2011 International Conference on Computer Science and Information Technology (ICCSIT 2011)
IPCSIT vol. 51 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V51.49

A Distributed Storage Access System for Mass Data using 3-tier Architecture

Lannan Luo[a], Gang Sun[a], Wei Yu[b]

[a]Univrtsity of Electronic Science and Technology of China,Chengdu,611731,P.R. China
[b]Shanda Interactive Entertainment Limited Company, Shanghai, 201203, P.R. China

Abstract. For efficiently handling mass user data, we incorporate key technologies including data segmentation, server clustering, reading and writing independency, load balancing, and Coroutine concurrent to propose a Distributed Storage Access System (DSAS) with a 3-tier architecture. We give the detailed description of implementation of DSAS and simulation results to show that our approach has high performance on user data handling.

Keywords: distributed storage access; data segmentation; reading and writing independency; consistent hash algorithm; load balancing

1. Introduction

As Internet is widely available, more and more enterprises are facing an important issue that how to construct a high-performance, high-availability, manageable and secure storage access systems.

Since the beginning of 21st century, there have emerged many storage access systems using peer-to-peer mechanism [1], such as PAST [2] storage system, OceanStore [3] storage system. But these systems have some disadvantages, for example, the download flow can occupy vast bandwidth resources, which will affect certain customers to use normal business, such as web service, email and video.

In order to conquer these problems, Reference [1] provides an architecture of a distributed storage system named LandHouse. This architecture consists of three parts: a manager, storage nodes and accessing clients. In [1], the authors proposed a simple version of the realization of storage nodes, in which storage nodes are separated into two classes including the deployed storage servers and client computers with sufficient storage. Fig. 1 shows the architecture of LandHouse storage system [1].

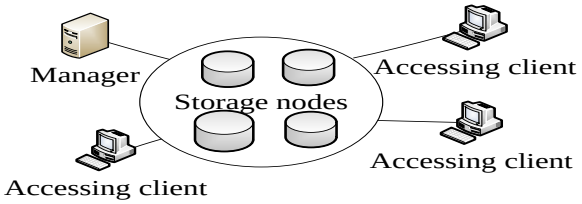

Fig. 1. Architecture of LandHouse storage system

However, this architecture has some shortcomings, such as: (1) The structure is not clear. (2) In reality, the proportion of reading operations to writing operations is unequal in the Internet, and the number of reading operations is more than the number of writing operations, so the efficiency of reading and writing is not high. Therefore, we propose an improved distributed access storage system with a 3-tier architecture.

* Lannan Luo. Tel.: +86-18908178176.
E-mail address: luo_lannan@126.com.

The remainder of this paper is organized as follows. Section II describes the architecture of the 3-tier Distributed Storage Access System (DSAS). Section III describes the key techniques employed in DSAS. The implementation of the system is discussed in Section IV. In Section V we demonstrate the results of experiments. In Section Ⅵ we conclude the paper.

2. Architecture

According to the actual needs, our goal is to achieve a high-performance data access, and the servers can respond the requests fleetly, i.e. the average response time is less than 100ms, and the shortest response time is less than 10ms.

The proposed DSAS is a clustering system and its internal structure is divided into three tiers showed in Fig. 2.

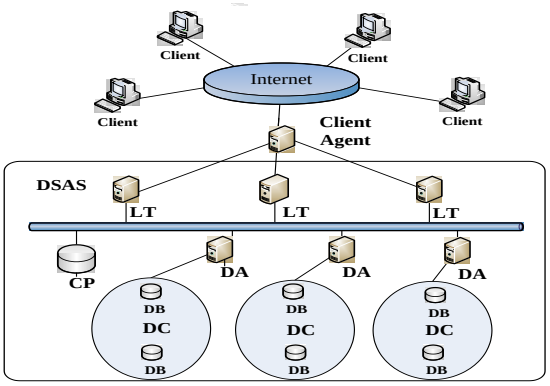

Fig. 2. The structure of DSAS

Figure 2 shows the structure of DSAS, which has three tiers and four modules. We name the four modules as Logical Transaction (LT), Cache Pool (CP), DB Agent (DA), and DB Cluster (DC). The first tier is LT, the second tier consists of two parts: CP and DA, and the third tier is DC. LT is the central manager. It can assemble SQL statements, search CP or DA by using data segmentation technology. CP is used to improve the reading rate and reduce the pressure on DC. DA takes charge of shielding details of DC, separating the reading and writing, and has the function of positioning the storage information and controlling the flow. Finally, DC is a cluster composed of several databases (DB), which store large amounts of data. The numbers of each module in the structure and the numbers of DB in each DC are variable according to the current situation.

Client Agent (CA) in Fig. 2 is responsible for client access, and transforms client requests into the package format defined by DSAS, and distributes requests to DSAS. This paper mainly focused on the structure and implementation of DSAS, so in Section IV we will only mention the package format of DSAS briefly.

3. Key Technologies

To design and implement DSAS technologies have been adopted. The main technologies include data segmentation technology, server cluster strategy, reading and writing independency strategy, load balancing strategy, and Coroutines concurrent mode. These technologies employed in DSAS are introduced as follows.

3.1. Data segmentation

At present, most mass data storage access systems designed by Internet service providers have adopted data segmentation technology. Through operating for many years prove these systems possess the operation efficiency and production effectiveness.

Data segmentation is mainly used in two aspects: in LT module to determine DA's position and in DC to store data dispersedly in different DB.

The main purpose of data segmentation is to alleviate a single DB's load and enhance the overall response rate. The key issue is how to choose an appropriate routing policy. Data segmentation has two basic scenarios: vertical segmentation and horizontal segmentation. Vertical segmentation puts different tables into different DB. But its shortcoming is if one table is under prodigious demand, the imbalance of DA' and DC' load will occur. Horizontal segmentation is another kind of scenarios that slice data via table rows, in other words, it puts some lines of a table into a DB and stores other lines in another DB. Of course, to implement this requires certain data segmentation rules. With horizontal segmentation the imbalance of DA's and DC's load will disappear. However the segmentation rules are complex. According to the situation that the access frequencies of active users and inactive users are different, so we choose horizontal segmentation method.

There are three main data segmentation rules as follows:

- Access DB by using the primary key. For example, if user_id is the primary key of a table, we can put the data corresponds to the value of user_id between 1 and 10000 into DB-1, and store the data corresponds to the value of user_id between 10001and 20000 in DB-2, and so on. But the weakness is that data distribution is not uniform.
- Hash mode. This is to use hash algorithm on certain field of a table, and use the same algorithm on certain segment of requests. By using this algorithm, it will return a result, and according to the result, LT decides to distribute the requests into certain DA.
- Keep a DB configuration in LT. Namely establish another table. This table keeps the mapping relationship between the needed data and the relevant DB. In this case, each time you should inquire the table first to obtain specific information of DB, and then can you go along the operation you demand.

These three ways are the general ways chosen in existing researches. In DSAS we take hash mode and also keep a configuration about DA instead of DB in LT, and the algorithm used in this mode is Consistent hash, since it can avoid changing the hash value if the numbers of server cluster increase or decrease. Its basic idea is to use the same hash function on objects and server clusters.

The specific algorithm is shown in Fig. 3. The key in Fig.3 is the value used by hash function; it should be sent together with the incoming data from client. The dbroute is used to find the information of DA from a configuration in LT.

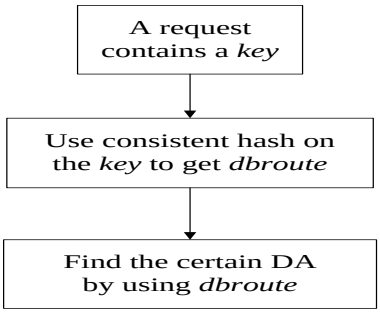

Fig. 3. Algorithm flowchart of searching DB and table

3.2. Server cluster strategy

In this structure, we use server cluster strategy in two cases: the Whole System and the DC. The structure of the later is shown in Fig. 4, and the structure of the Whole System will be discussed in Section IV.

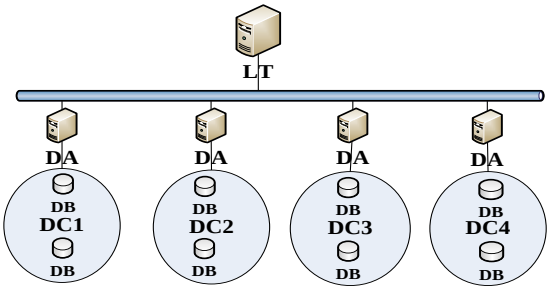

Fig. 4. Schematic of cluster

Fig. 4 describes the structure of DC. Here we introduce four clusters, DC1, DC2, DC3 and DC4. Of course, the number of clusters can be changed just based on the configuration in LT. These four clusters are the result of using horizontal segmentation on database. Certainly, these four clusters compose a database containing a complete data. Each cluster has a master and several slaves, and the data in the master is as same as the slaves.

The process of one query without adopting cluster is as follows: a request is send to CA, and CA distributes the request to LT after transforming it into the package format defined by DSAS. Then LT searches the needed data in CP first. If CP doesn't have the needed data, LT will send the request to a specific DA via data segmentation technology. Finally, the specific DA will search the data in DB.

But how will it be if cluster is introduced? From the graph, we know that we can only route the request to a virtual cluster, which isn't a certain physical server. Therefore, what we will do next is to find a specific physical DB.

The function of DA is to determine a specific DB's position, and distribute requests to the DB. Therefore, it also realizes load balancing. In addition, the implementation of DA we will discuss in Section IV.

3.3. Reading and writing independency strategy

After introducing server cluster strategy, we can see that a cluster consists of one master and several slaves for realizing reading and writing independency. The responsibility of master and slaves are writing operation and reading operation, respectively. Consequently, we can enhance the efficiency of reading and writing through the strategy.

By using MySQL Proxy, we can realize reading and writing independency strategy easily. We can see it in Fig. 5 briefly.

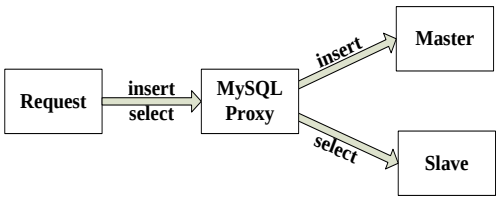

Fig. 5. Working schematic of MySQL Proxy

We realize the algorithm of DA based on the idea of MySQL Proxy. This will be discussed in section IV.

But it will bring about a new problem that how to keep the data in master as same as slaves. It can also be solved by MySQL Proxy mechanism. A simple solution is that slaves send requests to master to get new data on their own initiative. Other more sophisticated solutions will not be discussed detailedly here since it is beyond the scope of this work.

3.4. Load balancing strategy

Load balancing strategy includes random load balancing and weighted load balancing. And the strategy is used in DC to find a certain slave. Random load balancing can be comprehended easily. A slave will be selected randomly. Such way doesn't consider machines' performance, while each machine's physical properties and configurations are different in application. Therefore, this method is not perfect.

Thus, the weighted load balancing is adopted in our system. We assign each slave a weight through certain interface in internal system. Then, distribute load to slaves according to the ratio of weights in the cluster when they are running. Of course the introduction of the concept will increase the system complexity.

3.5. Concurrent mode

In current languages, multithreading and event-driven programming are both adopted as a standard for concurrent programming, however, these two methods have some shortages mentioned in [7]. In order to overcome these shortages, we take an effective mode—coroutines, which give an significant performance benefits when compared to multithreading and event-driven programming, and are widely used in many fields, such as in Sensor Networks [4, 5] and in Genetics [6]. Contrary to multithreading, each task will not

fully occupy a coroutine, but will release the coroutine when needs I/O waiting or other waiting actions, so that other tasks can occupy the coroutine. The benefit is that you can create fewer processes or threads compare to multithreading. Moreover, Coroutines are self-switching in user space instead of kernel space, so the costs will be lower than multithreading and event-driven programming. Coroutines are a powerful control abstraction, and can be easily implemented and comprehended, especially in the domain of procedural languages.

First we introduce the basic operators of coroutines [11].

- Create a coroutine. If you want to create a new coroutine, you can transfer the function: $co = coroutine.create(fun)$. This function receives only one procedural argument, which corresponds to a function that represents the main body of the coroutine, and returns a reference to the created coroutine.
- Activate a coroutine. The operator resume can (re)activate a coroutine. It receives a coroutine reference as its first argument. If a coroutine is suspended, it will start executing at its saved continuation point and run until it suspends or its main function terminates, and the function will return true. If a coroutine is already dead or meets some mistakes, this function will return false and the error message will be sent as accessory. In other cases, if it's the first time a coroutine is activated, an argument to the coroutine main function will pass to the second argument given to the operator resume.
- Suspend a coroutine. The operator yield suspends a coroutine execution. When transferred, the coroutine's continuation point is saved so that its execution will continue from the exact point where it suspended when the next time the coroutine is resumed. The capability of keeping state between successive calls constitutes the generally and commonly adopted description of a coroutine construct.

Through these, we can see that coroutines behave like routines, in the sense that control is always transferred back to their invokers. Moreover, coroutines, which are simpler and arguably less error-prone compare to the considerable complexity and overhead brought by multithreading and event-driven programming, allow the development of more structured programs.

In DSAS, the concurrent model of each module are adopted Coroutines, which is suitable for distributed system, to deal with high concurrent access. Each module has a scheduler responsible for creating, resuming, and yielding coroutines, to finish all kinds of operations. The flow chart of coroutines is shown in Fig. 6.

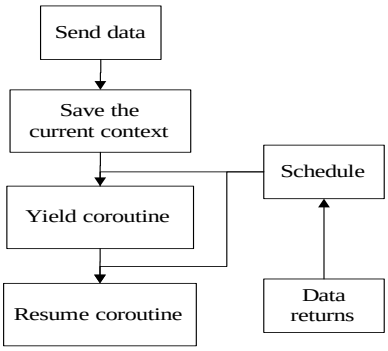

Fig. 6. Working schematic of Coroutines

We explain the coroutines execution briefly: when CA receives a request from client, it will spawn a coroutine, namely it will create a coroutine to deal with the current request. As we have mentioned above, the consumption of switching coroutines is tiny; also creating coroutines consumes few recourse. It only establishes a new stack to store the new coroutine variable. After managing the request, the coroutine sends data to LT. At the same time, it saves the current context, yields and puts itself into a suspend queue and waits for data returns. Meanwhile, it hands off the control to scheduler, and scheduler resumes a coroutine in the waiting queue. When the data from back-end comes back, the scheduler is triggered and put this coroutine into waiting queue for resuming next time.

4. System implementation

As we have described primarily, DSAS is a three-tier system composed of four models including Logical Transaction (LT), Cache Pool (CP), DB Agent (DA), and DB Cluster (DC). The first tier is LT, the second tier consists of two parts: CP and DA, and the third tier is DC.

4.1. Package format

When CA receives a request, it will transform the requests into the package format defined by the system, and then distributes requests to DSAS. And the package format is shown in Fig. 7.

package size	key	key value	cmd	cmd value	param1	…	paramN

Fig. 7. Package format of system

4.2. Logical Transaction

This module can access CP to enhance the response performance, and can route data to specific DA through data segmentation technology, shielding the logical details of internal access. And LT includes the following modules: CP access interface, DA access interface, protocol converter, timer manager and network interface. And the connection of each LT's module is shown in Fig. 8.

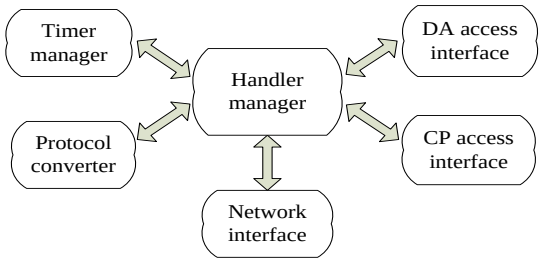

Fig. 8. Modules of LT

4.3. Database Agent

This module can access DC to improve server usability, keep connection with DC through the connection pool, and access DC asynchronously. DA supports MySQL.

The modules of DA include: connection pool control module, memory space control module, DC access interface. The flowchart of DA’s process is shown in Fig. 9.

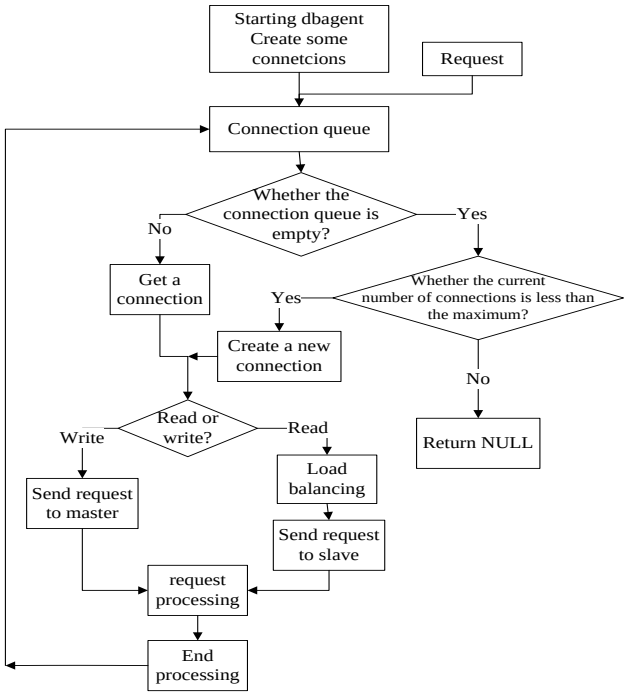

Fig.9. Flowchart of DA’s process

4.4. Cache Pool

This module is responsible for enhancing the reading and writing speed to reduce the pressure on DC. Cache access language includes read, write, delete, clear, and reset size of cache, etc.

Currently, there are many cache models, i.e. memory cache, compression cache, layered cache, etc. Memory cache refers to the cache using memory as storage. Compression cache uses compression storage Layered cache adopts mirror mode to enhance the usability of cache. And here, we take memory cache model.

This module is divided into three function parts cursorily: configuration management module, network communication module, memory cache module.

In the internal realization of CP, we use linked list and hash algorithm. Linked list is a data structure, realized by stringing the objects together. Therefore, if clients want to modify, delete, or search certain object, the object will be searched in the list firstly. Here, we use LRU list. The flowchart of algorithm is shown in Fig. 10.

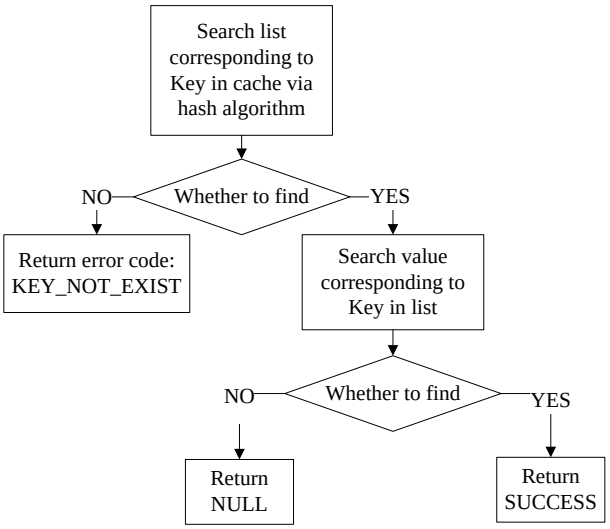

Fig. 10. Flowchart of cache mechanism

5. Simulation

We have to complete the simulation before deploying the system online. The performance of mass storage access system is characterized by measures such as system response time and servers throughput. Because we only need to test the performance of DSAS, so we use simulation clients to send request to DSAS directly in the testing below.

5.1. Function testing

The results of system response time are shown in Table 1.

Table 1. The response time

Events	Average response time		
	Without CP, data returned from DC (ms)	*With CP, cache doesn't have needed data (ms)*	*With CP, data returned from DC (ms)*
Select	14.59	11.55	3.86
Delete	44.92	64.40	
Update	50.90	57.09	
Insert	46.14	50.49	

Simulation results show that this system can perform simple SQL statements, and returns correct results. The response time is very short. The average response time is less than 100ms, and the shortest response time is 3.86ms which is less than 10ms.

5.2. Pressure testing

Because of the limiting numbers of concurrent connections in MySQL, therefore the needed data of simulation clients will be obtained from CP in this set of tests.

We perform the testing in two machines with the configurations and responsibilities as follows:

- Both machines with Linux operating system, 8 CPUs, 16 GB memory and 600GB hard disk.
- Testing Machine 1 is responsible for deploying one LT which occupies one CPU, and the simulation clients are deployed in testing machine 1, too.
- Testing Machines 2 is responsible for deploying one CP which also occupies one CPU.

In the pressure testing, there are a number of simulation clients send "SELECT" requests to LT, and one of the requests is "SELECT * FROM TABLE1 WHERE id=1". Then LT will search the data in CP. And we have known that the delay that a machine connects itself is 0.007ms, and the delay that a machine connects other machines is about 0.08ms. Therefore, through theoretical analysis, the throughput of DSAS can reach about 11,494 without considering the time spent in the processing of nodes.

The results of server throughput in pressure testing are showed in Table 2.

Table 2. System performance

Concurrent connection numbers	*Concurrent access numbers*	*Concurrent total packages*	*Time-consuming (s)*	*Throughput (packages/s)*
0	100	100*1000	9	11111
100,000	100	100*1000	14	7142
0	1000	1000*1000	101	9900
100,000	1000	1000*1000	125	8000
0	2000	2000*1000	203	9852
100,000	2000	2000*1000	252	7936
100,000	3000	3000*1000	410	7317

In Table 2, *Concurrent connection numbers* are the numbers of simulation clients which only connect LT without sending any requests. *Concurrent access numbers* are the numbers of simulation clients which also connect LT, and each client send 1000 requests in all constantly. *Concurrent total packages* are the numbers of the total requests send by access clients, obtained by multiplying *concurrent access numbers* and 1000. *Time-consuming* is the time that DSAS takes on dealing with all the *concurrent total packages*, starting with the first package arrives and ending by the last package is processed. Finally, the throughput is obtained by the following equation.

$$Throughput = \frac{Concurrent \quad total \quad packages}{Time \quad consuming} \tag{1}$$

Testing results show that this system can provide stable and reliable storage service. And in the case that only one LT and CP are deployed, when concurrent connection numbers is 100,000 per second, the throughput will be about 8000 per second with increasing number of concurrent access numbers, and when concurrent connection numbers is zero, the throughput will be about 9000 per second with increasing number of concurrent access numbers. Under extreme pressure circumstance, the various performance of the system can satisfy the basic requirements.

The limitation of the proposed design is the Memory size. The consumption of Memory will be increased with more and more requests. And when the number of requests reaches a certain amout, the Memory is not enough for further requests, so in that case, the system will refuse the requests.

Because DSAS is a distributed system, so if we deploy more nodes in network, the throughput of the system will increase. For example, if we deploy four LT, the throughput will reach more than 20,000 packages per second, and this we have tested. Through pressure testing, we can see that the performance on user data handling of DSAS is efficient.

6. Conclusion

This paper proposes a three-tier structure of storage access system with four models named LT, DA, CP and DC. In the structure, the first tier is LT, the second tier consists of two parts: CP and DA, and the third tier is DC. Key technologies including data segmentation technology, server cluster strategy, reading and writing independency strategy, load balancing strategy and Coroutines concurrent mode are used in the system. At the same time the system introduces cache mechanism to enhance server response speed. Testing results show that our approach for implementing manageable mass storage access system has high-performance, high-usability and high-handling ability. Now, this system has been applied in a storage access system of SNDA Company, and the system has been online successfully for more than 5 months, and its condition is good. The number of users is nearly 30 million, and is expected to reach 50 million by the end of 2011. According to the current conditions of system payload and the results of performance testing, it can operate at the end of 2011 without hardware update.

7. Acknowledgements

This work was supported by Open Platform Department of Shanda Interactive Entertainment Limited Company.

8. References

[1] J. Wang, L. Wang, L. Song. LandHouse: A Network Storage Structure Combining Regular Graph with Distributed Hash Table and Its Data Accessing Methods. *International Symposium on Intelligent Information Technology Application* 2009, 360–365.

[2] P. Druschel, A. Rowstro. PAST: a large-scale, persistent peer-to-peer stroage utility. *Hot Topics in Operating Systems* 2001, Schoss Elmau.

[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, et al. OceanStore: An Architecture for Global-Scale Persistent Storage. *International Conference on Architectural Support for Programming Languages and Operating Systemss* 2000, 190–201.

[4] Marcelo Cohen, Thiago Ponte, Silvana Rossetto, Noemi Rodriguez. Using Coroutines for RPC in Sensor Networks. *Parallel and Distributed Processing Symposium* 2007. 1-8.

[5] Slivana Rossetto, Noemi Rodriguez. A cooperative multitasking model for network sensors. *Distributed Computing Systems Workshops* 2006. 91-96.

[6] Sidney R.Maxwell III. Experiments with a coroutine execution model for genetic programming. *IEEE World Congress on Computational Intelligence* 1994. 413-417.

[7] H. Li, Y. Peng, X. Lu. A Programming Pattern for Distributed Systems. *Computer Engineering and Science* 2008. 142-145. (in Chinese).

[8] X. Chen, J. Warren, F. Hua, X. He. Characterizing the Dependability of Distributed Storage Systems Using a Two-layer Hidden Markov Model-Based Approach. *International Conference on Networking, Architectuer, and Storage* 2010. 31-40.

[9] T. Cai, S. Ju, D. Niu. Two-layered Access Conctrol for Storage Area Network. *International Conference on Grid and Cooperative Computing* 2009. 331-336.

[10] B. Yan, D. Qian, Y. Huang. Data Currency in Replicated Distributed Storage System. *International Conference on Grid and Cooperative Computing* 2009. 57-63.

[11] D. Moura, A. Lucia, R. Ierusalimschy. Revisiting Coroutine. *ACM Transactions on Programming Languages and Systems* 2009. Vol. 31(2).