process, per an example of the present techniques.

**[0009]** FIG. 5 shows a process for replication lag-constrained deletion of data, per an example of the present techniques.

**[0010]** FIG. 6 shows a process for adjusting a pause interval based on measured replication lag, per an example of the present techniques.

**[0011]** FIG. 7 illustrates a basic hardware machine that may be used utilized to implement an example of the present techniques.

**[0012]** FIG. 8 illustrates a non-limiting example of a basic software system for controlling the operation of the basic hardware machine.

#### DETAILED DESCRIPTION OF EMBODIMENTS

**[0013]** An online service provider may need to delete data from a large-scale distributed data storage system. Deleting data may be needed for various reasons. A common reason is to free up data storage space for new data. For example, data stored for users that no longer use the online service may be deleted to make room for new user data.

**[0014]** One challenge that may be faced by the online service provider when deleting data is that there may be a large amount of data to delete. For example, the distributed data storage system may store many petabytes of data or more. Thus, deleting even a fraction of the total amount of stored data can still involve deleting a substantial amount of data (e.g., many terabytes or more).

**[0015]** Given the large amount of data that may be targeted for deletion, there may be a tendency to attempt to delete all the targeted data in a single database transaction to delete the data as quickly as possible and thereby free up storage space as quickly as possible. For example, a single database transaction may be submitted to each database server of the storage system which may process the operation on its respective database. However, because of the large amount of data targeted by the transaction, this approach can quickly consume substantial computing resources of the database servers possibly even to the point of detrimental impact on the processing of requests from the application tier. The impact may be noticeable to end-users of the online service by the service's lack of usual responsiveness.

**[0016]** Another possible approach may be to delete portions of the targeted data in separate transactions. In this case, the separate transactions may be submitted to each database server on a regular predefined time interval. However, if a predefined time interval is selected that is too small, then the above-mentioned problems associated with a single large transaction may be encountered. On the other hand, if a predefined time interval is selected that is too large, it may take too long to delete all the targeted data.

**[0017]** OVERVIEW OF REPLICATION-LAG CONSTRAINED DELETION OF DATA

**[0018]** The foregoing problems and other problems with possible approaches for deleting data in a large-scale distributed data storage system, may be addressed by techniques described, suggested, and implied herein, which include systems and methods for replication lag-constrained deletion of data in a large-scale distributed data storage system. The techniques may be used in place of or in conjunction with the existing approaches for deleting data in a large-scale distributed data storage system.

**[0019]** The techniques account for a recognition that deleting a large-amount of data from a database in a distributed data storage system too quickly can cause a replication sub-system of the distributed data storage system to suffer substantial performance degradation or even fail. The replication sub-system may be used to replicate data stored in “master” databases of the distributed data storage system to “slave” databases of the distributed data storage system. If the delete rate is too high, the replication sub-system may suffer substantial performance degradation or even fail because of the processing load the delete rate places on disk and network I/O and CPU resources used by the replication sub-system when processing the replication events generated as consequence of deleting the data.

**[0020]** In one exemplary arrangement, the rate at which data is deleted from a database is constrained by measured replication lag. Generally, replication lag refers to the time delay between when data is stored in a master database and when the data replicated by the replication sub-system is stored in a slave database. Some replication lag is expected. However, a large replication lag can be indicative of a replication sub-system that is under stress and might be about to fail.

**[0021]** In one exemplary arrangement, the techniques include a method for replication-lag constrained deletion of data in a distributed data storage system. The method is performed by a “data vacuum” computing system comprising one or more processors and memory. The method includes the data vacuum serially and periodically submitting commands to a database server of the distributed data storage system. The commands are to delete data from a corresponding database. After each submission of a command, the data vacuum pauses (sleeps) for a time before submitting the next command. The length of the time that the data vacuum pauses between submissions of commands is referred to herein as the “pause interval.”

**[0022]** In one exemplary arrangement, the pause interval has an initial default value and is automatically adjusted by the data vacuum thereafter based on measured replication lag. The data vacuum may periodically shorten the pause interval starting from the initial default value by a decrement amount while both a local slave database replication lag metric is below

a local slave replication lag threshold and a geo-slave database replication lag metric is below a geo-slave replication lag threshold. The local slave database replication lag metric may be based on a single measurement or periodic measurements of the replication lag between a master database server and a slave database server that are co-located in the same data center. The geo-slave database replication lag metric may be based on a single measurement or periodic measurements of the replication lag between a master database server and a slave database server that are in different, geographically distributed data centers.

**[0023]** In one exemplary arrangement, while a local slave database replication lag metric is above the local slave replication lag threshold and/or a geo-slave database replication lag metric is above the geo-slave replication lag threshold, the data vacuum may periodically lengthen the pause interval starting from the then current value by an increment amount. The data vacuum can return to periodically shortening the pause interval after both a local slave database replication lag metric and a local slave replication lag metric and are again below their respective thresholds. The data vacuum may continue this process of periodically shortening and lengthening the pause interval based on local and geo slave replication lag metrics thereby maintaining a delete rate constrained by the local and geo slave replication lag thresholds.

**[0024]** The techniques disclosed herein for replication lag-constrained deletion of data in a distributed data storage system improve a computing system comprising one or more processors and memory for deleting data in the distributed data storage system. The improvement results from preventing too high of a delete rate that causes severe replication lag while at the same time adjusting the delete rate over time to a maximum allowable delete rate constrained by measured replication lag in terms of both local replication lag and geographic replication lag.

**[0025]** TERMINOLOGY

**[0026]** Reference will now be made in detail to embodiments, examples of which are illustrated in the accompanying drawings. In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the various described embodiments. However, it will be apparent to one of ordinary skill in the art that the various described embodiments may be practiced without these specific details. In other instances, well-known methods, procedures, components, circuits, and networks have not been described in detail so as not to unnecessarily obscure aspects of the embodiments.

**[0027]** It will also be understood that, although the terms first, second, etc. are, in some instances, used herein to describe various elements, these elements may not be limited by these terms, depending on context. These terms may be used only to distinguish one element from another, depending on context. For example, a first database server may be termed a

As used herein, being within a threshold means that a value for an item under comparison is between two specified other values, that an item under comparison is among a middle specified number of items, or that an item under comparison has a value within a middle specified percentage range. Relative terms, such as high or unimportant, when not otherwise defined, can be understood as assigning a value and determining how that value compares to an established threshold. For example, the phrase "severe replication lag" can be understood to mean a replication lag metric that is above a threshold.

#### **[0032] LARGE-SCALE DISTRIBUTED DATA STORAGE SYSTEM ENVIRONMENT**

**[0033]** While the present techniques may be implemented using a single computing machine, the present techniques are preferably implemented using multiple computing machines in a distributed computing environment. FIG. 1 shows an example of a distributed computing environment 100, per an exemplary arrangement.

**[0034]** Environment 100 includes two data centers labeled DC1 and DC2. Each data center DC1 and DC2 may include a facility or building for co-locating computing systems and associated components such as data network communications equipment, data storage equipment, and cooling equipment. Data centers DC1 and DC2 may be located at a geographic distance from one another. The geographic distance may be many miles. For example, data center DC1 may be in San Francisco, California, U.S.A. and data center DC2 may be in New York, New York, U.S.A. It is also possible for data centers DC1 and DC2 to be in different countries. In general, however, the geographic distance between data centers DC1 and DC2 may be at least a few miles.

**[0035]** While in an embodiment the distributed computing environment includes only two data centers, the distributed computing environment may include more than two data centers in another embodiment. In this case, the distributed computing environment may be viewed as being composed of pairs of data centers (data center peers) of which the environment 100 depicted in FIG. 1 is representative of each such pair (peers).

**[0036]** As shown in FIG. 1, each data center DC1 and DC2 may include an application tier, labeled Application Tier-1 and Application Tier-2 in FIG. 1, respectively. Each application tier may be composed of multiple computing machines that execute processes that operate as network clients of a respective distributed data storage system cluster. As shown in FIG. 1, a distributed data storage system cluster labeled Cluster-1 serves client processes in Application Tier-1 and a distributed data storage system cluster labeled Cluster-2 serves client processes in Application Tier-2. Each distributed data storage system cluster is also composed of multiple computing machines that execute respective database server engines.

The computing machines of Cluster-1 execute database server engines Server Engine 1-1, Server Engine 1-2, ... Server Engine 1-N and the computing machines of Cluster-2 execute Server Engine 2-1, Server Engine 2-2, ... Server Engine 2-M. The distributed data storage system clusters Cluster-1 and Cluster-2 may have the same or a different number of server engines. Details of an example database server engine are described below with respect to FIG. 2.

**[0037]** While in an embodiment different computing machines are used to execute the client processes of the application tier and the server engines of the distributed data storage system cluster, the same computing machine may execute one or more of the client processes and one or more of the server engines in another embodiment.

**[0038]** Various computing machines of the application tier and the distributed data storage system cluster in a data center may be interconnected by one or more data communications networks. Such a data communications network may support various network communications protocols for sending and receiving network messages (e.g., network requests and network responses thereto) between the various computing machines. Non-limiting examples of network communications protocol suitable for implementing an example of the present techniques include the Hyper Text Transfer Protocol (HTTP), the Secure Hyper Text Transfer Protocol (HTTPS), and/or other Internet Protocol (IP)-based network communications protocol.

**[0039]** In an embodiment, data stored in a distributed data storage system cluster is sharded (horizontally partitioned) over the server engines of the distributed data storage system cluster. As used herein, a shard 110 refers to a horizontal partition of a database. Each of the server engines may store multiple of the shards. In a non-limiting exemplary embodiment, a distributed data storage system cluster contains approximately two thousand (2,000) shards distributed across approximately two hundred and fifty (250) server engines.

**[0040]** Per an embodiment, in operation, a client process that executes in an application tier may connect to and request data from any of the server engines of the distributed data storage system cluster in the data center. The server engine receiving the request from the client process may inspect the request to determine which shard stores the data that the request pertains to. If the server engine receiving the request does not store the target shard, the server engine may redirect the request to one of the other server engines in the cluster that does store the target shard. In an embodiment, the distributed data storage system cluster may include a cache (not shown in FIG. 1) to improve performance of processing read requests from the application tier. The cache may be partitioned and replicated for high-availability.

another embodiment. For example, databases 206, 210, and 214 may be logically structured per a hierarchical, network, object, document, graph, key-value, or another logical database data model. Similarly, while in an embodiment the database servers 204, 208, and 212 are capable of processing database requests that are formulated per SQL or like query language, the database servers 204, 208, and 212 are configured to process database requests that are structured per a different query language in another embodiment. In general, virtually any database query language that supports commands for creating, reading, updating, and deleting data in the databases 206, 210, and 214 may be used.

**[0047]** Database servers 204, 208, and 212 may be configured to replicate data between the databases 206, 210, and 214 in a master-slave configuration. For example, data stored in first database 206 may be replicated to second database 210 and data stored in second database 210 may be replicated to third database 214. In this example, database 206 is a “master” database with respect to “slave” database 210 and database 210 is a “master” database with respect to “slave” database 214. Thus, database data changes applied to first database 206 may first be replicated to second database 210 and then from second database 210 to third database 214.

**[0048]** While in an embodiment such as shown in FIG. 2 a database server engine comprises three database servers and three databases, a database server engine comprises just two database servers and two databases in another embodiment. For example, database server engine 200 may comprise just database servers 204 and 208 and respective databases 206 and 210 arranged in a master-slave replication configuration.

**[0049]** While in an embodiment such as shown in FIG. 2 a database server engine comprises a core server for translating client requests from client processes in the application tier to database requests sent to the database servers 204, 208, and 212, a database server engine does not include a core server. In this case, the client processes in the application tier may send database request directly to the database servers 204, 208, and 212.

#### **[0050] LOCAL REPLICATION**

**[0051]** FIG. 3 shows a local replication process 300, per an exemplary arrangement. The process 300 involves a master database server 302 and a slave database server 304 of the same database server engine 306. Master database server 302 receives (S308) create, update, and delete commands from client processes in an application tier and delete commands from a data vacuum computing system as described in greater detail below. The create, update, and delete commands may be formulated per the Structure Query Language (SQL) or the like. For example, the create, update, and delete commands may be SQL INSERT, UPDATE, and DELETE commands respectively. The database commands may be

implemented on a second different computing machine. Alternatively, all the components may be implemented on one computing machine.

**[0061]     GEOGRAPHIC REPLICATION**

**[0062]**     FIG. 4 shows a geographic replication process 400, per an exemplary arrangement. The geographical replication process 400 uses an event stream processor to move replication events between data centers. As shown, a first data center 406 houses a master database server 402, a master database 414, and a binary log 418. A second geographically distant data center 408 houses a slave database server 404, a slave database 434, and a relay log 428. Replication of database events stored in binary log 418 to the relay log 428 is facilitated by an event stream processor. The event stream processor may include an event producer 420, an event stream log 422, and an event consumer 424. While the components of the event stream processor are not shown in FIG. 4 as residing in any data center, some or all the components may reside in data center 406 and/or data center 408.

**[0063]**     Steps S410, S414, and S416 of the geographic replication process 400 may be like Steps S308, S310, and S314 of the local replication process 300, respectively. However, for the geographical replication process 400, the event producer 420 may read or otherwise obtain (S436) replication events from the binary log 418 in a first in first out order and store them or otherwise (S438) cause the replication events to be stored in the event stream log 422 in that order. Event stream log 422 is implemented as an append-only distributed commit log in an embodiment. The event consumer 424 reads or otherwise obtains the replication events (S440) from the event stream log in a first in first out order and stores the replication events in the relay log 428 in that order. The slave database server 404 may then read (S430) the replication events from the relay log 428 and apply them (S432) to the slave database 434 in first in first out order. In this way, replication events may be “streamed” from the master database 414 to the slave database 434 via an event stream processor.

**[0064]**     Various factors can contribute to replication lag in the geographic replication process 400 including time spent by the master database server 402, the event producer 420, the event consumer 424, and the slave database server 404 performing operations that are generally performed serially with respect to a given replication event. The operations that may contribute to geographic replication lag may include the master database server 402 writing or otherwise causing (S416) the replication event to be stored in the binary log 418, the event producer reading or otherwise obtaining (S436) the replication event from the binary log 418 and writing or otherwise causing (S438) the replication event to be stored in the event stream log 422, the event consumer reading or otherwise obtaining (S440) the replication event from the event stream log 422 and writing or otherwise causing (S426) the

replication event to be stored in the relay log 428, and the slave database server 404 reading or otherwise obtaining (S430) the replication event from the relay log 326, and applying (S434) the replication event to the slave database 434.

**[0065]** The various components depicted in FIG. 4 may be implemented on a computing system comprising one or more processors and memory. The one or more processors and memory of the computing system may be provided by computing machines. Although components are shown separately in FIG. 4, various components may be implemented on different computing machines or the same computing machine. For example, master server 402, master database 414, binary log 418, and event producer 420 may be implemented on a first computing machine in data center 406 and slave server 404, slave database 434, relay log 428, and event consumer 424 may be implemented on a second different computing machine in data center 408.

#### **[0066] PROCESS FOR REPLICATION LAG-CONSTRAINED DELETION OF DATA**

**[0067]** FIG. 5 shows a process 500 for replication-lag constrained deletion of data in a distributed data storage system, per an exemplary arrangement. The process 500 may be implemented by a computing system comprising one or more processors and memory. The one or more processors and memory of the computing system may be provided by one or more computing machines. For purposes of providing clear examples, the process 500 is described below as being performed by a “data vacuum” computing system.

**[0068]** At operation S502, the data vacuum obtains a delete task to perform. The data vacuum may obtain the delete task via a command line interface, a graphical user interface, and/or a configuration file interface, per an embodiment. The delete task targets data objects stored in a database of a distributed data storage system that is a master database for the data objects. The delete task may target many data objects. For example, the delete task may target tens of thousands of database table rows or more.

**[0069]** To delete the target data objects, the data vacuum serially submits a series of database commands to the master database server of the master database for the data objects. After the submission of each command (S508), the data vacuum sleeps (S510) for a pause interval. Sleeping may be accomplished by invoking a system call or a standard library call that pauses execution of the data vacuum process, or a thread thereof, for a length of time (e.g., the pause interval) where the length of time is specified as a parameter to the call.

**[0070]** The pause interval may be periodically adjusted by the data vacuum as described below with respect to FIG. 6. After submitting a database command (S508) and sleeping (S510), if there are still more target data objects to delete (S504), then the process 500

continues with submission (S508) of another database command. Otherwise, if all the target data objects have been deleted, then the process 500 ends (S506).

**[0071]** In an embodiment, each database command is a Structure Query Language (SQL) DELETE command that specifies a maximum number of data objects to delete by the database command. The maximum number of data objects to delete may be specified by the SQL LIMIT clause, in an embodiment. In an embodiment, the maximum number of data objects to delete ranges between 1,000 to 10,000. The master database server may execute the command in the context of a database transaction.

**[0072]** The data vacuum can perform process 500 in parallel against multiple master database servers if the data objects targeted by the delete task are mastered by more than one database in the distributed data storage system. In this case, the database commands may be submitted serially against each of the respective master database servers.

#### **[0073] ADJUSTING THE PAUSE INTERVAL**

**[0074]** Initially, the pause interval may start a default value for a given delete task. Thereafter, the pause interval may be adjusted by the data vacuum during execution of the delete task based on measured replication lag. FIG. 6 shows a process 600 for adjusting the pause interval based on measured replication lag, per an exemplary arrangement.

**[0075]** Process 600 may be performed by the data vacuum for each master database server operating on a database that master's data objects targeted for deletion by the delete task. If there are multiple such master database servers, the process 600 may be performed concurrently for each such master database server.

**[0076]** At operation S602, the data vacuum determines a local slave replication lag metric. The metric may be determined based on a single local slave replication lag measurement or multiple local slave replication lag measurements. If multiple local slave replication lag measurements are used to determine the local slave replication lag metric, then the metric may be computed as an average, mean, weighted average, or weighted mean of the multiple measurements. In an embodiment, a local slave replication lag measurement is a time-based value such as seconds or millisecond representing the local replication lag.

**[0077]** Various techniques may be employed by the data vacuum to measure the local replication lag. In one embodiment, a "heartbeat" technique is used. Per the heartbeat technique, a database object (e.g., a database table row of a database table) is periodically insert or updated (e.g., every few seconds) in the master database by the data vacuum with a current timestamp reflecting a current system clock time at the time of the update. After the insert or update to the data object is replicated to the local slave database, the timestamp is

**[0081]** The data vacuum may periodically perform process 600 to adjust the pause interval such that the delete rate of data from the master database when executed the delete task is constrained by the local and geographic replication lag. By doing so, the data vacuum avoids overwhelming the distributed data storage system, including the replication sub-system, with delete commands.

**[0082]** While in an embodiment the data vacuum can sleep for a pause interval after the submission of each delete command in step S510 irrespective of whether the local slave replication lag metric or the geographical slave replication lag metric is currently above its respective threshold, the data vacuum sleeps for a pause interval only while either the local slave replication lag metric or the geographical slave replication lag metric is currently above its respective threshold in another embodiment. In this other embodiment, the data vacuum may not sleep for any length of time after submission of delete commands so long as and while both the local slave replication lag metric and the geographical slave replication lag metric remain below their respective thresholds. Once one or both the metrics is above a threshold, then the data vacuum may sleep for a pause interval after submission of delete commands. By doing so, the data vacuum may delete data from the distributed data storage system at a faster delete rate compared to a configuration where the data vacuum pauses for some time after submission of delete commands even when both the local slave and geographical slave replication lag metrics are below their respective thresholds.

**[0083] BASIC IMPLEMENTING MECHANISMS**

**[0084]** The present techniques may be implemented using a computing system comprising one or more processors and memory. The one or more processors and memory may be provided by one or more hardware machines. FIG. 7 illustrates an example of a basic hardware machine 700 that may be used to implement the present techniques, per an exemplary arrangement. Hardware machine 700 and its hardware components, including their connections, relationships, and functions, is meant to be exemplary only, and not meant to limit implementations of the present invention. Other hardware machines suitable for implementing the present techniques may have different components, including components with different connections, relationships, and functions.

**[0085]** Hardware machine 700 includes a bus 702 or other communication mechanism for addressing a main memory 706 and for transferring data between and among the various components of hardware machine 700.

**[0086]** Hardware machine 700 also includes a processor 704 coupled with bus 702 for processing information. Processor 704 may be a general-purpose microprocessor, a system on a chip (SoC), or another hardware processor.

**[0093]** A cursor control 716, such as a mouse, a trackball, touchpad, touch-sensitive surface, or cursor direction keys for communicating direction information and command selections to processor 704 and for controlling cursor movement on display 712, may be coupled to bus 702. Cursor control 716 may have two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane. Cursor control 716 may have more degrees of freedom with a third axis (e.g., z). For example, cursor control 716 may have three translational degrees of freedom (e.g., surge, heave, and sway) in three perpendicular axes, that allows the device to specify position in the three axes. Cursor control 716 may have three rotational degrees of freedom (e.g., pitch, yaw, roll) about three perpendicular axes, that allows the device to specify an orientation about the three axes.

**[0094]** While one or more of display 712, input device 714, and cursor control 716 may be external components (i.e., peripheral devices) of hardware machine 700, some or all of display 712, input device 714, and cursor control 716 may be integrated as part of the form factor of hardware machine 700.

**[0095]** A function or operation of the present techniques may be performed by hardware machine 700 in response to processor 704 executing one or more programs of software instructions contained in main memory 706. Such software instructions may be read into main memory 706 from another storage medium, such as a storage device 710. Execution of the software instructions contained in main memory 706 cause processor 704 to perform the function or operation.

**[0096]** While a function or operation of the present techniques may be implemented entirely with software instructions, hard-wired or programmable circuitry of hardware machine 700 (e.g., an ASIC, a FPGA, or the like) may be used in place of or in combination with software instructions to perform the function or operation.

**[0097]** The term “storage media” as used herein refers to any non-transitory media that store data and/or software instructions that cause a hardware machine to operate in a specific fashion. Such storage media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, non-volatile random access memory (NVRAM), flash memory, optical disks, magnetic disks, or solid-state drives, such as storage device 710. Volatile media includes dynamic memory, such as main memory 706. Common forms of storage media include, for example, a floppy disk, a flexible disk, hard disk, solid-state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, flash memory, any other memory chip or cartridge.

**[0103]** Hardware machine 700 can send messages and receive data, including program code, through network 722, network link 720, and communication interface 718. In the Internet example, a server might transmit a requested code for an application program through Internet, ISP, and network 722 and communication interface 718.

**[0104]** The received code may be executed by processor 704 as it is received, and/or stored in storage device 710, or other non-volatile storage for later execution.

**[0105]** FIG. 8 illustrates basic software system 800 that may be employed for controlling the operation of hardware machine 700 of FIG. 7, per an exemplary arrangement. Software system 800 and its software components, including their connections, relationships, and functions, is meant to be exemplary only, and not meant to limit implementations of the present techniques. Other software systems suitable for implementing the present techniques may have different components, including components with different connections, relationships, and functions.

**[0106]** Software system 800 is provided for directing the operation of hardware machine 700. Software system 800 may be stored in system memory (RAM) 706 and on fixed storage (e.g., hard disk or flash memory) 710.

**[0107]** Software system 800 includes a kernel or operating system (OS) 810. OS 810 manages low-level aspects of computer operation, including managing execution of processes, memory allocation, file input and output (I/O), and device I/O.

**[0108]** Software system 800 includes one or more application programs, represented as 802A, 802B, 802C ... 802N, that may be “loaded” (e.g., transferred from fixed storage 710 into memory 706) for execution by hardware machine 700. The applications or other software intended for use on hardware machine 700 may also be stored as a set of downloadable computer-executable instructions, for example, for downloading and installation from an Internet location (e.g., a Web server, an app store, or other online service).

**[0109]** Software system 800 includes a graphical user interface (GUI) 815, for receiving user commands and data in a graphical (e.g., “point-and-click” or “touch gesture”) fashion. These inputs, in turn, may be acted upon by the system 800 in accordance with instructions from operating system 810 and/or application(s) 802. GUI 815 also serves to display the results of operation from the OS 810 and applications 802, whereupon the user may supply additional inputs or terminate the session (e.g., log off).

**[0110]** Software system 800 can execute directly on bare hardware 820 (e.g., machine 700). Alternatively, a “Type-1” hypervisor 830 may be interposed between the bare hardware 820 and OS 810 as part of software system 800. Hypervisor 830 acts as a software “cushion” or virtualization layer between the OS 810 and bare hardware 820. Hypervisor 830