US 20170351639A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2017/0351639 A1**

Borikar (43) **Pub. Date: Dec. 7, 2017**

(54) **REMOTE MEMORY ACCESS USING MEMORY MAPPED ADDRESSING AMONG MULTIPLE COMPUTE NODES**

(71) Applicant: **CISCO TECHNOLOGY, INC.**, San Jose, CA (US)

(72) Inventor: **Sagar Borikar**, San Jose, CA (US)

(73) Assignee: **CISCO TECHNOLOGY, INC.**, San Jose, CA (US)

(21) Appl. No.: **15/174,718**

(22) Filed: **Jun. 6, 2016**

Publication Classification

(51) **Int. Cl.**

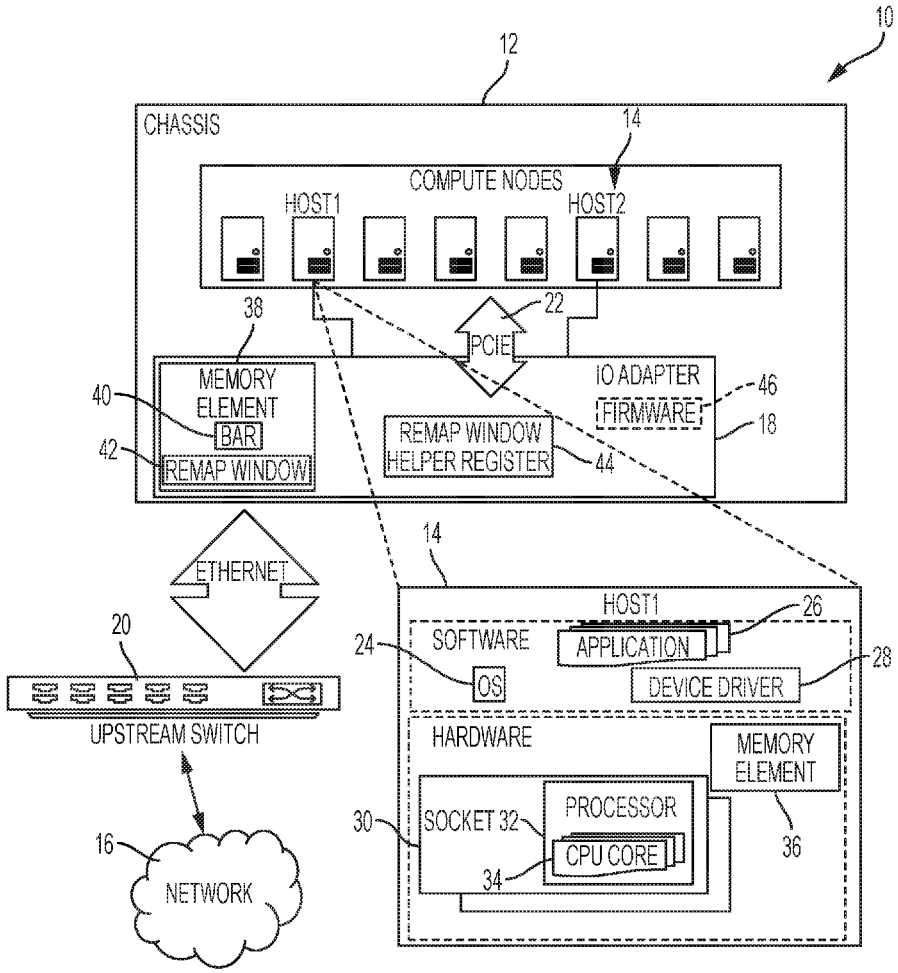
G06F 15/173	(2006.01)
G06F 13/42	(2006.01)
H04L 29/08	(2006.01)
H04L 29/06	(2006.01)

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

CPC ***G06F 15/17331*** (2013.01); ***H04L 69/16*** (2013.01); ***G06F 13/4282*** (2013.01); ***H04L 67/1097*** (2013.01)

(57) **ABSTRACT**

An example method for facilitating remote memory access with memory mapped addressing among multiple compute nodes is executed at an input/output (IO) adapter in communication with the compute nodes over a Peripheral Component Interconnect Express (PCIE) bus, the method including: receiving a memory request from a first compute node to permit access by a second compute node to a local memory region of the first compute node; generating a remap window region in a memory element of the IO adapter, the remap window region corresponding to a base address register (BAR) of the second compute node; and configuring the remap window region to point to the local memory region of the first compute node, wherein access by the second compute node to the BAR corresponding with the remap window region results in direct access of the local memory region of the first compute node by the second compute node.



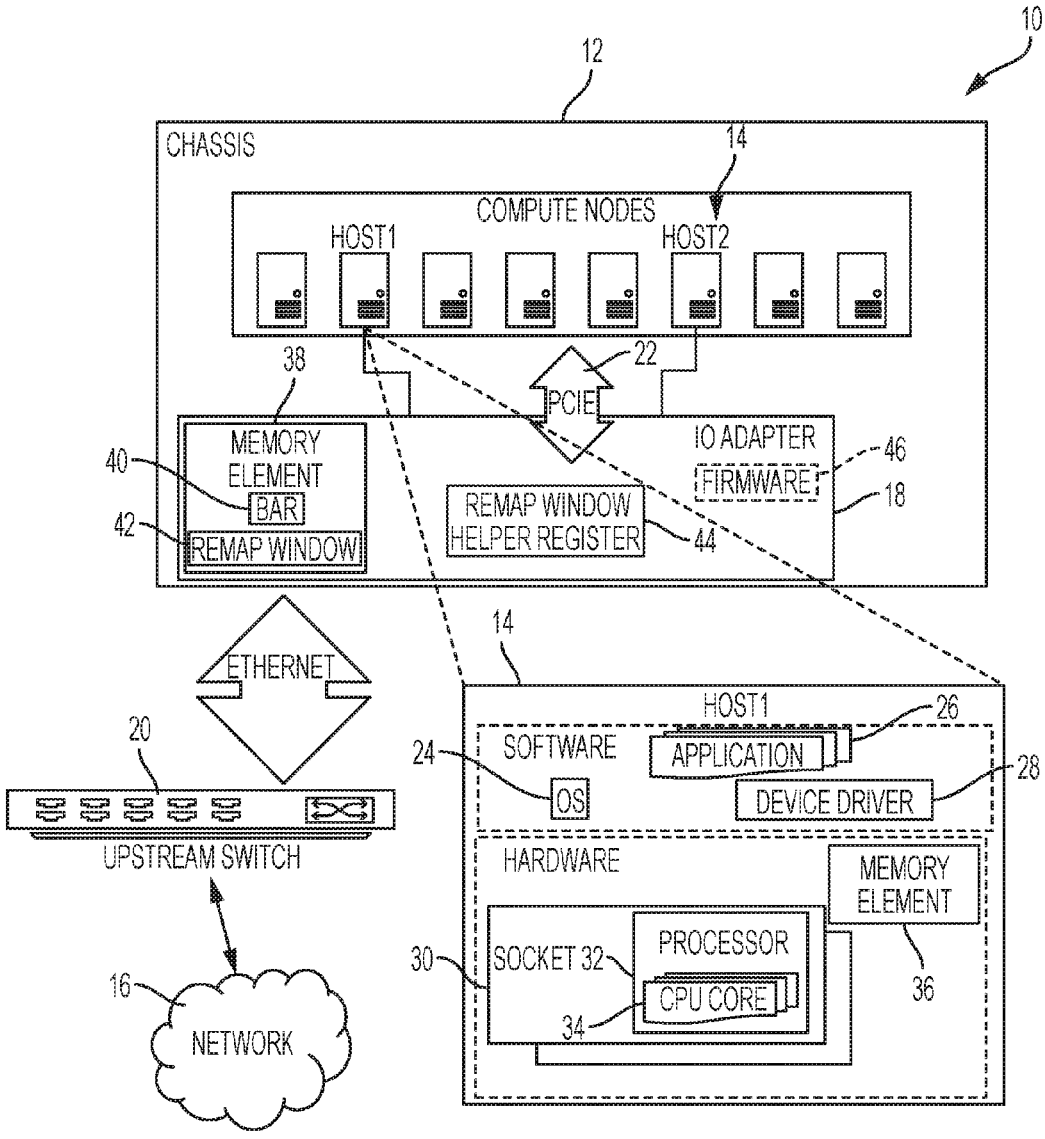

10
12
CHASSIS
14
COMPUTE NODES
HOST1
HOST2
38
22
PCIE
MEMORY ELEMENT
BAR
REMAP WINDOW
40
42
IO ADAPTER
FIRMWARE
46
18
REMAP WINDOW HELPER REGISTER
44
ETHERNET
14
HOST1
26
SOFTWARE
APPLICATION
24
OS
DEVICE DRIVER
28
20
UPSTREAM SWITCH
HARDWARE
MEMORY ELEMENT
SOCKET 32
PROCESSOR
CPU CORE
30
34
36
16
NETWORK

FIG. 1

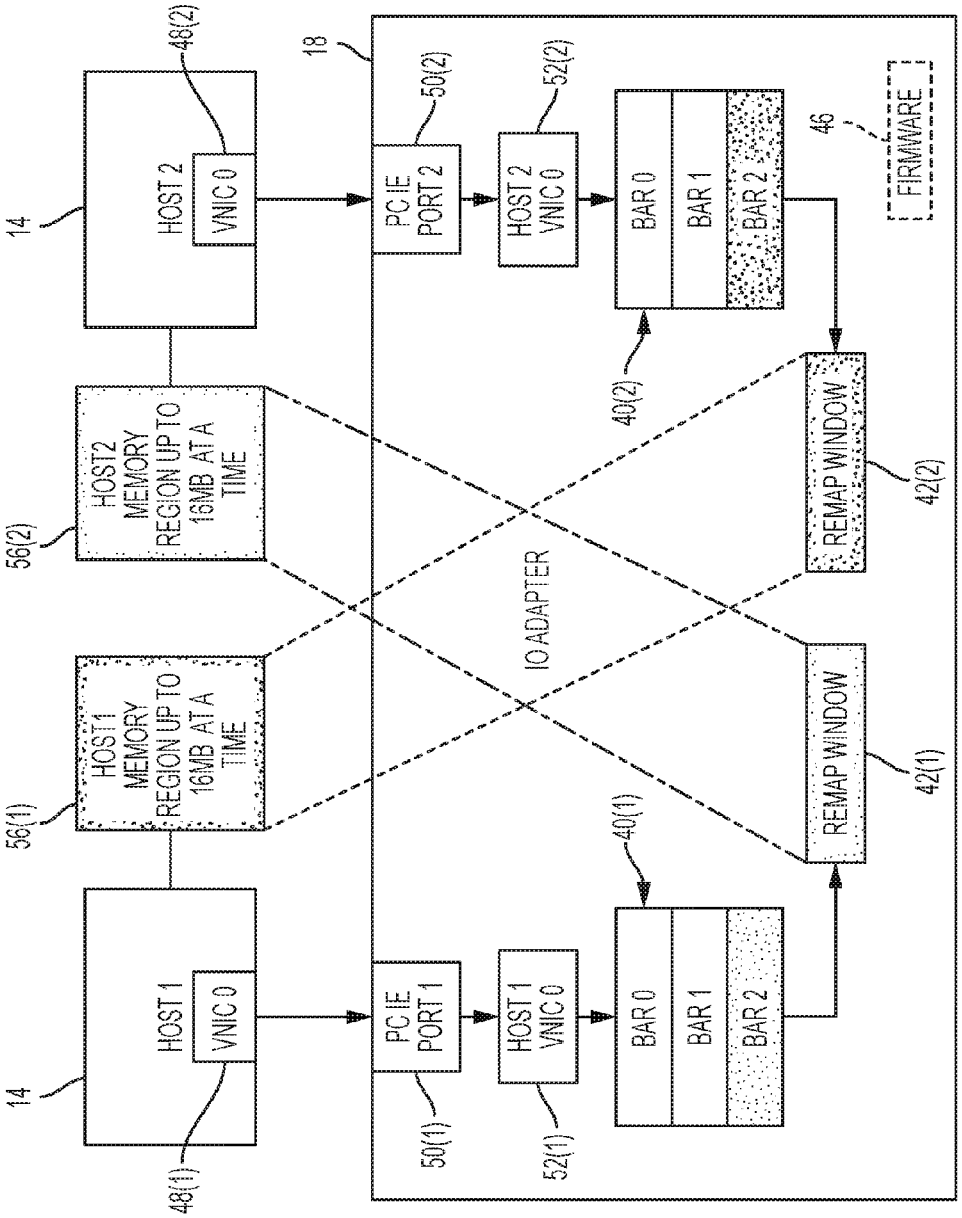

14
HOST 1
VNIC 0
48(1)
14
HOST 2
VNIC 0
46(2)
56(1)
HOST1 MEMORY REGION UP TO 16MB AT A TIME
56(2)
HOST2 MEMORY REGION UP TO 16MB AT A TIME
18
IO ADAPTER
PC IE PORT 1
50(1)
PC IE PORT 2
50(2)
HOST 1 VNIC 0
52(1)
HOST 2 VNIC 0
52(2)
BAR 0
BAR 1
BAR 2
40(1)
40(2)
REMAP WINDOW
42(1)
REMAP WINDOW
42(2)
46
FIRMWARE

FIG. 2

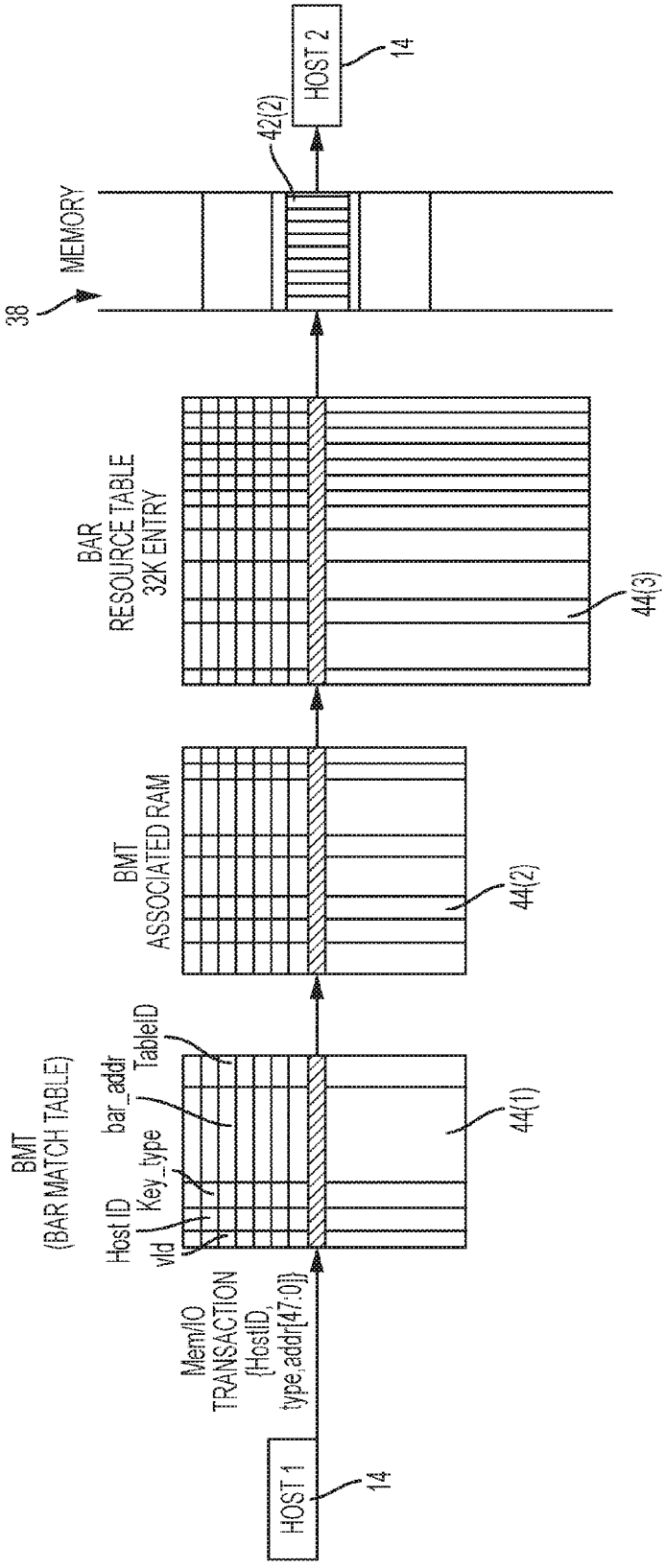

HOST 1
14
Mem/IO TRANSACTION {HostID, type,addr[47:0]}
Host ID
vld
Key_type
bar_addr
TableID
BMT (BAR MATCH TABLE)
44(1)
BMT ASSOCIATED RAM
44(2)
BAR RESOURCE TABLE 32K ENTRY
44(3)
38
MEMORY
42(2)
HOST 2
14

FIG. 3

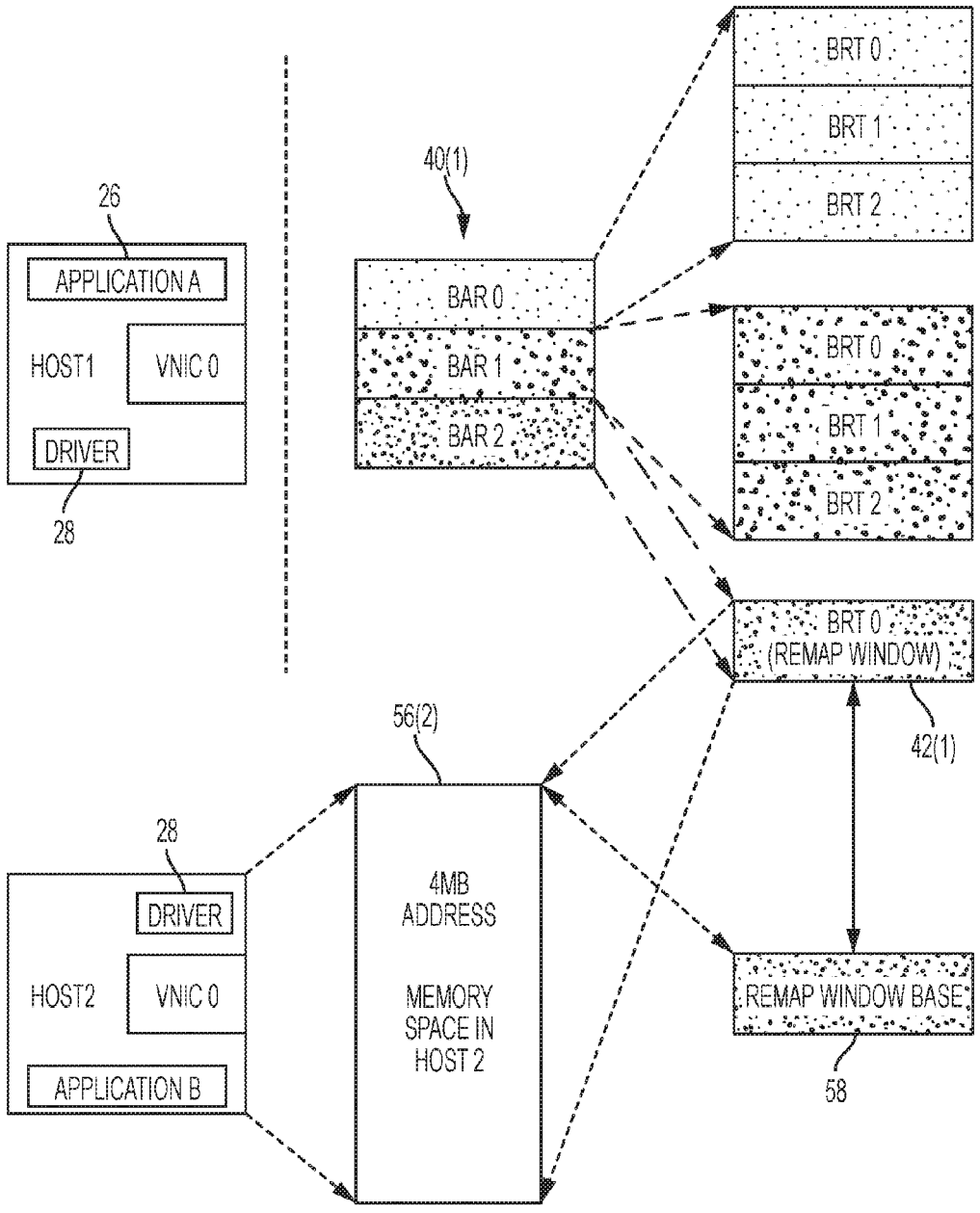

26
APPLICATION A
HOST1
VNIC 0
DRIVER
28
40(1)
BAR 0
BAR 1
BAR 2
BRT 0
BRT 1
BRT 2
BRT 0
BRT 1
BRT 2
BRT 0
(REMAP WINDOW)
42(1)
56(2)
28
DRIVER
HOST2
VNIC 0
APPLICATION B
4MB
ADDRESS
MEMORY
SPACE IN
HOST 2
REMAP WINDOW BASE
58

FIG. 4

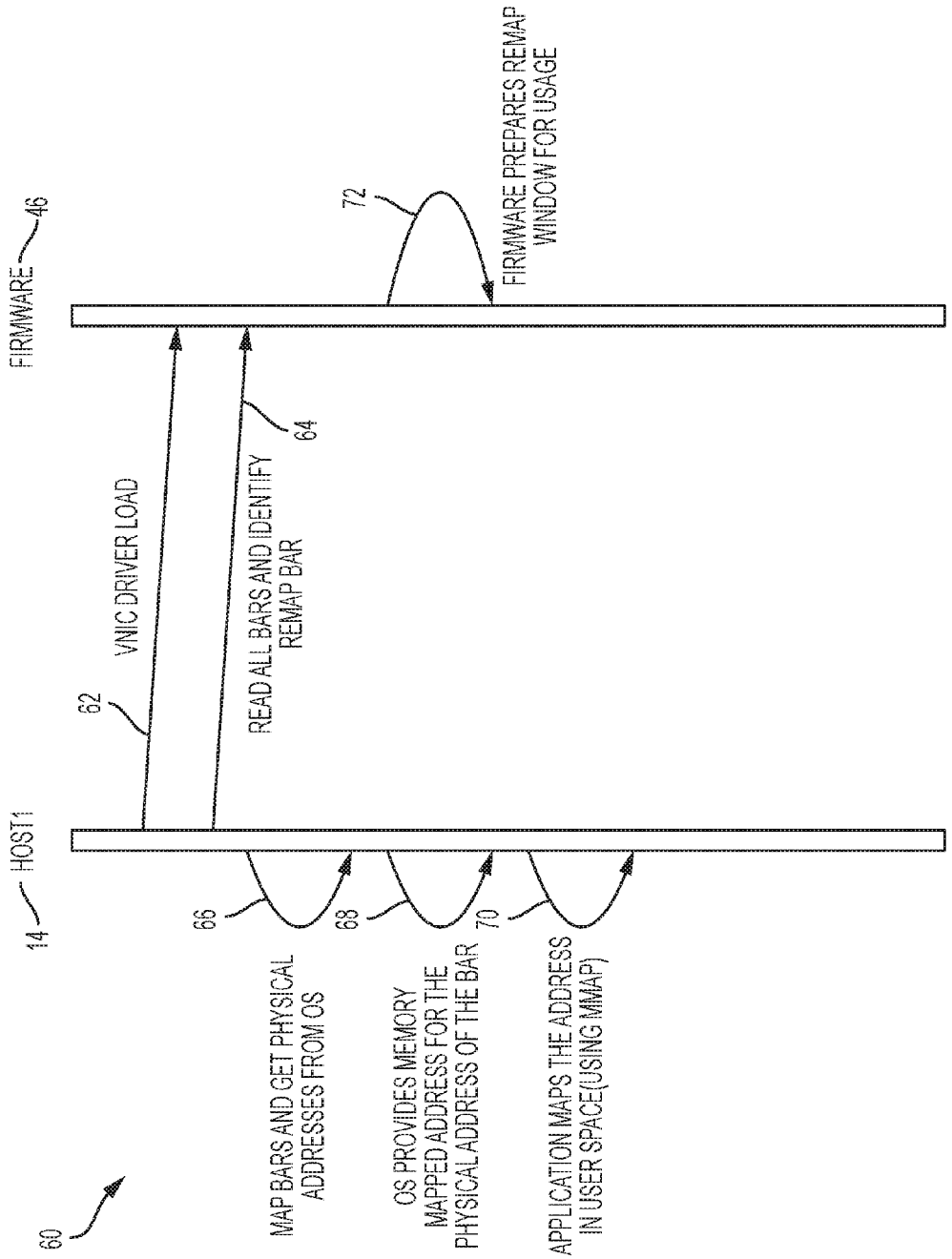

FIRMWARE
46
72
FIRMWARE PREPARES REMAP WINDOW FOR USAGE
64
READ ALL BARS AND IDENTIFY REMAP BAR
62
VNIC DRIVER LOAD
HOST11
14
66
MAP BARS AND GET PHYSICAL ADDRESSES FROM OS
68
OS PROVIDES MEMORY MAPPED ADDRESS FOR THE PHYSICAL ADDRESS OF THE BAR
70
APPLICATION MAPS THE ADDRESS IN USER SPACE(USING MMAP)
60

FIG. 5

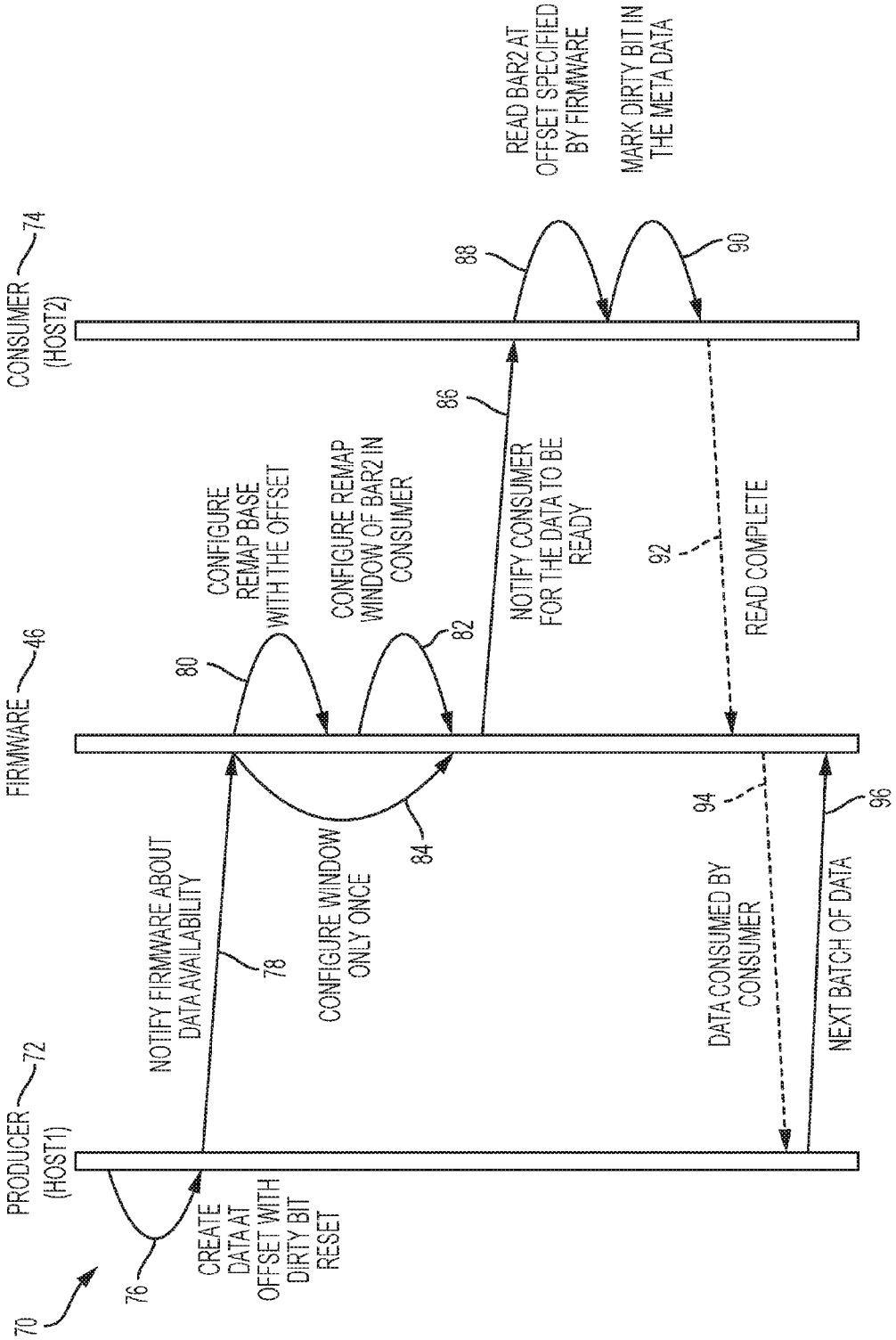

PRODUCER (HOST1)
72
FIRMWARE
46
CONSUMER (HOST2)
74
70
76
CREATE DATA AT OFFSET WITH DIRTY BIT RESET
NOTIFY FIRMWARE ABOUT DATA AVAILABILITY
78
CONFIGURE REMAP BASE WITH THE OFFSET
80
CONFIGURE WINDOW ONLY ONCE
84
CONFIGURE REMAP WINDOW OF BAR2 IN CONSUMER
82
NOTIFY CONSUMER FOR THE DATA TO BE READY
86
READ BAR2 AT OFFSET SPECIFIED BY FIRMWARE
88
MARK DIRTY BIT IN THE META DATA
90
READ COMPLETE
92
DATA CONSUMED BY CONSUMER
94
NEXT BATCH OF DATA
96

FIG. 6

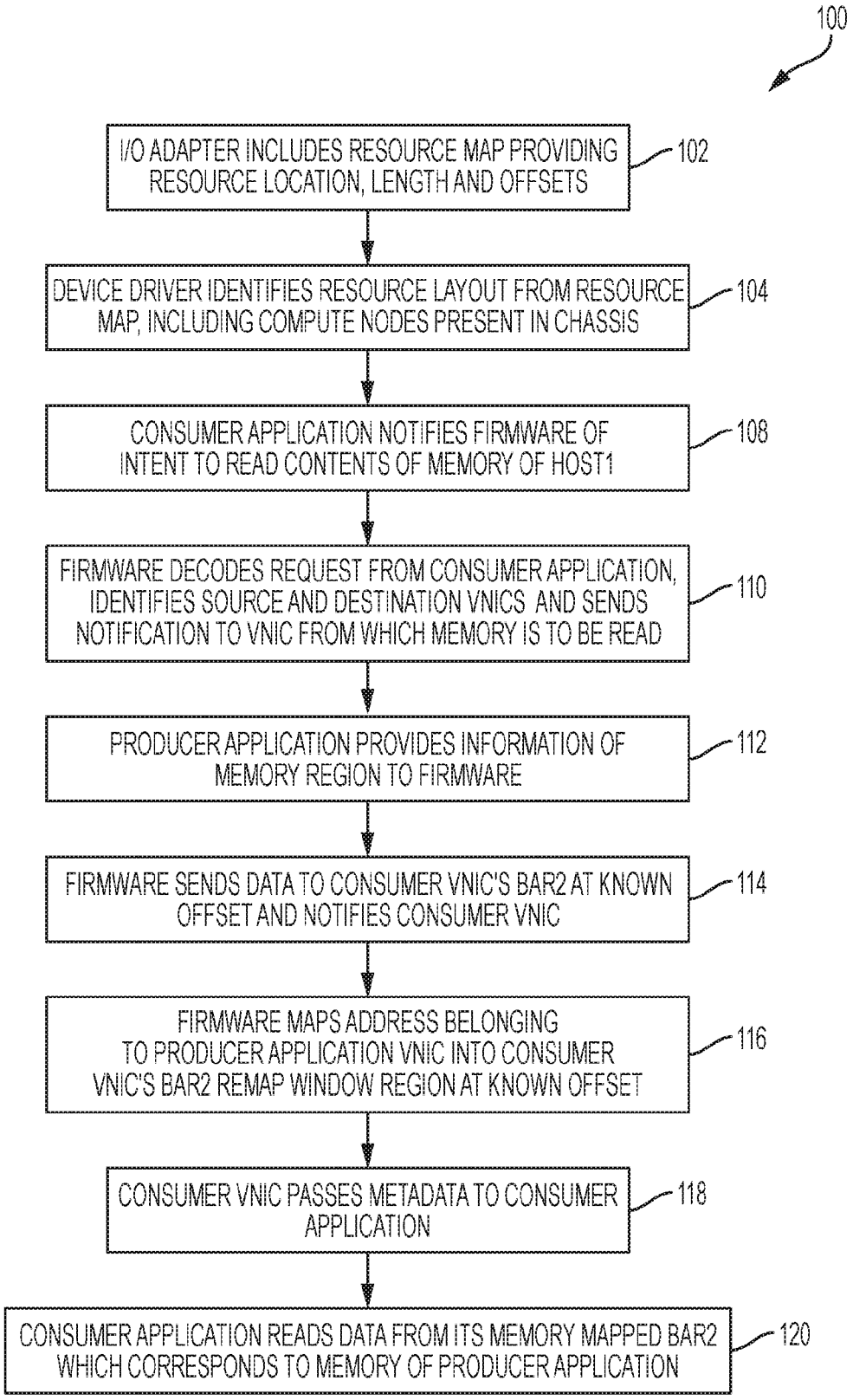

100
I/O ADAPTER INCLUDES RESOURCE MAP PROVIDING RESOURCE LOCATION, LENGTH AND OFFSETS
102
DEVICE DRIVER IDENTIFIES RESOURCE LAYOUT FROM RESOURCE MAP, INCLUDING COMPUTE NODES PRESENT IN CHASSIS
104
CONSUMER APPLICATION NOTIFIES FIRMWARE OF INTENT TO READ CONTENTS OF MEMORY OF HOST1
108
FIRMWARE DECODES REQUEST FROM CONSUMER APPLICATION, IDENTIFIES SOURCE AND DESTINATION VNICS AND SENDS NOTIFICATION TO VNIC FROM WHICH MEMORY IS TO BE READ
110
PRODUCER APPLICATION PROVIDES INFORMATION OF MEMORY REGION TO FIRMWARE
112
FIRMWARE SENDS DATA TO CONSUMER VNIC'S BAR2 AT KNOWN OFFSET AND NOTIFIES CONSUMER VNIC
114
FIRMWARE MAPS ADDRESS BELONGING TO PRODUCER APPLICATION VNIC INTO CONSUMER VNIC'S BAR2 REMAP WINDOW REGION AT KNOWN OFFSET
116
CONSUMER VNIC PASSES METADATA TO CONSUMER APPLICATION
118
CONSUMER APPLICATION READS DATA FROM ITS MEMORY MAPPED BAR2 WHICH CORRESPONDS TO MEMORY OF PRODUCER APPLICATION
120

FIG. 7

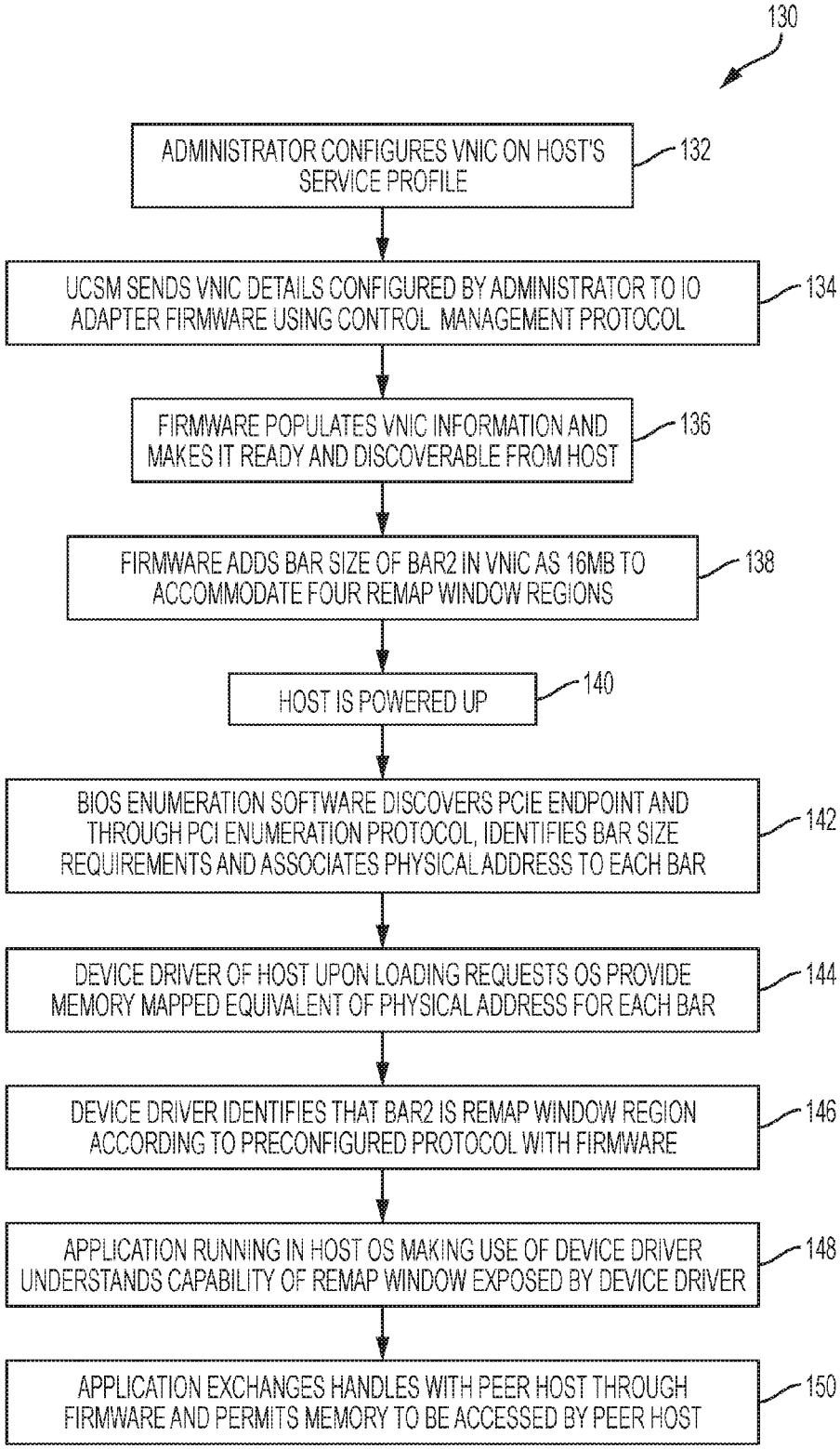

130
ADMINISTRATOR CONFIGURES VNIC ON HOST'S SERVICE PROFILE
132
UCSM SENDS VNIC DETAILS CONFIGURED BY ADMINISTRATOR TO IO ADAPTER FIRMWARE USING CONTROL MANAGEMENT PROTOCOL
134
FIRMWARE POPULATES VNIC INFORMATION AND MAKES IT READY AND DISCOVERABLE FROM HOST
136
FIRMWARE ADDS BAR SIZE OF BAR2 IN VNIC AS 16MB TO ACCOMMODATE FOUR REMAP WINDOW REGIONS
138
HOST IS POWERED UP
140
BIOS ENUMERATION SOFTWARE DISCOVERS PCIE ENDPOINT AND THROUGH PCI ENUMERATION PROTOCOL, IDENTIFIES BAR SIZE REQUIREMENTS AND ASSOCIATES PHYSICAL ADDRESS TO EACH BAR
142
DEVICE DRIVER OF HOST UPON LOADING REQUESTS OS PROVIDE MEMORY MAPPED EQUIVALENT OF PHYSICAL ADDRESS FOR EACH BAR
144
DEVICE DRIVER IDENTIFIES THAT BAR2 IS REMAP WINDOW REGION ACCORDING TO PRECONFIGURED PROTOCOL WITH FIRMWARE
146
APPLICATION RUNNING IN HOST OS MAKING USE OF DEVICE DRIVER UNDERSTANDS CAPABILITY OF REMAP WINDOW EXPOSED BY DEVICE DRIVER
148
APPLICATION EXCHANGES HANDLES WITH PEER HOST THROUGH FIRMWARE AND PERMITS MEMORY TO BE ACCESSED BY PEER HOST
150

FIG. 8

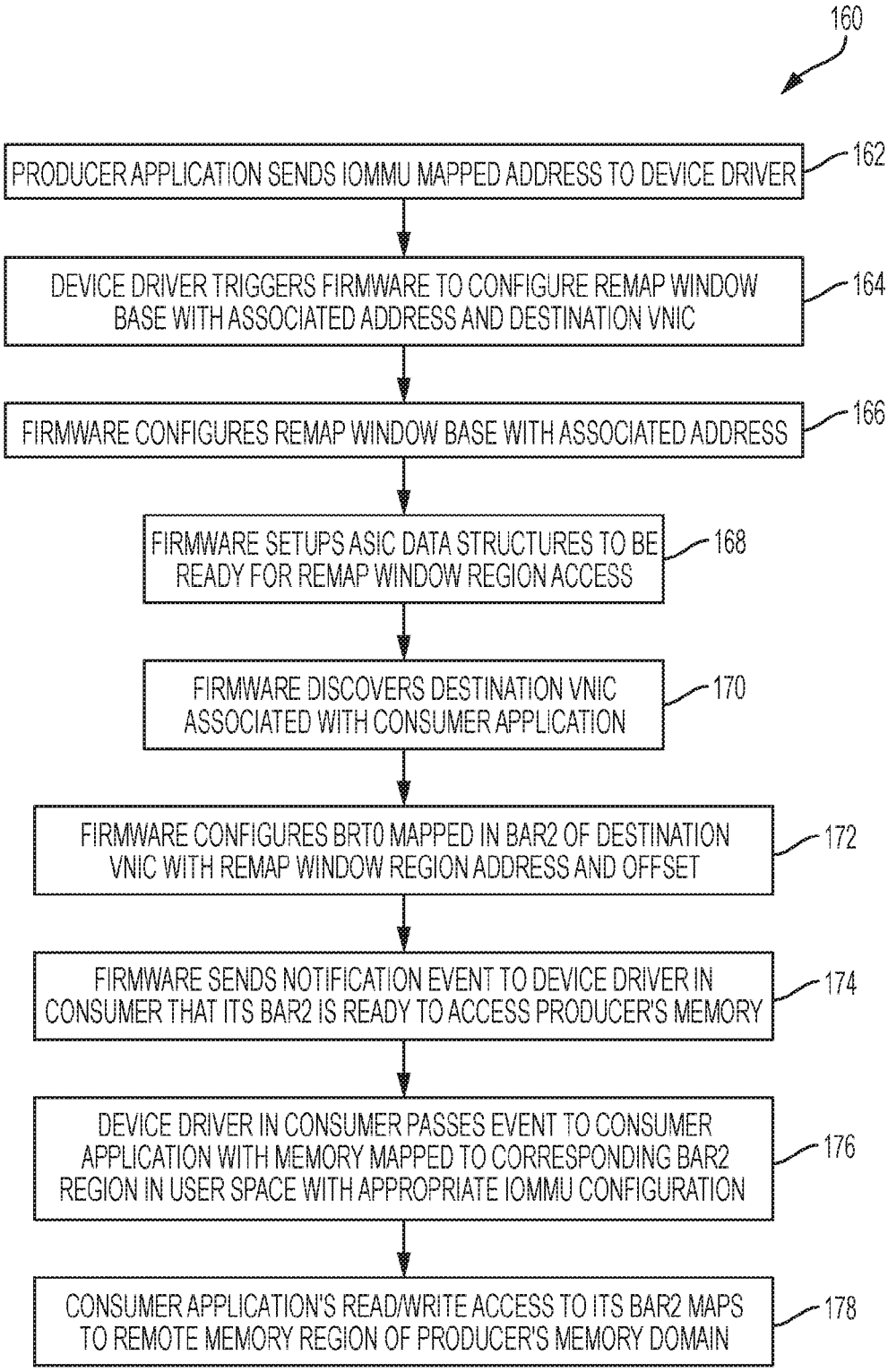

160
PRODUCER APPLICATION SENDS IOMMU MAPPED ADDRESS TO DEVICE DRIVER
162
DEVICE DRIVER TRIGGERS FIRMWARE TO CONFIGURE REMAP WINDOW BASE WITH ASSOCIATED ADDRESS AND DESTINATION VNIC
164
FIRMWARE CONFIGURES REMAP WINDOW BASE WITH ASSOCIATED ADDRESS
166
FIRMWARE SETUPS ASIC DATA STRUCTURES TO BE READY FOR REMAP WINDOW REGION ACCESS
168
FIRMWARE DISCOVERS DESTINATION VNIC ASSOCIATED WITH CONSUMER APPLICATION
170
FIRMWARE CONFIGURES BRT0 MAPPED IN BAR2 OF DESTINATION VNIC WITH REMAP WINDOW REGION ADDRESS AND OFFSET
172
FIRMWARE SENDS NOTIFICATION EVENT TO DEVICE DRIVER IN CONSUMER THAT ITS BAR2 IS READY TO ACCESS PRODUCER'S MEMORY
174
DEVICE DRIVER IN CONSUMER PASSES EVENT TO CONSUMER APPLICATION WITH MEMORY MAPPED TO CORRESPONDING BAR2 REGION IN USER SPACE WITH APPROPRIATE IOMMU CONFIGURATION
176
CONSUMER APPLICATION'S READ/WRITE ACCESS TO ITS BAR2 MAPS TO REMOTE MEMORY REGION OF PRODUCER'S MEMORY DOMAIN
178

FIG. 9

REMOTE MEMORY ACCESS USING MEMORY MAPPED ADDRESSING AMONG MULTIPLE COMPUTE NODES

TECHNICAL FIELD

[0001] This disclosure relates in general to the field of communications and, more particularly, to remote memory access with memory mapped addressing among multiple compute nodes.

BACKGROUND

[0002] Compute nodes such as microservers and hypervisor-based virtual machines executing in a single chassis can provide scaled out workloads in hyper-scale data centers. Microservers are an emerging trend of servers for processing lightweight workloads with large numbers (e.g., tens or even hundreds) of relatively lightweight server nodes bundled together in a shared chassis infrastructure, for example, sharing power, cooling fans, and input/output components, eliminating space and power consumption demands of duplicate infrastructure components. The microserver topology facilitates density, lower power per node, reduced costs, and increased operational efficiency. Microservers are generally based on small form-factor, system-on-a-chip (SoC) boards, which pack processing capability, memory, and system input/output onto a single integrated circuit. Unlike the relatively newer microservers, hypervisor-based virtual machines have been in use for several years. Yet, sharing data across the compute nodes with more effective and efficient inter-process communication has always been a challenge.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] To provide a more complete understanding of the present disclosure and features and advantages thereof, reference is made to the following description, taken in conjunction with the accompanying figures, wherein like reference numerals represent like parts, in which:

[0004] FIG. **1** is a simplified block diagram illustrating a communication system for facilitating remote memory access with memory mapped addressing among multiple compute nodes;

[0005] FIG. **2** is a simplified block diagram illustrating other example details of embodiments of the communication system;

[0006] FIG. **3** is a simplified block diagram illustrating yet other example details of embodiments of the communication system;

[0007] FIG. **4** is a simplified block diagram illustrating yet other example details of embodiments of the communication system;

[0008] FIG. **5** is a simplified sequence diagram illustrating example operations that may be associated with an embodiment of the communication system;

[0009] FIG. **6** is a simplified sequence diagram illustrating other example operations that may be associated with an embodiment of the communication system;

[0010] FIG. **7** is a simplified flow diagram illustrating yet other example operations that may be associated with an embodiment of the communication system;

[0011] FIG. **8** is a simplified flow diagram illustrating yet other example operations that may be associated with an embodiment of the communication system; and

[0012] FIG. **9** is a simplified flow diagram illustrating yet other example operations that may be associated with an embodiment of the communication system.

DETAILED DESCRIPTION OF EXAMPLE EMBODIMENTS

Overview

[0013] An example method for facilitating remote memory access with memory mapped addressing among multiple compute nodes is executed at an input/output (IO) adapter in communication with the compute nodes over a Peripheral Component Interconnect Express (PCIE) bus, the method including: receiving a memory request from a first compute node to permit access by a second compute node to a local memory region of the first compute node; generating a remap window region in a memory element of the IO adapter, the remap window region corresponding to a base address register (BAR) of the second compute node in the IO adapter; and configuring the remap window region to point to the local memory region of the first compute node, wherein access by the second compute node to the BAR corresponding with the remap window region results in direct access of the local memory region of the first compute node by the second compute node. As used herein, the term "compute node" refers to a hardware processing apparatus, in which user applications (e.g., software programs) are executed.

Example Embodiments

[0014] Turning to FIG. **1**, FIG. **1** is a simplified block diagram illustrating a communication system **10** for facilitating remote memory access with memory mapped addressing among multiple compute nodes in accordance with one example embodiment. FIG. **1** illustrates a communication system **10** comprising a chassis **12**, which includes a plurality of compute nodes **14** that communicate with network **16** through a common input/output (I/O) adapter **18**. An upstream switch **20** facilitates north-south traffic between compute nodes **14** and network **16**. Shared IO adapter **18** presents network and storage devices on a Peripheral Component Interconnect Express (PCIE) bus **22** to compute nodes **14**. In various embodiments, each compute node appears as a PCIE device to other compute nodes in chassis **12**.

[0015] In a general sense, compute nodes **14** include capabilities for processing, memory, network and storage resources. For example, as shown in greater detail in the figure, compute node Host**1** runs (e.g., executes) an operating system **24** and various applications **26**. A device driver (also referred to herein as a driver) **28** operates or controls a particular type of device that is attached to compute node **14**. For example, each PCIE device visible to (e.g., accessible by) Host**1** may be associated with a separate device driver in some embodiments. In another example, all PCIE endpoints visible to Host**1** may be associated with a single PCIE device driver. In a general sense device driver **28** provides a software interface to hardware devices, enabling operating system **24** and applications **26** to access hardware functions (e.g., memory access) without needing to know precise details of the hardware being used.

[0016] In many embodiments, substantially all PCIE endpoints appear as hardware device to the accessing compute

node, irrespective of its actual form. For example, in some embodiments, compute nodes **14** may comprise virtual machines; however, because one compute node is visible as a PCIE device to another compute node, they appear as hardware devices to each other and are associated with corresponding device drivers. Driver **28** communicates with the hardware device through PCIE bus **22**. When one of applications **26** invokes a routine in driver **28**, driver **28** issues commands to the hardware device it is associated with. Thus, driver **28** facilitates communication (e.g., acts as a translator) between its associated hardware device and applications **26**. Driver **28** is hardware dependent and operating-system-specific.

[0017] In various embodiments, each of compute nodes **14**, as shown using example Host**1**, includes various hardware components, such as one or more sockets **30** (e.g., socket refers to a hardware receptacle that enables a collection of central processing unit (CPU) cores with a direct pipe to memory); each socket holds one processor **32**; each processor comprises one or more CPU cores **34**; each CPU core **34** executes instructions (e.g., computations, such as Floating-point Operations Per Second (FLOPS)); a memory element **36** may facilitate operations of CPU cores **34**.

[0018] Common IO adapter **18** facilitates communication to and from each of compute nodes **14**. In various embodiments, IO adapter **18** services both network and storage access requests from compute nodes **14** in chassis **12**, facilitating a cost efficient architecture. In various embodiments, a memory element **38** may be associated with (e.g., accessed by) IP adapter **18**. Memory element **38** includes various base address registers (BARs) **40** and remap windows **42** for various operations as described herein. A remap window helper register **44** and firmware **46** are also included (among other components) in IO adapter **18**. As used herein the term "firmware" comprises machine-readable and executable instructions and associated data that are stored in (e.g., embedded in, forming an integral part of, etc.) hardware, such as a read-only memory, or flash memory, or an ASIC, or a field programmable gate array (FPGA) and executed by one or more processors (not shown) in IO adapter **18** to control the operations of IO adapter **18**. In a general sense, firmware **46** comprises a combination of software and hardware used exclusively to control operations of IO adapter **18**.

[0019] In a general sense, network traffic between compute nodes **14** and network **16** may be termed as "North-South Traffic"; network traffic among compute nodes **14** may be termed as "East-West Traffic". Note that compute nodes **14** are unaware of the physical location of other compute nodes, for example, whether they exist in same chassis **12**, or are located remotely, over network **16**. Thus, compute nodes **14** are agnostic to the direction of network traffic they originate or terminate, such as whether the traffic is North-South, or East-West, and thereby use the same addressing mechanism (e.g., L2 Ethernet MAC address/IP address) for addressing nodes located in same chassis **12** or located in a remote node in same L2/L3 domain.

[0020] According to various embodiments of communication system **10**, a memory access scheme using low latency and low overhead protocols implemented in IO adapter **18** allows any one (or more) compute nodes **14**, for example, Host**1**, to share and access remote memory of another compute node (e.g., across different servers; across a hypervisor; across different operating systems), for example, Host**2**. Host**2** may include an operating system different from that of Host**1** without departing from the scope of the embodiments. The protocols as described herein do not require any particularized (e.g., custom) support from the operating systems or networking stack of Host**1** or Host**2**. The scheme is completely transparent to the operating systems of Host**1** and Host**2**, allowing suitable throughput while communicating in different memory domains.

[0021] For purposes of illustrating the techniques of communication system **10**, it is important to understand the communications that may be traversing the system shown in FIG. **1**. The following foundational information may be viewed as a basis from which the present disclosure may be properly explained. Such information is offered earnestly for purposes of explanation only and, accordingly, should not be construed in any way to limit the broad scope of the present disclosure and its potential applications.

[0022] In any server ecosystem, typical challenges to achieving better inter-process communication or sharing the data across the servers include reliable tunnels for the data sharing, low latency for the communication, low overhead while working with remote server etc. There are several solutions available in the market that predominantly use network tunnels to communicate in two distinct physical servers. Typical examples would be Remote Direct Memory Access (RDMA), RDMA over Converged Ethernet (RoCE) and InfiniBand. Although proven and in use for several years, such network based communication can be limited by various parameters, such as latency, networking stack dependency (e.g., network stack awareness), OS interference (e.g., OS dependency, OS awareness, OS configuration), IO semantics and key exchanges for security, necessity for protocol awareness, complex channel semantics and tedious channel setup procedures. Moreover, not all operating systems support RDMA (and its variants).

[0023] For example, RDMA communication is based on a set of three queues: (i) a send queue and (ii) a receive queue, comprising a Queue Pair (QP) and (iii) a Completion Queue (CQ). Posts in the QP are used to initiate the sending or receiving of data. A sending application (e.g., driver) places instructions, called Work Queue Elements (WQE), on its work queues that generate buffers in the sender's adapter to send data. The WQE placed on the send queue contains a pointer to the message to be sent; a pointer in the WQE on the receive queue contains a pointer to a buffer where an incoming message can be placed. The sender's adapter consumes WQE from the send queue at the egress side and streams the data from the memory region to the remote receiver. When data arrives at the remote receiver, the receiver's adapter consumes the WQEs at the receive queue at the ingress side and places the received data in appropriate memory regions of the receiving application. Any memory sharing or access between a sending compute node and the receiving compute node thus requires tedious channel setup, RDMA protocols, etc.

[0024] Moreover, in a chassis where several compute nodes share a common IO adapter, such remote memory access sharing protocols can have unnecessary overhead. For example, every packet from any compute node, say Host**1**, has to hit a port of upstream switch **20** and then return on the same pipe back to IO adapter **18**, which then redirects it to the destination compute node, say Host**2**. Such east-west data sharing can cause inefficient utilization of bandwidth in the common pipe, which is potentially used by

various other compute nodes performing extensive north-south traffic with network **16**. The east-west traffic pattern also increases application response latency, for example, due to longer path to be traversed by network packets.

[0025] Communication system **10** is configured to address these issues (among others) to offer a system and method for facilitating remote memory access with memory mapped addressing among multiple compute nodes **14** sharing IO adapter **18**. In various embodiments, PCIE, which is typically supported by almost all operating systems, is used to share data from a memory region on one compute node, say Host**1**, with a different memory region of another compute node, say Host**2**. As used herein, the term "memory region" comprises a block (e.g., section, portion, slice, chunk, piece, space, etc.) of memory that can be accessed through a contiguous range of memory addresses (e.g., a memory address is a unique identifier (e.g., binary identifier) used by a processor for tracking a location of each memory byte stored in the memory). As used herein, the term "window" in the context of memory regions refers to a memory region comprising a contiguous range of memory addresses, either virtual or physical.

[0026] In various embodiments, IO adapter **18** is connected to compute nodes **14** by means of PCIE bus **22**. IO adapter **18** includes an embedded operating system hosting multiple VNICs configured with memory resources of memory element **38**. Each VNIC accesses a separate, exclusive region of memory element **38**. Each PCIE endpoint, namely VNICs is typically associated with a host software driver, namely device driver **28**. In an example embodiment, each VNIC that requires a separate driver is considered a separate PCIE device.

[0027] For ease of explanation of various embodiments, a brief overview of PCIE protocol is provided herein. A PCIe data transfer subsystem in a computing system (such as that of an IO adapter) includes a PCIe root complex comprising a computer hardware chipset that handles communications between the PCIE endpoints. The root complex enables PCIe endpoints to be discovered, enumerated and worked upon by the host operating system. The base PCIe switching structure of a single root complex has a tree topology, which addresses PCIe endpoints through a bus numbering scheme. Configuration software on the root complex detects every bus, device and function (e.g., storage adapter, networking adapter, graphics adapter, hard drive interface, device controller, Ethernet controller, etc.) within a given PCIe topology.

[0028] The IO adapter's operating system assigns address space in the IO adapter memory element **38** to each PCIe endpoint (e.g., VNIC) so that the PCIe endpoint can understand at what address space it is identified by the IO adapter and map the corresponding interrupts accordingly. After the configuration of the PCIe endpoint device is complete, the PCIe's device driver **28** compatible with the host operating system **24** can work efficiently with the PCIe endpoint and facilitate appropriate device specific functionality.

[0029] Each PCIE endpoint is enabled on IO adapter **18** by being mapped into a memory-mapped address space in memory element **18** referred to as configuration space (e.g., register, typically consisting of **256** bytes). The configuration space contains a number of base address registers (BARs) **40**, comprising the starting address of a contiguous mapped address in IO adapter memory element **38**. For example, a 32-bit BAR**0** is offset 10 h in PCI Compatible Configuration Space—and post enumeration would contain the start address of BAR. Any other PCIE endpoint, to access (e.g., read data from or write data to) the PCIE endpoint associated with a specific BAR, would submit a request with the address of that BAR. An enumeration software allocates memory for the PCIE endpoints and writes to corresponding BARs. Firmware **46** programs the PCIe endpoint's BARs to inform the PCIe endpoints of its address mapping. When the BAR for a particular PCIe endpoint is written, all memory transactions generated to that bus address range are claimed by the particular PCIe endpoint.

[0030] Typically, when a PCIE endpoint, say a flash memory device, is discovered on one of the compute nodes, say, Host**1**, OS **24** provides a physical address to BAR **40** and allocates the address space for device driver **28** to interact with the flash memory device. When device driver **28** is loaded, it requests the memory mapped address from OS **24** corresponding to the physical address so that it can work with the flash memory device using the address handle. Subsequent accesses to BAR **40** from device driver **28** are completely transparent to OS **24** as it has already carved out the address space sufficient to work with the flash memory device. Thus, typical PCIE data access is between application **26** and the PCIE endpoint, such as the flash memory device. PCIE data access is not typically used across two different compute nodes **14**. In other words, one compute node typically cannot share its memory space with another compute node using native PCIE protocols.

[0031] Nevertheless, according to various embodiments, appropriate configuration of IO adapter **18** with multiple ports and remap window feature can support memory sharing between compute nodes **14** using PCIE. Remap window feature includes a remap window base and remap window region for memory mapping for the purpose of remapping root complex IO and memory BARs to address ranges that are directly addressable by the processor. Remap window base is used to configure a start address of a memory region which can be mapped to any other memory region. The remap window region refers to the mapped region in memory element **38** differentiated according to a virtual network interface card (VNIC) identifier (ID) configured in remap window helper register **44** in IO adapter **18**.

[0032] The VNIC ID could map to any host-based VNIC or root complex VNIC. In some embodiments, four remap window regions, each capable of addressing 4 MB may be allocated for the remap window feature, permitting easy access of up to 16 MB of memory either in host memory or Root Complex endpoint device memory. Moreover, multiple PCIE ports on PCIE bus **22** distinguish different PCIE lanes associated with distinct compute nodes **14**. Each memory region in Root Complex endpoint device memory is associated with a distinct PCIE lane that is completely independent of each other such that no two memory regions share any PCIE activity with each other.

[0033] In an example embodiment, an administrator configures the VNIC ID of computing nodes **14** through respective service profiles. A unified computing system manager (e.g., network management application such as Cisco® UCSM) programs the VNIC ID in IO adapter **18** through appropriate control management protocol. Upon reception, firmware **46** populates the VNIC ID information in remap window helper register **44** and also makes the VNICs ready and discoverable from corresponding computing nodes **14**.

Firmware **46** adds the BAR size of a specific BAR, for example, BAR**3**, to the memory region allocated with each VNIC ID. In an example embodiment, 16 MB may be added to accommodate all four remap window regions.

[0034] After one of computing nodes **14**, say Host**1** is powered up, the enumeration software (BIOS) of IO adapter **18** discovers the new PCIE device. Through PCI enumeration protocol, the BIOS identifies BAR size requirements, and associates a physical address to corresponding Host**1** in BAR **40**. In various embodiments, three separate BARs are provided for each VNIC, namely, BAR**0**, BAR**1** and BAR**2**. Device driver **28** upon loading in Host**1** requests OS **24** to provide memory mapped equivalent of the physical address for each BAR. It identifies that BAR**2** is the remap window region according to a preconfigured protocol between firmware **46** and driver **28**. The memory mapped IO address comprises an address handle given by OS **24** to access the BAR**2** region of memory element **38** in IO adapter **18**. Applications **26** using device driver **28** understands the capability of remap window exposed by device driver **28**. Similar sequence of events occurs in another compute node, say Host**2**, when it powers up and its device driver is loaded in its OS. Through a pre-determined protocol, applications **26** in compute nodes **14**, say Host**1** and Host**2**, exchange their respective address handles through firmware **46** and request corresponding memory access.

[0035] The memory access mechanisms described herein can present one of the lowest latency protocols to communicate with different servers, virtual machines, or other such compute nodes **14**. In some embodiments, the memory access mechanisms described herein can also be used as IPC between two compute nodes **14**. Note that the operating system or network stacks do not need any separate, or distinct configuration to enable such remote memory access. In some embodiments, IO adapter **18** servicing a hypervisor can use the described mechanisms to allow various applications executing in separate virtual machines (e.g., guest domains) to communicate with each other without having to go through specially installed IPC software (e.g., VMWARE ESX/ESXi) or other external memory management/sharing applications.

[0036] In an example embodiment wherein compute nodes **14** comprise microservers the storage and network is shared across multiple servers (e.g., in some cases sixteen servers). The network ecosystem (e.g., of network **16**) may support different classes and QoS policies for network traffic, which can result in different priority flows. However, storage traffic does not typically have any associated QoS. Such differentiated traffic types (e.g., some traffic having QoS, other traffic not having QoS) can create imbalance of traffic performance across different servers causing some servers using (or allocated) larger bandwidths and other servers using (or allocated) poor bandwidth. With large amounts of input/output among (or from/to) servers, the condition can become worse with performance drops becoming noticeable in some servers. In other words, performance of some servers drops when other unrelated servers are experiencing heavy network traffic.

[0037] To have balanced throughput across the servers, a cooperative I/O scheduling across the servers may be implemented. For example, every server monitors and records a number of IO requests issued to IO adapter **18**. Such IO statistics are shared with other servers through the local memory mapped scheme in BAR**3** as described herein. Such data sharing can facilitate decisions at the individual servers regarding whether to send a SCSI_BUSY message to its OS storage stack. Thus, even though the associated storage VNIC has bandwidth to push the IOs to IO adapter **18**, it will not schedule the IO requests, voluntarily relinquishing claim on storage for some time, until the network traffic bottleneck clears up. Such actions can lead to other VNICs balancing out storage traffic pattern in chassis **12**, maintaining the IO equilibrium therein.

[0038] In various embodiments, IO adapter **18** receives a memory request from one of compute nodes **14**, say Host**1**, to permit access by another of compute nodes **14**, say Host**2**, to a local memory region of Host**1** (assume the local memory region is in memory element **36**). The memory request comprises a host identifier of Host**2** and address of the local memory region of Host**1** in some embodiments. The host identifier can be obtained from a resource map providing identifying information of compute nodes **14** in communication with IO adapter **18** over PCIE bus **22**.

[0039] Firmware **46** in IO adapter **18** generates remap window region **42** in memory element **38** of IO adapter **18**, remap window region **42** corresponding to BAR **40** (e.g., BAR**2**) of Host**2** in IO adapter **18**. Firmware **46** configures remap window region **42** to point to the local memory region of Host**1**, access by Host**2** to BAR**2** corresponding with remap window region **42** resulting in direct access of the local memory region of Host**1** by Host**2**. Note that compute nodes **14** are associated with unique PCIE endpoints on PCIE bus **22**; therefore, each has distinct BARs **40** associated therewith. Moreover, the direct access of the local memory region of Host**1** by Host**2** does not involve operating systems of Host**1** and/or Host**2**. BAR**2** associated with remap window region **42** can comprise one of a plurality of BARs associated with Host**2**.

[0040] In various embodiments, device driver **28** of Host**2** associates BAR**2** with remap window region, such that application **26** executing in Host**2** can access the local memory region of Host**1** through appropriate access requests to BAR**2** using device driver **28**. In various embodiments, configuring remap window region **42** comprises configuring a remap window base in a BAR Resource Table (BRT) to be a start address of the local memory region.

[0041] Turning to the infrastructure of communication system **10**, network topology of the network including chassis **12** can include any number of compute nodes, servers, hardware accelerators, virtual machines, switches (including distributed virtual switches), routers, and other nodes inter-connected to form a large and complex network. A node may be any electronic device, client, server, peer, service, application, or other object capable of sending, receiving, or forwarding information over communications channels in a network. Elements of FIG. **1** may be coupled to one another through one or more interfaces employing any suitable connection (wired or wireless), which provides a viable pathway for electronic communications. Additionally, any one or more of these elements may be combined or removed from the architecture based on particular configuration needs.

[0042] Communication system **10** may include a configuration capable of TCP/IP communications for the electronic transmission or reception of data packets in a network. Communication system **10** may also operate in conjunction with a User Datagram Protocol/Internet Protocol (UDP/IP) or any other suitable protocol, where appropriate and based

on particular needs. In addition, gateways, routers, switches, and any other suitable nodes (physical or virtual) may be used to facilitate electronic communication between various nodes in the network.

[0043] Note that the numerical and letter designations assigned to the elements of FIG. **1** do not connote any type of hierarchy; the designations are arbitrary and have been used for purposes of teaching only. Such designations should not be construed in any way to limit their capabilities, functionalities, or applications in the potential environments that may benefit from the features of communication system **10**. It should be understood that communication system **10** shown in FIG. **1** is simplified for ease of illustration.

[0044] The example network environment may be configured over a physical infrastructure that may include one or more networks and, further, may be configured in any form including, but not limited to, local area networks (LANs), wireless local area networks (WLANs), VLANs, metropolitan area networks (MANs), VPNs, Intranet, Extranet, any other appropriate architecture or system, or any combination thereof that facilitates communications in a network.

[0045] In some embodiments, a communication link may represent any electronic link supporting a LAN environment such as, for example, cable, PCIE, Ethernet, wireless technologies (e.g., IEEE 802.11x), ATM, fiber optics, etc. or any suitable combination thereof. In other embodiments, communication links may represent a remote connection through any appropriate medium (e.g., digital subscriber lines (DSL), telephone lines, T1 lines, T3 lines, wireless, satellite, fiber optics, cable, Ethernet, etc. or any combination thereof) and/or through any additional networks such as a wide area networks (e.g., the Internet).

[0046] In various embodiments, chassis **12** may comprise a rack-mounted enclosure, blade enclosure, or a rack computer that accepts plug-in compute nodes **14**. Note that chassis **12** can include, in a general sense, any suitable network element, which encompasses computers, network appliances, servers, routers, switches, gateways, bridges, load-balancers, firewalls, processors, modules, or any other suitable device, component, element, or object operable to exchange information in a network environment. Moreover, the network elements may include any suitably configured hardware provisioned with suitable software, components, modules, interfaces, or objects that facilitate the operations thereof. This may be inclusive of appropriate algorithms and communication protocols that allow for the effective exchange of data or information.

[0047] Compute nodes **14** may comprise printed circuit boards, for example, manufactured with empty sockets. Each printed circuit board may hold more than one processor (e.g., within the same processor family, differing core counts, with a wide range of frequencies and vastly differing memory cache structures may be included in a single processor/socket combination). In some embodiments, compute nodes **14** may include hypervisors and virtual machines. IO adapter **18** may include an electronic circuit, expansion card or plug-in module that accepts input and generates output in a particular format. IO adapter **18** facilitates conversion of data format and electronic timing between input/output streams and internal computer circuits of chassis **12**. In some embodiments, IO adapter **18** may comprise a hypervisor, and compute nodes **14** may comprise separate virtual machines.

[0048] Turning to FIG. **2**, FIG. **2** is a simplified block diagram illustrating example details according to an embodiment of communication system **10**. Assume, merely for example purposes and not as a limitation that computing nodes **14**, namely Host**1** and Host **2** respectively, are to share data across memory regions according to embodiments of communication system **10**. Each compute node **14**, namely Host**1** and Host**2** connects to IO adapter **18** through a respective virtual network interface card (VNIC) **48**(**1**) and **48**(**2**) at the compute node side and a respective PCIE port **50**(**1**) and **50**(**2**) at the IO adapter side. Firmware **46** exposes (e.g., creates, generates, provides, etc.) a separate VNIC **52**(**1**) and **52**(**2**) for corresponding PCIE ports **50**(**1**) and **50**(**2**). VNIC **52**(**1**) and **52**(**2**) at IO adapter **18** act as standalone Ethernet network controller adapters for network traffic and/or as storage controller adapters for storage traffic from and to respective compute nodes **14**(**1**) and **14**(**2**). For example, all traffic from VNIC **48**(**1**) on Host**1** is sent to corresponding PCIE port **50**(**1**), through VNIC **52**(**1**), to the external facing port, if needed. VNICs **48**(**1**), **48**(**2**), **52**(**1**) and **52**(**2**) are created based on user configurations, for example, as specified in a service profile and policy configured at the UCSM and deployed therefrom. Each VNIC **52**(**1**) and **52**(**2**) at IO adapter **18** is associated with BAR **40**(**1**) and **40**(**2**) respectively, each comprising three separate memory spaces denoted as: BAR**0**, BAR**1** and BAR**2**. BARs **40**(**1**) and **40**(**2**) predominantly expose hardware functionality, such as memory spaces that can be used by host software, such as applications **26**, to work with VNIC **52**(**1**) and **52**(**2**).

[0049] To explain further, consider Host**1**. Note that the descriptions herein for Host**1** apply equally for Host**2**. Operating system **24** in Host**1** enumerates BARs **40**(**1**) associated with Host**1** and maps IO address space in host memory **36** to each BAR such that any access to the corresponding mapped addresses in the mapped IO address space in Host**1** will point to (e.g., correspond with, associate with) the appropriate one of BARs **40**(**1**): BAR**0**, BAR**1** and BAR**2** in IO adapter **18**. Note that whereas mapped addresses in Host**1** may be virtual, they point to the physical memory region in IO adapter **18**. Device driver **28** accesses BARs **40**(**1**) using the memory mapped addresses returned by OS **24**.

[0050] In various embodiments, BAR**2** is reserved for remap window **42**, which is identified by the device driver in respective compute nodes **14**. For example, BAR**2** of BAR **40**(**1**) is reserved for remap window region **42**(**1**) and BAR**2** of BAR **40**(**2**) is reserved for remap window region **42**(**2**). In other words, device driver **28** in Host**1** understands BAR**2** of **40**(**1**) to be associated with remap window **42**(**1**). When device driver **28** (or application **26**) in Host**1** wants to allow another compute node, such as Host**2**, to access its local memory **56**(**1**), firmware **46** configures remap window **42**(**2**) of Host**2** to point to memory addressed space **56**(**1**) of Host**1**. Similarly, when device driver **28** (or application **26**) in Host**2** wants to allow Host**1** to access its local memory **56**(**2**), firmware **46** configures remap window **42**(**1**) to point to memory addressed space **56**(**2**) of Host**2**.

[0051] In other words, BAR**2** of BAR **40**(**1**) associated with Host**1** refers to memory space **56**(**2**) of Host**2**; likewise, BAR**2** of BAR **40**(**2**) associated with Host**2** refers to memory space **56**(**1**) of Host**1**. Anything written to BAR**2** of BAR **40**(**1**) by Host**1** will be as if written directly into memory space **56**(**2**) of Host**2**, without any intervening

protocols or communication. Thus applications in separate compute nodes can easily access the memory present in their peer's memory domain.

[0052] Turning to FIG. **3**, FIG. **3** is a simplified block diagram illustrating example details according to an embodiment of communication system **10**. Memory and I/O requests in IO adapter **18** are handled using remap window helper register **44** comprising three cascaded hardware tables: BAR Match Table (BMT) **44**(**1**), BMT associated random access memory (RAM) **44**(**2**), and BAR Resource Table (BRT) **44**(**3**). These tables attempt to resolve memory and I/O transactions to a IO adapter memory address in memory element **38** without involving any processor of IO adapter **18** or operating system of compute nodes **14**. BMT **44**(**1**) provides a mechanism to determine whether a memory request (e.g., transaction) received from Host**1** matches a valid PCIE device, such as Host**2**. BMT **44**(**1**) uses a search key comprising (among other parameters) a host ID and a BAR address, including length and offset. A hit in BMT **44**(**1**) outputs a Hit Index, which indexes into an associated RAM entry in table **44**(**2**). BRT **44**(**3**) provides a mechanism to flexibly map a single BAR to one or more possibly non-contiguous, adapter memory-mapped resources. In some embodiments, BRT **44**(**3**) comprises a logical table implemented in the hardware RAM of IO adapter **18**.

[0053] Firmware **46** of IO adapter **18** presents a virtualized view of PCIE endpoints' configuration space to compute nodes **14**. When Host**1** configures memory/IO bar window (s) in the VNIC's configuration space, Host**1**'s BAR address windows are translated by remap window helper register **44** to map them to the local root complex endpoint's BAR windows in IO adapter's local address space. For example, memory region **56**(**2**) of Host**1** is mapped to remap window region **42**(**2**) of Host**2** in memory element **38**. After enumeration and virtualization of the configuration space of the PCIE endpoints, the device drivers running on compute nodes **14** may post work requests using their assigned memory bar windows.

[0054] During operation, a memory request from Host**1** to allow access to a specific memory region **56**(**1**) by a remote PCIE endpoint, say Host**2** may proceed as follows. Host**1** sends a memory request to firmware **46**, including HostID=identifier of remote peer, say Host**2**; type=remote_memory_access; address=address of local memory **56**(**1**). The memory request is converted into a search key to BMT **44**(**1**), triggering a lookup (e.g., ternary content-addressable memory (TCAM)) of BMT **44**(**1**), which outputs a hit index to RAM **44**(**2**) that activates a read of appropriate entry in BRT **44**(**3**). In some embodiments, the memory request from Host**1** may reference a VNIC number, which may be converted into the corresponding host identifier by suitable modules. Firmware **46** programs the appropriate entry in BRT **44**(**3**) to point to the provided address **56**(**1**) of Host**1**. The specific memory region of the appropriate entry in BRT **44**(**3**) is already pre-mapped to BAR**2** of Host**2** as remap window **42**(**2**). In other words, the entry in BRT **44**(**3**) references remap window region **42**(**2**), which now directly points to memory space **56**(**1**) of Host**1** after configuration by firmware **46**. Any memory requests going through remap window region **42**(**1**) will be tagged with the VNIC of the destination compute node **14**. Any writes by Host**1** into local memory region **56**(**1**) can be directly accessed by Host**2** through its mapped remap window region **42**(**2**) without any intervention by operating systems or CPUs.

[0055] Turning to FIG. **4**, FIG. **4** is a simplified block diagram illustrating example details according to an embodiment of communication system **10**. Assume that application A in Host**1** and application B in Host**2** exchanges data according to mechanisms as described herein. Application B takes the following actions: Application B sends a memory mapped address (e.g.,

[0056] IOMMU mapped address) of memory space **56**(**2**) to driver **28** in Host**2** requesting access to the PCIE endpoint corresponding to Host**1**. Driver **28** triggers firmware **46** in IO adapter **18** to configure a remap window base **58** in BRT **44**(**3**) with the memory mapped address and associate it with the destination VNIC of Host**1** as identified through a predetermined protocol.

[0057] Firmware **46** configures remap window base **58** with the given address and sets up application specific integrated circuit (ASIC) data structures to be ready for remap window region access. Firmware **46** discovers the destination VNIC of Host**1** that wants to access the memory region as given by driver **28**. Firmware **46** configures BRT**0**, corresponding to BAR**2** of the destination VNIC Host**1**, with the remap window region address and offset that would correspond to the remap window base **58**. Configured BRT**0** corresponds to remap window **42**(**1**) and points to memory region **56**(**2**) of Host**2**.

[0058] After remap window region **42**(**1**) is configured on behalf of the destination VNIC, firmware **46** sends notification to driver **28** running in Host**1** that its BAR**2** is ready to access the Host**2** memory. Upon receiving the notification from firmware **46**, driver **28** running in Host**1** passes the notification to application A. Application A already has memory mapped the BAR**2** region with appropriate IOMMU configuration (e.g., addresses). Subsequently application A's read/write access to BAR**2** of Host**1** maps to remote memory region **56**(**2**) present in Host**2**'s memory domain. Likewise, application B's read/write access to memory region **56**(**2**) maps to BAR**2** of Host**1**. Thus both application A and application B running in different compute nodes **14** can communicate with each other without any OS intervention.

[0059] Turning to FIG. **5**, FIG. **5** is a simplified sequence diagram illustrating example operations **60** according to an embodiment of communication system **10** associated with a driver load scenario and discovery of various resources presented to driver **28** including remap window **42** mapped in BAR **40**. At **62**, driver **28** corresponding to VNIC**0** of one of compute nodes **14**, say Host**1**, is loaded. At **62**, driver **28** reads BAR **40** and identifies BAR**2** as the remap BAR. At **66**, driver **28** maps the BARs and gets physical addresses from OS **24**. At **68**, OS **24** provides memory mapped address for the physical address of the BAR. At **70**, application **26** maps the address in user space (e.g., using MMAP). At **72**, firmware **46** prepares remap window **42** for usage by driver **28**.

[0060] Turning to FIG. **6**, FIG. **6** is a simplified sequence diagram illustrating example operations **70** according to an embodiment of communication system **10** between applications **26** running on two different compute nodes **14** and firmware **46** to enable the remap window configuration for the purpose of accessing remote memory. Assume, merely for example purposes and not as a limitation that Host**1** includes application **26**, which produces data, and is referred to as producer **72**; Host**2** includes another application **26**, which consumes the data, and is referred to as consumer **74**.

[0061] IO adapter **18** includes a resource map providing resource information, for example, its memory offset and length, associated with the corresponding VNIC. In some embodiments, the resource map associates memory address offsets (also referred to herein as "memory offsets," or simply "offsets") with the BAR of one or more I/O resources (the I/O resource corresponding to a PCIE device, such as VNIC). For example, the resource map may include information identifying each PCIE device on PCIE bus **22** and its corresponding BARs. In many embodiments, the resource map may be comprised in remap window help register **44**. In various embodiments, BAR**0** of each PCIE endpoint may point to the resource map stored in IO adapter **18**. The PCIE endpoints may be identified using host indices, or other suitable identifiers. On parsing the resource map, device driver **28** in producer **72** identifies other compute nodes **14** present in chassis **12**. Consumer **74** notifies firmware **46** of its intent to read the contents of the memory of Host**1** through a resource update. Firmware **46** decodes the request, identifies the source and destination VNICs and sends notification to the VNIC whose associated memory is to be read.

[0062] At **76**, producer **72** creates data at memory offset with dirty bit reset. (Note that the dirty bit is well known in the art to be associated with a block of memory and indicates whether or not the corresponding block of memory has been modified; if the bit is set (or reset), the data has been modified since the last time it was read). At **78**, device driver **28** in Host**1** notifies firmware **46** about the data availability. The notification's meta-data includes the address to be read from at its local memory space **56**(**1**) including: address, length, destination host index of consumer and a key. At **80**, firmware **46** configures remap window base **58** with the memory offset. At **82**, firmware **46** configures remap window region **42**(**2**) of BAR**2** associated with Host**2** to point to memory space **56**(**1**) of Host**1**. At **84**, firmware **46** configures remap window **42**(**2**) only once (e.g., for all transactions between producer **72** and same consumer **74**).

[0063] At **86**, firmware **46** notifies consumer **74** that the data is ready. At **88**, consumer **74** reads BAR**2** at the memory offset specified by firmware **46**. Reading BAR**2** at the memory offset is identical to accessing memory region **56**(**1**) of producer **72**. At **90**, consumer **74** marks the dirty bit in the meta data, indicating that the data has been read. At **92**, consumer **74** may notify firmware **46** that consumer **74** has completed reading the data. At **94**, firmware **46** may notify producer **72** that data has been consumed by consumer **74**. At **96**, producer **72** writes the next set of data to the memory offset, and the operations resume from **76** and continue thereafter.

[0064] Turning to FIG. **7**, FIG. **7** is a simplified flow diagram illustrating example operations **100** according to an embodiment of communication system **10**. Assume that produce **72** in Host**1** is providing data to consumer **74** in Host**2**, both Host**1** and Host**2** being connected over PCIE bus **22** with IO adapter **18**. At **102**, IO adapter **18** includes a resource map providing resource information, such as resource location (e.g., PCIE host index), length and offsets where firmware data is present. At **104**, device driver **28** in Host**1** identifies the resource layout from the resource map, including other compute nodes **14** in chassis **12**. In an example embodiment, identifying the resource layout comprises parsing the resource map.

[0065] At **108**, consumer application **74** notifies firmware **46** of intent to read contents of memory **56**(**1**) of Host**1** through a resource update message (or other suitable mechanism). At **110**, firmware **46** decodes the request, identifies the source and destination VNICs and sends notification to Host**1** VNIC from which the memory is to be read. At **112**, producer application **72** provides to firmware **46** the address of memory region **56**(**1**) to be read from, through an appropriate memory request. At **114**, firmware **46** sends the remap window region information (e.g., remap window base **58**, remap window region **42**(**2**)) to consumer application **74** (e.g., through associated VNIC) BAR**2** at known offset and notifies the consumer VNIC.

[0066] At **116**, firmware **46** maps the address belonging to the producer application VNIC, namely, address of memory region **56**(**1**) into the consumer VNIC's BAR**2** remap window region **42**(**2**) at known offset. At **118**, consumer VNIC passes the remap window information to consumer application **74**. At **120**, consumer application **74** reads the data from its memory mapped BAR**2** which corresponds to memory **56**(**1**) of producer application **72**.

[0067] Turning to FIG. **8**, FIG. **8** is a simplified flow diagram illustrating example operations **130** according to an embodiment of communication system **10**. At **132**, an administrator configures VNIC on Host**1**'s service profile. At **134**, UCSM sends VNIC details configured by the administrator to IO adapter **18**'s firmware **46** using a suitable control management protocol. At **136**, firmware **46** populates VNIC information and makes it ready and discoverable from Host**1**. At **138**, firmware **46** adds BAR size of BAR**2** in VNIC as 16 MB to accommodate four remap window regions.

[0068] At **140**, Host**1** is powered up. At **142**, BIOS enumeration software discovers PCIE endpoint and through PCIE enumeration protocol, identifies BAR size requirements and associates physical address to each BAR. At **144**, device driver **28** of Host**1** upon loading, requests OS **24** to provide memory mapped equivalent of physical address for each BAR. At **146**, device driver **28** identifies that BAR**2** is remap window region according to preconfigured protocol with firmware **46**. At **148**, application **26** running in Host**1**'s operating system **24** making use of device driver **28** understands capability of remap window exposed by device driver **28**. At **150**, application **26** exchanges handles with peer host (e.g., Host**2**) through firmware **46** and permits memory to be accessed by the peer host.

[0069] Turning to FIG. **9**, FIG. **9** is a simplified flow diagram illustrating example operations **160** according to an embodiment of communication system **10**. At **162**, producer application **72** (e.g., application **26** running in Host**1**) sends a memory request comprising a destination host (e.g., Host**2**) and IOMMU mapped address of memory region **56**(**1**) to device driver **28**. At **164**, device driver **28** triggers firmware **46** to configure remap window base **58** with destination VNIC (e.g., corresponding to Host**2**) and associated address. At **166**, firmware **46** configures remap window base **58** with associated address. At **168**, firmware **46** sets up ASIC data structures to be ready for access to remap window region **42**(**2**). At **170**, firmware discovers destination VNIC associated with consumer application **74**. At **172**, firmware **46** configured BRT**0** mapped in BAR**2** of destination VNIC with remap window region address and offset in BRT **44**(**3**). In other words, firmware **46** maps appropriate entry in BRT **44**(**3**) corresponding to remap window region **42**(**2**) to point to memory region **56**(**1**). At **174**, firmware **46** sends notification event to device driver **28** in Host**2** that its

BAR2 is ready to access producer's memory 56(1). At 176, device driver 28 in Host2 passes event to consumer application 74 running therein with memory mapped to corresponding BAR2 region in user space with appropriate IOMMU configuration. At 178, consumer application 72's read/write access to its BAR2 maps to remote memory region 56(2) of producer application 74's memory domain.

[0070] Note that in this Specification, references to various features (e.g., elements, structures, modules, components, steps, operations, characteristics, etc.) included in "one embodiment", "example embodiment", "an embodiment", "another embodiment", "some embodiments", "various embodiments", "other embodiments", "alternative embodiment", and the like are intended to mean that any such features are included in one or more embodiments of the present disclosure, but may or may not necessarily be combined in the same embodiments. Furthermore, the words "optimize," "optimization," and related terms are terms of art that refer to improvements in speed and/or efficiency of a specified outcome and do not purport to indicate that a process for achieving the specified outcome has achieved, or is capable of achieving, an "optimal" or perfectly speedy/perfectly efficient state.

[0071] Embodiments described herein may be used as or to support firmware instructions executed upon some form of processing core (such as the processor of IO adapter 18) or otherwise implemented or realized upon or within a machine-readable medium. A machine-readable medium includes any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium can include such as a read only memory (ROM); a random access memory (RAM); a magnetic disk storage media; an optical storage media; and a flash memory device, etc. In addition, a machine-readable medium can include propagated signals such as electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.).

[0072] In example implementations, at least some portions of the activities outlined herein may be implemented in software in, for example, IO adapter 18. In some embodiments, one or more of these features may be implemented in hardware, provided external to these elements, or consolidated in any appropriate manner to achieve the intended functionality. The various network elements may include software (or reciprocating software) that can coordinate in order to achieve the operations as outlined herein. In still other embodiments, these elements may include any suitable algorithms, hardware, software, components, modules, interfaces, or objects that facilitate the operations thereof.

[0073] Furthermore, IO adapter 18 described and shown herein (and/or their associated structures) may also include suitable interfaces for receiving, transmitting, and/or otherwise communicating data or information in a network environment. Additionally, some of the processors and memory elements associated with the various nodes may be removed, or otherwise consolidated such that a single processor and a single memory element are responsible for certain activities. In a general sense, the arrangements depicted in the FIGURES may be more logical in their representations, whereas a physical architecture may include various permutations, combinations, and/or hybrids of these elements. It is imperative to note that countless possible design configurations can be used to achieve the operational objectives outlined here. Accordingly, the associated infrastructure has a myriad of substitute arrangements, design choices, device possibilities, hardware configurations, software implementations, equipment options, etc.

[0074] In some of example embodiments, one or more memory elements (e.g., memory element 38, memory element 36) can store data used for the operations described herein. This includes the memory element being able to store instructions (e.g., software, logic, code, etc.) in non-transitory media, such that the instructions are executed to carry out the activities described in this Specification. A processor can execute any type of instructions associated with the data to achieve the operations detailed herein in this Specification. In one example, processors could transform an element or an article (e.g., data) from one state or thing to another state or thing. In another example, the activities outlined herein may be implemented with fixed logic or programmable logic (e.g., software/computer instructions executed by a processor) and the elements identified herein could be some type of a programmable processor, programmable digital logic (e.g., a field programmable gate array (FPGA), an erasable programmable read only memory (EPROM), an electrically erasable programmable read only memory (EEPROM)), an ASIC that includes digital logic, software, code, electronic instructions, flash memory, optical disks, CD-ROMs, DVD ROMs, magnetic or optical cards, other types of machine-readable mediums suitable for storing electronic instructions, or any suitable combination thereof.

[0075] These devices may further keep information in any suitable type of non-transitory storage medium (e.g., random access memory (RAM), read only memory (ROM), field programmable gate array (FPGA), erasable programmable read only memory (EPROM), electrically erasable programmable ROM (EEPROM), etc.), software, hardware, or in any other suitable component, device, element, or object where appropriate and based on particular needs. The information being tracked, sent, received, or stored in communication system 10 could be provided in any database, register, table, cache, queue, control list, or storage structure, based on particular needs and implementations, all of which could be referenced in any suitable timeframe. Any of the memory items discussed herein should be construed as being encompassed within the broad term 'memory element.' Similarly, any of the potential processing elements, modules, and machines described in this Specification should be construed as being encompassed within the broad term 'processor.'

[0076] It is also important to note that the operations and steps described with reference to the preceding FIGURES illustrate only some of the possible scenarios that may be executed by, or within, the system. Some of these operations may be deleted or removed where appropriate, or these steps may be modified or changed considerably without departing from the scope of the discussed concepts. In addition, the timing of these operations may be altered considerably and still achieve the results taught in this disclosure. The preceding operational flows have been offered for purposes of example and discussion. Substantial flexibility is provided by the system in that any suitable arrangements, chronologies, configurations, and timing mechanisms may be provided without departing from the teachings of the discussed concepts.

[0077] Although the present disclosure has been described in detail with reference to particular arrangements and

configurations, these example configurations and arrangements may be changed significantly without departing from the scope of the present disclosure. For example, although the present disclosure has been described with reference to particular communication exchanges involving certain network access and protocols, communication system **10** may be applicable to other exchanges or routing protocols. Moreover, although communication system **10** has been illustrated with reference to particular elements and operations that facilitate the communication process, these elements, and operations may be replaced by any suitable architecture or process that achieves the intended functionality of communication system **10**.

[0078] Numerous other changes, substitutions, variations, alterations, and modifications may be ascertained to one skilled in the art and it is intended that the present disclosure encompass all such changes, substitutions, variations, alterations, and modifications as falling within the scope of the appended claims. In order to assist the United States Patent and Trademark Office (USPTO) and, additionally, any readers of any patent issued on this application in interpreting the claims appended hereto, Applicant wishes to note that the Applicant: (a) does not intend any of the appended claims to invoke paragraph six (6) of 35 U.S.C. section 112 as it exists on the date of the filing hereof unless the words "means for" or "step for" are specifically used in the particular claims; and (b) does not intend, by any statement in the specification, to limit this disclosure in any way that is not otherwise reflected in the appended claims.

What is claimed is:

1. A method executed at an input/output (IO) adapter in communication with a plurality of compute nodes over a Peripheral Component Interconnect Express (PCIE) bus, the method comprising:

receiving a memory request from a first compute node to permit access by a second compute node to a local memory region of the first compute node;

generating a remap window region in a memory element of the IO adapter, the remap window region corresponding to a base address register (BAR) of the second compute node in the IO adapter; and

configuring the remap window region to point to the local memory region of the first compute node, wherein access by the second compute node to the BAR corresponding with the remap window region results in direct access of the local memory region of the first compute node by the second compute node.

2. The method of claim **1**, wherein the compute nodes are associated with unique PCIE endpoints on the PCIE bus.

3. The method of claim **1**, wherein the direct access of the local memory region of the first compute node by the second compute node does not involve operating systems of the first compute node and the second compute node.

4. The method of claim **1**, wherein the BAR of the second compute node comprises one of a plurality of BARs associated with the second compute node.

5. The method of claim **4**. wherein a device driver of the second compute node associates the BAR with the remap window region, such that an application executing in the second compute node can access the local memory region of the first compute node through appropriate access requests to the BAR using the device driver.

6. The method of claim **1**, wherein configuring the remap window region comprises configuring a remap window base in a BAR Resource Table (BRT) to be a start address of the local memory region.

7. The method of claim **1**, wherein the memory request comprises a host identifier of the second compute node and address of the local memory region of the first compute node.

8. The method of claim **7**, wherein the host identifier is obtained from a resource map providing identifying information of the compute nodes in communication with the IO adapter over the PCIE bus.

9. The method of claim **1**, wherein the compute nodes comprise microservers executing in a single chassis.

10. The method of claim **1**, wherein the compute nodes comprise virtual machines executing in a single hypervisor.

11. Non-transitory tangible media that includes instructions for execution, which when executed by a processor of a IO adapter in communication with a plurality of compute nodes over a PCIE bus, is operable to perform operations comprising:

receiving a memory request from a first compute node to permit access by a second compute node to a local memory region of the first compute node;

generating a remap window region in a memory element of the IO adapter, the remap window region corresponding to a BAR of the second compute node in the IO adapter; and

configuring the remap window region to point to the local memory region of the first compute node, wherein access by the second compute node to the BAR corresponding with the remap window region results in direct access of the local memory region of the first compute node by the second compute node.

12. The media of claim **11**, wherein the BAR of the second compute node comprises one of a plurality of BARs associated with the second compute node.

13. The media of claim **11**, wherein a device driver of the second compute node associates the BAR with the remap window region, such that an application executing in the second compute node can access the local memory region of the first compute node through appropriate access requests to the BAR using the device driver.

14. The media of claim **11**, wherein configuring the remap window region comprises configuring a remap window base in a BRT to be a start address of the local memory region.

15. The media of claim **11**, wherein the compute nodes comprise microservers executing in a single chassis.

16. An apparatus, comprising:

an IO adapter;

a plurality of compute nodes in communication with the IO adapter over a PCIE bus;

a physical memory for storing data; and

a processor, wherein the processor executes instructions associated with the data, wherein the processor and the physical memory cooperate, such that the apparatus is configured for:

receiving a memory request from a first compute node to permit access by a second compute node to a local memory region of the first compute node;

generating a remap window region in a memory element of the IO adapter, the remap window region corresponding to a BAR of the second compute node in the IO adapter; and

configuring the remap window region to point to the local memory region of the first compute node, wherein access by the second compute node to the BAR corresponding with the remap window region results in direct access of the local memory region of the first compute node by the second compute node.

17. The apparatus of claim **16**, wherein the BAR of the second compute node comprises one of a plurality of BARs associated with the second compute node.

18. The apparatus of claim **16**, wherein a device driver of the second compute node associates the BAR with the remap window region, such that an application executing in the second compute node can access the local memory region of the first compute node through appropriate access requests to the BAR using the device driver.

19. The apparatus of claim **16**, wherein configuring the remap window region comprises configuring a remap window base in a BRT to be a start address of the local memory region.

20. The apparatus of claim **16**, wherein the compute nodes comprise microservers executing in a single chassis.

* * * * *