ChordMR: A P2P-based Job Management Scheme in Cloud

Jiagao Wu, Hang Yuan, Ying He, and Zhiqiang Zou
College of Computer, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu, China
Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing, Jiangsu, China
Email: jgwu@njupt.edu.cn, {gegeyyhh, rainbowgirlgo}@163.com, zouzq@njupt.edu.cn

Abstract—MapReduce is a programming model and an associated implementation for processing parallel data, which is widely used in Cloud computing environments. However, the traditional MapReduce system is based on a centralized master-slave structure. While, along with the increase of the number of MapReduce jobs submitted and system scale, the master node will become the bottleneck of the system. To improve this problem, we have proposed a new MapReduce system named ChordMR, which is designed to use a peer-to-peer Chord network to manage master node churn and failure in a decentralized way. More importantly, we propose a job management scheme which contains the aspects of job assignment, job backup, job recovery, etc. In addition, in ChordMR, the job backup scheme makes a number of \(2^{n-1}\) identical copies for each job and classifies the jobs into \(K_2\) classes to achieve the goals of fault tolerance and load balance. The theoretical and simulation results show that our ChordMR is effective.

Index Terms—Cloud Computing; P2P; Chord; MapReduce; Fault Tolerance; Load Balancing

I. INTRODUCTION

Cloud Computing is the development of parallel computing, distributed computing and grid computing. And a lot of technologies are used by Cloud Computing, including virtualization technology, data storage, data management, programming model and task scheduling, to provide computing and storage service. MapReduce, which was put forward by Google in 2004 [1], is a remarkable parallel programming model and method for dealing with large input data which are generated by Internet-based applications. And it has achieved a huge success because of its excellent fault tolerance features, scalability and ease of use.

The MapReduce facilitates easy parallel program implementation for many real-world tasks on clusters through a simple interface with two functions, map and reduce, which are defined by user. And the map function processes a input \(\text{(key, value)}\) pair to produce a set of intermediate \(\text{(key, value)}\) pairs, and the reduce function accepts the intermediate key and a set of values for that key and merges these values to form a smaller set of values. Today, Google [2] and open source Cloud Computing platform Hadoop [3] both use the MapReduce parallel computing model which is based on centralized master-slave architecture. However, this centralized system cannot be handled well in the high dynamic Cloud environment. Since all the jobs submitted by users are processed by only one master which is the bottleneck of the whole network, resulting that the scalability of the Cloud cluster will be limited. In addition, this centralized master-slave structure will lead to the problem of single point failure, once the master fails, the whole system will be paralyzed. Therefore, to solve the problems talked above, we have proposed a new MapReduce system named ChordMR, which is designed to use a peer-to-peer Chord [4] network to enhance the scalability of Cloud cluster and manage master failure in a decentralized way. More specifically, for solving the problem of scalability limited, the one master node is extended to multiple masters that are organized as Chord network are responsible for jobs assignment and execution. In order to tackle the problem of jobs lost caused by master failure, we propose a complete P2P-based job management scheme in ChordMR, which contains MapReduce job assignment, job backup, job recovery and masters join/leave scheme. Among them, the job assignment scheme can help to assign jobs to ChordMR evenly on the basis of the SHA-1, reaching the goal of load balancing. The job backup scheme makes a number of \(2^{n-1}\) identical copies for each job and classifies the jobs into \(K_2\) classes to achieve the goals of fault tolerance and load balance. In addition, based on the rule of job backup scheme, the corresponding job recovery and masters join/leave scheme are given out, too. The theoretical and simulation results show that the ChordMR can improve the success rate of jobs and have a better performance on load balance.

The remainder of this paper is organized as follows: section 2 we discuss related works. Section 3 we will give an introduction to the system model and architecture of ChordMR. In section 4, we will describe the Job management scheme. In section 5, we will give the performance analysis. Experimental results will be provided in section 6. And we will give final conclusions and future works in section 7.

II. BACKGROUND AND RELATE WORK

A. P2P Networks

Peer-to-peer [5] networks and applications are distributed systems without any centralized control or
hierarchical organization, where the software running at each node is equivalent in functionality. P2P networks have a lot of features, such as efficient, load balance, high scalability. There are mainly three kinds of P2P networks today: unstructured, semi-structured and structured P2P networks. Among them, the structured P2P networks are characterized by static logic and high-performance routing. One of the most famous structured P2P networks is called Chord network, which is one of the most simple and accurate P2P ring structure was put forward by MIT in 2001. Chord organizes nodes in a ring based on node identifiers, and each node has a finger table with a specific size. Node and data object are hashed to a unique identifier using Secure Hashing Standard (SHA-1 [6]) base hash function and mapped in the Chord network. The burden of each node in the Chord network is roughly same because of the SHA-1.

Although P2P networks have a lot of advantages, they also have their faults, for example, data loss caused by peer joins or leaves at a high rate. To improve availability in unreliable P2P systems, a well-known technique is replication. Several peer-to-peer derived backup systems have appeared in the past few years. Anglano et al. [7] pointed out that the given C identical replicas of the same object are stored on the k successors in the Chord network. Emils et al. [8] presented that add a central server in the P2P network, and the data recovery could be solved by accessing the central server. Peerstore [9] is a model that divides the system into two layers: metadata layer and symmetric trading layer. The metadata layer is responsible for recording data block and location of backup blocks, and the symmetric trading layer is answer for data storage and data backup.

B. Mapreduce Implementations

A number of MapReduce implementations have been proposed in the last years. Besides the original one by google, several other MapReduce implementations such as Hadoop [3], MapSharp [10], GridGain [11], Nimbus [12], Eucalyptus [13] and Enomaly ECP [14] also have been realized. However, as we known, the traditional MapReduce system is based on a centralized master-slave architecture, which cannot be handled well in the Cloud environment with high dynamic and overload characteristics. So a lot of works have been done to solve this problem. Wang et al. [15] presented that we could use metadata replication to avoid the failure of single point and higher the availability of Hadoop. Murthy [16-17] put forward a method that transferred a part of works from master node to slave nodes to execute. It could reduce the workload of master and the rate of master failure. There were also some studies focus on combining Cloud with P2P architecture, such as P2P-MapReduce [18], which basic idea differed from centralized architecture for it used of a peer-to-peer approach both for job and system management. But it was a kind of unstructured P2P models, and did not have the benefits such as loading balance, structural management and high performance route. Gangeshwari et al. [19] presented HPCloud, which consisted of eight masters connected to one another in a structured peer-to-peer framework on a Cloud environment. And its fault tolerance was achieved by dynamically making back up for each of the masters. Whereas, the number of masters limited to eight resulted the expandability of HPcloud was restricted.

III. ARCHITECTURE

As mentioned above, the purpose of ChordMR is to enable a reliable execution of MapReduce applications and achieve a better solution for master failure and churn. To achieve this goal, our design adopts a peer-to-peer model in which a set of nodes whose load capacity are higher act as master nodes.

As shown in Fig. 1, the architecture of ChordMR we present includes three basic roles: User, Master and Slave. The User nodes are responsible for submitting jobs. The Master nodes are organized as Chord network are responsible for jobs assignment and execution. Nevertheless, the Slave nodes, which are in charge of executing Map/Reduce tasks, are still maintained in traditional Cloud structure, such as Hadoop platform. When a User submits a job, which is identified with a unique job id by SHA-1, to the system, the job is routed to the Master which is the successor node of the job’s id in Chord. In this way, the number of jobs each Master receives is approximately equal, and we can achieve the goal of load balancing. Meanwhile, to prevent the loss of jobs data caused by failure of Master, each Master further acts as two roles: the first role is Primary Master which is responsible for maintaining original executing states of a job User submits; the second role is Backup Master which just stores the copy of states of the job.

Each job executed by its Primary Master should be replicated multiple identical copies of states to its backup Master. If a Primary Master fails, the successor Master in Chord will detect the failure, and start a recovery procedure to recover the job's data from Backup Master. At last, the failure Master's role will be replaced by its successor in Chord. This process is transparent to both the User and the job itself. In addition, we don't take Slave failure into consideration in our system because of it could be solved by the existing Cloud mechanisms, such as Hadoop platform.
In the following we will describe, through an example, how a job is submitted to the system and how a Master failure is handled in the ChordMR architecture.

As shown in Fig. 2(a), a User node (U) submits a job J, which is identified with a unique job id by SHA-1, to the system. Then the job is routed to the Master which is the successor node of the job's id in Chord as its Primary Master. Meanwhile, based on the backup scheme proposed in this paper later, multiple identical copies of the job will be replicated to a number of corresponding Masters which act as Backup Masters of J. In this example, M_1 is selected as the Primary Master for job J and (M_2 - M_3) as Backup Master.

M_1 divides J into several Map/Reduce tasks, and selects several Slaves to execute these tasks. The choice of slaves could follow various polices, based on current workload, highest reliability, and so on. In this example, S_0, S_2, S_3 are chosen as Slaves to execute the Map/Reduce tasks.

While executing of Job J, M_1 will dynamically transfer the update states of J to all its Backup Masters (M_2-M_3). And to prevent excessive overhead, the updates do not contain whole job information, but the information has updated.

As soon as the Map/Reduce tasks of job J is completed, M_0 returns the result to U.

If M_1 fails, as show in Fig. 2(b), M_0, which is the successor of M_1 in Chord, will detect the failure. And with the recovery scheme proposed in this paper later, M_0 will find the all jobs data executed and backed up by M_1 from corresponding Master nodes. Then, these parts of jobs data will be transmitted to M_0 and recovered.

After receiving the recovered jobs data, M_0 will become the new Primary Master node for the jobs executed by M_1 before and continue to execute these jobs.

IV. JOB MANAGEMENT SCHEME

In this section, we provide a detailed description of our Job management scheme. Let Master Id is a Master identifier, which represents the id of a Master node. jobId is a job identifier, which represents the id of a MapReduce job.

The job management scheme we will talk about consists of four aspects, which are the followings: Jobs Allocation Scheme, Jobs Backup Scheme, Jobs Recovery Scheme and Masters Join/Leave Scheme. The detail of our scheme depicted as Fig. 3.

A. Job Assignment Scheme

Based on the Chord routing algorithm used in our Job Assignment Scheme, we could prevent some Masters from excessive overhead and reach the goal of load balancing. The amount of jobs each Master executing is approximately equal, so we could reduce the possibility of Master failure caused by excessive overhead. The process of MapReduce Job Assignment Scheme includes the following steps:

A User node submits a job, which is identified with a unique jobId by SHA-1, to the system. Then the job is routed to the Master which is the successor node of the jobId in Chord as its Primary Master.

And then the job will be divided into several MapReduce tasks by the Primary Master and assigned to some Slaves to execute. The choice of Slaves could follow various polices, based on current workload, highest reliability and so on.

As soon as the MapReduce job is completed, the Primary Master will send the results back to the User who submits this job.

B. Job Backup Scheme

In order to avoid loss of jobs data caused by Master failure, proper job backup schemes are required. In this paper, we propose the Job Backup Scheme which contains the following steps:

The Job Backup Scheme utilizes modulus to classify the MapReduce jobs to K_2 classes based on jobId. Meanwhile, each MapReduce job has a number of 2^k - 1 identical copies. Based on our backup scheme, the 2^k - 1 copies will be distributed in the Chord network with roughly equal intervals. The calculation method of job backup could be described as (1):

\[ B_{jobId} = (jobId + (jobId \% 2^k) \cdot 2^{M-k} - k_i) \cdot 2^{M-k} - k_i, \quad 1 \leq i \leq 2^k - 1 \]
where BjobId, represents the id of the i-th backup of the MapReduce job. M represents the hash value length of secure hash function SHA-1 used by Chord network. Both \(K_1\) and \(K_2\) are belong to the set of natural numbers \(N\).

According to the BjobId, we got, the Primary Master will back up the data of MapReduce jobs to the corresponding successor Master in Chord as the job’s Backup Master.

Set a timer \(T\), the Primary Master of one MapReduce job will send the job data updates to all its Backup Masters every time \(T\), to ensure data consistency. In order to prevent excessive overhead, the updates does not contain whole job information, but only that part of information that has changed.

C. Job Recovery Scheme

In a P2P network, Master node may leave at any time. When a Master node left the Chord network, the lost jobs data belong to the failure Master could be divided to two kinds to recover. The first one contains the jobs which act the failure Master as their Primary Master, the second one contains the jobs which act the failure Master as their Backup Master. The process of MapReduce Job Recovery Scheme includes the following steps.

Recovery of first kind of jobs:

Once a Master node fails, the successor of the failure Master in the Chord network will detect the failure and utilize (2) to figure out a MasterId set, BID, which contains the Masters act as the Backup Master of the first kind of jobs.

\[
\text{BID} = \{id \mid (X + 2^M - j) \% 2^M < id \leq (Y + 2^M) \% 2^M, \quad 0 \leq j \leq 2^{K_1} - 1\} \tag{2}
\]

where \(Y\) is the MasterId of the failure Master, and \(X\) is the MasterId of the failure’s processor Master in the Chord network.

Then, the successor will find out the backup data of the first kind of jobs, which satisfy the formula \(\text{jobId} \in (X, Y]\), from the corresponding Backup Masters whose MasterId \(\in \text{BID}\).

At last, the jobs data, which are found out in step (b) will be transmitted to the successor of the failure Master in the Chord network. The successor will become the Primary Master of these jobs.

Recovery of second kind of jobs:

The successor of the failure Master in the Chord network will figure out BJID, which is a jobId set contains the jobs, which act the failure Master as their Backup Master, through (3):

\[
\text{BjobId} = \{\text{jobId} + (\text{jobId} \% 2^{K_1}) \cdot 2^{M-K_1} \% 2^M, \quad 1 \leq i \leq 2^{K_1} - 1\} \tag{3}
\]

where \(Y\) is the MasterId of the failure Master. And \(X\) is the predecessor’s MasterId of the failure Master in the Chord network.

Based on the scheme of job backup and symmetry we store our job data, the job data backed up in the failure Master must be also all backed up in the list of Masters BBID, which could be figured out through (4):

\[
\text{BBID} = \{\text{id} \mid (X + 2^M - j) \% 2^M < id \leq (Y + 2^M - j) \% 2^M, \quad 0 \leq j \leq 2^{K_1} - 1\} \tag{4}
\]

According to step (b), the corresponding successor Master of BBID in Chord network are selected as the candidates, and the successor of the failure Master in the Chord network will take out the jobs data whose \(\text{jobId} \in \text{BJID}\), from these candidates, and then become the new Backup Master of these jobs.

D. Masters Join/Leave Scheme

Masters Join Scheme

When a new Master joins in Chord network, some jobs executed or backed up by its successor in the Chord network should be transmitted to the new Master to execute or back up. The Master Join Scheme is described as follows:

Firstly, When a new Master joins in our network, the successor of the new Master in the Chord network will detect it, and then will find out the list of jobs which should be executed by the new Master and transmit them to the new Master as their Primary Master. The jobId of these jobs satisfy the formula \(\text{jobId} \in (X, Y]\), where \(Y\) is the MasterId of the new Master and \(X\) is the MasterId of new Master’s processor in the Chord network.

Secondly, the successor of the new Master will figure out the backup list of jobs which should be backed up in the new Master based on (5), take out these jobs data and transmit them to the new Master as their Backup Master.

\[
\text{JBID} = \{\text{id} \mid (X - i \cdot 2^{M-K_1} - j \cdot 2^{M-K_1} - 2^M) \% 2^M, \quad 1 \leq i \leq 2^{K_1} - 1, 0 \leq j \leq 2^{K_2} - 1\} \tag{5}
\]

At last, the successor of the new Master will delete these jobs data, which are transmitted to new Master, from its own space.

Masters Leave Scheme

When a Master left the network, the successor of this Master in the Chord network will detect it and utilize the Job Recovery Scheme, as in Section IV-C, to recover the jobs which act the left Master as Primary Master or Backup Master, respectively.

V. PERFORMANCE ANALYSIS

In this section, we will provide some performance analysis for ChordMR in success rate and load balance respectively.

A. Analysis of Success Rate

Whether the MapReduce job could be finished depends on the failure rate of the Masters, and the number of replicas had in the system. Therefore, we develop a
formula to calculate the success rate of a MapReduce job with the parameter talked above.

The success rate of a MapReduce job with Job Backup Scheme could be calculated through (6),

\[ P_A = 1 - P^{K_1} \]

where \( K_1 \) is the number of replicas of a MapReduce job, \( P \) is the failure rate of a Master.

While, if there is not Job Backup Scheme in system, \( K_1 = 0 \). So the success rate of a MapReduce job without Job Backup Scheme is:

\[ P_B = 1 - P \]

It is obvious that \( P_A \geq P_B \). It means that, compared to the system without backup mechanism, the success rate of MapReduce job with Job Backup Scheme enhances significantly. Therefore, the Cloud Platform will be more stable.

B. Analysis of Load Balance

Chord network uses a variant of consistent hashing [5] to assign keys to Chord nodes. Consistent hashing used in Chord tends to balance load, since it has the ability to allocate MapReduce jobs to Primary Masters evenly, therefore the number of jobs each Primary Master receive is approximately equal. Meanwhile, the Job Backup Scheme we talked above have a parameter named \( K_2 \), we classify the jobs to \( K_2 \) classes by performing modulo \( K_2 \) arithmetic. It could achieve the goal that disperses all the MapReduce jobs executed by one Primary Master to much more Backup Masters in the system to back up.

We calculate the average number of Backup Masters required backing up all the jobs executed by one Primary Master, as shown in (8):

\[ B_{Masters} = \begin{cases} \frac{(m_p / s) - (2^K / m_p + 2^M / K_2)}{(2^K / s)} = 1 + \frac{m_p}{2^K / s}, & K_2 \neq 0 \\ \frac{(m_p / s) - (2^K / m_p)}{(2^K / s)} = 1 & K_2 = 0 \end{cases} \]

where \( s \) is the total number of Masters in system. \( M \) represents the hash value length of secure hash function SHA-1 used by Chord network. \( m_p \) is the total number of MapReduce jobs processed in our system. Both \( K_1 \) and \( K_2 \) are belong to the set of natural numbers \( N \).

From (8), we could find that if we use of the Job Backup Scheme with parameter \( K_2 \neq 0 \), \( B_{Masters} \) is bigger than that of \( K_2 = 0 \). It means that we could back up all the MapReduce jobs executed by one Primary Master to much more Backup Masters in the network based on the Job Backup Scheme having parameter \( K_2 \neq 0 \), having a benefit that the load of each Backup Master is approximately equal. Moreover, if one Master fails, we could achieve the goal that parallel data transmission from much more Backup Master, improving the speed of recovery and avoiding network congestion.

VI. PERFORMANCE EVALUATION

In this section, we will carry out a set of simulations by using Peersim [20] which is improved by us; we add a MapReduce processing module into Peersim to evaluate the behavior of ChordMR system in terms of three aspects. The first is Success Rate, which represents the probability of success for a MapReduce job processed in ChordMR system with different failure rate of Master. The second one is Recovery Time, when a Master fails, we simulate the total recovery time of all jobs which act the failure Master as Primary Master or Backup Master. The last aspect is Execution Time with different number of Masters for processing three kinds of jobs respectively.

In our experiment, we simulate a 1024-Master ChordMR network and the job scheduling strategy is FIFO.

A. Success Rate

As mentioned earlier, one of the goals of our simulations is to compare the ChordMR system with the traditional centralized system in terms of fault tolerance. As Fig. 4 shows, we simulate the failure rate of Masters in our system ranges from 0.02 to 0.25 and \( K_2 = 2 \). We compare the Success Rate of a job executed by a Primary Master both in centralized system and ChordMR system with different value of parameter \( K_1 \) respectively. As expected, from the result of our simulation we can find out that the ChordMR system has a better fault tolerance compared to the central system, and the bigger the \( K_1 \) is, the higher the Success Rate will be achieved. Moreover, we can observe that the Success Rate of MapReduce job will be close to 1 when \( K_2 \geq 2 \). This result is also accordance with the analysis of (6) and (7) above. It indicates that, Job Backup scheme has a good performance on fault tolerance and improve the stability of ChordMR.

B. Recovery Time

Based on the Job Backup scheme in ChordMR, jobs executed in system can be restored if their Primary Masters failed. In order to check the influence of parameters \( K_1 \) and \( K_2 \) on Recovery Time, we fix the value of \( K_1 \) and \( K_2 \) in our experiment respectively in the scenario of one Primary Master fails. As show in Fig. 5.
and Fig. 6, we simulate the total number of jobs executed in our system ranges from 15000 to 128000. And we assume that the average transmission time of one job from Backup Master to the successor of failure Master is 1s. During the simulation, we also take into account the route query time consumption and assume the routing time of every hop is 0.005s.

It means that, the bigger the backups number of each job increasing, the average intervals of corresponding adjacent Backup Masters will decrease, that the route query consumption time will be decreased. Therefore, we can achieve the goal of reducing the overall recovery time.

Fig. 6 shows the Recovery Time of all the jobs executed by one failed Primary Master with $K_2=2$ and different value of $K_1$. We can find out that the bigger $K_1$ is, the less Recovery Time will be obtained accordingly. It means that the larger the number of back-ups is, the shorter the recovery time will be achieved. This is because with the back-ups number of each job increasing, the average intervals of corresponding adjacent Backup Masters will decrease, that the route query consumption time will be decreased. Therefore, we can achieve the goal of reducing the overall recovery time.

C. Execution Time

In our simulation, we assume that the Masters in our system processes jobs with single CPU. In order to evaluate the behavior of the system under different loads, we define three kinds of job types, A, B and C. Type A represents Compute-Intensive jobs, B represents Balanced jobs and C represents Allocation-Intensive jobs. As detailed in Table I, Job types are characterized by different processing time.

![Figure 5](image1.png) Recovery Time vs. Number of jobs with different K1

![Figure 6](image2.png) Recovery Time vs. number of jobs with different K2

![Figure 7](image3.png) Execution Time for processing jobs vs. number of Master

To comparing the ChordMR system with traditional centralized MapReduce system, we use several scenarios characterized by different number of Masters varying from 1 to 100. Meanwhile, we use different job types with A, B and C for our simulation and the number of jobs in system is 500. Compared with the central MapReduce system (the number of Master is 1) from Fig. 7, we can figure out that the system with multiple Masters have a better performance on job processing. In addition, we also find out our system has a better performance for processing A jobs compared with C jobs. What is more, no matter the jobs submitted to the system simulator belong either to type A, B or C, the Execution Time all remains at a low level based on our ChordMR system with multiple Masters. Therefore, we can figure out that ChordMR has a better performance in processing MapReduce jobs than traditional system.

VII. CONCLUSION AND FUTURE WORK

In this Paper, we put forward ChordMR system, which exploits a peer-to-peer model to manage node churn, Master failures, and job recovery. In order to prevent the jobs loss caused by Master failure and achieve the goal of load balancing, we propose the job management scheme, which could help us to reduce the rate of Master failure...
and improve the processing efficiency of MapReduce jobs. In the meantime, the theoretical and simulation results show that, differently from centralized Master-Slave system, the ChordMR system does not suffer from job failures even in the presence of very high node failure rates, and have a good performance on job processing and recovery.

However, in our experiment, we do not take account into the impact of network congestion and request packet loss. So in our future work, we will add these factors to our experiment, aims at achieving a more accurate result. In addition, we will apply ChordMR to spatial query algorithm, such as dynamic range query and continuous k-nearest neighbor query, to improve the query efficiency and accuracy.

ACKNOWLEDGMENTS

This research is supported by Open Research Fund from Key Laboratory of Computer Network and Information Integration in Southeast University, Ministry of Education, China, under Grant No. K93-9-2010-06; National Natural Science Foundation of China under Grants Nos. 61373139, 40801149, 60903181; Natural Science Foundation of Jiangsu Province under Grants No. BK2012833; Natural Science Foundation of Jiangsu Higher Education under Grants No. 12KJB520011; Scientific Research Foundation of Nanjing University of Posts and Telecommunications under Grant No. NY211113.

REFERENCE


Jiagao Wu was born in Nanjing, China in October 1969. He received the B. S., M. S. in Physics from Nanjing University in 1992, 1995 and Ph. D degrees in Computer Science from the Southeast University in 2006, respectively.

He has worked as an associate professor in College of Computer, Nanjing University of Posts and Telecommunications. His research interest includes Cloud Computing, Mobile Computing and P2P Networks. Dr. Wu is a member of CCF and ACM. He has authored or co-authored over 20 papers in journals and conferences.

Hang Yuan was born in PingDingshan, Henan, China in April 1988. He received the B. S. in software engineering from Nanjing University of Posts and Telecommunications in 2011. Now he is reading the M. S. degree in Technology of Computer Application in NUPT. His research interest includes Cloud Computing and P2P Networks.

Ying He was born in Huai'an, Jiangsu, China in July 1988. She received the B. S. degree in Electrical Engineering and Automation from Nanjing University of Posts and Telecommunications in 2011, and she is reading the M. S. degree in Technology of Computer Application in NUPT. Her research interest includes Mobile Computing and P2P Networks.

© 2014 ACADEMY PUBLISHER
Zou Zhiqiang, was born in Nanjing, China in October 1967. He received his Ph.D. degree from the Graduate School of Chinese Academy of Sciences in 2007. He has been worked as an associate professor in College of Computer, Nanjing University of Posts and Telecommunications. His research interest includes networks architecture, distributed heterogeneous spatial data index and sensor networks. Dr. Zou is a member of CCF and ACM. He has authored or co-authored over 20 papers in journals and conferences.