(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 January 2017 (12.01.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/005761 Al

(51) International Patent Classification:
G06F 12/08 (2006.01) *G06F 12/12* (2006.01)

(21) International Application Number:
PCT/EP2016/065897

(22) International Filing Date:
6 July 2016 (06.07.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
15002008.9 6 July 2015 (06.07.2015) EP

(71) Applicant: **ALCATEL LUCENT** [FR/FR]; 148/152 Route de la Reine, 92100 Boulogne-Billancourt (FR).

(72) Inventors: **GALLO, Massimo;** Alcatel-Lucent Bell Labs Franc, 1 Route De Villejust, Centre De Villarceaux, 91620 Nozay (FR). **PERINO, Diego;** Alcatel-Lucent Bell Labs Franc, 1 Route De Villejust, Centre De Villarceaux, 91620 Nozay (FR). **SAINO, Lorenzo;** Room B7.03.3, New Hall, 465 Caledonian Road, London N79GU (GB).

(74) Agent: **BERTHIER, Karine;** Alcatel-Lucent International, 148/152 Route de la Reine, 92100 Boulogne-Billancourt (FR).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available):* AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available):* ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(54) Title: METHOD FOR MANAGING A DISTRIBUTED CACHE

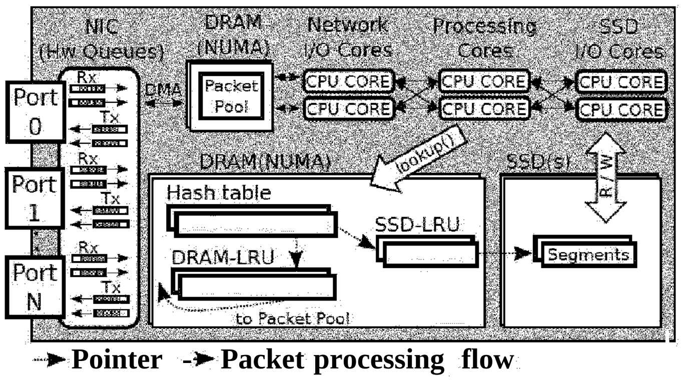

Figure 1

(57) Abstract: A method for managing a multiple level cache of a host comprising a primary cache which is a volatile memory such as a DRAM memory and a secondary cache which is a non-volatile memory such as a SSD memory. The method comprises, if a segment identification data has been computed in said segment hash table, a corresponding processing core checks whether a corresponding packet is stored in a first portion of a primary cache or in a second portion of a secondary cache, - if the packet is stored in said first portion, said corresponding packet is sent back to a requester and a request counter is incremented, a DRAM segment map pointer entering in a DRAM-LRU linked list, the DRAM segment map pointer being prioritized by being moved on top of said DRAM-LRU linked list, - if the packet is stored in said second portion, said corresponding packet is passed to an SSD core so as to copy the entire given segment from the secondary cache to the primary cache; then said request is passed back to said corresponding processing core in order to create the DRAM segment map pointer for pointing to the first portion storing said corresponding packet so as to be entered in said DRAM-LRU linked list, the SSD segment map pointer being also entered in said SSD-LRU linked list, the DRAM segment map pointer and the SSD segment map pointer being respectively prioritized by being respectively moved on top of said DRAM-LRU linked list and said SSD-LRU linked list; then said corresponding packet is sent back to said requester.

METHOD FOR MANAGING A DISTRIBUTED CACHE

FIELD OF THE INVENTION

The invention relates to a method for managing a multiple level cache of a host.

The invention also concerns a computer program product comprising computer-executable instructions for performing the method and a computer readable medium having loaded thereupon such computer program.

BACKGROUND OF THE INVENTION

Several existing solutions require packet-level caching capabilities. This is for instance the case of redundancy elimination or data deduplication services or Named Data Networking (NDN). Such services pose severe requirements on their caching system. In particular, they need to perform operations at packet level, sustain multi-Gbps line rate, provide terabytes (TB) of caching memory and ideally run on commodity hardware. However, existing caching solutions do not meet these requirements. They deploy caches in large data-centers to scale throughput and storage, or sustain limited throughput. Key value store designs either achieve high throughput on single Dynamic Random Access Memory (DRAM) technology, or provide large storage capacity on Hard Disk Drive (HDD) or Solid-State Drive (SSD) but they do not operate at wire speed.

Prior art techniques attempt to design a system targeting a two layers packet-level cache able to store several terabytes of data while serving requests at line rate. According to a first prior art technique, there is disclosed a high level hierarchical design tailored to video streaming applications over NDN and evaluate it by theoretical modeling and simulation. A second prior art technique discloses a preliminary design to integrate SSD technology in an NDN forwarding engine and experimentally analyzes the read/write throughput only.

SUMMARY OF THE INVENTION

The object of the present invention is to alleviate at least partly the above mentioned drawbacks, notably by providing a system taking into account the constraints imposed by the implementation environment and by proposing optimized data structures and indexing techniques for the management of the two caching layers.

In a first aspect, this aim is achieved with a method for managing a distributed cache of a host comprising a primary cache which is a volatile memory such as a DRAM memory and a secondary cache which is a non-volatile memory such as a SSD memory, the method comprising the steps of:

- dividing a content object into N segments, each of said N segments being composed of K consecutive packets, each of the N segments being identified by a segment identification data (segment^), said packets being stored in a first portion of said primary cache and/or in a second portion of said secondary cache, said first portion and said second portion being assigned to a corresponding processing core so that packets belonging to a same segment are always assigned to the same processing core,
- building a segment hash table in order to index the packets of said N segments stored in said first portion and/or in said second portion, the entries of said segment hash table being related to a DRAM segment map pointer and/or a SSD segment map pointer depending on which portion among said first portion and said second portion a corresponding packet is cached in, the DRAM segment map pointer being configured to point to the first portion storing the packets of a given segment, the SSD segment map pointer being configured to

point to the second portion storing the packets of said_given segment, the DRAM segment map pointer of each of the N segments being managed as a DRAM-LRU linked list which is ordered, each of the DRAM segment map pointers containing a request counter for counting the number of times said given segment has been requested, the SSD segment pointer of each of the N segments being managed as a SSD-LRU linked list which is ordered,

- receiving a request from a requester and performing a look up in said segment hash table so as to determine if said segment identified by segment identification data (segment^) has already been built (i.e., if said segment identified by segment identification data is stored in the local cache of the host),

if said segment identification data has been computed in said segment hash table, said corresponding processing core checks whether the corresponding packet is stored in said first portion of said primary cache or in said second portion of said secondary cache,

- if the packet is stored in said first portion, said corresponding packet is sent back to said requester and said request counter is incremented, the DRAM segment map pointer entering in said DRAM-LRU linked list, the DRAM segment map pointer being prioritized by being moved on top of said linked DRAM-LRU list,
- packet is passed to an SSD core so as to copy the entire given segment from the secondary cache to the primary cache ; then said request is passed back to said corresponding processing core in order to create the DRAM segment map pointer for pointing to the first portion storing said corresponding packet so as to be entered in said DRAM-LRU linked list,the SSD segment map pointer being also entered in said SSD-LRU linked list, the DRAM segment map pointer and the SSD segment map pointer being respectively prioritized by being respectively moved on top of said DRAM-LRU

linked list and said SSD-LRU linked list; then said corresponding packet is sent back to said requester.

According to an embodiment, the method further comprises:

- if the primary cache is full, evaluating the value of said request counter of the DRAM segment map pointer which is at the tail of said DRAM-LRU linked list,
- comparing the value of the request counter with a predetermined threshold T_R,
- if the value is above said predetermined threshold, said packets of the entire segment pointed to by the DRAM segment map pointer are transferred from the primary cache to the secondary cache and the SSD segment map pointer is prioritized by being moved on top of said SSD-LRU linked list,
- if the value is below said predetermined threshold, the DRAM segment map pointer and said packets of the entire segment pointed to by the DRAM segment map pointer are respectively deleted from the primary cache and from said DRAM-LRU linked list.

According to an embodiment, the method comprises:

- if the secondary cache is full, the SSD segment map pointer of the segment to be deleted which is at the tail of said SSD-LRU linked list is deleted from said SSD-LRU linked list and from the entries of said segment hash table if there is no DRAM segment map pointer corresponding to the segment to be deleted in the DRAM-LRU linked list.

According to an embodiment, the method comprises:

- receiving a data and performing a lookup in said segment hash table so as to determine if said segment identified by segment identification data (segment^) has already been stored, if said segment identification data has not been computed in said segment hash table, an entry is created in said segment hash table and a corresponding DRAM segment map pointer is created so as to be entered and prioritized in said DRAM-LRU linked list by the assigned processing core.

According to an embodiment, the method comprises, when said corresponding packet is passed to an SSD core so as to copy the entire given segment comprising the said packet from the secondary cache to the primary cache, said corresponding segment is not removed from the secondary cache.

According to an embodiment, if said segment identification data has not been stored in said segment hash table, said request is forwarded to a next hop.

According to an embodiment, the primary cache is a DRAM memory and the secondary cache is a SSD memory.

According to an embodiment, said DRAM-LRU linked list and said SSD-LRU linked list are ordered according to a Least Recently Used policy.

According to an embodiment, the method comprises:

- each of the DRAM segment map pointers further containing pointers to previous and next segments in said DRAM-LRU linked list,
- each of the SSD segment map pointers further containing pointers to previous and next segments in said SSD-LRU linked list.

A second aspect of the invention provides a computer program product comprising computer-executable instructions for performing a method according to any of the embodiments herein mentioned.

A third aspect of the invention provides a computer readable medium having recorded thereon a computer program comprising program instructions, the computer program being loadable into a data-processing unit and adapted to cause execution of the method according to any of the embodiments herein mentioned when the computer program is run by the data-processing unit.

The dependent claims define various embodiments of the invention.

Further features and advantages of the invention will appear from the following description of embodiments of the invention, given as non-limiting examples, with reference to the accompanying drawings listed hereunder.

BRIEF DESCRIPTION OF THE DRAWING

Fig.l represents a flowchart of an embodiment of the proposed method and system.

Fig. 2 illustrates the structure of content objects to identify packets.

Fig. 3 illustrates the results of an experiment wherein the hit ratio is in function of the threshold for objects sizes of 1.5 MB and 15 MB.

Fig. 4 illustrates the results of an experiment wherein one single SSD memory is used.

DETAILED DESCRIPTION OF THE INVENTION

Specific embodiments of the invention will now be described in detail with reference to the accompanying figures. Like elements in the various figures are denoted by like reference numerals for consistency.

In general, embodiments of the invention are directed at managing a multiple level cache of a host comprising a primary cache which is a volatile memory such as a DRAM memory and a secondary cache which is a non-volatile memory such as a SSD memory.

Figure 1 shows a packet processing flow implemented by a data storage system configuration (not shown) which is suitable for use by the invention. The data storage system configuration includes a data storage system assembly (not shown) and a host. The data storage assembly includes a set of storage devices (not shown) such as hard disk drives. The host includes a host control circuit, a primary cache and a secondary cache, a central processing unit (CPU). In one or more embodiments of the invention, the CPU is a hardware processor component for processing instructions of the host. The CPU may include multiple hardware processors. Alternatively or additionally, each hardware processor may include multiple processing cores. In general, the CPU is any device configured to execute instructions on the host.

A requester connected to the data storage system through the host (and possibly other NDN host machines) can obtain the data by sending a request to the data storage. The data storage assembly responds by retrieving the data from the set of storage devices and transmitting the data to the requester. The host control circuit of the host then caches the data in the primary cache.

If the host receives a new request for the same, the host may not need to re-read the data from the data storage assembly. Rather, the host control circuit can simply access the copied data residing within the primary cache or in the secondary cache. Accordingly, the host can avoid incurring the communication latencies and the overhead associated with re-reading the same data from the data storage assembly.

It should be understood that memory spaces within the primary cache and the secondary cache are resources which may be re-consumed for other operations. Accordingly, the host may encounter a situation in which it needs to re-access the data.

Two types of packets can be received by this data storage system configuration: request and data. As explained above, the first type is the request that a client uses to express an interest in a specific data and the second type is the data itself that satisfies the corresponding request. Both packets (request and data) contain the name of the packet, i.e. content object name plus packet identifier. The host control circuit includes cores (CPU) that work in poll mode in an asynchronous mode. Packets are passed among cores via dedicated per core software input queues in the shared memory with zero-copy. A core processes a new packet from the input queue when it has finished the processing of the previous packet.

The Hierarchical 2 level cache (H2C) divides content objects in N segments each composed of K consecutive packets. Segments are identified by a segment identification data segment^ (i.e. content object name plus the integer part of packetid / N; the packetid, as illustrated in figure 2, is the sequential number of the packet composing the file). H2C performs operations at segment granularity rather than at packet granularity.

The main H2C data structure is a hash table used for indexing all the segments stored in a first portion of the primary cache and/or in a second portion of the secondary cache assigned to the corresponding processing core. It is implemented using an open-addressed hash table similar to those well known to the person skilled in the art. The hash table is configured so as to minimize access time. For this purpose, every hash table bucket contains multiple values avoiding chaining in case of collision (i.e., two items to be stored in the same bucket) and is dimensioned to a cache line. It is therefore possible to read all elements stored in a bucket with a single access to the memory. Bucket overflow is managed with linked lists but it is expected to be rare if the number of buckets is large enough.

For instance, each entry of the hash table is composed of 6 fields for a total of 16B. In Figure 2, the "active" field indicates whether the entry is valid or not (used for fast deletion). The "segment hash" field is the 32-bit hash value of the segment identifier. The "SSD flag" field indicates whether the entire segment is stored in the secondary cache or not (in this example, the secondary cache is SSD). The "DRAM packet bitmap" indicates which packets of the segment are stored in the primary cache (in this example, the primary cache is DRAM. Finally, the DRAM segment map pointer and the *SSD segment map pointer* store a pointer to the DRAM/SSD segment map respectively.

Every hash table entry is related to a DRAM segment map and/or a SSD segment map depending on which level the packet is cached at.

The DRAM segment map primarily contains an array of K elements (i.e., the number of packets in a segment) pointing to the buffers storing the packets of a given segment. DRAM segment maps of various segments are organized as a linked list and ordered according to Least Recently Used (LRU) policy. We call this list DRAM-LRU. Every segment map also contains a Rc field counting the number of times a segment has been requested, and pointers to the previous and next segments in the DRAM-LRU.

The SSD segment map contains a pointer to the SSD address where the segment is stored, and pointers to previous and next elements. SSD segment maps of various segments are organized as a linked list and ordered according to LRU policy. We call this list SSD-LRU.

Finally, the DRAM packet pool is a lock-free pool of available pre-allocated packet buffers stored in DRAM. Whenever a packet is retrieved from the Network Interface Card (NIC) or the SSD memory, it is stored in DRAM memory by the dedicated CPU core using a free buffer of the packet pool. Conversely, when a packet is evicted from DRAM, the corresponding object is added back to the packet pool.

First, packets are extracted in batch from the Software queues (SW queues) by input/output (I/O) cores. Hardware queues (HW queues) are integrated into the NIC and I/O cores leverage on fast packet processing techniques such as Intel DPDK to receive incoming packets. SW queues are integrated into the DRAM, also known as TW/RW rings. Transferring from SW queues to HW queues (or reciprocally) is transparent to the CPU and done using Direct Memory Access (DMA) techniques leveraging a set of libraries and drivers for fast packet processing, such as Intel® DPDK. The I/O core receives the batch of packets and dispatches them to the different processing cores according to the hash value of the hash table of their segmenti_D, computed as the integer part of packetid / N, so that packets belonging to the same segments are always assigned to the same processing core. This enables lockless operations on the data structures above described.

Processing cores extract batches of packets from software queues and process them according to their type.

If the received packet is a request, a lookup in the hash table of the segment is performed using the segment^ hash value already computed. In case the segment identification data is not present in the segmented hash table the request is forwarded to the next hop. In case the segment identification is present, the processing core checks whether the packet is stored in the primary cache (DRAM) or in the secondary cache (SSD) using the DRAM packet bitmap and the SSD flag.

If the packet is stored in the primary cache (DRAM), the corresponding data packet is prepared and sent back to the requester and a request counter Rc is incremented. The segment entry is also moved on top of the list DRAM-LRU.

If the packet is stored in the secondary cache (SSD), the request is passed to an SSD core (associated with the secondary cache, the SSD core being a core which is dedicated to manage inpu/ouput operation with the SSD) that fetches the entire segment from the SSD. Once the segment is moved to the first level cache the request is passed back to the processing core that creates and populates the DRAM segment map. The segment map entry is then moved to the top of the DRAM-LRU and SSD-LRU, and the response is sent back to the requester. Notice that segments copied from the second to the first level of the cache are not removed from the secondary cache.

If the received packet is data and it is not already in first or second level cache of H2C, a hash table entry is created and the corresponding DRAM segment map entry is created and inserted on top of the DRAM-LRU by the target processing core. The data is then sent to the requester.

During these packet processing operations, segment eviction from the primary cache or secondary cache may be required. When the primary cache is full the segment at the tail of DRAM-LRU is evaluated. If its value Rc is above a predefined threshold T_R the segment is demoted from the primary cache to the secondary cache and placed at the top of the SSD-LRU, otherwise it is discarded. When the secondary cache is full, the element at the tail of the SSD-LRU is removed from the list; it is also removed from the hash table if it is not stored in DRAM-LRU. The rationale for selective SSD insertion mechanism is to reduce SSD writes caused by unpopular objects which degrade SSD read throughput and increase wear.

We implemented H2C integrated to an NDN forwarded as application example. Specifically, we implement H2C on a general purpose server running Linux Ubuntu 12.04 LTS and equipped with two 4-core CPUs (Intel® Xeon® E5540 2.53 GHz), 32 GB of DRAM, one dual-port lOGbE card, and two 200 GB SSD drives.

For our evaluation we connected H2C to a commercial traffic generator equipped with lOGbE interfaces via optical fibers. As evaluation workload we used a one-day trace recording 10% of the user requests served by Wikipedia website. This trace contains 8 million requests over a catalog of 1.3 millions unique items. We artificially vary the average content size from 1.5 MB to 15 MB to study the impact of different catalog sizes on H2C performance. We split content items in packets of 4 KB.

Unless otherwise specified H2C is configured with a DRAM cache of 20 GB, an SSD cache of 200 GB (100 GB on every SSD), and segments composed of K=8 consecutive packets (i.e., segments are of 32 KB each).

# SSD	T_R	Packet Throughput	Total hits	DRAM hits	SSD hits
1	2	20 Gbps	11.91 bps	9.76 Gbps	2.15 Gbps
2	1	20 Gbps	12.44 bps	9.76 Gbps	2.68 Gbps

Table 1

We first analyze the overall system throughput and report the results in Table 1; We perform our evaluation using one and two SSD drives and operating with content objects of 1.5 MB. In both cases, we selected the value of T_R that maximizes the overall cache throughput (DRAM + SSD) for the scenario considered. The throughput is measured as the Data rate served by H2C averaged over a 60 second period.

The most striking result is that our design can operate at line speed (20 Gbps) even with only one SSD drive, thus validating the soundness of our design. In both cases considered overall throughput is limited by the network bandwidth. In the first case, however, the SSD drive operates near to the maximum throughput that we measured in isolation (2.15 out of 2.48 Gbps). Differently, in the second case, the two SSD drives operate far from their maximum throughput (2.68 out of 4.94 Gbps). Therefore there is a margin to further increase system throughput if the network bottleneck is removed.

We now investigate the hit ratio of H2C, which we break down into DRAM and SSD cache hit ratio in order to understand the contribution of each caching layer. We define DRAM and SSD hit ratios as the ratios of requests served by object located respectively in DRAM and SSD caches over the total number of requests. The total hit ratio is given by the sum of DRAM and the SSD hit ratios. Fig.3 reports the hit ratio as a function of the T_R threshold for object sizes S of 1.5 MB and 15 MB.

We expectedly observe that the DRAM hit ratio is not influenced by variations of T_R, while the SSD hit ratio determines the total hit ratio shape. Increasing the T_R threshold from 0 to 1 improves the SSD hit ratio, as it effectively prevents one-timers (i.e. content objects requested only once) from entering the SSD cache and evicting more popular contents.

Further increasing the T_R threshold degrades SSD hit ratio. However, selecting greater values of T_R has the desirable effect of reducing SSD write load, improving SSD read bandwidth and reducing wear.

It is also interesting to notice that the second layer of the cache significantly contributes to the overall throughput, i.e., from 13% for 1.5 MB content sizes to 33% for 15 MB contents.

Finally, we analyze the SSD throughput and how it is impacted by the choice of the T_R threshold.

To do that, we measure the read-write workload at the SSD layer interface and then we replay it locally to bypass the network bottleneck and evaluate the performance of the SSD layer in isolation.

The results of our experiments, conducted using one single SSD, are depicted in Fig.4. Results for the multi-SSD cases can be easily obtained multiplying throughput values of the single SSD case by the number of drives.

It can be immediately noticed that the maximum read/write throughput remains fairly constant independently on the read-write mix, as expected.

The most important aspect however is the impact of T_R selection on read-write mix and, as a direct consequence, on read and write throughput. In fact, increasing T_R from 0 to 1 results in a 3.66X read throughput gain, caused by SSD write reduction triggered by selective SSD insertion. Further increasing T_R improves read performance but, also leads to reduced cache hits as previously noticed.

The dotted line in Fig.4 represents the cumulative read-write throughput that the SSD layer must provide for the system to operate at line speed (20 Gbps). As shown in the graph, in this configuration, for T_R less than or equal to 1, the system throughput is limited by SSD bandwidth, which is encumbered by a heavy write workload. For T_R greater than or equal to 2, the bottleneck is at the network interface. Table 1, we selected T_R equal to 2 for experiments with one SSD exactly because it is the value maximizing cache hits while supporting line-speed operation. In the case of two SSD drives, the available SSD bandwidth being twice as high, only T_R equal to 1 is required.

CLAIMS

1. A method for managing a multiple level cache of a host comprising a primary cache which is a volatile memory such as a DRAM memory and a secondary cache which is a non-volatile memory such as a SSD memory, the method comprising the steps of:
   - dividing a content object into N segments, each of said N segments being composed of K consecutive packets, each of the N segments being identified by a segment identification data (segment^), said packets being stored in a first portion of said primary cache and/or in a second portion of said secondary cache, said first portion and said second portion being assigned to a corresponding processing core so that packets belonging to a same segment are always assigned to the same processing core,
   - building a segment hash table in order to index the packets of said N segments stored in said first portion and/or in said second portion, the entries of said segment hash table being related to a DRAM segment map pointer and/or a SSD segment map pointer depending on which portion among said first portion and said second portion a corresponding packet is cached in, the DRAM segment map pointer being configured to point to the first portion storing the packets of a given segment, the SSD segment map pointer being configured to point to the second portion storing the packets of said given segment, the DRAM segment map pointer of each of the N segments being managed as a DRAM-LRU linked list which is ordered, each of the DRAM segment map pointers containing a request counter for counting the number of times said given segment has been requested, the SSD segment pointer of each of the N segments being managed as a SSD-LRU linked list which is ordered,

- receiving a request from a requester and performing a look up in said segment hash table so as to determine if said segment identified by segment identification data (segment^) has already been built,

if said segment identification data has been computed in said segment hash table, said corresponding processing core checks whether the corresponding packet is stored in said first portion of said primary cache or in said second portion of said secondary cache,

- if the packet is stored in said first portion, said corresponding packet is sent back to said requester and said request counter is incremented, the DRAM segment map pointer entering in said DRAM-LRU linked list, the DRAM segment map pointer being prioritized by being moved on top of said DRAM-LRU linked list,
- if the packet is stored in said second portion, said corresponding packet is passed to an SSD core so as to copy the entire given segment from the secondary cache to the primary cache ; then said request is passed back to said corresponding processing core in order to create the DRAM segment map pointer for pointing to the first portion storing said corresponding packet so as to be entered in said DRAM-LRU linked list, the SSD segment map pointer being also entered in said SSD-LRU linked list, the DRAM segment map pointer and the SSD segment map pointer being respectively prioritized by being respectively moved on top of said DRAM-LRU linked list and said SSD-LRU linked list; then said corresponding packet is sent back to said requester.

2. Method according to claim 1, the method comprising :
- if the primary cache is full, evaluating the value of said request counter of the DRAM segment map pointer which is at the tail of said DRAM-LRU linked list,

- comparing the value of the request counter with a predetermined threshold T_R,
- if the value is above said predetermined threshold, said packets of the entire segment pointed to by the DRAM segment map pointer are transferred from the primary cache to the secondary cache and the SSD segment map pointer is prioritized by being moved on top of said SSD-LRU linked list,
- if the value is below said predetermined threshold, the DRAM segment map pointer and said packets of the entire segment pointed to by the DRAM segment map pointer are respectively deleted from the primary cache and from said DRAM-LRU linked list.

3. Method according to any one of the preceding claims, the method comprising:
- if the secondary cache is full, the SSD segment map pointer of the segment to be deleted which is at the tail of said SSD-LRU linked list is deleted from said SSD-LRU linked list and from the entries of said segment hash table if there is no DRAM segment map pointer corresponding to the segment to be deleted in the DRAM-LRU linked list.

4. Method according to any one of the preceding claims, the method comprising :
- receiving a data and performing a lookup in said segment hash table so as to determine if said segment identified by segment identification data (segment^) has already been stored, if said segment identification data has not been computed in said segment hash table, an entry is created in said segment hash table and a corresponding DRAM segment map pointer is created so as to be entered and prioritized in said DRAM-LRU linked list by the assigned processing core.

5. Method according to any one of the preceding claims, wherein, when said corresponding packet is passed to an SSD core so as to copy the entire given segment comprising the said packet from the secondary cache to the primary cache, said corresponding segment is not removed from the secondary cache.

6. Method according to one of the preceding claims, wherein, if said segment identification data has not been stored in said segment hash table, said request is forwarded to a next hop.

7. Method according to one of the preceding claims, wherein the primary cache is a DRAM memory and the secondary cache is a SSD memory.

8. Method according to one of the preceding claims, wherein said DRAM-LRU linked list and said SSD-LRU linked list are ordered according to a Least Recently Used (LRU) policy.

9. Method according to one of the preceding claims, wherein :
   - each of the DRAM segment map pointers further containing pointers to previous and next segments in said DRAM-LRU linked list,
   - each of the SSD segment map pointers further containing pointers to previous and next segments in said SSD-LRU linked list .

10. A computer program product comprising computer-executable instructions for performing a method according to any of the claims 1 to 11.

11. A computer readable medium having a computer program comprising program instructions, the computer program being loadable into a data-processing unit and adapted to cause execution of the method according

to any of the claims 1 to 9 when the computer program is run by the data-processing unit.

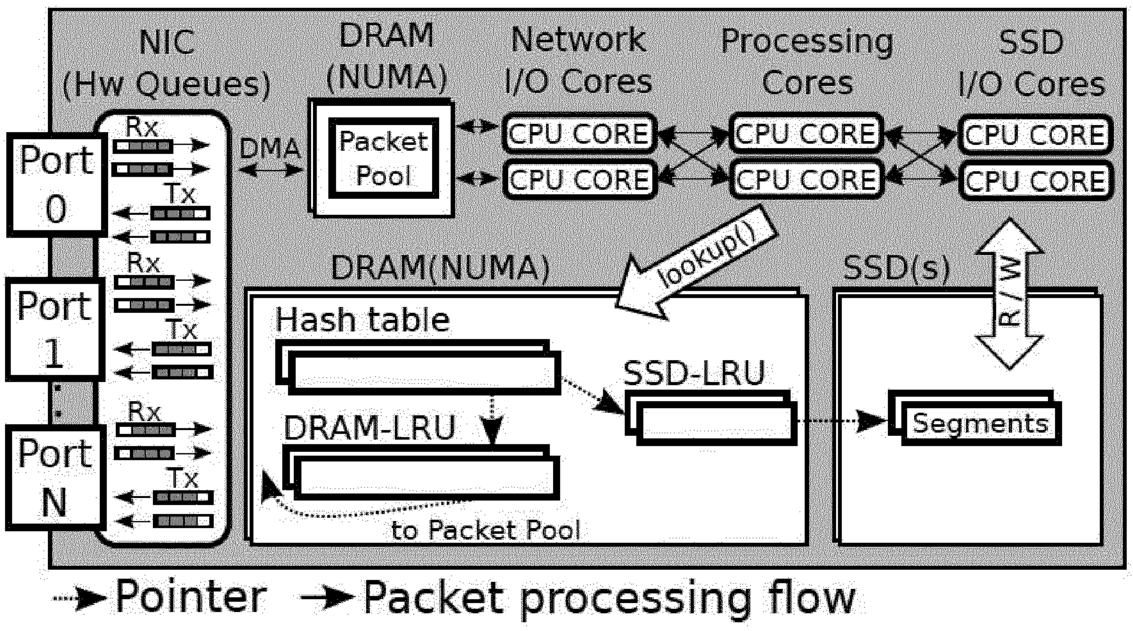

Figure 1

Name: $\underbrace{\text{/usenix/hotstorage15/stream.avi}}_{\text{content object name}}\underbrace{\text{/10}}_{\text{packet identifier}}$

Hash Table entry:

Active (1B)	Segment hash (4B)	SSD flag (1B)	DRAM packet bitmap(2B)	SSD-LRU pointer(4B)	DRAM-LRU pointer(4B)

DRAM Segment Map:

Req. (1B)	Prev (4B)	Next (4B)	Hash table pointer(4B)	Chunk 1 index(4B)	...	Chunk N index(4B)

SSD Segment Map:

Prev (4B)	Next (4B)	Segment ID (var. len.)	Hash table pointer(4B)

Figure 2

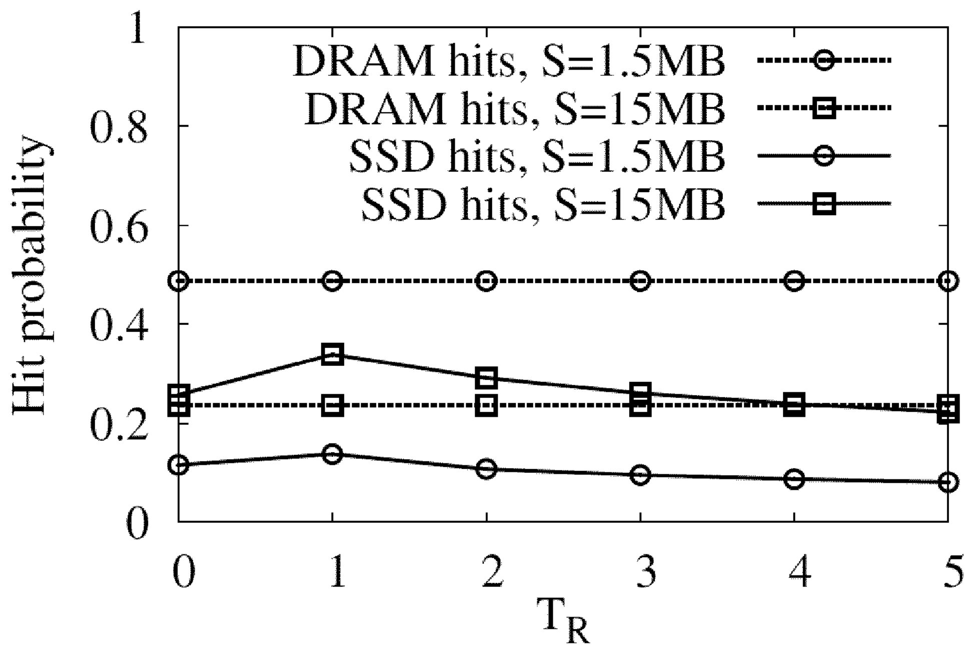

Figure 3

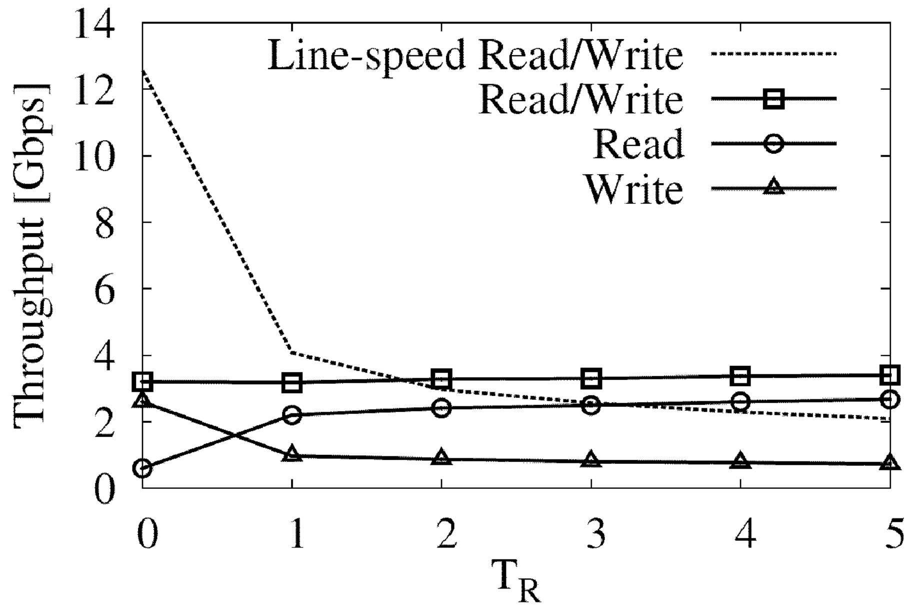

Figure 4

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2016/065897

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F12/08 G06F12/12
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal , WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2014/201448 Al (BENHASE MICHAEL T [US] ET AL) 17 July 2014 (2014-07-17) paragraphs [0006] - [0008], [0027] - [0038]; claim 1; figure 3 -----	1-11

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" documentwhich may throw doubts on priority claim(s) orwhich is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search 19 July 2016	Date of mailing of the international search report 01/08/2016
Name and mailing address of the ISA/ European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Authorized officer Jardon , Stephan

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2016/065897

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014201448 A1	17-07-2014	DE 102012219098 A1	02-05-2013
		GB 2505969 A	19-03-2014
		US 2013111134 A1	02-05-2013
		US 2013185512 A1	18-07-2013
		US 2014201448 A1	17-07-2014
		US 2015286580 A1	08-10-2015