International Journal of Automation and Computing 13(6), December 2016, 541-551
DOI: 10.1007/s11633-016-1002-6

Hybrid Collaborative Management Ring on Mobile Multi-agent for Cloud-P2P

Xiao-Long Xu[1] Nik Bessis[2] Peter Norrington[3]

[1]College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
[2]School of Computing and Mathematics, University of Derby, Derby DE22 1GB, UK
[3]Institute for Research in Applicable Computing, University of Bedfordshire, Bedfordshire LU1 3JU, UK

Abstract: In order to fully utilize all potential available network resources and make the interoperability of systems possible, we propose to integrate cloud computing and peer-to-peer (P2P) computing environments together. We utilize the mobile multi-agent technology to construct an effective hierarchical integration model named Cloud-P2P. As the original management mechanisms for traditional cloud computing and P2P computing systems are no longer applicable to Cloud-P2P, we propose a novel hybrid collaborative management ring based on mobile multi-agent in order to ensure the efficiency and success rate of task implementation in the Cloud-P2P environment. This mechanism needs to divide the system into core ring, cloud inner rings and several peer rings. In each ring, every node is in collaboration with its neighbor nodes with multi-agent, or uses mobile agent moving from node to node with string or parallel methods to monitor the statuses and performances of all nodes, in order to avoid problems of performance bottleneck and single point failure. This paper analyses the node conditions of cloud computing and P2P computing environments in-depth, then elaborates on Cloud-P2P and the hybrid collaborative management ring based on mobile multi-agent (HCMRMMA). After that, the construction method of the network ring topology for Cloud-P2P is introduced. Finally, experimental results and performance analysis of HCMRMMA are presented.

Keywords: Cloud computing, peer-to-peer (P2P) computing, mobile multi-agent, integration, management.

1 Introduction

New network computing technologies are constantly emerging. Two distributed computing models, cloud computing and peer-to-peer (P2P) computing, have attracted extensive attentions.

In the last few years, cloud computing as a new computing paradigm has gone through significant development[1, 2]. Cloud computing can support application systems obtaining computation, storage and information services as they require transparency and cost-effectiveness by distributing tasks on the resource pool consisting of cluster servers. Three cloud service models emerged: infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS)[3]. Cloud computing platforms have some particular features, such as easy-to-program, high fault-tolerance, expandable, which make large-scale distributed data processing possible[4, 5]. Google, Yahoo, IBM, Amazon have all developed their own cloud computing platforms[6], supporting large-scale information retrieval, data mining, business information processing, scientific computing, and other applications.

P2P computing also changes the traditional, unequal network computing models such as client/server computing (C/S) and browser/server computing (B/S). P2P computing can fully utilize network edge computing and storage resources. In the P2P computing environment, each peer can be a server and a client at the same time. P2P computing offers a flexible and expandable platform for information sharing, direct communication, and collaboration[7−9]. At present, P2P technology is primarily applied to the Internet-based file sharing systems, such as Napster, Gnutella, Freenet, BitTorrent, eMule, etc. The SETI@Home project at the University of California utilizes P2P computing power to analyze radio signals to search for extraterrestrial intelligence.

As stated above, cloud computing focuses on the resources of network center servers, while P2P computing focuses on the resources of network edge nodes. Although there are differences between cloud computing and P2P computing, both are adapted to solve distributed computing and resource sharing problems. If they can be combined together efficiently, Internet infrastructure and applications will be utilized with best efficiency and low-cost.

In order to fully consider and utilize all the potential available resources of cloud computing and P2P computing environments, we propose to integrate the cloud computing environment and the P2P computing environment together. The main contributions are summarized as follows:

1) We analyze the nodes of cloud computing and P2P

Research Article
Manuscript received April 23, 2014; accepted July 27, 2015; published online September 2, 2016
This work was supported by National Natural Science Foundation of China (Nos. 61472192 and 61202004), Special Fund for Fast Sharing of Science Paper in Net Era by CSTD (No. 2013116), and Natural Science Fund of Higher Education of Jiangsu Province (No. 14KJB520014).
Recommended by Associate Editor Matjaz Gams

computing environments in depth, and adopt mobile multi-agent technology to construct an effective hierarchical integration model named Cloud-P2P in this paper and make the interoperability of integrated Cloud-P2P systems possible.

2) As the original management mechanisms for traditional cloud computing systems are no longer applicable to Cloud-P2P, we also propose a novel hybrid collaborative management ring based on mobile multi-agent (HCMRMMA) in order to ensure the efficiency and success rate of the task implementation in the Cloud-P2P environment. The ring topology suitable for Cloud-P2P and its management mechanism are also presented.

The rest of the paper is organized as follows. In Section 2, we analyze the cloud computing and P2P computing environments in depth. Section 3 elaborates on the hierarchical integration model of Cloud-P2P based on mobile multi-agent. In Section 4, HCMRMMA is proposed. Experiments and performance evaluation are presented in Section 5. Finally, Section 6 concludes the paper by summarizing the main contributions of this paper and giving future directions of this work.

2 Analysis of cloud computing and P2P computing environments

In cloud computing systems, almost all complicated functions are transferred to cluster servers, greatly simplifying client-side workloads. The operation procedure of cloud computing is usually as follows: Firstly, the terminal user accesses the network and requests his requirement to the cloud computing system. If the requirement is accepted, the system organizes resources and divides tasks to provide services to the clients via the Internet. The system takes these terminals as stupid nodes, which actually regresses to the old centralized computing mainframe. The target of virtualization for cloud computing is to optimize the configuration of resources rapidly to do more computing tasks concurrently with low-cost[10, 11].

Current cloud systems usually deploy both high-performance nodes and cheap cluster servers at the same time. High-performance nodes are very expensive and responsible for the whole system management. For example, Google file system (GFS) and BigTable technologies are used for data storage and MapReduce technology is used for large-scale parallel computing on the Google cloud platform. The Google cloud platform integrates thousands of ordinary chunk servers to store data and execute tasks. In order to prevent data loss caused by node failures, Amazon′s Dynamo and Google′s BigTable both backup their data with three copies at different nodes to maintain system operating stably and constantly[12−14].

However, in P2P computing environments, resources do not come from the server nodes. Each peer is not only a consumer but also a service provider[7]. Because the applications on many network edge nodes generally process human-computer interaction transactions, computing devices are always at the idle state, resulting in huge waste of computing and storage resources. Obviously, different from those high-performance server nodes operating 7×24h continuously, and cluster servers under the centralized management and the direct control, peers can dynamically and randomly join or leave P2P computing environments. When a peer joins the P2P computing environment, it may contribute its idle resources and provide services, but this behavior is clearly not reliable, and the quality of service is difficult to ensure. Even so, the number of nodes in the P2P computing environment is always very huge, so utilization of redundancy to improve performance is entirely possible.

In short, the research on cloud computing and P2P computing has made a series of important achievements. Some researchers are even trying to apply the P2P architecture and technologies into cloud systems for making them scalable, efficient and fault-tolerant computing platforms[15, 16]. However, effective integration of resources of cloud computing and P2P computing is almost non-existent for both academia and industry.

3 Integration model of cloud and P2P computing

3.1 System architecture

As the above analysis shows, current cloud computing systems continue to ignore the computational capabilities and storage resources of edge nodes. The resource utilization rate remains in an unbalanced state. As more and more applications based on massive computing and storage resources are constructed and deployed, systems will soon face the performance bottleneck problem once again. In order to solve the contradiction, we propose a new integration model of cloud and P2P computing.

The integration model of cloud and P2P computing (Cloud-P2P) is composed of three layers including the stable core management layer, the less stable cloud resource layer and the unstable P2P resource layer. The core management layer consists of master servers responsible for the management of all Cloud-P2P nodes, resource and task deployment. The cloud resource layer consists of cluster servers. And the P2P resource layer consists of peer nodes. The three layers together constitute a huge information, storage and computation resource pool.

There are two kinds of peer in the P2P resource layer: One is stable and can provide reliable services. The other is unstable and strongly random which cannot ensure the quality of service (QoS). These two kinds of peer may also switch in a kind of "convection" situation. The system should pick honest and reliable peers to execute tasks as much as possible.

3.2 Cloud-P2P model based on mobile multi-agent

Agent can be defined in broad and narrow senses. In the

broad sense, agent generally refers to physical robots in the real world and software robots in the information world. In this paper, based on the mobile agent system interoperability facility (MASIF) and the foundation for intelligent physical agents (FIPA) specifications of agent, we define agent in the narrow sense as follows:

Definition 1. Agent is a kind of software entity, which has bigger granularity than the object, encapsulating methods, data, attributes and states. Agent has several characteristics, such as autonomy, sociality and mobility.

Agency is the container of agent which provides the communication service, the registry service, the migration mechanism, the persistence mechanism and the security mechanism.

Mobile multi-agent systems contain not only static multiple agents which can collaborate with each other but also mobile agents which can migrate among nodes, as shown in Fig. 1.

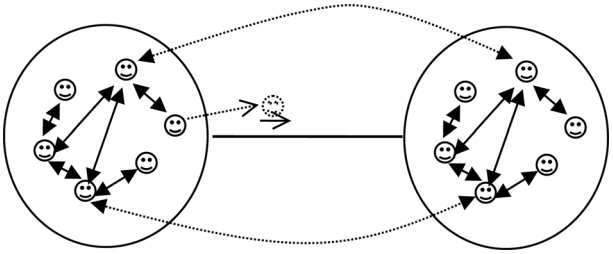

Fig. 1 Mobile multi-agent system

In order to build the Cloud-P2P model based on mobile multi-agent, agents and their operating platform should be set up on each node. This means that it is possible to utilize a group of decentralized, loosely coupled distributed intelligent agents in a Cloud-P2P computing environment to simulate the organization of human society in order to achieve highly-efficient intra-or-inter-group collaboration and effective solutions to solve a variety of conflicts and contradictions. As shown in Fig. 2, the Cloud-P2P model can be classified into five layers:

1) Network layer (NL): Master servers and cluster servers compose the local area networks of cloud computing environment and peers compose P2P computing environments at the edge of the Internet, which have great diversity in transmission mediums, communication protocols and interface methods. Especially, peers can join and leave the network at their own will, making many useless short connections with very negative influence on network performance. In order to make information exchange smooth, the network layer should dynamically change the topology of the network.

2) Resource layer (RL): There are many resources in the resource layer distributed on servers and peers. In order to achieve the highest level of automation on demand, resources must be virtualized. The large-scale virtualized resource pool based on a network platform can supply users with all kinds of information service. Virtualization improves the efficiency and usability of the application and resource, achieving the optimized allocation of various resources to process more parallel tasks on the platform.

3) Management layer (ML): This is mainly used for the management and maintenance of the entities, resources, security, robustness and accounting of the system, ensuring tasks can be deployed and scheduled efficiently. A task scheduler can be applied to multiple nodes with parallel operation or deep computing. Service management is used to manage the services and its related modules provided by the system compared with a traditional distributed system. The Cloud-P2P system is more dynamic (especially the peer nodes, which are highly autonomous), making the system′s security and reliability difficult to maintain. On the other hand, the enormous number of nodes in a P2P network can be used to ensure the reliability and stability of the system with redundancy.

4) Collaboration layer (CL): This is an agent society built with the multi-agent technology. This kind of design means that all coordination, cooperation and negotiation tasks are the duty of agents of the collaboration layer. It makes the collaboration layer become an independent unit and makes complex collaboration flexible and reliable.

5) Application layer (AL): In the Cloud-P2P model, this layer provides users with application-oriented advanced functions based on basic services of fundamental layers. The layer consists of different components. Instead of being installed on nodes in advance, components are deployed by mobile agents dynamically based on the negotiation results of multi-agents. Components can be made and distributed by task owners. This mechanism ensures that nodes can cooperate flexibly with each other in unpredictable environments.

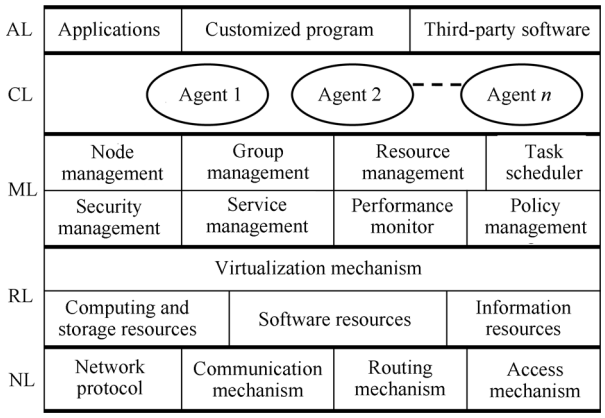

Fig. 2 Five layers of the Cloud-P2P model based on mobile multi-agent

4 Hybrid collaborative management ring based on mobile multi-agent

4.1 Problem analysis

Most P2P network computing systems use the decentralized structure without the central management node. Therefore, this structure makes the system looks like a sim-

ple nodes collection. The system′s dynamic and random response relies on node cooperating themselves. As the number of nodes and resources increasing, the system tends to anarchy, resources are hard to share, and complex cooperation is hard to set up.

Compared with P2P computing systems, cloud computing systems usually utilize the centralized management mechanism. For example, in Hadoop, the MapReduce framework consists of a single master JobTracker and slave-TaskTrackers. The master is the only one responsible for scheduling jobs′ component tasks on the slaves, monitoring them, and re-executing any failed tasks. The management cost increases significantly as the number of nodes increases[17]. The Google cloud platform uses the traditional server-farm model, and deploys one or several master servers to manage and monitor the statuses of servers, achieving the load balancing and the failure detection. The IBM′s Blue Cloud platform uses Tivoli provisioning manager and Tivoli monitoring, which is also based on a centralized architecture. Jerome Boulon designed and implemented a large-scale monitoring system based on Hadoop named Chukwa[18, 19], which still utilizes the centralized architecture for monitoring task process, device performance and system malfunction. UC Berkeley initiated an open source cluster monitoring project called Ganglia[20], which is used to measure the operation of thousands of nodes and provide performance data for cloud computing systems. Each node in the Ganglia system collects information and sends a daemon called gmond. All these repeated data collection and sending processes will affect the system performance. Carnegie Mellon University developed DSMon which can collect the resource status and load information of each node in distributed computing systems[21]. Somasundaram and Govindarajan[22] designed a cloud resource broker (CLOUDRB) for efficiently managing cloud resources and completing jobs for scientific applications within a user-specified deadline, which was implemented and integrated with the deadline-based job scheduling and particle swarm optimization (PSO)-based resource allocation mechanism, intending to achieve the objectives of minimizing both execution time and cost based on the defined fitness function.

The main defects of the centralized management mechanism are the bottleneck problem and the single point failure problem[23]. The advantages of the centralized management mechanism includes the powerful controllability and the convenient and flexible maintenance.

Povedano-Molina[24] proposed DARGOS, a distributed architecture for resource management and monitoring in clouds, which ensures an accurate measurement of physical and virtual resources in the Cloud keeping a low overhead at the same time. DARGOS has been integrated into a real cloud deployment based on OpenStack. In Amazon′s elastic cloud computing (EC2), there is CloudWatch[25], which is used to monitor the states of EC2 instances, resource utilization, CPU utilization, network traffic, etc. Amazon′s cloud storage architecture Dynamo adopts the distributed hash table (DHT)-based P2P network architecture, emphasizing the decentralized structure. Dynamo uses the Gossip-based distributed fault detection scheme to monitor and control nodes and their workloads automatically, reducing the manual administration involved. However, Dynamo is not suitable for highly dynamic, large-scale systems. Too many nodes can make the system performance dramatically worse.

As stated previously, the number of nodes in a cloud is limited, which means the centralized management mechanism is still workable. But, Cloud-P2P obviously cannot use the "heartbeat" mechanism, because the huge number of peer nodes sending cycle heartbeats will bring too much network communication burden, and exhaust the master nodes′ resources.

4.2 Hybrid composite collaborative management ring based on mobile multi-agent

In order to solve the problems discussed above, we propose a novel hybrid collaborative management ring based on mobile multi-agent, including the collaborative management ring model (CCMR) and the mobile-agent-based management ring model (MAMR).

As shown in (1), the hybrid ring is composed of three kinds of rings: the master ring (MR), which consists of master nodes, the server ring (SR), which consists of clustered servers, and the peer ring (PR), which consists of a group network edge nodes.

$$\begin{cases} MR = \{m_1, m_2, \cdots, m_x\} \\ SR = \{s_1, s_2, \cdots, s_y\} \\ P = \{PR_1, PR_2, \cdots, PR_z\},\ PR_i = \{P_{i,1}, P_{i,2}, \cdots, P_{i,j}\}. \end{cases} \tag{1}$$

As shown in Fig. 3, the hybrid collaborative management ring requires nodes staying aware of each other. In this way, the nodes have the self-management ability to avoid the single point failure problem.

There are static and dynamic agents in HCMRMMA. The nodes′ collaboration inside and outside rings are converted to the agents′ interaction and collaboration.

Each node contains several static agents to monitor the performance of the node itself and other neighbor nodes. The aim of building this management mechanism is to improve the efficiency and success rate of the system. Each node may be in one of the following three statuses:

Status 1 (available). The node is online and able to undertake tasks.

Status 2 (unavailable). The node is online, but cannot (no capacity or unwilling) undertake any task.

Status 3 (offline). The node is offline.

Nodes can switch from one status to another dynamically.

Generally, each node would like to share its spare CPU and memory and other idle resources only on the premise of ensuring its normal work. Therefore, we set some appropriate strategies for nodes. One of them is to set the threshold for their own CPU and memory. If the threshold is exceeded, the node will not undertake any extra tasks

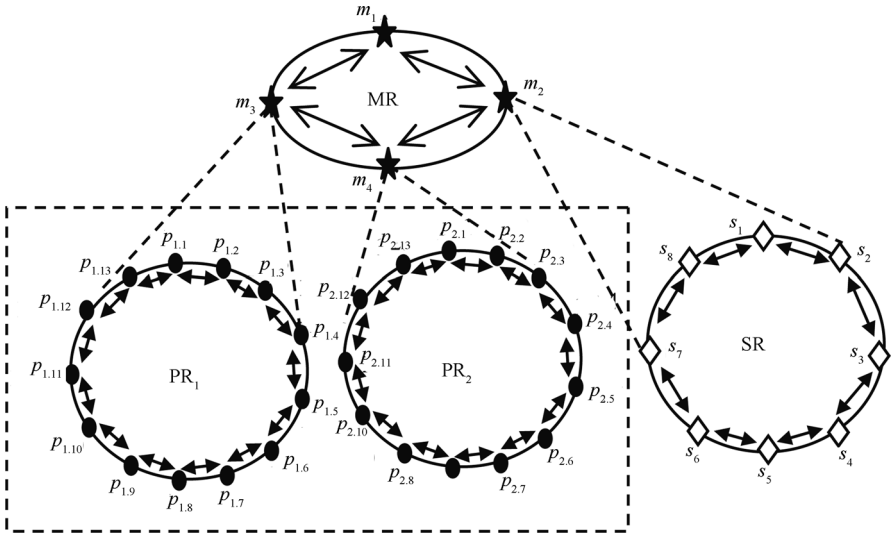

Fig. 3 Hybrid collaborative management ring

and switches to Status 2. When its own CPU and memory becomes non-busy and this continues for a certain period of time, the node will switch back to Status 1.

Unlike the traditional grouping strategy focusing on regions and resources, we group peers based on the time they spend online and the patterns of their activities. This is mainly based on considerations to maintain the stability of the peer ring, which system will be easier to manage even in the case of nodes joining and leaving the ring frequently.

The master server works as the system manager, responsible for segmenting jobs into tasks and scheduling tasks at the same time. In order to avoid the single point failure problem, the shadow nodes of the master server must be set as back-up. Shadow nodes hold the replicas of all information on the master server nodes. When the information on a master server node changes, the information on its shadow nodes will update at the same time. The master server and its shadow nodes must monitor each other′s current situations by regularly sending heartbeats to each other.

1) CCMR

Each node in CCMR contains several static agents running in the agency. Agents realize the coordination activities with the agent communication language (ACL), which is based on the theory of speech acts by Cohen and Levesque. The following is an ACL message example: node m_1 sends heartbeat information to node m_2, and asks m_2 to return its status:

(inform-one
: sender
: content (heartbeat)
: receiver
: reply-with (status)
: language Standard_Prolog
: ontology Cloud & P2P management)

In Fig. 3, for m_1, both m_2 and m_3 are its shadow nodes. For m_2, m_1 and m_4 are its core shadow nodes. If m_1 crashes, the shadow node m_2 or m_3 can immediately take on the work of m_1. m_2 or m_3 can be used to restore m_1′s data when m_1 is restarted, repaired or replaced. Any information on the master server has three copies, since the possibility of simultaneous failure of three nodes is 10^{-9}, which means the system is robust.

The P2P resource layer is divided into several distributed hash table (DHT)-based peer rings, which can make the system more structured and more easily integrated with the cloud computing system.

For example, m_3 is responsible for the management of peer ring PR_1, but m_3 does not need to interact with the nodes in the ring frequently, because the peer node state monitoring relies on the interaction between nodes themselves.

As shown in Fig. 3, $p_{1,1}$ is monitored by $p_{1,13}$ and $p_{1,2}$. Both $p_{1,13}$ and $p_{1,2}$ are monitored by $p_{1,1}$ at the same time. If $p_{1,1}$ went offline abnormally before it completes its tasks, $p_{1,13}$ and $p_{1,2}$ would not receive heartbeat messages sent from $p_{1,1}$ in a certain time, they will report the abnormal information to m_3, and then set up the supervision relationship between themselves.

2) MAMR

The agent transfer protocol (ATP) allows an agent to migrate among nodes, assigning the implementation environment and service interface for agent.

As shown in Fig. 4, agents communicate with each other and access the services provided by the agency through ACL.

ATP defines the mobile agent transfer syntax and semantics, implementing the migration mechanism of mobile agent between the service facilities. Based on IBM′s ATP framework, the basic operations used in Cloud-P2P include dispatch, retract, message and response.

If SR contains a larger number of nodes, it can also use the CCMR architecture. However, if the number of nodes in SR was not that large, MAMR, as shown in Fig. 5, would be a better choice.

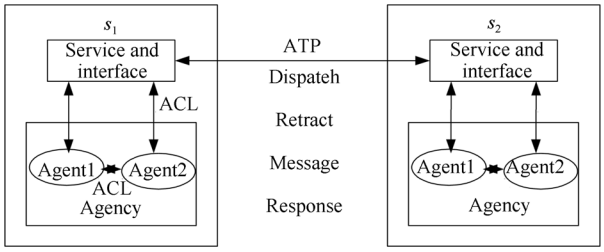

Fig. 4 Structure of the mobile agent system

As shown in Fig. 5, the master server sends several mobile agents into a SR in parallel. Each agent migrates clockwise along the ring serially. Supposing SR contains n nodes, the master server node sends j agents uniformly into SR in parallel. If the number n happens to be a multiple k (k is an integer) of j, the cruising path of agent sent to node S_i by the master sever is

$$m_2 \rightarrow S_i \rightarrow S_{i+1} \rightarrow \cdots \rightarrow S_{i+k-1} \rightarrow m_2.$$

The cruising path of the agent can be based on its routing strategy. If n happens to be a multiple r ($k < r < k+1$) of j, the cruising path of Agent_k, which is sent to node $S_{k\times j+1}$, is

$$m_2 \rightarrow S_{k\times j+1} \rightarrow S_{k\times j+2} \rightarrow \cdots \rightarrow S_n \rightarrow m_2.$$

If S_{i+l+1} does not respond when Agent i is cruising from S_i to S_{i+l} and about to move to S_{i+l+1}, S_{i+l} will immediately send a message that S_{i+l+1} has no-response to m_2. Then, the agent tries to migrate to S_{i+l+2} according to its routing table.

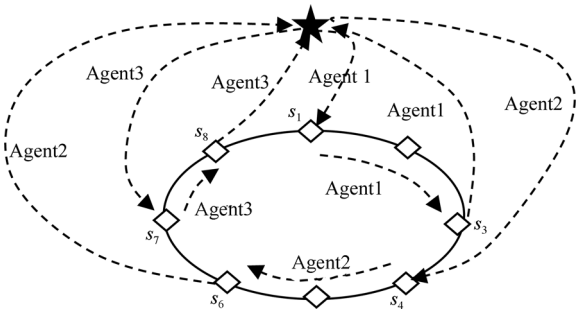

Fig. 5 Mobile-agent-based management ring model

4.3 Topology of HCMRMMA

A major problem of HCMRMMA is how to build the topology structure of the ring. In the system, there are three kinds of ring, MR, SR and PR, which have significant differences:

1) The performance of nodes in MR is very stable. As MR contains fewer nodes, the system can configure the node ID manually and put the nodes into a bidirectional ring just based on their IDs.

2) As both SR and PR contain a much larger number of nodes, the topology of the two rings must be able to adapt to the requirements of a large-scale network computing environment′s effective positioning, load balancing, better scalability, etc.

In order to achieve HCMRMMA in the integration environment of cloud and P2P computing, we designed a new ring topology which is suitable for MR, SR and PR. Firstly, the system creates the *RingID* for each ring uniquely which identifies the type and other attributes of ring. Then, the system creates the *NodeID* for each node in the ring. So the identifier of each node consists of two parts, *RingID* and *NodeID*. *NodeID* is the node identifier in the ring. Nodes are aligned clockwise from small to large by key value from 0 to $2^m - 1$(m is the digits of *NodeID*), and then make up a loop topology. For node with *NodeID*= k, the online node in the clockwise direction of the ring within a specified range is its successor node, denoted as Successor(k). The online node in the counterclockwise direction of the ring is its predecessor node, denoted as Predecessor(k). Each node in the ring establishes a monitoring relationship with its predecessor and successor. It is obvious that they need to know the situation of their predecessor and successor nodes in real time.

However, if a node only knew the situation of its direct predecessor and successor, then when either of them failed, especially for a large number of node failures, the node would not be able to establish the monitoring relationship with its new precursor and successor nodes as quickly as possible. In this paper, we use RingTable and PartRingTable to solve this problem.

1) RingTable and PartRingTable

There are two kinds of tables in the system: One is called RingTable, deployed in core nodes, storing the information of all nodes of all rings. The other is called PartRingTable. Each node has its own PartRingTable which stores the information of its direct and indirect neighbor nodes in the same ring. If a ring contains n nodes, the space complexity of each PartRingTable is $\log n$. The content of PartRingTable is shown in Table 1.

Table 1 Definition of each element in PartRingTable

PartRingTable	Definition
$NodeID$	The node identifier
$RingID$	The ring identifier
Distance	The network distance to the node where this PartRingTable exists
Place	Predecessor or Successor
Status	available, unavailable or offline

"Distance" in Table 1 means the relative network distance to the node where this PartRingTable exists. The network distance of one node to its direct predecessor and successor nodes is "−1" and "1", respectively. The network distance of the node to its first indirect predecessor and successor node is "−2" and "2", respectively.

As shown in Fig. 6, supposing there is a node whose

NodeID is 4 (i.e., $Node_4$) in a ring whose RingID is x and contains 16 nodes, the PartRingTable in $Node_4$ has 4 entries. $Node_4$'s direct predecessor is $Node_2$ and its direct successor is $Node_6$. In addition, the PartRingTable also includes the information about the direct predecessor of $Node_2$ and the direct successor of $Node_6$. In this way, if $Node_2$ or $Node_6$ fails, $Node_4$ can build the new relationship of $Node_1$ with $Node_{10}$ or quickly update its PartRingTable straightaway.

2) Nodes joining and exiting a ring

When a node wants to join a ring, it needs the core node to introduce it, help it to initialize its PartRingTable and update its neighbor nodes' PartRingTables. For example, as shown in Fig. 5, if $Node_3$ wants to join $Ring_x$, it should follow these processes:

Step 1. $Node_3$ sends a message containing its identifier to the core node to request to join $Ring_x$. The core node checks the message and returns a digital certificate and a response message to $Node_3$. The response message contains the information of $Node_2$ and $Node_4$.

Step 2. When $Node_3$ receives the response message, it will contact $Node_2$ and $Node_4$ immediately to initialize its own PartRingTable.

Step 3. When $Node_2$ and $Node_4$ receive the message called UpdatePredReq and UpdateSuccReq from $Node_3$, they will update their PartRingTables and inform $Node_1$ and $Node_6$ to update their PartRingTables.

If a node tries to exit a ring, there are two possibilities: One is the node exiting the ring normally, and the other is the node exiting abnormally. The normally exiting node has to send a message requiring to logout to a core node, and inform its predecessor and successor that it is going to exit the ring. After its predecessor and successor receive the notification, they will update their own PartRingTables. However, if the node fails to do those things due to some unexpected reasons, such as the link to the network disconnects, system failure or random user action, its predecessor and successor need to update their own PartRingTables initiatively to maintain the stability of the ring.

The ring proposed in this paper not only applies to parse-node failure, but also applies to the continuous-node failure. When continuous nodes in a ring exit the ring abnormally, the core node will receive one report from the nodes predecessor or successor, then the server will send a message to the nodes successor or predecessor to check its status. If the nodes successor or predecessor also exit abnormally, the core node will take appropriate measures.

As shown in Fig. 6, supposing $Node_0$, $Node_1$ and $Node_2$ abnormally exit the ring almost together, the core node will receive a report from $Node_{15}$ about the failure of $Node_0$ and a report from $Node_4$ about the failure of $Node_5$. As the core node does not receive the report from $Node_1$ which is the successor of $Node_0$, the core node will send a message to $Node_1$ to check its status. If $Node_1$ fails to respond to the core node, the core node will know that $Node_1$ also exits and deal with the report sent by $Node_{15}$ and confirm the exit of $Node_0$. The same process is also used to confirm the failure of $Node_2$.

5 Experiments and performance analysis

The following are the simulation experiments on HCMRMMA, including CCMR and MAMR. Especially, we compared HCMRMMA with the traditional centralized management mechanism (CM) on the load of nodes, the efficiency of failure detection, etc.

5.1 Load of nodes

We built the simulation experimental platform for HCMRMMA in the Cloud-P2P system in an intranet environment. The experiment parameters are set as follows:

1) The interval time that a task execution node sends the heartbeat message to a core node: 10 s

2) The interval time that a task execution node sends the heartbeat message to each other: 6 s

3) The failure rate of task execution nodes: 5 %

4) The reconstruction rate of task execution nodes: 50 %

5) The period of the experimental time: 10 000 s

6) The judgment time of node failure: 30 s.

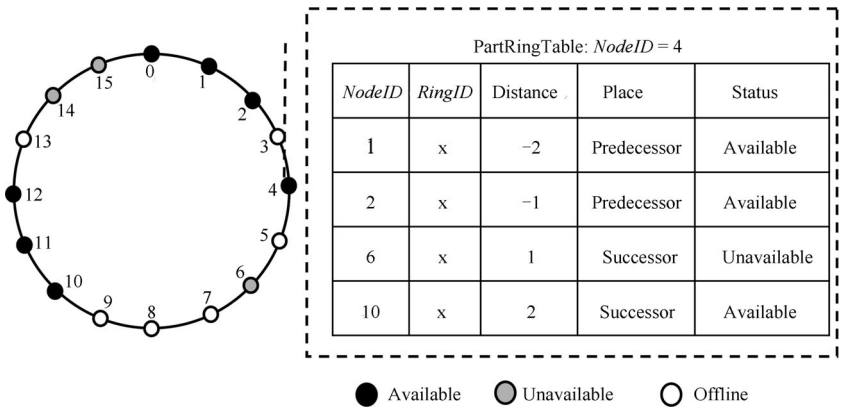

Fig. 6 Topology of hybrid collaborative management ring

As shown in Fig. 7, with increasing number of nodes, the number of monitoring reports increases both with CM and with CCMR, but the growth rate with CM is significantly greater with CCMR.

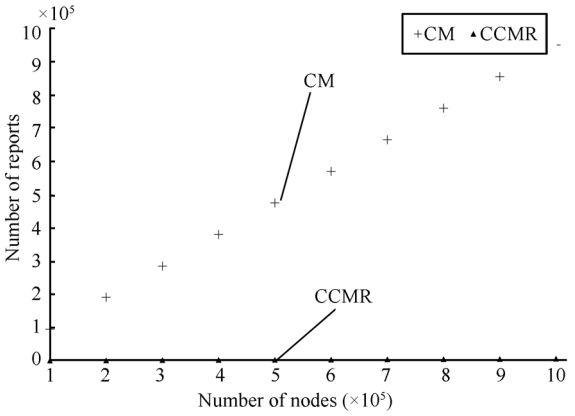

Fig. 7 Number of reports the core node receives under different node scales

Fig. 8 shows that the number of reports the core node receives with CCMR is significantly less than that with CM. In summary, CCMR can significantly reduce the resource consumption of the core node.

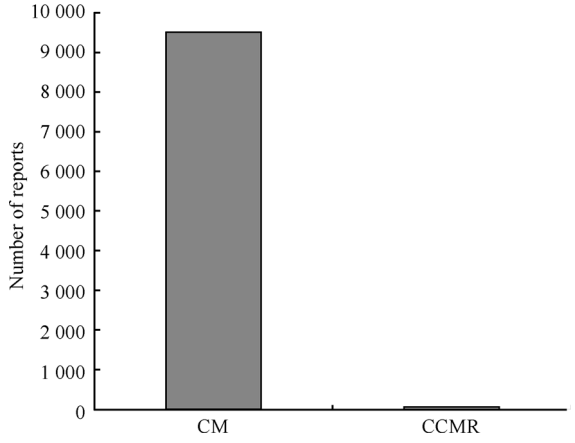

Fig. 8 Number of reports the core node receives with CM and CCMR

The system switches the traditional method of task execution nodes reporting regularly to the management node into regular communication between task execution nodes. Fig. 9 shows the statistics for resource consumption. In CCMR, each node is monitored by its predecessor and successor nodes, so the consumption is greater than that in the CM model. But for all the nodes in the system, the average cost of each node is very low.

5.2 Efficiency of failure detection

In the following, we show the experiment results for the efficiency of node failure detection with CCMR, MAMR and CM. The experiment parameters are set as follows:

1) The total number of nodes: 10 000

2) The interval time that a task execution node sends the heartbeat message to a core node in CM: 9 s

3) The interval time that a task execution node sends the heartbeat message to each other: 9 s

4) The interval time of determining node failure: 3×9 s.

5) The time that a task execution node sends the heartbeat message to a core node in CCMR: 800 ms

6) The time that an agent migrates from one node to another in MAMR: 800 ms.

Fig. 10 shows the experiment results under different node failure rates. The gray bar shows the average time of core nodes finding out task execution node failure in the situation where the node failure rate is 1 %. The black bar shows the average time of core nodes finding out task execution node failure when the node failure rate is 10 %. We can see that the failure rate does not significantly affect the performance of the three management models. In MAMR, the time cost on finding a disabled node depends on its position in the agent migration route.

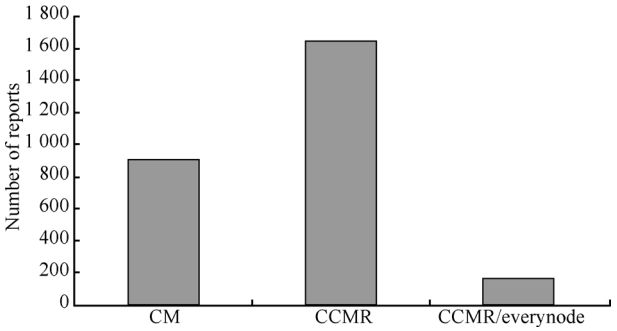

Fig. 9 Number of reports in the system with CM and CCMR

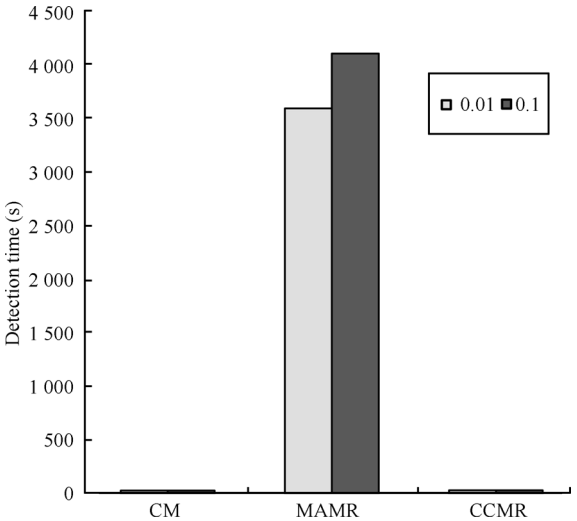

Fig. 10 Average detection time of failed nodes under different rates of node failure

The later the agent migrates to a node, the longer the time cost to find its failure. Supposing the number of nodes in a ring is n, the number of agents dispatched is m, and

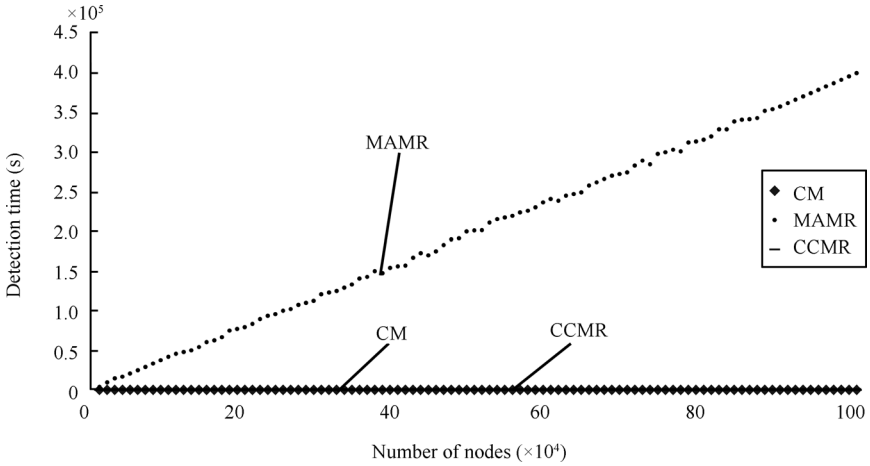

Fig. 11 Detection time of failed nodes under different node scales

the time an agent migrates from one node to another is t, the average detection time of node failure is $\frac{n}{2m} \times t$.

Fig. 11 shows the experimental results of detection time under different nodes scales. We find that increasing nodes does not significantly affect the performance of CM and CCMR. In CM, the failed node will be found after several heartbeat cycles. In CCMR, the failed node can be detected in the same way. However, in MAMR, a failed node can only be found while an agent tries to migrate to the node, the node scale significantly affects the detection efficiency of node failure. With increasing node scale, the migration path of an agent becomes longer, which makes the detection efficiency lower. The average detection time is inversely proportional to the number of nodes. Even so, we can dispatch more agents migrating in parallel in the ring.

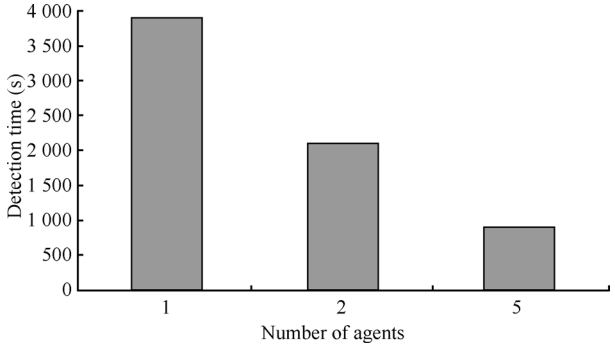

Fig. 12 The detection time of failed nodes with different number of agents

As shown in Fig. 12, increasing the number of agents can reduce the detection time, which is inversely proportional to the number of agents. At the same node scale, the increasing number of agents reduces the migration distance of each agent and increases the test frequency of each node. As a result, the detection efficiency is improved. It can be found that MAMR is applicable for the stable Cloud Resource Layer.

6 Conclusions

One of the core objectives of modern network computing and information communication is to eliminate information islands and maximize aggregation of all WAN and LAN computing, storage and information resources to meet the requirements of large-scale computing and massive data processing. Compared with traditional cloud systems, the Cloud-P2P system proposed in this paper can aggregate a wider range of resources and further eliminate the information island problem.

However, Cloud-P2P is an open large-scale computing environment. Security, reliability, stability and scalability are its major problems. HCMRMMA has been proved by experiments that it is applicable for large-scale computing environments, especially Cloud-P2P. HCMRMMA can effectively maintain the stability and reliability of Cloud-P2P.

Follow-up works should focus on how to improve service security and credibility of Cloud-P2P. It is important to consider the possibility that malicious network edge nodes hiding in Cloud-P2P systems may attack the networks via fake services, conspiracy, non-cooperation and other malicious behaviors. Simple, efficient and quantitative trust mechanism and incentive mechanism will become our focus of future research for realizing a more robust Cloud-P2P system.

Acknowledgments

We would like to thank the reviewers for their detailed comments and suggestions throughout the reviewing process that helped us significantly improve the quality of this paper.

References

[1] W. J. Fan, S. L. Yang, H. Perros, J. Pei. A multi-dimensional trust-aware cloud service selection mechanism based on evidential reasoning approach. *International Journal of Automation and Computing*, vol. 12, no. 2, pp. 208–219, 2015.

[2] Y. K. Guo, L. Guo. IC cloud: Enabling compositional cloud. *International Journal of Automation and Computing*, vol. 8, no. 3, pp. 269–279, 2011.

[3] J. F. Zhao, J. T. Zhou. Strategies and methods for cloud migration. *International Journal of Automation and Computing*, vol. 11, no. 2, pp. 143–152, 2014.

[4] P. Mell, T. Grance. The NIST definition of cloud computing. *Communications of the ACM*, vol. 53, no. 6, pp. 50, 2010.

[5] Y. Huang, N. Bessis, P. Norrington, P. Kuonen, B. Hirsbrunner. Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm. *Future Generation Computer Systems*, vol. 29, no. 1, pp. 402–415, 2013.

[6] C. L. Cheng, C. J. Sun, X. L. Xu, D. Y. Zhang. A multidimensional index structure based on improved VA-file and CAN in the cloud. *International Journal of Automation and Computing*, vol. 11, no. 1, pp. 109–117, 2014.

[7] G. Chmaj, K. Walkowiak. A P2P computing system for overlay networks. *Future Generation Computer Systems*, vol. 29, no. 1, pp. 242–249, 2013.

[8] D. Castellá, F. Giné, F. Solsona, F. J. L. Lérida. Analyzing locality over a P2P computing architecture. *Journal of Network and Computer Applications*, vol. 36, no. 6, pp. 1610–1619, 2013.

[9] T. Ghafarian, H. Deldari, B. Javadi, M. H. Yaghmaee, R. Buyya. CycloidGrid: A proximity-aware P2P-based resource discovery architecture in volunteer computing systems. *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1583–1595, 2013.

[10] J. Z. Luo, J. H. Jin, A. B. Song, F. Dong. Cloud computing: Architecture and key technologies. *Journal on Communications*, vol. 32, no. 7, pp. 3–21, 2011. (in Chinese)

[11] Q. Liang, Y. Z. Wang, Y. H. Zhang. Resource virtualization model using hybrid-graph representation and converging algorithm for cloud computing. *International Journal of Automation and Computing*, vol. 10, no. 6, pp. 597–606, 2013.

[12] L. A. Barroso, J. Dean, U. Holzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.

[13] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber. BigTable: A distributed storage system for structured data. *ACM Transactions on Computers*, vol. 26, no. 2, pp. 205–218, 2008.

[14] J. Dean, S. Ghemawat. Distributed programming with Mapreduce. *Beautiful Code*, A. Oram, G. Wilson, Eds., Sebastopol, USA: OReilly Media, Inc., pp. 371–384, 2007.

[15] F. Marozzo, D. Talia, P. Trunfio. A peer-to-peer framework for supporting MapReduce applications in dynamic cloud environments. *Cloud Computing: Principles, Systems and Applications*, N. Antonopoulos, L. Gillam, Eds., London, UK: Springer-Verlag, pp. 113–125, 2010.

[16] R. Ranjan, L. Zhao, X. M. Wu, A. N. Liu, A. Quiroz, M. Parashar. Peer-to-peer cloud provisioning: Service discovery and load-balancing. *Cloud Computing: Principles, Systems and Applications*, N. Antonopoulos, L. Gillam, Eds., London, UK: Springer-Verlag, pp. 195–217, 2010.

[17] H. Jin, S. Ibrahim, T. Bell, L. Qi, H. J. Cao, S. Wu, X. H. Shi. Tools and technologies for building clouds. *Cloud Computing: Principles, Systems and Applications*, N. Antonopoulos, L. Gillam, Eds., London, UK: Springer-Verlag, pp. 3–20, 2010.

[18] J. Boulon, A. Konwinski, R. P. Qi, A. Rabkin, E. Yang, M. Yang. Chukwa: A large-scale monitoring system, [Online], Available: http://mmm.csd.uwo.ca/courses/CS9842/papers/Paper-13-Ariel-Rabkin.pdf, October 1, 2008.

[19] M. Madhury, P. M. Dhanya. An exploratory survey of Hadoop log analysis tools. *International Journal of Computer Applications*, vol. 75, no. 18, pp. 33–36, 2013.

[20] M. Massie, B. Li, B. Nicholes, V. Vuksan , R. Alexander, J. Buchbinder, F. Costa, A. Dean, D. Josephsen, P. Phaal, D. Pocock. *Monitoring with Ganglia*. Sebastopol, USA: O′Reilly Media, Inc., 2012.

[21] IBM Systems Group. An introduction to DSMon, [Online], Available: ftp://public.dhe.ibm.com/eserver/zseries/zos/racf/pdf/r05_dsmon_introduction.pdf, May 20, 2005.

[22] T. S. Somasundaram, K. Govindarajan. CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud. *Future Generation Computer Systems*, vol. 34, pp. 47–65, 2014.

[23] B. K′onig, J. M. A. Calero, J. Kirschnick. Elastic monitoring framework for cloud infrastructures. *IET Communications*, vol. 6, no. 10, pp. 1306–1315, 2012.

[24] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, L. Foschini. DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds. *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2041–2056, 2013.

[25] Amazon Web Services, Inc. Amazon CloudWatch, [Online], Available: http://aws.amazon.com/cn/cloudwatch/, December 16, 2014.

Xiao-Long Xu received the B. Sc. degree in computer and its applications, the M. Sc. degree in computer software and theories, and the Ph. D. degree in communications and information systems at Nanjing University of Posts & Telecommunications, China in 1999, 2002 and 2008, respectively. He worked as a postdoctoral researcher at Center of Electronic Science and Technology, Nanjing University of Posts & Telecommunications from 2011 to 2013. He is currently a professor in College of computer, Nanjing University of Posts & Telecommunications, China. He is a senior member of China Computer Federation, China. He has published about 110 refereed journal and conference papers.

His research interests include cloud computing, mobile computing, intelligent agent and information security.

E-mail: xuxl@njupt.edu.cn (Corresponding author)
ORCID iD: 0000-0001-6254-5864

Nik Bessis received the B. A. degree from the Technological Educational Institute (TEI) of Athens, Greece, and the M. A. and the Ph. D. degrees at De Montfort University, Leicester, UK.

He joined Department of Computer Science and Technology at University of Bedfordshire, UK as a Lecturer in 2001. In 2004 he was promoted to be senior lecturer and the postgraduate course manager. In 2009, he promoted to be a principal lecturer. In December 2010, he joined the University of Derby, UK, as a professor of computer science and led the research within the School. Since January 2011, he is the head of the distributed and intelligent Systems (DISYS) research group and the REF UOA11 leader. He has published over 165 works and has received three Best Papers Awards (2009, 2009 and 2012).

His research interests include web-centric system developments, dynamic web applications, data integration, social networking, data analytics, and visualization and resource management.

E-mail: n.bessis@derby.ac.uk

Peter Norrington received the B. A. degree in linguistics and philosophy at the University of Sheffield, UK in 1990, the M. Sc. degree in internet technologies at University of Bedfordshire, UK in 2004, and the Ph. D. degree in computer & internet security at University of Bedfordshire, UK in 2009. He works for University of Bedfordshire, UK since 2008. He is now a Ph. D. research supervisor, higher education consultant and freelance.

His research interests include Internet, information security and usability, and project and programme management.

E-mail: p.norrington@gmail.com