



(51) International Patent Classification:
G06F 12/08 (2006.01) **G06F 9/46** (2006.01)
G06F 12/14 (2006.01)

(21) International Application Number:
PCT/EP2011/061513

(22) International Filing Date:
7 July 2011 (07.07.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10305868.1 6 August 2010 (06.08.2010) EP

(71) Applicant (for all designated States except US): **ALCATEL LUCENT** [FR/FR].

(72) Inventors; and

(75) Inventors/Applicants (for US only): **MULLENDER, Sape** [NL/BE]; Anjelierstraat 15, B-2600 Antwerp (BE). **MCKIE, James Balmer** [US/US]; 15, Chestnuthill Drive, Murray Hill, NJ 07974 (US). **PIANESE, Fabio** [IT/BE]; Rue de la Madeleine 61, B-1000 Brussels (BE). **EVANS, Noah** [US/BE]; Zwaardstraat 14, B-2000 Antwerp (BE).

(74) Agent: **MEYER ZU BEXTEN, Elmar**; Alcatel-Lucent Deutschland AG, Intellectual Property & Standards, Lorenzstrasse 10, 70435 Stuttgart (DE).

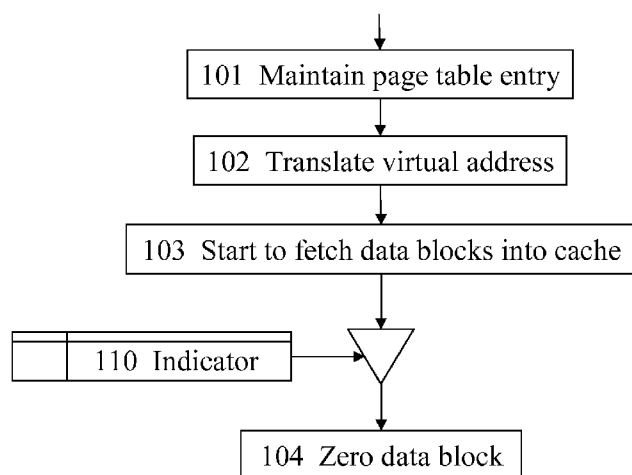
(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, 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, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, 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, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

[Continued on next page]

(54) Title: A METHOD OF MANAGING COMPUTER MEMORY, CORRESPONDING COMPUTER PROGRAM PRODUCT; AND DATA STORAGE DEVICE THEREFOR



(57) Abstract: The invention concerns a method of managing computer memory, the method comprising the steps of maintaining (101) a page table entry for mapping a virtual address to a physical address and a cache comprising a plurality of data blocks and, in response to a reference to the virtual address, translating (102) the virtual address into the physical address by means of the page table entry and fetching (103) data from the physical address into the cache, wherein the page table entry comprises a plurality of indicators, each data block corresponding to an indicator, and, upon fetching the data into the cache, the method comprises the further step of, in response to an indicator being set, zeroing (104) the corresponding data block. The invention further concerns a computer program product and a device therefor.

Fig. 1



-
- *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*
- *of inventorship (Rule 4.17(iv))*
- Published:**
- *with international search report (Art. 21(3))*

A method of managing computer memory,
corresponding computer program product,
and data storage device therefor

5 **Field of the Invention**

The invention relates to a method of managing computer memory comprising the steps of maintaining a page table entry for mapping a virtual address to a physical address and a cache comprising a plurality of data blocks and, in response to a reference to the virtual address, translating the virtual address into the physical address by means of the page table entry and fetching data from the physical address into the cache. The invention further relates to a computer program product comprising computer-executable instructions for performing said method when the program is run on a computer, and to a device programmed or configured to perform said method.

Background

In computing, memory management is the act of managing computer memory. In its simpler forms, this involves providing ways to allocate portions of memory to programs at their request and freeing it for reuse when no longer needed. The management of memory is critical to any computer system.

Virtual memory systems separate memory addresses used by a process from physical addresses, allowing separation of processes and increasing the effectively available amount of memory. The quality of the virtual memory manager has a significant impact on overall system performance.

Ludmila Cherkasova and Rob Gardner outline a virtual memory system in "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor", Proceedings of the USENIX Annual Technical Conference 2005, Anaheim, California, United States of America. To allow an operating system (OS) to retrieve data from secondary storage for use in main memory, this conventional system makes use of a memory management scheme known as paging, wherein the data is retrieved in fixed-size blocks called memory pages. Herein, by secondary storage, sometimes called external memory, is meant any storage not directly accessible by the central processing unit (CPU), such as a hard disk drive, optical storage, flash memory, floppy disk, magnetic tape, paper tape, punched card, or Zip drive.

To allow for efficient communications with other systems, commonly called input/output or I/O, the state-of-the-art virtual memory system implements a mechanism known as page flipping. According to this technique, an application specifies one or more memory pages for receiving input data, allowing the OS to supply that data by means of its built-in paging scheme. To this end, the OS "swaps" the specified memory pages in the application's working memory, commonly called its address space, with the requested input.

A major downside of this known system lies in the fact that the input data to be supplied rarely fits the target memory pages precisely. The remainder of each memory page swapped in thus needs to be zeroed to cater for applications that rely on memory initialization as well as to prevent the requesting application from obtaining unauthorized access to foreign, potentially sensitive data. In software development, by zeroing is meant overwriting data with a fixed, meaningless value, e.g., zero, to prevent its accidental disclosure, such disclosure potentially allowing a security breach by the requesting application.

According to US 5 920 895 A, the efficiency of writing files

that are cached using mapped file I/O is improved by suppressing zeroing of uninitialized data in cached pages of a file until the file is mapped by a user mode thread. In an operating system where paging operations are controlled by a virtual memory manager and memory based caching using mapped file I/O is administered by a cache manager, suppressing zeroing of mapped files on writes is implemented by a set of internal operating system interfaces for communications between the virtual memory manager and the cache manager. When a file being cached is not yet mapped by a user mode thread, the cache manager tracks the extent to which a cache page of the file is written so that any uninitialized data in the cache page can later be zeroed when the file is mapped by a user mode thread.

A method for preventing digital piracy in a computing environment according to US 2008/229117 A1 comprises loading an application into the computing environment, wherein the application is encrypted using a cryptographic key; assigning a virtual address space to the application; loading the cryptographic key for the application into a register which is accessible only by a central processing unit; and storing an index value for the key in the register in a page table entry which corresponds to the virtual address space for the application, thereby linking the virtual address space to the key for the application.

In MENON A., ET AL.: "Optimizing Network Virtualization in Xen", PROCEEDINGS OF THE 2006 USENIX ANNUAL TECHNICAL CONFERENCE, 29 May 2006 (2006-05-29), XP002623699, the authors propose and evaluate three techniques for optimizing network performance in the Xen virtualized environment. Their techniques retain the basic Xen architecture of locating device drivers in a privileged 'driver' domain with access to I/O devices, and providing network access to unprivileged 'guest' domains through virtualized network interfaces.

Summary

It is an objective of the invention to present an improved approach to virtual memory management that eliminates or reduces the need for zeroing upon page flipping. It is a further
5 objective to provide a method that is suitable for memory of limited bandwidth, that is, the rate at which data can be read from or stored in the memory by a CPU. An even further objective lies in the reduction of load imposed on the memory bus,
10 that is, the computer subsystem that connects the main memory to the memory controller managing the flow of data going to and from that main memory. Herein, by main memory, also called primary storage or internal memory, is meant any storage directly accessible by the CPU, such as random-access memory (RAM),
15 read-only memory (ROM), processor registers, or processor caches.

This objective is achieved by a method of managing computer memory comprising the steps of maintaining a page table entry for mapping a virtual address to a physical address and a cache comprising a plurality of data blocks and, in response to a
20 reference to the virtual address, translating the virtual address into the physical address by means of the page table entry and fetching data from the physical address into the cache, wherein the page table entry comprises a plurality of indicators, each data block corresponding to one of the plurality of
25 indicators, and, once fetching the data into the cache has started, the method comprises the further step of, in response to an indicator, selected from said plurality of indicators, being set, zeroing the corresponding data block. The objective further is achieved by a computer program product comprising
30 computer-executable instructions for performing said method when the program is run on a computer, or by a device programmed or configured to perform said method.

A main idea of the invention is to augment the page table, a

data structure used by the virtual memory system of the OS to store the mapping between a virtual address, that is, the index for a location in an application's working memory, and the corresponding physical address. In the context of virtual memory, a virtual address is unique only to the accessing process, whereas a physical address refers to the actual storage cell of main memory.

Further developments of the invention can be gathered from the dependent claims and the following description.

In the following the invention will be explained further making reference to the attached drawing.

To manage computer memory according to an embodiment of the invention, a page table entry is maintained to map a virtual address to a physical address. Further, a cache comprising a plurality of data blocks is maintained. The page table entry comprises a plurality of indicators, each data block corresponding to an indicator. In response to a reference to the virtual address, the latter is translated into the physical address by means of the page table entry, and data is fetched from that physical address into the cache. Upon fetching the data, in response to an indicator being set, the corresponding data block is zeroed.

Brief Description of the Figures

Fig. 1 shows a flowchart depicting a method according to an embodiment of the invention.

Description of the Embodiments

In the following, a method according to the invention is elucidated by way of example, referencing Fig. 1.

The flowchart 100 of Fig. 1 comprises a first processing step 101, a second processing step 102, a third processing step 103,

and a fourth processing step 104. A set of arrows represents the flow of control passing through the processing steps 101 to 103, merging with internal storage 110, and finally passing into the fourth processing step 104. Within this flow, an arrow starting at a first symbol and ending at a second symbol indicates that control passes from the first symbol to the second symbol.

In the embodiment at hand, the method is applied by a dedicated memory management unit (MMU). In microprocessor design, by MMU is meant any computer hardware component responsible for handling access to memory requested by the CPU. Besides virtual memory management, that is, the translation of virtual addresses to physical addresses, the MMU may support techniques such as memory protection, bus arbitration, and, in simpler computer architectures such as 8-bit systems, bank switching.

To allow for processes to be based on a notion of contiguous working memory, commonly called an address space, while in fact that memory may be physically fragmented and may even overflow on to secondary storage, the MMU supports paging, thus taking the form of what is known in the art as a paged MMU (PMMU).

To reduce the average time required by the CPU to access memory, the PMMU further controls a CPU cache, that is, a smaller, faster memory configured to store copies of data from the most frequently used main memory locations. When the CPU needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the CPU immediately reads from or writes to the cache, which is inherently faster than reading from or writing to main memory. In the context of caching, the main memory is sometimes referred to as the cache's backing store.

More specifically, the CPU cache controlled by the PMMU comprises a data cache to speed up data fetch and store as well as a translation look-aside buffer (TLB) to speed up virtual mem-

ory management for both executable instructions and data. The TLB comprises a fixed number of slots that contain page table entries (PTEs) mapping virtual addresses to physical addresses. In a typical implementation, each PTE comprises between 32 and 64 bits.

In the first processing step 101, the PMMU maintains a PTE for mapping a virtual address to a physical address. Further, the PMMU maintains a cache comprising a plurality of data blocks, commonly called cache lines or cache blocks in the context of CPU caching. Typically, each data block ranges in size from 8 to 512 bytes. For maximum hardware acceleration, the PMMU at hand employs data blocks larger than the amount of data that can be requested by a single CPU instruction, which typically ranges from 1 byte to 16 bytes in state-of-the-art 32-bit and 64-bit CPU architectures.

The PTE maintained in the first processing step 101 comprises a plurality of indicators, each data block corresponding to an indicator, indicating whether the corresponding data block needs to be zeroed before providing it to a requesting application. Consequently, the indicator takes the form of a flag, that is, a Boolean variable having either of the values "true" or "false". To minimize the storage capacity required by the PTE, each indicator takes the form of a bit, each of its binary values having an assigned meaning. To further allow the CPU to set multiple indicators in a single bitwise operation, the bits are grouped into a suitably sized vector known in computer arithmetic as a bitmask. In an alternative embodiment, to further decrease the size of the PTE at the expense of average write speed, each bit may be interpreted to correspond to a plurality of data blocks, effectively reducing the number of bits that constitute the bitmask.

In an intermediate step (not depicted), the OS, on request by an application, receives a data block over a computer network by means of an associated network interface controller (NIC),

also known as a network interface card, network adapter, or Local Area Network (LAN) adapter. For maximum throughput, the data block is transmitted by means of a Gigabit Ethernet (GbE, 1 GbE) connection, that is, at a rate of one gigabit per second, as defined by the IEEE 802.3-2008 standard. To further speed up the operation by avoiding the overhead of copying the I/O data by means of the CPU, the OS employs page flipping to store it in a memory page specified by the application. Assuming that only some of the data blocks that make up the memory page are modified by the I/O operation, instead of instantly zeroing the remaining, unaffected data blocks, the OS sets the bits in the PTE's bitmask that correspond to the data blocks to be zeroed while clearing those bits that correspond to the modified data blocks.

Ethernet being a packet switching protocol, the I/O data takes the form of a data transmission unit called a packet or, in this context, Ethernet frame. The frame may vary in size between a lower limit of 64 bytes, such as required for a simple acknowledgment, and an upper limit, commonly called maximum transmission unit (MTU) in the art of computer networking, of 9000 bytes or more. Taking into account a typical memory page size of up 4 to 64 kilobytes, a conventional memory management system, in this scenario, would impose a significant zeroing overhead on the CPU and memory due to the significant portion of the memory page that needs to be purged upon receiving each packet. A further benefit of the invention thus lies in its particular aptitude for the receipt of small to medium-sized packet data.

In most cases, the amount of data received through the NIC may not coincide with a multiple of the fixed size of a data block as defined by the PTE. In such cases, at least part of the first and/or last data block affected by the I/O remain undefined. To avoid disclosure of the data contained therein to the requesting application, the OS needs to zero that remaining

part of the data block instantly prior to page flipping.

In an alternative embodiment, instead of receiving the data through a NIC, the OS may be requested to load a program into main memory for execution by the CPU. In this case, to avoid a physical transfer of the entire executable, the program loader updates the page table and TLB, effectively declaring a mapping of the virtual address designated for the executable to the contents of the associated object file. The segment of virtual memory thus mapped is commonly called a memory-mapped file, bearing the advantage that unused program code may never need to be loaded into main memory at all.

As the case may be, the program to be loaded comprises a section known as a block started by symbol (BSS), often referred to as a BSS segment, containing statically allocated variables that are expected to be filled with zero-valued data initially, that is, when execution of the program begins. In this case, to avoid zeroing of the entire BSS section prior to execution, the OS employs the above mechanism analogously by setting the bits in the PTE's bitmask that correspond to the data blocks to be zeroed while clearing those bits that correspond to other data segments of the program.

In the second processing step 102, in response to the virtual address being referenced by a CPU instruction, the PMMU fulfills its primary function of virtual memory management by translating the virtual address into the physical address. To this end, the PMMU employs the PTEs contained in the TLB of the first processing step 101. More particularly, the PMMU first searches the TLB for a PTE that corresponds to the virtual address at hand. If a match is found, a situation known in the art of virtual memory management as a TLB hit, the physical address can be retrieved directly from the matching PTE. Otherwise, the situation is commonly called a TLB miss, requiring the PMMU to consult the page table maintained by the OS. If this page table contains a matching PTE, the PMMU caches it in

the TLB and re-executes the faulting CPU instruction.

In the third processing step 103, the PMMU accesses the physical address derived in the second processing step 102 and starts to fetch data blocks contained therein into the CPU's data cache. If the data cache is already full at this point, an occupied cache line may need to be overwritten and its prior contents discarded to accommodate the newly fetched data block. Various algorithms are known in the art for selecting which cache line to evict.

To speed up any write operation on the data cache, the latter takes the form of a write-back cache, sometimes called write-behind cache. According to this policy, writes are not immediately mirrored to the backing store. Instead, the PMMU tracks which cache lines have been overwritten by the CPU and accordingly marks these locations as "dirty". Prior to eviction of a dirty cache line, the respective data block is written back to the backing store to preserve its updated contents for future read access. This approach is known in computer science as a "lazy write". Upon writing back a data block, the PMMU needs to clear the corresponding bit in the PTE's bitmask to prevent that data block from being zeroed in case it is ever retrieved.

Once a free cache line has been identified or an occupied one has been freed, for each data block to be fetched, the PMMU consults the bitmask contained in the PTE found in its internal storage 110. If its corresponding bit is set, the PMMU, instead of retrieving the data block from main memory, zeros the respective cache line in the fourth processing step 104 prior to making it available to the requesting application.

A person of skill in the art would readily recognize that steps of various above-described methods can be performed by programmed computers. Herein, some embodiments are intended to cover program storage devices, e.g., digital data storage media, which are machine or computer readable and encode machine-

executable or computer-executable programs of instructions where said instructions perform some or all of the steps of methods described herein. The program storage devices may be, e.g., digital memories, magnetic storage media such as a magnetic disks or tapes, hard drives, or optically readable digital data storage media. The embodiments are also intended to cover computers programmed to perform said steps of methods described herein.

The present inventions may be embodied in other specific apparatus and/or methods. The described embodiments are to be considered in all respects as only illustrative and not restrictive. In particular, the scope of the invention is indicated by the appended claims rather than by the description and figures herein. All changes that come within the meaning and range of equivalency of the claims are to be embraced within their scope.

Claims

1. A method of managing computer memory, the method comprising the steps of
 - 5 maintaining (101) a page table entry for mapping a virtual address to a physical address and a cache comprising a plurality of data blocks and,
 in response to a reference to the virtual address, translating (102) the virtual address into the physical
10 address by means of the page table entry and fetching (103) data from the physical address into the cache,
 characterized in that the page table entry comprises a plurality of indicators, each data block corresponding to one of the plurality of indicators, and, once fetching the
15 data into the cache has started (103), the method comprises the further step of,
 in response to an indicator (110), selected from said plurality of indicators, being set, zeroing (104) the corresponding data block.
- 20 2. A method according to claim 1, characterized in that the page table entry comprises a bitmask and the indicators are bits contained in the bitmask.
3. A method according to claim 1, characterized in that the page table entry is associated with a memory page comprising the plurality of data blocks and the method comprises the intermediate steps of
 - 25 receiving a data block,
 storing the data block in the memory page, and
 clearing the indicator to which the data block corresponds.
30
4. A method according to claim 1, characterized in that the method comprises the further step of

setting the remaining ones of said plurality of indicators, to which the data block does not correspond.

5. A method according to claim 3 or 4, characterized in that the method comprises the subsequent steps of
5 overwriting a further data block and
 clearing the indicator to which the further data block corresponds.

6. A computer program product comprising computer-executable instructions for performing a method according to claim 1
10 when the program is run on a computer.

7. A computer program product according to claim 6, characterized in that the computer program product comprises an operating system.

8. A device comprising
15 means for maintaining (101) a page table entry for mapping a virtual address to a physical address and a cache comprising a plurality of data blocks and means for,
 in response to a reference to the virtual address, translating (102) the virtual address into the physical
20 address by means of the page table entry and fetching (103) data from the physical address into the cache,
 characterized in that the page table entry comprises a plurality of indicators, each data block corresponding to one of the plurality of indicators, and the device comprises further means for, upon fetching (103) the data in-
25 to the cache and
 in response to an indicator (110), selected from said plurality of indicators, being set, zeroing (104) the corresponding data block.

9. A device according to claim 8, characterized in that the device comprises at least one of the following:
30 a central processing unit,

a memory management unit, and
a data cache.

- 5 10. A device according to claim 8 or 9, characterized in that
the device further comprises a translation look-aside
buffer configured to store the page table entry, wherein
the virtual address is translated (102) by means of the
translation look-aside buffer.

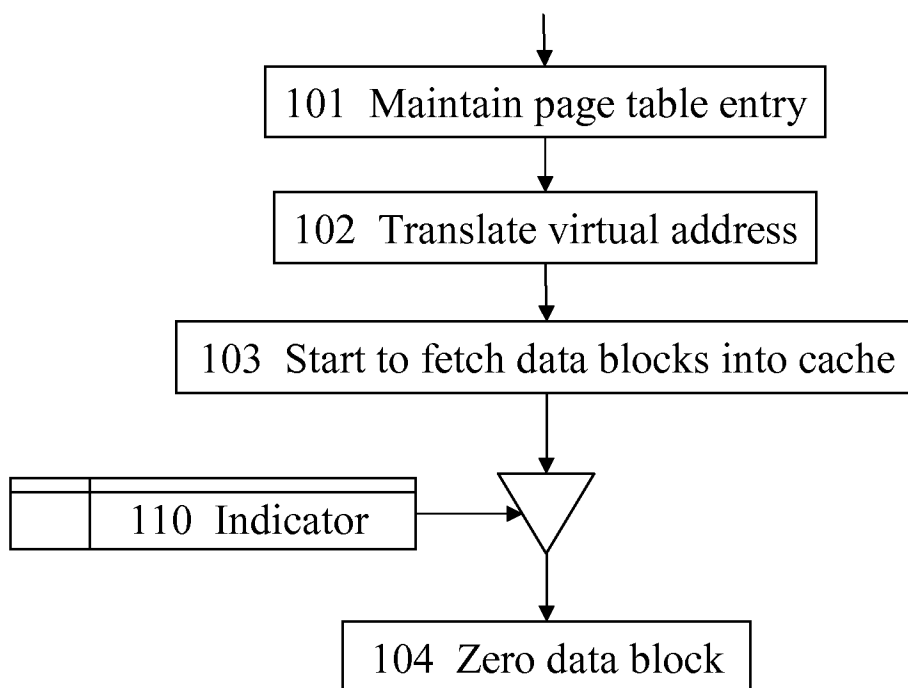
100

Fig. 1

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2011/061513

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06F12/08 G06F12/14 G06F9/46
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 practical, search terms used)

EPO-Internal, WPI Data, COMPENDEX, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 920 895 A (PERAZZOLI JR FRANK LOUIS [US] ET AL) 6 July 1999 (1999-07-06) abstract column 2, line 42 - column 4, line 30 -----	1-10
A	US 2008/229117 A1 (SHIN KANG G [US] ET AL) 18 September 2008 (2008-09-18) abstract paragraph [0023] ----- -/--	1-10



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 document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which 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

25 August 2011

Date of mailing of the international search report

02/09/2011

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

Mandato, Davide

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2011/061513

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	MENON A., ET AL.: "Optimizing Network Virtualization in Xen", PROCEEDINGS OF THE 2006 USENIX ANNUAL TECHNICAL CONFERENCE, 29 May 2006 (2006-05-29), pages 15-28, XP002623699, abstract page 16, right-hand column, line 21 - page 17, left-hand column, line 1 page 18, right-hand column, line 44 - page 19, right-hand column, line 1 -----	1-10
A	CHERKASOVA, L. ET AL.: "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor", PROCEEDINGS OF THE USENIX ANNUAL TECHNICAL CONFERENCE, 20 May 2001 (2001-05-20), XP002623700, cited in the application the whole document -----	1-10

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/EP2011/061513

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
US 5920895	A	06-07-1999	NONE	

US 2008229117	A1	18-09-2008	NONE	
