(12)Indian Patent Application

(21) Application Number: 202341051971

(22) Filing Date: 02/08/2023 (43) Publication Date: 11/07/2025

(71) Applicant(s): L&T TECHNOLOGY SERVICES LIMITED

(72) Inventor(s): Singha, Zubraj

(51) International Classifications: G06F 9/44 G06F 12/08 G06F 12/0871 G06F 12/084

(54) Title: METHOD AND SYSTEM OF CACHING IN A VIRTUAL COMPUTING ENVIRONMENT USING NESTED LRU TECHNIQUES

(57) Abstract: A method of caching memory in a virtual computing environment is disclosed. The method includes enabling access of at least one persistent disk (140) by at least one container of at least one virtual machine (104) via container cache (202). The container cache (202) is enabled by a hypervisor (102) in virtual computing environment (100) separate from a hypervisor cache (114). The access of at least one persistent disk (140) by at least one container via container cache (202) is based on caching of data of at least one persistent disk (140) based on container cache counter (204) implementing first least-recently-used technique. Further, the caching of data of at least one persistent disk (140) based on container cache counter (204) includes caching data of at least one persistent disk (140) via one of plurality of sub-caches (206) of container cache (202). Caching data of at least one persistent disk (140) is based on sub-cache counter (214) of one of the plurality of sub-caches (206) implementing second least recently-used technique.

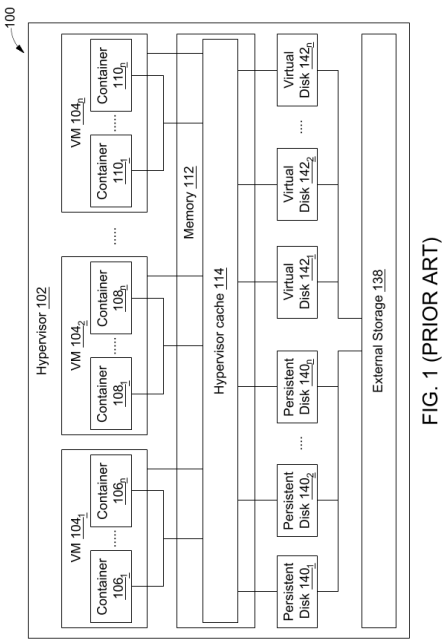


FIG. 1 (PRIOR ART)

# FORM 2

THE PATENTS ACT 1970
(39 OF 1970)
&
The Patent Rules, 2003

# Complete Specification
(See Section 10 and Rule 13)

**1. TITLE OF THE INVENTION**

**METHOD AND SYSTEM OF CACHING IN A VIRTUAL COMPUTING ENVIRONMENT USING NESTED LRU TECHNIQUES**

**2. APPLICANT(S)**


(a) NAME : **L&T TECHNOLOGY SERVICES LIMITED**
(b) NATIONALITY : **INDIAN**
(c) ADDRESS : DLF IT SEZ Park, 2nd Floor – Block 3
1/124, Mount Poonamallee Road,
Ramapuram, Chennai – 600 089,
INDIA.


**3. PREAMBLE TO THE DESCRIPTION**

**COMPLETE**

The following specification describes the invention and the manner in which it is to be performed

## DESCRIPTION

Technical Field

**[001]** This disclosure relates generally to a virtual environment in a server, more particularly to a method and system of caching in a virtual environment.

## BACKGROUND

**[002]** With advent of the new technological reforms, a hypervisor is used to run multiple Virtual Machines (VM) independently of each other in a virtual environment. Further, most of the applications and software running on the VM has been broken down from monolithic to microservices architecture that uses multiple containers to do the processing independently. Traditionally, the hypervisor deploys an in-memory hypervisor cache to optimize reads on the storage and accordingly, the hypervisor cache caters to millions of I/O requests from each of the VM and the containers. Further, the containers and the VMs access the same storage from the hypervisor and have to use the same hypervisor cache which is shared between all the other VM's and the containers. In some cases, the hypervisor cache uses a Least Recent Used (LRU) technique. However, the LRU technique is proven to be ineffective in case the data is sparse and spread across many data stores and disks.

**[003]** Since the hypervisor cache has to cater to all the read I/O requests on the hypervisor, the containers read I/O's are not fully optimized and due to this the container read I/O's have to bypass multiple layers of hypervisor storage to fetch the required data. Thus, when a large number of containers frequently issue reads through the hypervisor cache, there is a performance bottleneck for the container read requests.

**[004]** Thus, there is a need to provide a faster method and system of caching in a virtual environment, which may cater to the container I/O in an efficient and faster manner.

## SUMMARY OF THE INVENTION

**[005]** In an embodiment, a method caching memory in a virtual computing environment is disclosed. The method may include enabling a processor to access at least one persistent disk by at least one container of at least one virtual machine (VM) via a container cache. In an embodiment, the container cache may be enabled by a hypervisor in the virtual

computing environment separate from a hypervisor cache. Further, the access of the at least one persistent disk, by the at least one container via the container cache may be based on caching of data of the at least one persistent disk based on a container cache counter implementing a first least-recently-used technique. Further, the caching of the data of the at least one persistent disk based on the container cache counter may include caching the data of the at least one persistent disk via one of a plurality of sub-caches of the container cache. In an embodiment, the caching of the data of the at least one persistent disk may be based on a sub-cache counter of the one of the plurality of sub-caches implementing a second least-recently-used (LRU) technique.

**[006]** In another embodiment, a system of caching memory in a virtual environment is disclosed. The system may include a processor and a memory coupled to the processor. The memory may store processor-executable instructions which when executed may cause the processor to enable access of at least one persistent disk by at least one container of at least one virtual machine (VM) via a container cache. In an embodiment, the container cache may be enabled by a hypervisor in the virtual computing environment separate from a hypervisor cache. Further, the access of the at least one persistent disk, by the at least one container via the container cache may be based on caching of data of the at least one persistent disk based on a container cache counter implementing a first least-recently-used technique. Further, the caching of the data of the at least one persistent disk based on the container cache counter may include caching the data of the at least one persistent disk via one of a plurality of sub-caches of the container cache. In an embodiment, the caching of the data of the at least one persistent disk may be based on a sub-cache counter of the one of the plurality of sub-caches implementing a second least-recently-used (LRU) technique.

**[007]** Various objects, features, aspects, and advantages of the inventive subject matter will become more apparent from the following detailed description of preferred embodiments, along with the accompanying drawing figures in which like numerals represent like components.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[008]** The accompanying drawings, which are incorporated in and constitute a part of this disclosure, illustrate exemplary embodiments and, together with the description, serve to explain the disclosed principles.

**[009]** **FIG. 1** illustrates an architecture diagram for caching memory in a virtual computing environment, in accordance with a prior art embodiment.

**[010]** **FIG. 2** illustrates an architecture diagram for caching memory in a virtual computing environment using nested LRU techniques, in accordance with an embodiment of the current disclosure.

**[011]** **FIG. 3** illustrates a working of an exemplary LRU technique for caching memory in a virtual computing environment, in accordance with an embodiment of the current disclosure.

**[012]** **FIG. 4** illustrates a flowchart describing the methodology for caching memory in a virtual computing environment using nested LRU techniques, in accordance with an embodiment of the current disclosure.

## DETAILED DESCRIPTION OF THE DRAWINGS

**[013]** Exemplary embodiments are described with reference to the accompanying drawings. Wherever convenient, the same reference numbers are used throughout the drawings to refer to the same or like parts. While examples and features of disclosed principles are described herein, modifications, adaptations, and other implementations are possible without departing from the scope of the disclosed embodiments. It is intended that the following detailed description be considered as exemplary only, with the true scope being indicated by the following claims. Additional illustrative embodiments are listed.

**[014]** In the figures, similar components and/or features may have the same reference label. Further, various components of the same type may be distinguished by following the reference label with a second label that distinguishes among the similar components. If only the first reference label is used in the specification, the description is applicable to any one of the similar components having the same first reference label irrespective of the second reference label.

**[015]** Presently, hypervisor may be responsible for coordinating VM operations and for managing physical resources of the host server. A hypervisor cache may be deployed by the hypervisor such as CBRC cache in ESX hypervisor to optimize the read requests from plurality of Virtual Machines (VMs) running in a virtual computing environment. However,

there may be a plurality of containers running in each of the plurality of VM's which may also issue I/O requests to the hypervisor cache, due to which the hypervisor cache may be overburdened. In some embodiments, the hypervisor cache may use a Least Recently Used (LRU) technique that may cause frequent replacing of the nodes in the hypervisor cache and the I/O may not be serviced from the hypervisor cache as it may not include the data reference due to lot of intermediate read requests from multiple sources other than the containers. This may cause a huge performance bottleneck for the container I/O requests.

**[016]** Referring to **FIG. 1**, an architecture diagram for caching memory in a virtual computing environment 100 is illustrated, in accordance with a prior art embodiment. As shown in **FIG. 1** a virtual environment 100 of a host which may be enabled on a server grade hardware such as an x86 architecture platform. The virtual environment may be enabled by a hypervisor 102 as shown in **FIG. 1**.

**[017]** In an embodiment, the hypervisor 102 may be enabled on a server (not shown) and may be communicatively coupled via network to one or more virtual machines. As discussed above, server may be implemented on one or more commercially available computer servers. For example, such a server may include an I/O device such as a network card/controller connected by a PCI bus to a motherboard, a computer processor and memory (e.g., any combination of non-volatile memory such as hard disks, flash memory, optical disks, and the like, and volatile memory such as SRAM, DRAM, SDRAM, etc.). Hypervisor software, one or more virtual machines, and virtual file system software may be stored in the memory for use by a server operating system, and one or more of these programs may operate directly on the server hardware or on the operating system of the server.

**[018]** As shown in **FIG. 1**, the hypervisor 102 may include a plurality of virtual machines (VMs) $104_1$, $104_2$, … $104_n$ (collectively referred to as VMs 104 and individually referred to as VM 104) and running in same host. In an embodiment, each of the plurality of the VMs 104 may be a virtualization or an emulation of a computer system based on a computer architecture. Each of the plurality of the VM 104 may provide the functionalities of a physical computer. In simpler terms, each of the plurality of the VM 104 may be a virtualized instance of the computer system running of operating system such as MICROSOFT® Windows, MACINTOSH® OS X®, LINUX®, or UNIX®. An operating system manages computer hardware resources and provides common services for execution of various software applications.

**[019]** The VMs 104 may be configured to store and retrieve data from an external storage 138 coupled to the host. The VMs 104 may include containers $106_1$-$110_1$, $106_1$-$110_2$, … $106_1$-$110_n$ (collectively referred to as containers 106-110 and individually referred to as container 106, container 108, container 110 and so on). Further, each of the plurality of containers 106-110 of each of the plurality of the VMs 104 may be a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. In simpler words, nowadays most applications or software are developed on microservices based architectures that may break down the application or the software from a monolithic architecture to the microservice architecture independent of each other but working together to form the application or the software. These microservices architecture of the application or the software uses plurality of containers to do the processing of read I/O requests for this application or software.

**[020]** The hypervisor 102 may enable a hypervisor cache 114 in physical memory 112 (e.g., random access memory (RAM)) configured within the host. The hypervisor cache 114 may act as a small, fast memory that stores recently accessed data items and can be used to satisfy data requests without accessing an external storage 138. Accordingly, data requests satisfied by the cache may be executed with less latency as the latency associated with accessing the storage 138 directly is avoided. In an embodiment, the external storage 138 of the hypervisor 102 may be an auxiliary storage or secondary storage which may include addressable data storage. The external storage 138 of the hypervisor 102 may be, but not limited to, a virtual storage, a cloud storage, or a hardware storage.

**[021]** In an embodiment, the hypervisor cache 114 may exploit temporal locality by storing recently accessed data of the memory 112. Further, the hypervisor cache 114 may cache the data of one or more persistent disks $140_1$, $140_2$….$140_n$ (collectively referred to as 140) and a plurality of virtual disks $142_1$, $142_2$….$142_n$ (collectively referred to as virtual disks 142 and individually referred to as virtual disc 142) of the external storage 138. In an embodiment, the virtual discs 142 may include virtual disc files that may be stored in the external storage 138. In an embodiment, the persistent disks 140 may be independent of the VMs 104 and may be accessible by the containers 106-110 of the VMs 104.

**[022]** Hypervisor 102 may be connected to the external storage 138 or other plurality of storage devices via a network. In an embodiment, the network may establish a computing cloud (e.g., the software implementing the virtual machines and storage devices are hosted by

a cloud provider and exists “in the cloud”). Moreover, network can be a combination of public and/or private networks, which can include any combination of the internet and intranet systems that allow the hypervisor 102, and a plurality of VMs 104 operating thereon, to access storage servers; and for client to access the VMs 104. For example, network may connect one or more of the system components using the internet, a local area network (“LAN”) such as Ethernet or WI-FI, or wide area network (“WAN”) such as LAN to LAN via internet tunneling, or a combination thereof, using electrical cable such as HomePNA or power line communication, optical fiber, or radio waves such as wireless LAN, to transmit data. In this regard, the server and storage devices may use standard internet protocols for communication (e.g., iSCSI). In some embodiments, hypervisor 102 may be connected to the communications network using a wired connection to the internet.

**[023]** In an embodiment, the read I/O request from each of the plurality of VM 104 may be serviced by the hypervisor cache 114 to retrieve the requested data from at least one of the virtual discs 142. In an embodiment, the hypervisor cache 114 may implement a replacement policy such as least recently used (LRU) technique that may define eviction logic of the cache. LRU is a cache eviction strategy, wherein if the cache size has reached the maximum allocated capacity, the least recently accessed objects in the cache will be evicted. Similarly, the read I/O request from at least one of the plurality of containers 106-110 may be serviced in the hypervisor cache 114 to retrieve the requested data from the hypervisor cache 114. In case the requested read I/O data by at least one of the plurality of containers 106-110 is not found in the hypervisor cache 114, the requested read I/O data is retrieved from the corresponding persistent disk 140 of the external storage 138. It should be noted that the plurality of the VM 104 and the plurality of the containers 106-110 of the hypervisor 102 may prompt millions of read I/O requests to the hypervisor cache 114 that may cause frequent replacing of the nodes in the hypervisor cache 114. In case the capacity of the hypervisor cache 114 is less than the number of I/O requests received simultaneously, some of the I/O requests may not be serviced from the hypervisor cache 114. This may cause a huge performance bottleneck for the container read requests. Accordingly, the hypervisor cache 114 may be overburdened due to I/O requests received from the both the VMs to access the virtual discs 142 and the containers 106-110 to access the persistent discs 140.

**[024]** Referring now to **FIG. 2,** an architecture diagram for caching memory in a virtual computing environment 200 using nested LRU techniques is illustrated, in accordance

with an embodiment of current disclosure. The description of **FIG. 2** is provided in conjunction with the description of **FIG. 1**. In order to address the problems discussed above with respect to the hypervisor cache 114 not being able to address the I/O requests received from the containers 106-110 and the VMs 104. The hypervisor 102 as shown in **FIG. 2** may implement a container cache 202 separate from the hypervisor cache 114 in the memory 112. The container cache 202 may cache data of at least one of the persistent disk 140 based on an I/O request from at least one of the containers 106-110 of the VMs 104. Accordingly, access of at least one persistent disk 140 by at least one container 106-110 of at least one VM 104 may be enabled via the container cache 202. In some embodiments, the virtual environment 200 implemented by the hypervisor 102 that may include the plurality of the VMs 104 that may be connected to the hypervisor cache 114 of the memory 112. The hypervisor cache 114 may cache the virtual disks 142 and the persistent disks 140 in an analogous manner as explained in FIG. 1. In addition, the container cache 202 may cater to the I/O requests from the containers 106-110 to the plurality of the persistent disks 140.

**[025]** In an embodiment, a virtual storage adapter (virtual SCSI or virtual NVME) may be exposed to the corresponding container 106-110 of the VMs 104 to perform a read I/O request. The virtual storage adapter may act as a Virtual Private Network (VPN) and may enable connectivity of the hypervisor 102 to the internet or the external storage 138.

**[026]** In an embodiment, the hypervisor 102 may reserve a portion of the memory 112 for the container cache 116 separate from the hypervisor cache 114 when the hypervisor 102 is booted for the first time. By way of an example, a memory space with scope of expanding the container cache 116 may depend on the workload and the amount of resource the hypervisor 102 is running on. In an embodiment, a hypervisor 102 may have more than 500 GB of RAM. Further, the hypervisor 102 may initiated the container cache 202 when initiated by the hypervisor 102 on boot may be empty and may be filled based on creation of the containers 106-110. Further, the sub-caches 206 may be added based on creation of the persistent disks 140. Accordingly, when any of the containers 106-110 issue I/O requests the container cache 116 may start to inflate with the I/O data that may be cached. The hash map index of the plurality of the persistent disks 140 may be initialized that may be empty before receiving the I/O requests.

**[027]** In an embodiment, each of the plurality of the persistent disks 140 of the hypervisor 102 may be assigned a unique tag and a disk number. Further, a hash map index

may be maintained in a container cache counter 204 of the container cache 202. The container cache counter 204 may include all the disk numbers corresponding to each of the plurality of the persistent disks 140. The hash map index may include the same information as assigned to the disk i.e., disk number of each of the plurality of the persistent disk 140 along with the corresponding unique tag of each of the plurality of the persistent disk 140. The disk number and the unique tag of the persistent disk 140 may be written as a node in the persistent disk 140.

```
Node {
        Uint64 uniqueTag;
        Uint64 serialNumber;
};
```

**[028]** In an embodiment, each of the plurality of the persistent disks 140 may further include reserved space like sectors (not shown) on each of the plurality of the persistent disks 140 to store the information or the existing metadata may be appended for the persistent disk 140 by putting two more uint64 numbers. By way of an example, for every read I/O data requested by each of the containers 106-110, the disk number in the hash map index may be fetched and that particular field may be marked as false or true to indicate the disk number being in any of the persistent disks 140.

**[029]** In an embodiment, the container cache 116 may include one or more sub-caches $206_1$, $206_2$, … $206_n$ (individually referred to as sub-cache 206 and collectively referred to as sub-caches 206). Data of each persistent disk 140 may be cached by a corresponding sub-cache 206. Accordingly, for each persistent disk 140 there is a corresponding sub-cache 206. The container cache counter 204 may implement a first LRU technique. In an embodiment, each of the plurality of the sub-caches 206 may be assigned a unique ID. The container cache counter 204 may include a first lookup table including unique IDs of the sub-caches 206 sorted based on a recentness of access of each of the plurality of sub-caches 206. Accordingly, the first lookup table of the container cache counter 204 may include unique ID of the sub-cache 206 on top which may have been most recently accessed and the least recently accessed sub-cache 206 may be removed from the list. Accordingly, each of the plurality of the sub-caches 206 may cache data of the corresponding persistent disk 140 by looking up the hash map index by looking up the sub-caches 206 sorted as per the first lookup table of the container cache counter 204. Accordingly, the unique ID of the corresponding persistent disk 140 that may

have been recently accessed would be listed on top of the first lookup table of the container cache counter 204. Therefore, an I/O request to the corresponding persistent disk 140 that may have been recently accessed would be accessed based on looking up the unique ID on the sub-cache 206 having the hash map index of the corresponding persistent disk 140 which may be present on top of the first lookup table due to implementation of LRU technique.

**[030]** In some embodiments, each of the plurality of the sub-caches 206 of the container cache 202 may include a plurality of sector cache $208_1 \ldots 208_n$. In an embodiment, each of the sector cache 208 may correspond to a corresponding sector from plurality of sectors of the corresponding persistent disk 140. Accordingly, data of each sector the of persistent disk may be cached by a sector cache 208 of the corresponding sub-cache 206.

**[031]** Accordingly, the sub-cache $206_1$ may include a plurality of sectors cache $208_1 \ldots 208_n$ (collectively referred to as 208 and individually referred to as 208). Similarly, each of the sub-cache $206_{2\text{-}n}$ may include a plurality of sectors cache referred to as 210-212). Further, each of the plurality of sectors cache $208_1 \ldots 208_n$ may be assigned a unique sector ID. Further, each of the plurality of sub-caches 206 may include a sub-cache counter 214. Each of the corresponding sub-cache counter 214 may include a second lookup table 205 including unique sector IDs of the sector caches $208_1 \ldots 208_n$ sorted based on a recentness of access of each of the plurality of sector caches $208_1 \ldots 208_n$. Accordingly, the second lookup table 205 of the sub-cache counter 214 may include unique sector ID of the sector cache 208 on top which may have been most recently accessed and the least recently accessed sector cache 208 may be removed from the list.

**[032]** In an embodiment, the each of the plurality of the sub-cache counters214 may include a count of a number of times each of the plurality of the corresponding sub-cache 206 is accessed based on I/O request from at least one of the plurality of the containers 106-110. In simpler words, a sub-cache counter 214 may increment by one, every time it may be accessed based on an I/O request received from at least one of the containers 106-110. In an embodiment, the first lookup table 203 of the container cache counter 204 may include a list of unique IDs of each of the sub-caches 206 in descending order of the sub-cache counters 214 of each of the sub-caches 206.

**[033]** Accordingly, each of the sectors cache counter 214 may include a count of a number of times each of the plurality of the corresponding sector-cache 208 is accessed based on I/O request from at least one of the plurality of the containers 106-110. Accordingly, a

sector-cache counter 214 may increment by one, every time it may be accessed based on an I/O received from at least one of the containers 106-110. In an embodiment, the second lookup table 205 of each of the plurality of the sub-caches counter 214 may include a list of sector IDs of each of the sector caches 208 in descending order of the sector-cache counter 214 of each of the sector caches 208.

**[034]** Accordingly, the hypervisor 102 may deploy the LRU technique in a nested manner. The hypervisor 102 may deploy the first LRU technique in the container cache 202 and the second LRU technique in each of the plurality of the sub-caches 206. By way of an example, the first lookup table 203 of the container cache counter 204 may include the first lookup table including a list of the unique ID's of the plurality of the sub-caches 206 sorted in descending order of the value of the sub-cache counter 214. In a similar way, the second lookup table 205 of each of the plurality of the sub-caches 206 may include a list of the sector ID's of the plurality of the sectors caches 208 sorted in descending order of each of the value of the sector cache counter 216. Thus, the container cache 202 upon receiving an I/O request from one of the plurality of the containers 106-110, may traverse the sorted list of the plurality of the sub-caches 206 in the first lookup table 203 which when found, the hypervisor 102 may traverse the sorted list of the plurality of the sectors caches 208 in the second lookup table 205 of the corresponding found sub-cache 206 to read the I/O data requested by the containers 106-110.

**[035]** In some embodiments, each of the plurality of the containers 106-110 may request the read I/O data from the plurality of the persistent disks 140 of the external storage 138. Upon receiving the read I/O data requests the container cache 114 may start to inflate with the read I/O data that may be cached. The hash map index of the plurality of the persistent disks 140 may be initialized with the index initially be null. Further, each of the read I/O request of the plurality of the containers 106-110 may be received by the virtual adapter layer in the hypervisor 102. The read I/O data corresponding to each of the read I/O request may be determined on the plurality of the persistent disk 140. Further, the virtual adapter logic of the virtual adapter layer may fetch the disk number and determine the disk number as the persistent disk number, if found in the hash map index. Further, the read I/O request may be redirected to the container cache 116 and the requested I/O data may be retrieved from the container cache 116 or the requested I/O data may be written on the container cache 116 from the relevant persistent disk 140 of the external storage 138.

**[036]** Referring now to **FIG. 3,** a flowchart depicting caching in a virtual computing environment 200 using the nested LRU methodology is illustrated, in accordance with an embodiment of the current disclosure. **FIG. 3** is explained in conjunction with FIGS. 1-2. Each step of the flowchart 300 may be executed in the container cache 202 of the hypervisor 102.

**[037]** In an embodiment, the container cache 202 may implement a Least Recently Used (LRU) technique which is a cache eviction policy which may remove the least recently used items from the container cache 202 when the container cache 202 is full. It may employ a doubly linked list to keep track of the accessed items in the container cache 202. Further, each of the items in the container cache 202 may be represented as a node in the linked list with arrows indicating the links between the plurality of nodes. It should be noted that the length of the linked list may vary as per the configuration and requirement of the container cache 202. Further, the most recently accessed node may be positioned at the front of the linked list and the least recently used node may be positioned at the back. In case a new node is accessed, the accessed node may be moved to the front of the linked list and in case the list is already full, then the last node which may represent the least recently used item may be evicted from the linked list.

**[038]** Referring for now FIG. 3, at step 302 of the flowchart 300, the container cache 202 may include an empty linked list with capacity of three items in container cache 202. Further, at step 304, the items A, B, C may be added in an order of receiving an I/O request for their access and may be updated in the linked list as [C] -> [B] -> [A]. At step 306, in case a new item [D] is requested, from the already full list, the container cache 202 may cause the eviction of the item [A] which was least recently used, and the list may be updated to include [D] at the top of the list as [D] -> [C] -> [B]. Further, at step 308, the item [C] may be accessed again, accordingly, the item [C] may be moved to the front of the list and the updated list may in the container cache may be depicted as [C] -> [D] -> [B]. At step 310, a new item [E] may be accessed which may be updated at the top and may cause the list to evict the last item from the list and the updated list may be depicted as [E] -> [C] -> [D]. Further, at step 312, the item [C] may be accessed which may move the item [C] to the top and the updated list may be depicted as [C] -> [E] -> [D].

**[039]** Accordingly, the flowchart 300 illustrates an exemplary progression of the container cache 202 using the first LRU technique. In simpler words, the items that are more frequently accessed are moved towards the front of the linked list, ensuring that the most

recently used items are retained in the container cache 202, while the least recently used items may be evicted from the container cache 202 as needed. Further, a second level of the LRU technique may be implemented inside each of the plurality of the sub-cache 206 of the container cache 202. In an embodiment, the container cache 202 may enable further levels of nested LRU for caching data of sub-units of sectors of the persistent disk 140 in case applicable, Accordingly, the nested LRU technique may allow for improved and faster reads from the cache.

**[040]** Referring now to **FIG. 4**, a flowchart 400 illustrates a method for caching memory in a virtual computing environment 200 using nested LRU techniques, in accordance with an embodiment of the current disclosure. **FIG. 4** is explained in conjunction with FIGS. 1-3. Each step of the flowchart 400 may be executed by a processor and various modules (same as the modules of the system 200).

**[041]** At step 402, access of at least one persistent disk may be enabled by at least one container of at least one virtual machine (VM) via a container cache. In an embodiment, the container cache 202 may be enabled by the hypervisor 102 in the virtual computing environment 200 separate from the hypervisor cache 114. Further, the access of the at least one persistent disk 140 by the at least one container 106 via the container cache 202 may be based on caching of data of the at least one persistent disk 140 based on a container cache counter 204 implementing a first least-recently-used technique.

**[042]** At step 404, the data of the at least one persistent disk 140 may be cached based on the container cache counter 204 which may include caching the data of the at least one persistent disk 140 via one of a plurality of sub-caches 206 of the container cache 202. Further, at step 406, the caching of the data of the at least one persistent disk 140 may be performed based on a sub-cache counter 214 of the one of the plurality of sub-caches 206 implementing a second least-recently-used technique.

**[043]** Thus, the disclosed method and system try to overcome the technical problem of caching memory in a virtual computing environment using nested LRU techniques. The method and system provide means to deploy a container cache for faster accessing external storage. Further, the method and system may cater to millions of read I/O requests generated by the plurality of containers and the plurality of VM. Further, the method and system provide a means to deploy a container cache to specifically cache the read I/O data requested by the plurality of the containers. Further, the method and system may deploy the Nested Least

Recently Used (LRU) technique to cater to the stateful workloads of the containers which may mean that the container requests may be local and spatial that may have very high performance gains for these containers.

**[044]** In light of the above mentioned advantages and the technical advancements provided by the disclosed method and system, the claimed steps as discussed above are not routine, conventional, or well understood in the art, as the claimed steps enable the following solutions to the existing problems in conventional technologies. Further, the claimed steps clearly bring an improvement in the functioning of the device itself as the claimed steps provide a technical solution to a technical problem.

**[045]** It is intended that the disclosure and examples be considered as exemplary only, with a true scope of disclosed embodiments being indicated by the following claims.

**WE CLAIM:**

1. A method (400) of caching memory in a virtual computing environment (100), the method comprising:

enabling, by a processor, access of at least one persistent disk (140) by at least one container (106, 108, 110) of at least one virtual machine (VM) (104) via a container cache (202),

wherein the container cache (202) is enabled by a hypervisor (102) in the virtual computing environment (100) separate from a hypervisor cache (114),

wherein the access of the at least one persistent disk (140) by the at least one container (106, 108, 110) via the container cache (202) is based on caching of data of the at least one persistent disk (140) based on a container cache counter (204) implementing a first least-recently-used technique, and

wherein the caching of the data of the at least one persistent disk (140) based on the container cache counter (204) comprises:

caching the data of the at least one persistent disk (140) via one of a plurality of sub-caches (206) of the container cache (202), and

wherein the caching the data of the at least one persistent disk (140) is based on a sub-cache counter (214) of the one of the plurality of sub-caches (206) implementing a second least-recently-used technique.

2. The method (400) as claimed in claim 1, wherein the container cache counter (204) implements the first least-recently-used technique by sorting the plurality of sub-caches (206) based on a recentness of access of each of the plurality of sub-caches.

3. The method (400) as claimed in claim 2, wherein each of the plurality of sub-caches (206) is assigned a unique ID, and

wherein the container cache counter (204) comprises a first lookup table comprising a list of the unique IDs sorted based on the recentness of access of each of the plurality of sub-caches.

4. The method (400) as claimed in claim 1, the caching the data of the at least one persistent disk (140) via the one of a plurality of sub-caches (206) comprises caching the data of the at

least one persistent disk (140) via one of a plurality of sector caches of the one of the plurality of sub-caches.

5. The method (400) as claimed in claim 4, wherein the sub-cache counter (214) implements the second least-recently-used technique by sorting the plurality of sector caches based on a recentness of access of each of the plurality of sector caches.

6. A system of caching memory in a virtual computing environment (100), the system comprises:

a processor; and

a memory coupled to the processor, wherein the memory stores processor-executable instructions, which, on execution, causes the processor to:

enable access of at least one persistent disk (140) by at least one container (106, 108, 110) of at least one virtual machine (VM) via a container cache (202),

wherein the container cache is enabled by a hypervisor (102) in the virtual computing environment (100) separate from a hypervisor cache (114),

wherein the access of the at least one persistent disk (140) by the at least one container (106, 108, 110) via the container cache (202) is based on caching of data of the at least one persistent disk (140) based on a container cache counter (204) implementing a first least-recently-used technique, and

wherein the data of the at least one persistent disk (140) is cached based on the container cache counter (204) by:

caching the data of the at least one persistent disk (140) via one of a plurality of sub-caches (206) of the container cache (202), and

wherein the caching the data of the at least one persistent disk (140) is based on a sub-cache counter (214) of the one of the plurality of sub-caches (206) implementing a second least-recently-used technique.

7. The system as claimed in claim 6, wherein the container cache counter (204) implements the first least-recently-used technique by sorting the plurality of sub-caches (206) based on a recentness of access of each of the plurality of sub-caches (206).

8. The system as claimed in claim 7, wherein each of the plurality of sub-caches (206) is assigned a unique ID, and

wherein the container cache counter (204) comprises a first lookup table (203) comprising a list of the unique IDs sorted based on the recentness of access of each of the plurality of sub-caches (206).

9. The system as claimed in claim 7, the caching the data of the at least one persistent disk (140) via the one of a plurality of sub-caches (206) comprises caching the data of the at least one persistent disk (140) via one of a plurality of sectors caches (208, 210, 212) of the one of the plurality of sub-caches (206).

10. The system as claimed in claim 9, wherein the sub-cache counter (214) implements the second least-recently-used technique by sorting the plurality of sector caches (208, 210, 212) based on a recentness of access of each of the plurality of sector caches (206).

Dated this 2nd day of August 2023

***-- Digitally Signed--***
Bhanu Prasad
(INPA No: **3253**)
Head, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

## ABSTRACT

### METHOD AND SYSTEM OF CACHING IN A VIRTUAL COMPUTING ENVIRONMENT USING NESTED LRU TECHNIQUES

A method of caching memory in a virtual computing environment is disclosed. The method includes enabling access of at least one persistent disk (140) by at least one container of at least one virtual machine (104) via container cache (202). The container cache (202) is enabled by a hypervisor (102) in virtual computing environment (100) separate from a hypervisor cache (114). The access of at least one persistent disk (140) by at least one container via container cache (202) is based on caching of data of at least one persistent disk (140) based on container cache counter (204) implementing first least-recently-used technique. Further, the caching of data of at least one persistent disk (140) based on container cache counter (204) includes caching data of at least one persistent disk (140) via one of plurality of sub-caches (206) of container cache (202). Caching data of at least one persistent disk (140) is based on sub-cache counter (214) of one of the plurality of sub-caches (206) implementing second least-recently-used technique.

*[To be published with FIG. 2]*

1/4

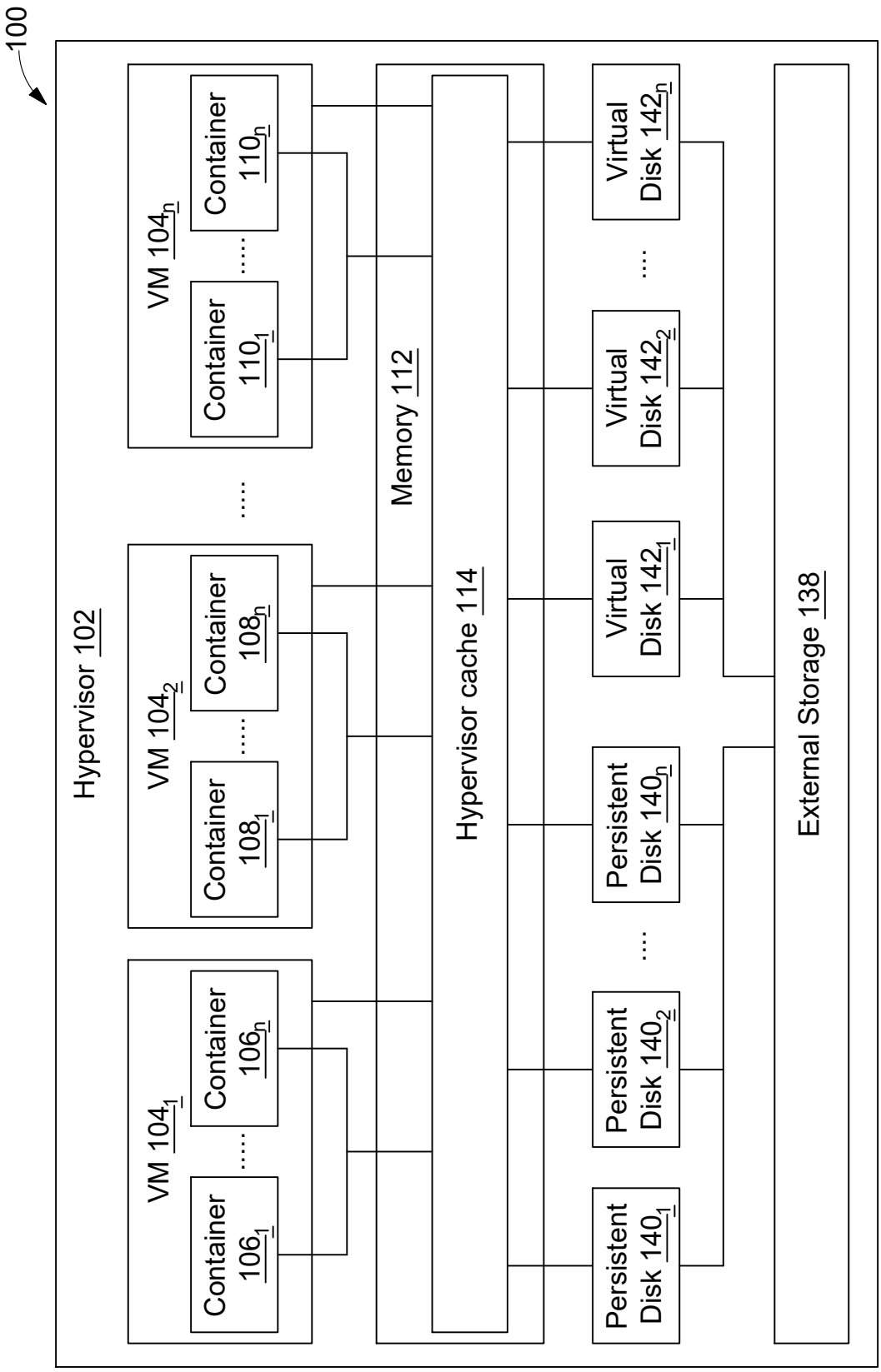


FIG. 1 (PRIOR ART)

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Head, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

2/4

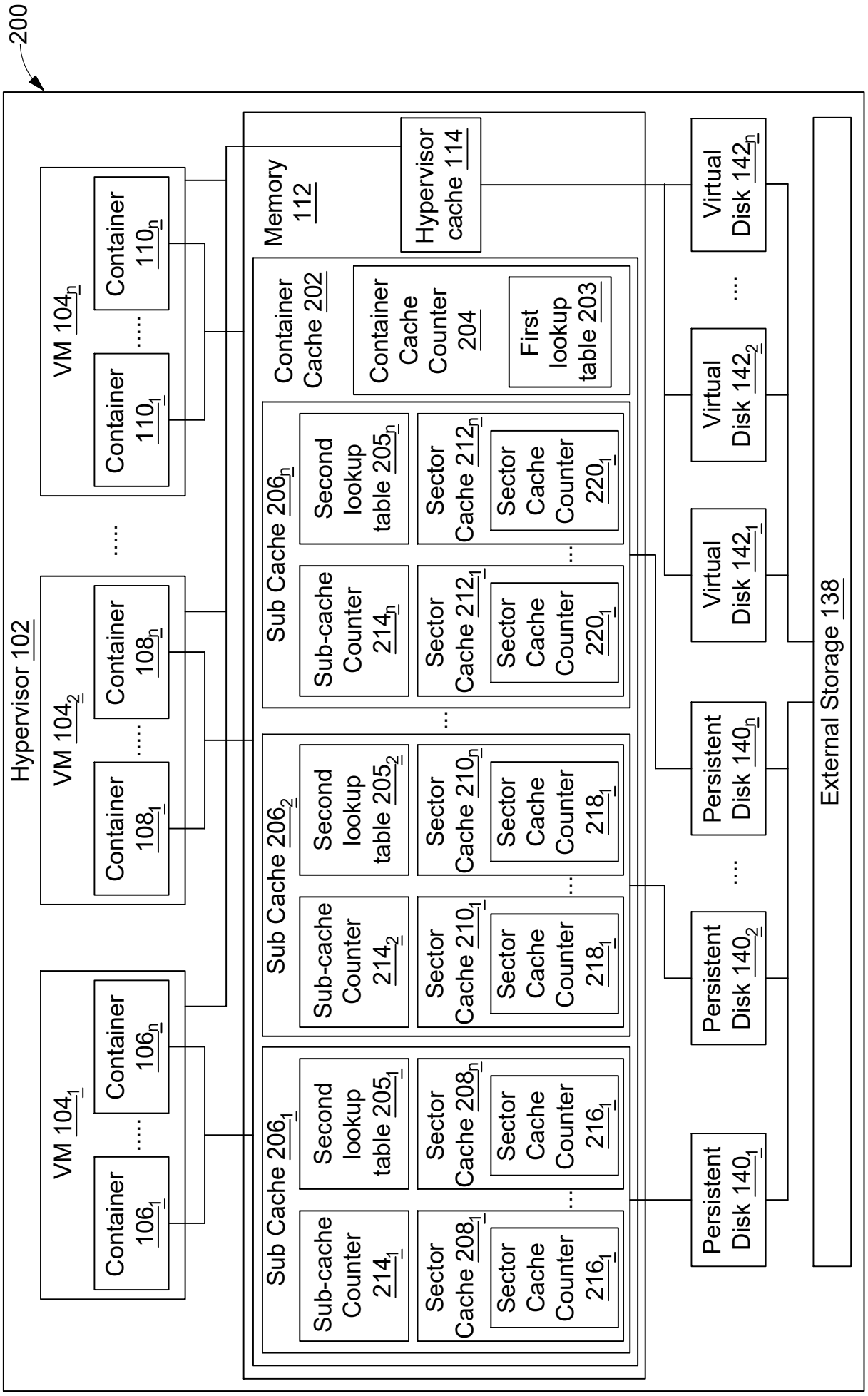


FIG. 2

3/4

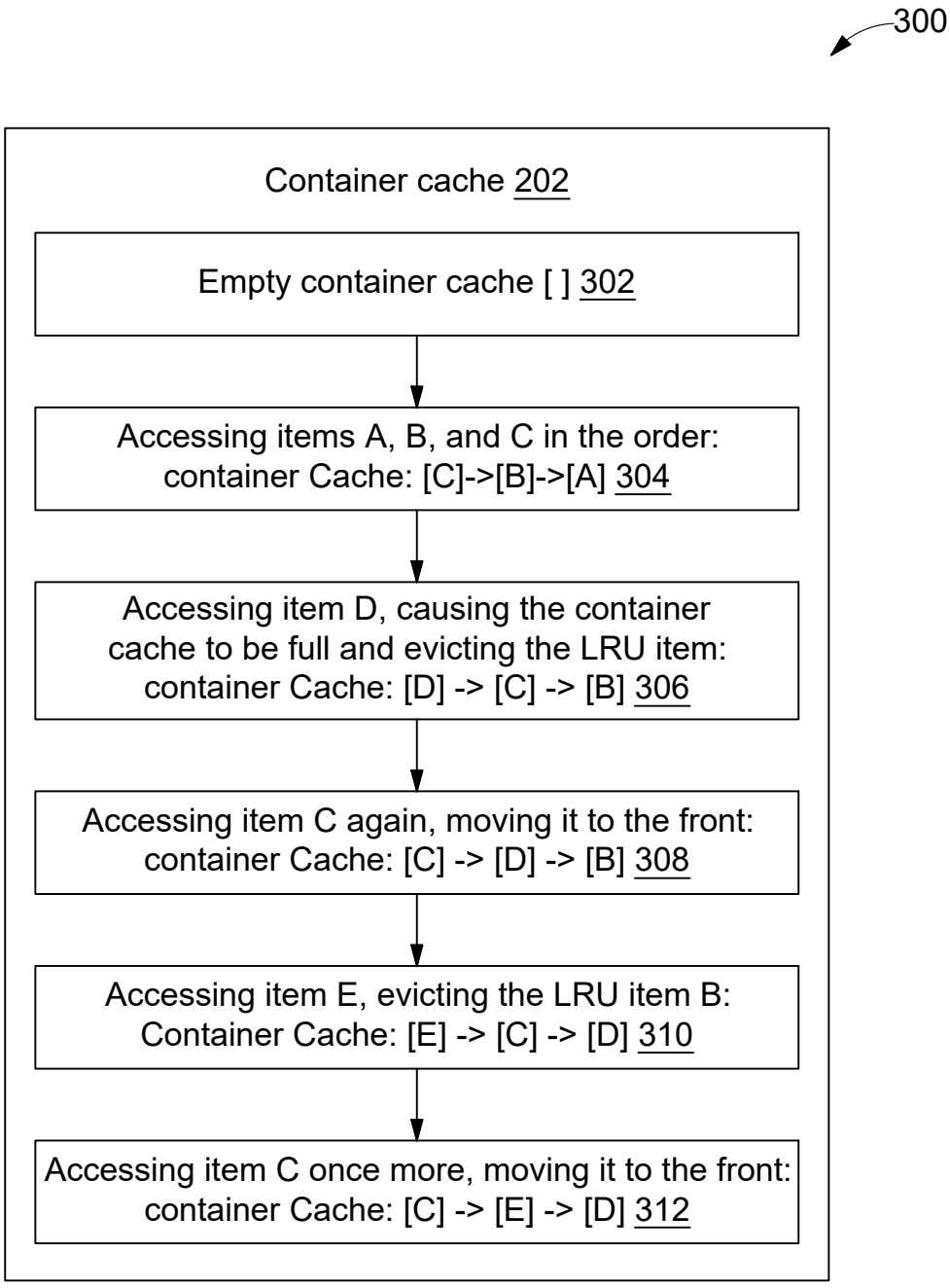


FIG. 3

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Head, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

4/4

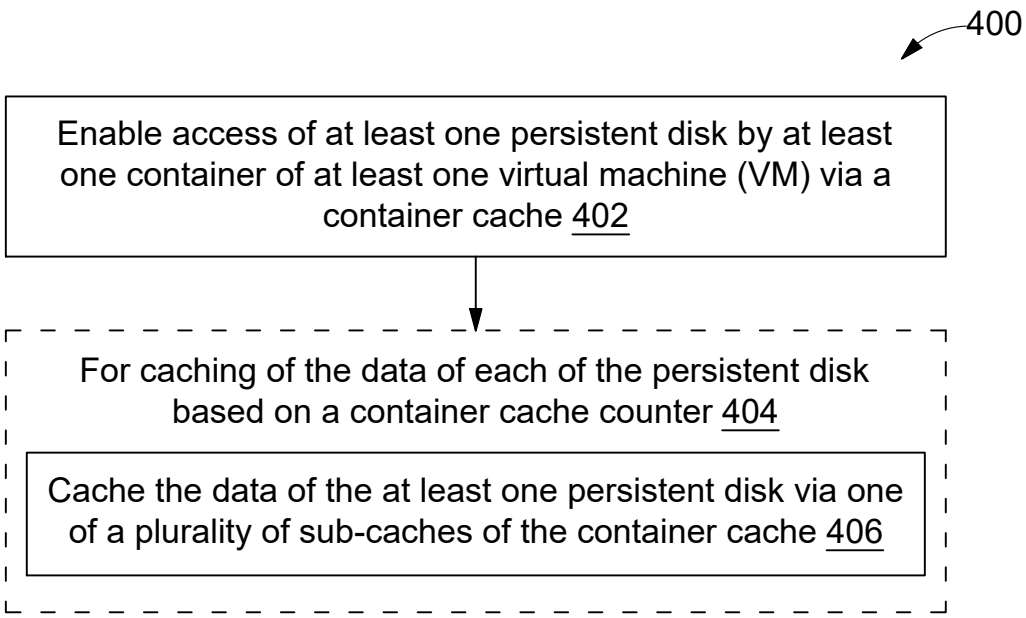


FIG. 4

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Head, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.