

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)
(19) World Intellectual Property Organization
(40) International Bureau
(43) International Publication Date
4 June 2015 (04.06.2015)

(51) International Patent Classification:
G06F 11/07 (2006.01)
(21) International Application Number:
PCT/US20 14/067815
(22) International Filing Date:
28 November 2014 (28.11.2014)
(25) Filing Language:
English
(26) Publication Language:
English
(30) Priority Data:
61/910,018 27 November 2013 (27.11.2013) US
(71) Applicant (for US only): FUTUREWEI TECHNOLOGIES INC. [US/US]; 5340 Legacy Drive, Suite 175, Plano, Texas 75024 (US).
(71) Applicant: HUAWEI TECHNOLOGIES CO., LTD. [CN/CN]; Huawei Administration Building, Bantian, Longgang District, Shenzhen, Guangdong 518129 (CN).
(72) Inventors: REN, Da Qi; 2330 Central Expressway, Santa Clara, California 95059 (US); WEI, Zhulin; 2330 Central Expressway, Santa Clara, California 95059 (US).

(74) Agent: MCCUTCHEON, Robert D.; Munck Wilson Mandala, LLP, 12770 Coit Road, 600 Banner Place Tower, Dallas, Texas 75251 (US).


(54) Title: FAILURE RECOVERY FOR TRANSPLANTING ALGORITHMS FROM CLUSTER TO CLOUD

(57) Abstract: A method (401) of providing failure recovery capabilities to a cloud environment (10) for scientific HPC applications. An HPC application with MPI implementation extends the class of MPI programs to embed the HPC application with various degrees of fault tolerance. An MPI fault tolerance mechanism realizes a recover-and-continue solution. If an error occurs, only failed processes re-spawn, and the remaining living processes remain in their original processors/nodes (12, 14, 16), and system recovery costs are thus minimized.

FIG. 4
Published:

— with international search report (Art. 21(3))
FAILURE RECOVERY FOR TRANSPLANTING ALGORITHMS FROM CLUSTER TO CLOUD

TECHNICAL FIELD

[0001] The present disclosure is generally directed to high performance computing (HPC) and cloud computing, and more specifically to fault tolerance to provide reliability of virtual clusters on clouds where high-performance and data-intensive computing paradigms are deployed.

BACKGROUND

[0002] High-performance computing (HPC) provides accurate and rapid solutions for scientific and engineering problems based on powerful computing engines and the highly parallelized management of computing resources. Cloud computing as a technology and paradigm for the new HPC era is set to become one of the mainstream choices for high-performance computing customers and service providers. The cloud offers end users a variety of services covering the entire computing stack of hardware, software, and applications. Charges can be levied on a pay-per-use basis, and technicians can scale their computing infrastructures up or down in line with application requirements and budgets. Cloud computing technologies provide easy access to distributed infrastructures and enable customized execution environments to be easily established. The computing cloud allows users to immediately access required resources without capacity planning and freely release resources that are no longer needed.

[0003] Each cloud can support HPC with virtualized Infrastructure as a Service (IaaS). IaaS is managed by a cloud provider that enables external customers to deploy and execute applications. FIGURE 1 shows the layer correspondences between cluster computing and cloud computing models. The main challenges facing HPC-based clouds are cloud interconnection speeds and the
noise of virtualized operating systems. Technical problems include system virtualization, task submission, cloud data input/output (I/O), security and reliability. HPC applications require considerable computing power, high performance interconnections, and rapid connectivity for storage or file systems, such as supercomputers that commonly use InfiniBand and proprietary interconnections. However, most clouds are built around heterogeneous distributed systems connected by low performance interconnection mechanisms, such as 0-Gigabit Ethernet, which do not offer optimal environments for HPC applications. Table 1 below shows the comparison of technical characters between cluster computing and cloud computing models. Differences in infrastructures between cluster computing and cloud computing have increased the need to develop and deploy fault tolerance solutions on cloud computers.

<table>
<thead>
<tr>
<th>Performance factors</th>
<th>Cloud Computing</th>
<th>Cluster Computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Storage cost</td>
<td>2. Communication latencies</td>
<td>2. Communication latencies</td>
</tr>
<tr>
<td>3. Data transfer (in or out for each service)</td>
<td>3. Data dependencies</td>
<td>3. Data dependencies</td>
</tr>
<tr>
<td>Performance Timing</td>
<td>1. Specifying a particular service for a particular task; 2. Archiving intermediate data on a particular storage device; 3. Choosing a set of locations for input and output data;</td>
<td>1. Defining the data size to be distributed 2. Scheduling the send and receive workload 3. Task synchronization</td>
</tr>
<tr>
<td>Goal</td>
<td>Minimizing the total cost of execution while meeting all the user-specified constraints.</td>
<td>Minimizing the total execution time; performing on users’ hardware platforms.</td>
</tr>
<tr>
<td>Reliability</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Task size</td>
<td>Single large</td>
<td>Small and medium</td>
</tr>
<tr>
<td>Scalable</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Switching</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Application</td>
<td>HPC, HPC</td>
<td>SME interactive</td>
</tr>
</tbody>
</table>
This disclosure is directed to a failure recovery solution for transplanting high-performance data-intensive algorithms from the cluster to the cloud.

According to one example embodiment, a method provides failure recovery capabilities to a cloud environment for scientific HPC applications. An HPC application with MPI implementation extends the class of MPI programs to embed the HPC application with various degrees of fault tolerance (FT). An MPI FT mechanism realizes a recover-and-continue solution; if an error occurs, only failed processes re-spawn, the remaining living processes remain in their original processors/nodes, and system recovery costs are thus minimized.
BRIEF DESCRIPTION OF THE DRAWINGS

[0006] For a more complete understanding of the present disclosure, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, wherein like numbers designate like objects, and in which:

[0007] FIGURE 1 illustrates layer correspondences between cluster computing and cloud computing;

[0008] FIGURE 2 illustrates three IaaS layers forming network resources from bottom to top: cloud resources and TCP networks; hosts and virtual machines; guest MPI applications;

[0009] FIGURE 3 illustrates pseudo-code of an algorithm setting up a TCP connection using MPI initialization;

[0010] FIGURE 4 shows the component launch and monitoring steps and automated recovery flow process:

[0011] FIGURE 5 illustrates pseudo-code of an algorithm for MPI error handler setup, and the main function;

[0012] FIGURE 6 illustrates pseudo-code for MPI spawning new communications; and

[0013] FIGURE 7A-7D illustrates a MPI/TCP failure recovery process establishing new TCP long connections.
DETAILED DESCRIPTION

[0014] Failure Model for HPC on Cloud

An HPC cloud platform provides a comprehensive set of integrated cloud management capabilities to meet users' HPC requirements. Deployed on top of a HPC cloud, the software manages the running of computing and data-intensive distributed applications on a scalable shared grid, and accelerates parallel applications to accelerate results and improve the utilization of available resources. An HPC cloud enables the self-service creation and management of multiple flexible clusters to deliver the performance required by computing-intensive workloads in multi-tenant HPC environments. This disclosure provides a failure recovery solution using a typical Message Passing Interface-Transmission Control Protocol (MPI-TCP) model.

[0016] HPC Fault Tolerant Model on Cloud

MPI provides a message-passing application programmer interface and supports both point-to-point and collective communication. MPI has remained the dominant model used for HPC for several decades due to its high performance, scalability, and portability.

[0017] Many of the current big data applications use Remote Procedure Call (RPC) to establish TCP connections for high-performance data intensive computing. Typical examples, such as MapReduce and Pregel, require long TCP connections to build up virtual cluster networks over the Internet or cloud. Hadoop RPC forms the primary communication mechanism between the nodes in the Hadoop cluster. The Hadoop Distributed File System (HDFS) enables multiple machines to implement functions. Hadoop NameNode receives requests from HDFS clients in the form of Hadoop RPC requests over a TCP connection. The listener object listens to the TCP port that serves RPC requests from the client.

[0019] In comparison, GraphLab simultaneously uses MPJ (MPI for Java) and TCP, and simplifies the update process because users
do not need to explicitly define the information flow from Map to Reduce and can just modify data in-place. For iterative computations, GraphLab's knowledge of the dependency structure directly communicates modified data to the destination. GraphLab presents a simpler API and data graph to programmers, and informs GraphLab of the program's communication needs. This implementation model uses collective synchronous MPIJ operations for communication.

Based on the applications listed above, this disclosure provides a modeled three-layer IaaS networked computing platform as shown at 10 in FIGURE 2, which, from bottom to top, are: cloud resources and TCP networks 12; hosts and virtual machines (VMs) 14; and guest MPI applications 16 (all collectively referred to herein as network resources). The cloud provider is responsible for administering services and cloud resources, such as hardware and VMs, that customers use to deploy and execute applications. FIGURE 2 summarizes the vertical cloud architecture and the scope of each cloud participant of the network resources. This disclosure identifies three types of failure of the network resources in the cloud platform: hardware/network failure, virtual machine failure, and application failure. Each of the above layers has exclusive fault tolerant functions; however, for optimal performance, collaborative failure management approaches including best effort must be considered.

Failure Detection

At the application level 16, MPI fault tolerance or virtual machine sensors can detect an application or virtual machine failure. Both the application layers 16 and virtual machine layers 14 collaborate to precisely locate errors. Errors can have three origins: MPI application, the virtual machine, and TCP network/hardware.

At the network/hardware level 12, TCP connections can be long-term as certain users maintain connections for hours or
even days at a time. The duration of TCP connections provides an excellent parallel platform for a group of virtual machines to run like a cluster on a cloud. If a problem occurs, heartbeating can check whether the network connection is alive because a connected network periodically sends small packets of data (heartbeats). If the peer does not receive a heartbeat for a specified time period, the connection is considered broken. However, the TCP protocol does not provide heartbeats and TCP endpoints are not notified of broken connections, causing them to live indefinitely, forever trying to communicate with the inaccessible peer. Higher level applications must then reinitialize certain applications. For many scientific computing applications, especially those with high-availability requirements, this missing failure notification is a critical issue that urgently requires a recovery mechanism.

[0024] Failure Recovery

[0025] If an error originates from the program at the application layer 16, the program itself should be able to self-recover e.g. the map-reduce implementation replicates data on the HDFS. If a node fails, tasks that are using the data on the failed node can restart on another node that hosts a replica of the data.

[0026] If an error occurs on a virtual machine due to a hardware host failure in layer 14, the cloud administration starts a new virtual machine with the same features, allowing users to redeploy tasks and restart and synchronize the new machine. In line with an application's properties, the checkpointing and recovery process is required after a new virtual machine is generated. The checkpointing process periodically takes system snapshots and stores application status information in persistent storage units. If a failure occurs, the most recent status can be retrieved and the system recovered. User directed checkpointing requires the application programmer to form the checkpoint and write out any
data needed to restart the application. Checkpoints must be saved to persistent storage units, which are typically cloud-based, that will not be affected by the failure of a computing element. However, there are two disadvantages in this scenario: first, the user is responsible for ensuring that all data is saved; second, the checkpoints must be taken at particular points in the program.

[0027] **MPI/TCP INFRASTRUCTURE WHEN SEVERING HIGH PERFORMANCE DATA-INTENSIVE COMPUTING ON CLOUD**

[0028] This disclosure provides a method to add failure recovery capabilities to a cloud environment for scientific HPC applications. An HPC application with MPI implementation is able to extend the class of MPI programs to embed the HPC application with various degrees of fault tolerance (FT). An MPI FT mechanism realizes a recover-and-continue solution; if an error occurs, only failed processes re-spawn, the remaining living processes remain in their original processors/nodes, and system recovery costs are thus minimized.

[0029] **MPI and TCP**

[0030] Users can initialize a low-level MPI/TCP communication model by enabling the communication group to use the MPI COMM to collect distributed system data, and then deliver it to the RPC to create a long-term TCP connection. Executing a distributed application over TCP connections and on a virtual cluster involves a similar process that requires three steps: 1) Initialize communicator groups using MPI; 2) Pass the data to RPC; 3) All devices with TCP connections complete connection setup and enter the established state. TCP software can then operate normally. FIGURE 3 shows pseudo-code for the steps of setting up a TCP connection using MPI initialization. The pseudo-code describes how MPI and TCP jointly build a Hadoop cluster.

[0031] **Fault Tolerant MPI Semantics and Interfaces**

[0032] The MPI Forum's Fault Tolerance Working Group has defined a set of semantics and interfaces to enable fault tolerant
applications and libraries to be portably constructed on top of the MPI interface, which enables applications to continue running and using MPI if one or multiple processes in the MPI universe fail. This disclosure assumes that MPI implementation provides the application with a view of the failure detector that is both accurate and complete. The application is notified of a process failure when it attempts to communicate either directly or indirectly with the failed process using the function's return code and error handler set on the associated communicator. The application must explicitly change the error handler to MPI_ERRORS_RETURN on all communicators involved in fault handling on the application.

MPI Recovery Procedure

To minimize the impact of the failure recovery process on an MPI/TCP task running on a cloud infrastructure, this disclosure provides a component that automates the launch and monitoring processes, periodically checks MPI health, and stops and re-launches the MPI/TCP process if a failure is detected. The component implements the following launch and monitoring steps and automated recovery flow process 400 as shown in FIGURE 4.

Step 401. The MPI_INIT pings and establishes connections with each virtual machine, builds a communication group comprising all communicators, and ensures that the communicators are up and available.

Step 402. The MPI process sends the size n node numbers, node names, folder path in which the MPI process will run, and file names with application instructions.

Step 403. RPC initializes independent, long-term TCP connections.

Step 404. Parallel execution enables each node to deploy multiple threads. A node is deemed to have failed if a virtual machine is in down status. MPI implementation must be able
to return an error code if a communication failure such as an aborted process or failed network link occurs.

[0039] Step 405. The management process uses MPI_Comm_Spawn to create workers and return an intercommunicator. This simplifies intercommunication formation in the scenario of parallel workers because one MPI_Comm_Spawn command can create multiple workers in a single intercommunicator's remote group. MPI_Comm_Spawn replaces dead workers, and continues processing with no fewer workers than before.


[0041] Step 407. The MPI process sends the size n node numbers node names, folder path in which the MPI process will run, and file names with application instructions. RPC initializes independent, long-term TCP connections. Checkpoints are copied from cloud storage.

[0042] Step 408. Parallel execution enables each virtual machine (VM) to deploy multiple threads. Note that the component is independent of any particular MPI application.

[0043] Fault Tolerance Implementation

[0044] Focusing on communication-level fault tolerance issues, FIGURE 5 and FIGURE 6 illustrate an example of a common scenario based on a well-known master/worker communication program. The scenario covers program control management failure detection and termination detection. FIGURE 5 illustrates the general procedure for setting up a fault tolerance MPI/TCP working environment using inter-communicators and MPI_ERROR_Handlers. FIGURE 6 shows how MPJ responds by spawning new nodes and removing dead nodes when a failure occurs.
FIGURES 7A-7D shows a diagram of the MPI/TCP failure recovery process.

FIGURE 7A illustrates a set of virtual machines running in parallel.

FIGURE 7B illustrates Node 2 fails.

FIGURE 7C illustrates MPI locating a new node 3.

FIGURE 7D illustrates establishing a new TCP long connection.

TCP Recovery

After the MPI connection is recovered, an RPC procedure is initialized. A client node calls its client stub using parameters pushed to the stack. The client stub packs these parameters into a message, and makes a system call to send the message from the client machine to the server machine. The operating system on the server machine passes the incoming packets to the server stub, and the server stub unpacks the parameters from the message. The server stub calls the server procedure, which forms the basis of establishing the TCP connection.

Higher Level Applications

An HPC-based cloud example is the design of a distributed master/worker finite element method (FEM) computing process. The FEM process involves mesh generation, refinement, and matrix assembly. The uncertainty of mesh refinement complicates the following operations: distributing basic functional modules that incorporate message synchronization, distributing matrices 'data, and collecting data; however, a MapReduce HDFS can maintain the consistency of FEM meshes in a distributed environment. Assuming that the computing capability of each node in a cloud is identical, the process for solving this problem is to map all tasks to a set of cloud mesh data. An independent task assigned to a worker process has a well-defined life cycle: First, the master sends a task to a worker, which the worker takes and works on; second, the worker returns the results after completing the task.
The fault tolerant scheme collaborates with checkpoint/restart techniques to handle failures during distributed processing. At least three lists task lists must be created: (1) Waiting (2) In progress (3) Done. The manager-part program should mark an intercommunicator as dead when a send or receive operation fails, maintain the task in-progress list, and send the operation to the next free worker. Global performance tuning optimization can be constructed from the timing and fault tolerant modes to identify the global minimum execution time for correct computing results.

While this disclosure has described certain embodiments and generally associated methods, alterations and permutations of these embodiments and methods will be apparent to those skilled in the art. Accordingly, the above description of example embodiments does not define or constrain this disclosure.

Other changes, substitutions, and alterations are also possible without departing from the spirit and scope of this disclosure, as defined by the following claims.
WHAT IS CLAIMED IS:

1. A method of performing fault tolerance at an infrastructure as a service (IaaS) layer on a cloud computing platform having network resources, comprising:
   - a component collecting system distributed data of the cloud computing platform using a message passing interface (MPI);
   - the component establishing long-term transmission control protocol (TCP) interconnections of the cloud computing platform using a remote procedure call (RPC);
   - the component automatically detecting a failure of one of the network resources; and
   - the component recovering the failure by adding a new network resource in place of the failed network resource using combined MPI and RPC functionalities.

2. The method as specified in Claim 1, wherein the failure is detected in a cluster of the network resources comprising a plurality MPI nodes in an MPI communication group, comprising the steps of:
   - the component calling the MPI nodes;
   - the component delivering information indicative of a failed MPI node to a MPI master node in order to spawn a new MPI node; and
   - the MPI master node broadcasting the information of the failed MPI node to all MPI nodes in the MPI communication group such that each MPI node is updated with the information.

3. The method as specified in Claim 2, further comprising the steps of:
   - the component determining a new MPI node as a new MPI communicator according to information at the MPI master node;
the component establishing a new connection with RPC to the new MPI node;
the component spawning a new communicator on the new MPI node; and
the component updating the new communicator with group member information and parallel processing information.

4. A method as specified in Claim 3, further comprising:
the component establishing checkpoints during parallel processing periodically; and
the component saving data of each checkpoint on a cloud storage.

5. The method as specified in Claim 4, further comprising:
the component updating the spawned new MPI node with current checkpoint data from the cloud storage; and
the component updating all the MPI members with the current checkpoint data from the cloud storage.

6. The method as specified in Claim 4, wherein:
the cloud storage has a definition in MPI; and
the cloud storage is one of the MPI members such that all the MPI nodes recognize the cloud storage and can copy data to/from the cloud storage.

7. The method as specified in Claim 2, further comprising the steps of:
defining a threshold time $T$ allowing the component to determine whether or not an MPI node has failed;
wherein when the master MPI node determines no response from an MPI node, the master MPI node waits a time length of time $T$, 

wherein if the MPI node with no response is recovered and responds to the master MPI node correctly within the time $\tau$, no new MPI node is spawned,

wherein if the MPI node with no response is not recovered within time $\tau$, the component spawns a new MPI node to replace the failed MPI node.

8. The method as specified in Claim 7 wherein a time $T_{\text{opt}}$ represents a time to establish the new MPI node, spawn the new MPI node, update the new MPI node information, and update the new MPI node with checkpoint data, wherein:

if $T > T_{\text{opt}}$, the master MPI node will not wait for the time $\tau$, and instead, the component spawns the new MPI node to replace the failed MPI node.

9. The method as specified in Claim 8, wherein if $T \leq T_{\text{opt}}$, the master MPI node waits until time $T$ to decide if the non-responsive MPI node has failed.
10. A method of performing failure recovery in a parallel cloud high performance computing (HPC) system having nodes, including the steps of:

- a component pinging and establishing connections with a plurality of virtual machines (VMs) having communicators, building a communication group that includes the communicators, and determining if the VMs are up and available;
- an message passing interface (MPI) process sending node numbers, node names, a folder path on which a MPI process can run, and file names with application instructions;
- a remote procedure call (RPC) initializing independent, long-term transmission control protocol (TCP) connections;
- the MPI process returning an error code to the component if a communication failure occurs in one of the communicators;
- the component spawning a new communicator if there is a failure in one of the communicators to replace the failed communicator;
- the RPC re-initializing independent, long-term TCP connections; and
- the MPI process loading checkpoints from storage.
Algorithm: MPI setup RPC call

/*MPI initialization communicator group */
1. MPI_INIT; MPI_COMM_WORLD=nprocs;
   /*MPI process to Create communicators */
2. MPI_COMM_SIZE(MPI_COMM_WORLD, &mpi_size);
3. MPI_Gather (IP Address, machines);
   /*Gather node informations */
   /*Create parameter table using information from MPI */
   /* Performs RPC call to the target machine in Hadoop Cluster */
4. FOR (size_t; i=1; i<nprocs; i++) {
5.    LISTEN ();
6. }
7. FOR (size_t; i=0; i<nprocs; i++) {
8.    Connect (i);
9. }
   /* Establish TCP connection with RPC; */
10. In_thread_num = machine_num/proc_per_thread;
11. Out_thread_num = machine_num/proc_per_thread;
    /* Create TCP long connections over the virtual machines identified by MPI communicators */
    /* Run distributed computing over the TCP connection */

FIG. 3
400

401. MPI initializes communicator group

402. RPC uses collected node information to build TCP communication group

403. Performing distributed processing over TCP connections

404. Node fails?

405. Replacing failure nodes with new nodes, MPI initialize new communicator group

406. Delivery communicator group information to RPC

407. Recovering data of last checkpoint from cloud storage

408. Performing distributed processing over TCP connections

FIG. 4
Algorithm: MPI error handler setup, the main function

1. int i, myrank, ini_size, curr_size;
2. MPI_Comm worker_comm [MAX_WORKERS];
3. MPI_Comm_rank (MPI_COMM_WORLD,
   &myrank);
4. MPI_Comm_size (MPI_COMM_WORLD, &ini_size);
5. curr_size = ini_size;
6. /* create intercommunicators, set error handlers*/
7. for (i = 1; i<curr_size; ++i)
8. {
9.   MPI_Intercomm_Create (MPI_COMM_SELF, 0,
10.    MPI_COMM_WORLD, i,
11.    IC_CREATE_TAG, &worker_comm[i-1]
12.    );
13.  MPI_Comm_Set_Errhandler (worker_comm[i-1],
14.    MPI_ERRORS_RETURN );
15. }
16. if (TCP application) {
17.   Don't do anything if the application is TCP based; */
18. }
19. if (MPI Application){
20.   manage MPI higher level fault tolerant mechanisms
21. }
22. for (i = 1; i<curr_size; ++i) {
23.   MPI_Comm_free (& worker_comm[i-1]);
24.   MPI_Finalize ( );
25. }
26. }

FIG. 5
Algorithm: MPI spawn new communicators

```
1. MPI_Comm Cgr = MPI_COMM_WORLD;
2. int get_current_state() { 
3.   MPI_RANK_info irs;
4.   int deadnode=0;
5.   for (n = 0; n<curr_size; ++n) { 
6.     /* Try to validate the status of each MPI rank, 
7.        find out who is dead, and how many new nodes 
8.        are needed; */
9.     MPI_Comm_validate_rank (Cgr, n, rs);
10.    if (MPI_RANK_OK!=rs.state){
11.       return n;
12.      deadnode++;
13.    }
14.  }
15.  int NUM_SPAWNS=deadnode;
16.  int errcodes[NUM_SPAWNS];
17.  MPI_Comm parentcomm, intercomm;
18.  MPI_Comm_get_parent (&parentcomm);
19.  if (parentcomm==Cgr) {
20.    MPI_Comm_spawn ("Exe", 
21.      MPI_ARGV_NULL,
22.      np, MPI_INFO_NULL, 0,
23.      MPI_COMM_WORLD,
24.      &intercomm, errcodes);
25.  }
```

FIG. 6
FIG. 7A

FIG. 7B

FIG. 7C

FIG. 7D
### INTERNATIONAL SEARCH REPORT

#### A. CLASSIFICATION OF SUBJECT MATTER

**IPC(8)**: G06F 1107 (2015.01)

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)

- **IPC(8)**: G06F 9/50, 11/07, 15/16, 15/173 (2015.01)

- **CPC**: G06F 11/0709, 11/0712, 11/0757, 15/16, 15/173

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

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

- PatSear (US, EP, WO, JP, DE, GB, CN, FR, KR, ES, AU, IN, CA, INPADOC Data); Espacenet

Keywords: fault tolerant, high availability, cloud, TCP, RPC, MPI, cluster, replace, failure, nodes, MPI, pinging, polling, virtual

#### C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category*</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>US 7,941,479 B2 (HOWARD, K et al.), 10 May 2011; figure 52; column 25, lines 50-61; column 27, lines 11-17.</td>
<td>1-10</td>
</tr>
<tr>
<td>Y</td>
<td>CN 1719831 A (TSINGHUA UNIV) 11 January 2006; (see machine translation).</td>
<td>2-9</td>
</tr>
<tr>
<td>Y</td>
<td>US 2013/031 1543 A1 (MASSIVELY PARALLEL TECHNOLOGIES, INC.) 21 November 2013; paragraphs [0111], [0116]; claim 14.</td>
<td>4-6, 8-9</td>
</tr>
<tr>
<td>Y</td>
<td>CN 101369241 A (INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES) 18 February 2009; (see machine translation).</td>
<td>10</td>
</tr>
</tbody>
</table>

- Special categories of cited documents:
  - "A" document defining the general state of the art which is not considered to be of particular relevance
  - "E" earlier application or patent but published on or after the international filing date
  - "L" 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
  - "Z" member document of the same patent family

#### Date of the actual completion of the international search

07 February 2015 (07.02.2015)

#### Date of mailing of the international search report

04 March 2015

**Name and mailing address of the ISA/US**

Mail Stop PCT, Attn: ISA/US, Commissioner for Patents

P.O. Box 1450, Alexandria, Virginia 22313-1450

Facsimile No. 571-273-3201

**Authorized officer:** Shane Thomas

PCT Helpdesk: 571-272-4300

PCT OSP: 571-272-7774

Form PCT/ISA/2.10 (second sheet) (July 2009)