

# A Scalable Byzantine Fault Tolerant Service in Grid System

Wang Xiuqun, Hou Honglun, Zhuang Yueting

College of Computer Science

Zhejiang University, Hangzhou, PR. China

xiuqunwang@zju.edu.cn [houl@cs.zju.edu.cn](mailto:houl@cs.zju.edu.cn) yzhuang@cs.zju.edu.cn

**Abstract** - *This paper describes the design, implementation and usage of a secure scalable Byzantine fault tolerant MDS system in the Grid. The scalable Byzantine fault tolerant MDS system provides a hierarchy GIIS servers, a local GIIS domain can require the resource it needs from remote GIIS domain. By using the state-machine replication approach and quorum system technique, the scalable Byzantine fault tolerant MDS system can tolerate not only benign faults, but also arbitrary (Byzantine) ones. An optimizing key management and caching mechanism is also explored in this paper.*

**Keywords:** Byzantine fault tolerance, Quorum system, Grid, MDS, LDAP

## 1 Introduction

Byzantine(arbitrary) fault tolerance[1,2,11] is a major problem in distributed computing system. A Byzantine fault tolerant system is required to keep operating regardless of not only failures of individual component but also arbitrary faults such as system component failure, application software errors and malicious attacks. The Byzantine fault issue is discussed pervasively in theoretic model, but seldom in real system. This paper, we will discuss the problem of a Byzantine fault tolerance in Grid system.

Fault tolerance in Grid is mainly involved in Meta Directory Service (MDS). The HBM (heartbeat monitor) technique[3] which is used to detect, handle and recover faults in MDS, can only deal with benign faults, such as, server crash, network crash and fail-stop attacks. But how can the Byzantine faults do, and how to tolerate such kind of faults in Grid is not considered in the literature.

We will design a scalable Byzantine fault tolerant MDS(SBFTMDS) system in this paper. This SBFTMDS system can tolerate Byzantine faults in the hierarchy architecture and provide the same interface as before. Meanwhile, it has good performance with 29.53% faster(about search) to 18.09%(about add) slow than formal operations. With the Castro-Liskov's Byzantine fault tolerant (CLBFT)[4-7] algorithms which uses the state-machine replication approach[8] and quorum system[9], our SBFTMD system can be built. A Scheme to Support Replicated Clients with Reduced Number of Messages and

a secure cache mechanism is also provided in our SBFTMDS system.

In section 2, the techniques to tolerate faults in MDS and its disadvantages in security are discussed. In section 3, the Byzantine fault model and the assumptions for is given. In section 4, the architecture of SBFTMDS, the key management and the caching mechanism is designed. The implementation and performance of the system is given in section 5. And we end this paper with a conclusion remark at the last section.

## 2 Organization and Security issues of MDS

There are many systems to implement the architecture of the Grid. The famous one is Globus. The Globus project [10] is in fact the criterion operating system of the Grid. We will discuss the secure issues of the Globus project in this section.

Globus consists of four parts, including GSI (Grid Security Infrastructure), MDS (Monitoring and Discovery System), GRAM (Resource Allocation Manager) and Data Management. GSI enables secure authentication and communication of all the system components over an open network. MDS is the information service component of the Globus toolkit, and it provides information about the available resource on the Grid and their status. GRAM provides a single interface for requesting and using system resources for the execution of "job". Data Management is in charge of data transfer in the system.

When a request from a client is sent to the server, MDS shows the available resource and GRAM do the "job" using this resource. The security of MDS is the key issue in Globus. If the MDS is attacked, it will provide wrong resource information or provide nothing. In Globus, the GSI, HBM techniques and the functionality implementation such as LDAP promise the security of the MDS. We will discuss such security issue of the MDS in the following of this section.

MDS consists of two directory services, the Globus Resource Information Service(GRIS) and the Grid Index Information Service(GIIS). The GRIS is a machine-specific service which contains information regarding the machine

on which it is running. The information includes IP address, software available, system administrator, network connected to, etc. The GIIS is the area-centric MDS service. A GRIS registers its local resource information to a GIIS in its domain, and withdraws itself from the GIIS when it leaves. This mechanism can implement the system scalability dynamically.

When a client asks for a service, he sends a request to its area-centric GIIS server to see which registered resource can be used. If the domain GIIS has no resource that the client needs, he can interact with the other domain GIIS to share resources. If a client happens to know the port of GRIS with the exact resource, it can search the GRIS running on a particular machine. The architecture of MDS is configured as Fig.1.

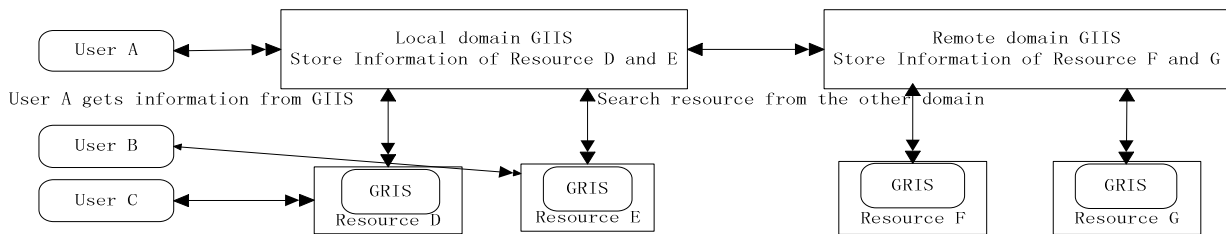


Fig.1 The Architecture of hierarchy MDS

All the communication between GIIS and GRIS, and the communication between Grid users and GRIS are protected by GSI. The GSI can protect the data integrity and tolerate the attacks from the Internet.

The GRIS is the local monitor which is responsible for observing the state of the registered process, and the GIIS is the data collector which receives HBM generated by local monitors and identifies faulty nodes [3]. When a failure or any change of local resource occurs, the GRIS on that machine will refresh its state, and send information to update the state of its domain's GIIS. This mechanism can not only monitors but also tolerates the fault of the local resource.

But how should we do about the GIIS service faults? A faulty GIIS service may provide non-correct information to the request.

In Globus, LDAP is used to implement GIIS and GRIS service. The resource information is stored, updated and searched by LDAP. The use of LDAP in GIIS is divided into three categories:

1. *Initialization*: using an information schema that describes the attributes associated with the different entry classes.
2. *Population*: a framework to populate the directory with information.
3. *Querying*: a framework to request information from the directory service.

The objects of LDAP can contain the attributes that are specifically useful for network computing environments. A set of simple UNIX scripts is available to populate the objects with information. Querying and updating information are easy while choosing the desired API for interfacing with LDAP server.

The security of GIIS depends on the security of LDAP. LDAP provides authentication mechanisms for a client to authenticate with a server, and provides authorization way for rich access control to protect the information the server contains. The method includes[12]:

- Simple bind: Client sends a DN and a password that

is stored in the userPassword attribute of that entry, and the password is sent in clear;

- TSL(SSL)[13] and SASL authentication[14,15]: Data integrity protection by means of the TSL(SSL) protocol or data-integrity SASL mechanisms;
- Authorization: Client authorization by means of access control list(ACL) based on the requestor's authenticated identity, including source IP address, encryption strength, the type of operation being requested, time of day, etc.

The authentication and authorization between the client and the server can protect information of servers from being attacked. The security mechanism can work well under the assumptions that the client and the connection between client and server can be faulty, but the servers are not corrupted themselves. If the server is attacked by hacker, and do Byzantine fault, such as the private key is lost or software error occurs, the current security mechanism can't work well. For example, the GIIS server may be corrupted itself, and offer faulty information to the requesting client. So we need a mechanism to tolerate Byzantine fault. A Byzantine fault tolerant GIIS system should be designed and implemented, which can promise to provide correct answer to every client's request, and the answer will finally arrive even in the presence of arbitrary fault.

### 3 The System Model and Assumption

This section, we will introduce a practical Byzantine fault tolerant algorithm, the CLBFT algorithm, and its system model and the assumptions.

*System model and the assumption*: The system consists of  $n$  tightly coupled replicas per server. It assumes an asynchronous distributed system where nodes are connected by an insecure network which may fail to deliver message, duplicate or deliver them out of order. The nodes of the system work independently, i.e., a faulty node does not necessarily cause other node's fault. The Byzantine fault nodes in the system may behave arbitrarily, but we limit the

ability of the faulty nodes to bounded computational power. The nodes act correctly is called as honest or non-faulty nodes, and the nodes depart from the protocol as faulty nodes.

*Algorithm:* We identify the  $n$  replicas with  $\{0,1,2,\dots,n-1\}$ . For simplicity, we assume  $n = 3f + 1$ , here  $f$  is the maximum number of replicas that may be faulty. To tolerate Byzantine faults, every step taken by a node is based on obtaining a certificate, which is a set of messages coming from different replicas. A certificate of size  $f + 1$  is sufficient to prove that the statement is correct and a certificate of size  $2f + 1$  ensures to convince other replicas of validity of the statement even when  $f$  replicas are faulty.

The system guarantees safety by using a primary-backup mechanism where replicas move through a succession of configurations called views. In a view  $v$ , the replica  $p$ , here  $p = v \bmod n$ , is designed as the primary and the others are backups. View changes are carried out to provide liveness by allowing the system to make progress when the current primary fails. The algorithm works roughly as follows:

- A client sends a request to the primary to invoke a service operation.
- The primary multicasts the request to the backups.
- After a three-phase agreement proceeding, the replicas execute the request and send a reply to the client. (Each replica maintains a copy of the service state and implements the service operation independently.
- The client waits for  $f + 1$  replies from different replicas with the same result: this is the result of the operation.

Fig.2 shows the three-phase agreement proceeding in the case of 4 replicas, in which replica 4 is faulty. Here, the *pre-prepare* and *prepare* phases are used to totally order requests sent in the same view even when the primary is faulty. The *prepare* and *commit* phases are used to ensure that requests are totally ordered across views.

All the messages between replicas in this three-phase protocol are authenticated by using MAC which makes the CLBFT system have good performance. In the CLBFT system, proactive recovery mechanism is used to make faulty replicas being correct again, that makes the system tolerate more than  $f$  faults over its lifetime. For more details about CLBFT algorithm are showed in[4-7].

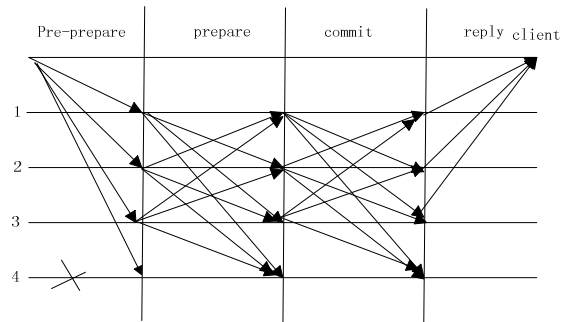


Fig. 2 Three-phase Agreement Proceeding

## 4 A Scalable Byzantine Fault Tolerant MDS Architecture

From section 2, we can know that, the GSI can protect the communication between MDS and outside, the HBM will tolerate the faults occurred by the GRIS, and Byzantine faults in GIIS will cause wrong service. In this section, we will design a scalable Byzantine fault tolerant MDS (SBFTMDS) system and explain how the GIIS can tolerate the Byzantine faults.

### 4.1 A MDS System with Byzantine Fault Tolerant Architecture

In order to tolerate Byzantine faults, there compose of  $3f + 1$  tightly coupled replicas per GIIS server in each domain, where  $f$  is the maximum number of faulty replicas the system can tolerate.

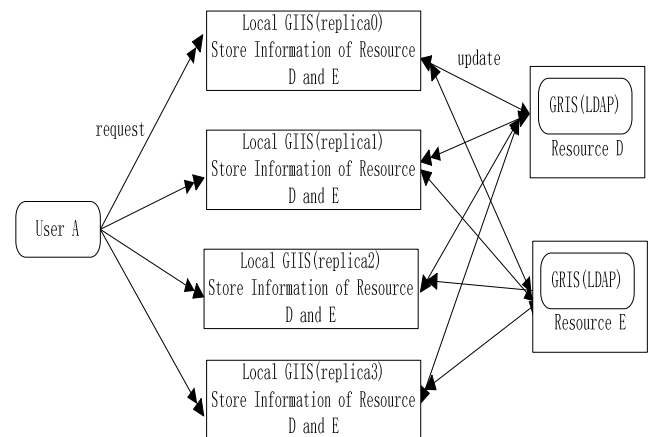


Fig.3 The Architecture of SBFTMDS

As figured in fig.3,  $3f + 1$  tightly coupled replicas compose the GIIS service side of SBFTMDS. A user requests the information it needed from the GIIS service of the SBFTMDS system, and the GRIS server of the local resource updates information to the replicas of the local GIIS service. The replicas of the GIIS service works like the CLBFT algorithm the section 3 figured.

Each replica maintains a copy of entries with identical initial state of the GIIS service and its own public-private key pair. The public key is stored in the replica's configure file, the private key is stored in a secure coprocessor attached to the replica and cannot be stolen. Each client (including the user and the GRIS) is statically configured with  $3f + 1$  public keys which belong to the  $3f + 1$  replicas respectively.

As we can see from the section 3, messages during communication are authenticated by using MAC, session key pairs must be set up between a client and a replica server, and between each of the replicas. By using their public keys they configured beforehand, session keys can be easily obtained. The client can refresh the key periodically, using the *new-key* message. If a client neglects to do so, a replica discards its current key for that client, which forces the client to refresh its key. Of course, the primary may be faulty, CLBFT uses *view-change* function to change the primary. An intruder needs to compromise more than  $f$  replicas to corrupt the GIIS service.

If the domain GIIS has no resource that the client needs, it can interact with the remote domain GIIS to share resources. However, how can the local GIIS interact with the remote GIIS server?

## 4.2 A Scalable MDS System with Byzantine Fault Tolerant

In our SBFTMDS system, requests are handled recursively through local and remote GIIS servers. A user makes a recursive query to the local GIIS. If the query cannot be dealt with locally, the local GIIS server makes a recursive query to a remote GIIS server, and so on, until the final result is found.

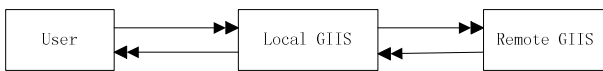


Fig.4 A Recursive query Scheme

As figured in Fig.4, The Byzantine-fault-tolerant local GIIS server, consisting of  $3f+1$  replicas, acts as a client to the remote GIIS server. If each replica in the local GIIS communicates with each replica in the remote GIIS as client, the number of messages will be large. Let us consider a system with two levels of Byzantine tolerant replicated server as illustrated in Fig.5, each server consists of four ( $3f+1=4$ ) replicas to tolerate up to one failure. Suppose a user wants to query resource, it sends out 4 messages to the 4 replicas in level 1. If the query could not be resolved by the level-1 replicas, each replica in level 1 can act as a Byzantine fault tolerant client and therefore independently sends requests to each of the level 2 replicas. In this way,

the number of messages increases geometrically with the number of replicas per set when the level of hierarchy increases. It is a heavy overhead on communication.

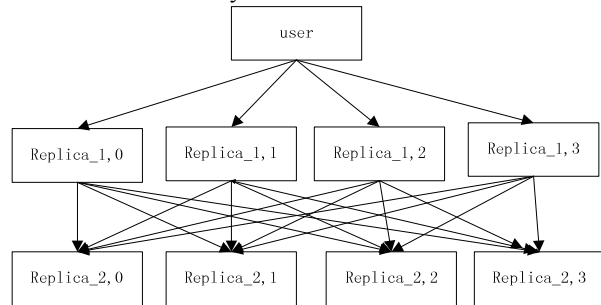


Fig.5 The Number of Messages Across Levels of Hierarchy

### 4.2.1 Key Management

In order to lighten the overhead, the primary of the client-group will perform the query on the behalf of other replicas. After getting the reply of the query from level 2, the primary sends the reply back to other replicas in level 1 as usual. As illustrated in Fig.6, the number of messages increases linearly with the number of replicas per set as the level of hierarchy increases.

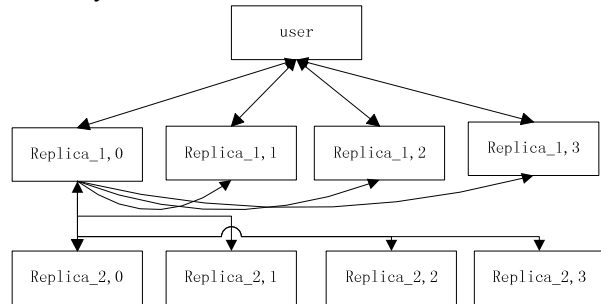


Fig.6 Scheme to Support Replicated Clients with Reduced Number of Messages

Of course, the primary in level 1 may be faulty. The replies sent by the level 2 of replicas are constructed in a special way, so that the replicas in level 1 can verify the integrity and authenticity of the reply independent of whoever performed the query on their behalf. In the MDS system, only the GRIS servers of local resources do update requests to the local GIIS server, so the requests between the local GIIS and the remote GIIS servers are only read-only queries. A read-only query does not modify the state of the system, it is not necessary for the level 2 replicas to verify that a read-only request came from at least  $f + 1$  replicas acting as clients. When the user sends out a read-only request to the level 1 GIIS replicas and the request can not be served at this level, the primary in level 1

authenticates the request with a special flag notifying the replicas in the level 2 that this request is on the behalf of a set of replicas. After replicas in level 2 receives the request, each replica deals with this request, authenticates the reply using the keys for each of the replica in level 1, and sends this reply back to the primary in level 1. When the primary in level 1 collects  $2f + 1$  correct replicas from the level 2, and then retransmit the replies to all backups in level 1. In this way, the system guarantees correctness to check against a malicious primary. If the primary is faulty, *view-change* mechanism can be used to change the primary, just like the technique described as section 3.

#### 4.2.2 Caching Mechanism

In order for the good performance, an efficient data caching system is needed. In general, caching exploits the idea of temporal locality since it is the case that an item referenced recently is likely to be referenced again in the near future. Caching the result of a query in the local GIID servers allows the servers to provide the answer to a later

reference of the same query immediately from its local cache without interacting with remote GIIS replicas.

Because a user always has a read-only query, the issue of cache consistency can be resolved by server-driven mechanism. When a data item changes, the server notifies clients that have cached copies. For the large-scale system like GIIS, information of resource data updates is rare, the server-driven invalidation mechanism can perform well.

## 5 Implementation of SBFTMDS System

In Globus, the GIIS encapsulates the interfaces provided by the LDAP and provides interface to application program. If we can provide the same LDAP interfaces which can tolerate Byzantine faults, then the same interface of the GIIS can be used. With such seamless interface, the application program needs no change and can tolerate Byzantine faults.

The CLBFT algorithm provides library. With the CLBFT replication library, the Byzantine fault tolerant LDAP system can be implemented. The CLBFT library has the client interfaces including *Byz\_int\_client* which initializes the client by using a configuration file, *byz\_invoke* which causes an operation to be executed

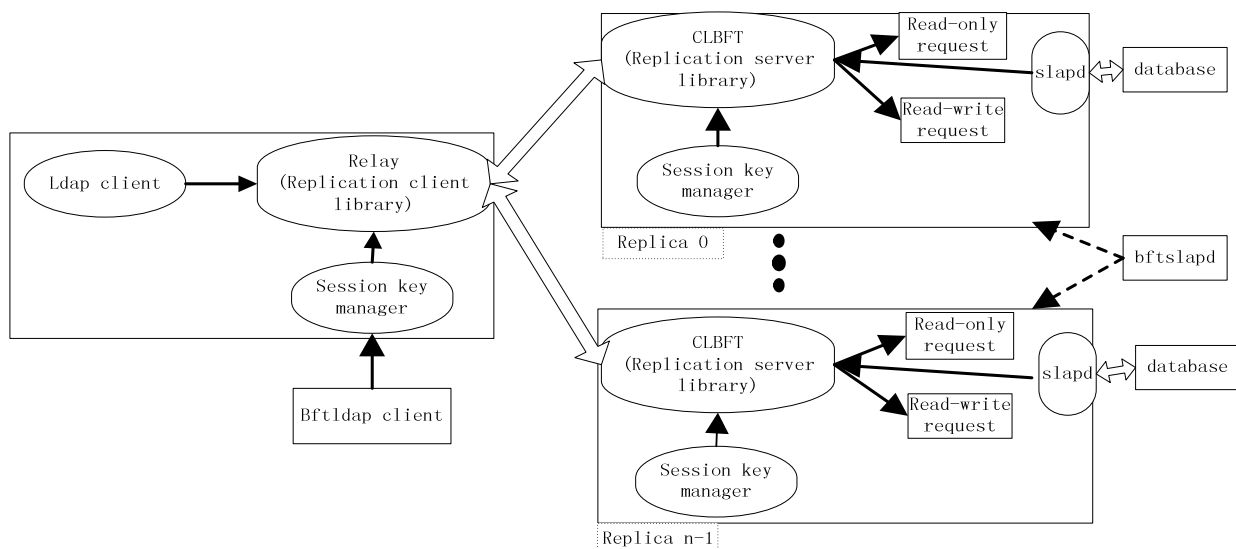


Fig.7 The Architecture of BFTLDAP

and carries out the client side of the protocol and returns the result when enough replicas have responded. The server-side interfaces have *Byz\_init\_replica()* with *conf*, *exec*, *get*, *put*, *restart* *shutdown*, procedures as arguments, and *Byz\_modify* to modify the service state.

As figured in Fig.7, a *relay* process is used to connect the standard LDAP client with the replication client library.

As the *relay* receives a LDAP request, it calls the *byz\_invoke* procedure of the replication library, which sends out the request to the replicas and returns the results. In the server side, an incoming request triggers the *exec* procedure in the replica, thereby executing a read-only or update request, depending upon the request type. Only the request which changes the state of the system needs three-phase agreement proceeding. When an updating request is coming, replicas do the three-phase protocol through replication library, and then deal with the updating

request and modify the system state. When a read-only request is coming, replicas deal with the request and return the reply to the client directly.

Being tolerating Byzantine fault, the BFTLDAP consists of a server daemon named `bftslapd`, a set of client-side utilities, and a set of library. Such interfaces of the BFTLDAP library are as same as those of the LDAP. In order to set up a SBFTMDS system, the server-side daemon of BFTLDAP has to be installed in the GIIS replicas, the utility of BFTLDAP is installed in clients and the public keys are configured in the configure files of clients and replicas. Because of these same interfaces of BFTLDAP library, the application program needs no change and can tolerate Byzantine faults.

## 6 Performance Evaluation

In this section, the performance evaluation of BFTMDS system is investigated. In fact, the performance of BFTMDS is the same as that of BFTLDAP. In this section, only the performance evaluation of BFTLDAP is given. We use `Openldap-2.1.22` as the LDAP service program, `Berkeley db-4.2.52` as the backend database, and `cyrus-sasl-2.1.20` as the Authentication software. We store the sasl password in the `sasldb`. The experiment runs on four Dell Precision 410 workstations with a single Pentium III processor, and 128 MB of memory. All machine ran `redhat7.3`, the process clock speed is 600MHz. The machines were connected by a 100Mbs/s switched Ethernet and had 3Com Sc905B interface cards. There are 10000 entries in the root DN, each entry has the same object and attributes. Two kinds of requests in LDAP are considered, one is Read-Only request, the other is Read-Write request. The Read-Only requests include `ldapsearch`, `ldapcompare`. The Read-Write requests includes `ldapmordn`, `ldapdelete`. Because a Read-Only request need not modify the state of the system, the BFTLDAP do optimization without verifying that a RO request came from at least  $f+1$  replicas acting as clients.

**Table.1 Performance of BFTLDAP query**

system	Ldapsearch seconds	Ldapadd seconds	Ldapdelete seconds	Ldapmordn seconds
LDAP/ simple	0.2381	0.00122	0.00118	0.00058
LDAP/ sasl	0.5855	0.00285	0.00267	0.00136
LDAP/ CLBFT	0.4126	0.00304	0.00293	0.00142

In our experiment, LDAP with password, LDAP with SASL authentication and LDALP with CLBFT are compared. The results show as the tables 1. We note that the experiment is set up on only one level system. From Table.1, we can see that the time of `ldapsearch` query in LDAP/CLBFT is 42.29% slower than that in LDAP/Simple, however, it is 29.53% faster than the time in LDAP/SASL. The time of `ldapadd`, `ldapdelete`, and `ldapmordn` queries in LDAP/CLBFT are 18.09%, 8.87% and 4.23% slower than those in LDAP/SASL respectively. Because the simple authentication is not recommended, only the time of

requests in LDAP/SASL and LDAP/CLBFT are compared. Through optimization, the read-only request in LDAP/CLBFT performs better than that in LDAP/SASL. The other requests perform well also. Because most of the requests of LDAP are search requests, and it can work well in the presence of Byzantine faults, BFTLDAP is practical and secure. So, the BFTMDS in Globus is practical and secure.

## 7 Conclusion and Future Work

In this paper, a scalable Byzantine-Fault-Tolerant MDS system is designed and implemented. It can not only tolerate malicious attacks on the client and the Internet, but also malicious attacks on server. It is practical and robust in the presence of Byzantine faults. With the growing pervasive use of Grid and the reliance of industry and government on online information, malicious attacks become more attractive and successful, and Byzantine fault tolerance is needed in grid applications.

## Acknowledgement

This work is supported by the National Natural Science Foundation of China (Grant No.60525108, No.60533090), 973 Program (No.2002CB312101), Science and Technology Project of Zhejiang Province (2005C13032, 2005C11001-05)

## 8 References

- [1] PEASE , R.SHOSTAK, AND L.LAMPORT "Reaching Agreement in the Presence of Faults", Journal of the Association for Computing Machinery.Vol 27, No 2, April 1980,pp 228-234
- [2] Leslie Lamport , Robert Shostak , and Marshall Pease "The Byzantine Generals Problem". ACM Transactions on Programming Languages and Systems . Vol.4, No.3, July 1982,Page382-401
- [3] Paul Stelling, Ian Foster, Carl Kesselman, Craig Lee, Gregor von Laszewski. "A fault detection service for wide area distributed computations". [citeseer.nj.nec.com/stelling98fault.html](http://citeseer.nj.nec.com/stelling98fault.html)
- [4] Miguel Castro,"Practical Byzantine Fault tolerant" PhD Thesis, Massachusetts Institute of Technology, November 2000
- [5] M.Castro, B.Liskov. "Authenticated Byzantine Fault Tolerance Without Public-key Cryptography" Technical Memo MIT/CS/TM-589, MIT Laboratory for Computer Science, June 1999

[6] M.Castro, B.Liskov. “Proactive Recovery in a Byzantine Tolerant System” In Proceeding of the Fifth Symposium on Operating Systems Design and Implementation(OSDI 2000), San Diego, USA, October 2000

[7] M.Castro, B.Liskov. “Practical Byzantine Fault Tolerance” In Proceeding of the Fourth Symposium on Operating Systems Design and Implementation(OSDI 2000), San Diego, USA, October 2000

[8] F.B.Schneider, “Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial”, ACM Computing Survey, Vol22, No.4, Dec. 1990, pp, 299-319

[9] Dahlia Malkhi, “Byzantine Quorum System”  
<http://citeseer.ist.psu.edu/malkhi98byzantine.html>

[10] I. Foster and C. Kesselman. “The Globus project: A progress report”. In Proceeding of the Heterogeneous Computing Workshop, 1998. to appear.

[11] Fischer , Nancy A.Lynch , S.Paterson“ Impossibility of Distributed Consensus with One Faulty Process ” . Journal of the Association for Computing Machinery , Vol.32, No.2, April 1985, pp.374-382

[12] Mark Wahl. “Authentication Methods for LDAP”, RFC2829

[13] Dierks, T.and C.Allen, “The TLS Protocol Version 1.0”, RFC 2246, January 1999

[14] Myers, J., “Simple Authentication and Security Layer(SASL)”, RFC2222, October 1997

[15] P.Leach, Innosoft. “Using Digest Authentication as a SASL Mechanism”, RFC2831