Available online at www.sciencedirect.com
ScienceDirect
Parallel Computing 34 (2008) 17–31
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Dynamic and scalable storage management architecture for Grid Oriented Storage devices

Yuhui Deng [a,*], Frank Wang [a], Na Helian [b], Sining Wu [c], Chenhan Liao [c]

[a] *Centre for Grid Computing, Cambridge-Cranfield High Performance Computing Facilities, Cranfield University Campus, Bedfordshire MK430AL, United Kingdom*
[b] *London Metropolitan University, United Kingdom*
[c] *Cranfield University Campus, Bedfordshire MK430AL, United Kingdom*

Received 21 January 2007; received in revised form 17 September 2007; accepted 16 October 2007
Available online 26 October 2007

Abstract

Most of currently deployed Grid systems employ hierarchical or centralized approaches to simplify system management. However, the approaches cannot satisfy the requirements of complex Grid applications which involve hundreds or thousands of geographically distributed nodes. This paper proposes a Dynamic and Scalable Storage Management (DSSM) architecture for Grid Oriented Storage (GOS) devices. Since large-scale data intensive applications frequently involve a high degree of data access locality, the DSSM divides GOS nodes into multiple geographically distributed domains to facilitate the locality and simplify the intra-domain storage management. Dynamic GOS agents selected from the domains are organized as a virtual agent domain in a Peer-to-Peer (P2P) manner to coordinate multiple domains. As only the domain agents participate in the inter-domain communication, system wide information dissemination can be done far more efficiently than flat flooding. Grid service based storage resources are adopted to stack simple modular service piece by piece as demand grows. The decentralized architecture of DSSM avoids the hierarchical or centralized approaches of traditional Grid architectures, eliminates large-scale flat flooding of unstructured P2P systems, and provides an interoperable, seamless, and infinite storage pool in a Grid environment. The DSSM architecture is validated by a proof-of-concept prototype system.

Keywords: Storage management; Data locality; Storage Grid; Peer-to-Peer; Grid service

1. Introduction

The explosive growth of data generated by information digitization has been identified as the key driver to escalate storage requirements (e.g. the data produced by protein folding, global earth system model and high energy physics is often measured in petabytes and even in exabyte). Network based storage systems such as Network Attached Storage (NAS) and Storage Area Network (SAN) offer a robust and easy method to

* Corresponding author. Tel.: +44 (0) 1234 750111x4773.
E-mail addresses: y.deng@cranfield.ac.uk, yuhuid@hotmail.com (Y. Deng).

0167-8191/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2007.10.003

control and access large amounts of storage. However, with the steady growth of client access demands and the required data sizes, it is a challenge to design an autonomous, dynamic, large-scale and scalable storage system which can consolidate distributed storage resources to satisfy both the bandwidth and storage capacity requirements [1].

Grid is a flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources [2]. The objective is to virtualize resources and allow users and applications to access resources in a transparent manner. Storage Grid is a new model for deploying and managing storage resources geographically distributed across multiple systems and networks, making efficient use of available storage capacity. The research community has been very active in the area of designing storage systems in the Grid environment. Storage Resource Broker (SRB) [3] supports shared collections that can be distributed across multiple organizations and heterogeneous storage systems. The SRB offers some Grid functionalities including a Virtual Organization (VO) structure for data and information, handling a multiplicity of platforms, resource and data types, and seamless authorization and authentication to data and information which are stored in distributed sites, etc. [4,5]. Distributed Parallel Storage System (DPSS) [6] provides high speed parallel access to remote data and the DPSS servers are transparent to applications. It is currently being integrated into Globus and the SRB. High Performance Storage System (HPSS) [7] provides hierarchical storage management and services for very large storage environments. Manabe et al. [8] discussed how to integrate the HPSS into a Grid architecture to handle petabytes of data produced by the Atlas experiment and to share such data among the regional collaborators. DataCutter is a middleware infrastructure that enables processing of large-scale scientific datasets stored in archival storage systems in a Grid environment [9,10]. However, many aspects of the deployed storage Grid systems are based on hierarchical or centralized approaches (e.g. the metadata organization of SRB and the storage management of HPSS) which may result in system bottlenecks.

A typical Grid environment (GT4) which is a VO consists of a Monitoring and Discovery System (MDS) [11], a Certificate Authentication (CA) centre and Grid service providers. All Grid services are registered in the MDS. The interaction between the Grid services and Grid users is mediated through the MDS. There is typically one MDS per VO, but in a large VO, several MDSs are normally organized in a hierarchy.

Grid Oriented Storage (GOS) is proposed to build a distributed and large-scale storage system to keep up with the ever increasing demands of bandwidth and capacity. GOS device is a thin server which can be directly connected to an existing Grid environment using a Globus XIO interface [12]. The server integrates Redundant Array of Independent Disks (RAID) subsystem to aggregate storage capacity, I/O performance and reliability based on data striping and distribution. A lot of applications can benefit from the GOS project, such as large-scale streaming media system, remote education, digital library and Internet Service Providers (ISPs), etc. Due to the increasing storage demands and system scale, hundreds of thousands of GOS devices may be involved to match the requirements imposed by all kinds of Grid applications. Because the GOS is based on an existing Grid environment, it is a challenge for the hierarchical architecture to manage the GOS devices involved in a large-scale and complex Grid application. It is important to decentralize their functionalities to avoid the bottlenecks of hierarchical or centralized approaches [13].

Peer-to-Peer (P2P) model offers a prospect of robustness, scalability and availability of a large pool of storage and computational resources. Recently, there have been some research efforts invested in designing P2P based large-scale and geographically distributed storage systems. PAST [14] is a large-scale P2P persistent storage utility. PAST is based on a self-organizing, Internet based overlay network of storage nodes that cooperatively route file queries, store multiple replicas of files, and cache additional copies of popular files. OceanStore [15] is a utility infrastructure designed to span the globe and provide continuous access to persistent information. The OceanStore is comprised of untrusted servers. Data is protected through redundancy and cryptographic techniques. Free Heaven [16] mainly focuses on distributed, reliable, and anonymous storage over efficient retrieval. The above efforts have made important strides in designing P2P based storage systems. However, the security, storage management, data consistency, etc. are all challenging problems to be solved. Fortunately, the P2P philosophy and techniques could be used to implement nonhierarchical and decentralized Grid systems. More recently, several research projects have investigated techniques for P2P Grid, which let us share resources (computers, databases, instruments, and so forth) distributed across multiple organizations. The P2P Grid is emerging as a promising platform for executing large-scale, resource intensive applications [17,18].

Based on our earlier research [12], this paper proposes a Dynamic and Scalable Storage Management (DSSM) architecture to organize GOS devices into a large-scale and geographically distributed storage system to meet the requirements imposed by all kinds of Grid applications. The DSSM divides GOS devices into multiple geographically distributed domains to facilitate the data access locality. The architecture consists of two levels. The bottom level adopts multicast to achieve dynamic, scalable, and self-organized physical domains. The method significantly simplifies the intra-domain storage resource management. The upper level is a virtual domain that consists of geographically distributed and dynamic GOS agents selected from each physical domain. P2P model is employed to discover the required storage resources. Due to the virtual two-tiered architecture, two-tiered parallelism and scalability of intra-domain and inter-domain are achieved. The DSSM removes the traditional hierarchy or centralized approaches of conventional Grid architecture, avoids large-scale flat flooding of unstructured P2P systems, and inherits the dynamic scalability of P2P naturally. Storage resources of GOS devices are wrapped into Grid services which are adopted as basic building blocks to stack an infinite storage pool.

The remainder of the paper is organized as follows. The DSSM architecture is introduced in Section 2. Section 3 describes the domain formation algorithm since the DSSM is based on physical GOS domains and a virtual GOS agent domain. Section 4 illustrates how to wrap storage resources into Grid services. The Grid service based storage resource discovery mechanism is depicted in Section 5. Section 6 illustrates the prototype system and measures the system performance. Section 7 concludes the paper with remarks on main contributions of the paper and indications of future research.

2. The DSSM architecture description

A large-scale and geographically distributed storage system may involve hundreds or even thousands of storage nodes to match the requirements imposed by all kinds of applications, which is challenging the storage resource management with the increasing of the system scale.

2.1. Data locality and domain of DSSM

Data locality is a measure of how well data can be selected, retrieved, compactly stored, and reused for subsequent accesses [19]. In general, there are two basic types of data locality: temporal and spatial. Temporal locality denotes that the data accessed at one point in time will be accessed in the near future. Temporal locality relies on the access pattern of different applications and can therefore change dynamically. Spatial locality defines that the probability of accessing data is higher if the data near it was just accessed (e.g. prefetch). Unlike the temporal locality, spatial locality is inherent in the data managed by a storage system, and is relatively more stable and does not depend on applications, but rather on data organizations which is closely related to the system architecture. Data locality is a property of both the access pattern of applications and data organization or system architectures. Reshaping access patterns can be employed to improve temporal locality [20]. Data reorganization is normally adopted to improve spatial locality.

Large-scale data intensive applications frequently involve a high degree of data access locality [19,21,22]. For example, the simulation data of earth's climate system is growing rapidly and going beyond 100 TB. The data has to be spread over several sites [21]. The data of Large Hadron Collider (LHC) is generated at one place (CERN), but the computational capacity required to analyze the data implies that the analysis must be performed at geographically distributed centres. It requires 10 petabytes of disk storage which has to be distributed across Europe, America and Asia [22]. Many research efforts have been invested in exploiting the impact of access pattern and data organization of applications on the data locality to achieve performance gains [19,23]. DSSM divides GOS devices into multiple domains to facilitate the locality and reduce unnecessary network traffic. Fig. 1 illustrates the DSSM architecture.

2.2. Criterion of domain division

Choosing a suitable criterion to form GOS nodes into domains is an important factor of the DSSM architecture. Jean-Baptiste Ernst-Desmulie et al. [24] proposed three criteria used for forming nodes into domains,

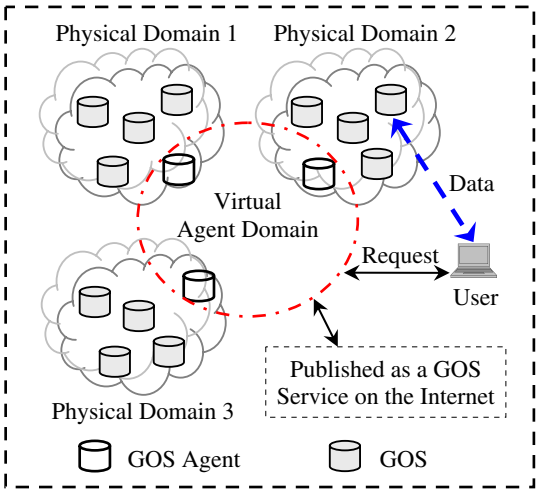

Fig. 1. Architecture of DSSM (GOS devices are organized into different domains to enhance data access locality in terms of geographical area. Multiple geographically distributed domains coordinate with each other through the virtual agent domain.).

namely distance, qualitative and quantitative. DSSM employs the distance criterion in a geographical way to divide GOS nodes into multiple domains to facilitate the data access locality and limit the amount of communication traffic seen by each node with performance guarantee. GOS devices belonging to the same geographical area (e.g. the same enterprise or the same LAN) are formed into a domain, because the GOS devices in the area are normally placed geographically close.

With a domain based GOS network, the scalability is achieved from a two-tiered architecture, namely, intra-domain and inter-domain, as shown in Fig. 1. Within a domain, each GOS device is equal in functions and capabilities, and can leave or join the domain dynamically. Any node can communicate with anyone else directly. Multicast is employed to achieve self-organizing and self-discovery features when the GOS devices leave or join the domains. Since DSSM is based on stable and trusted GOS devices that are less dynamic than pure P2P nodes, multicast does not result in much communication traffic. All GOS devices in a domain work collaboratively to construct a scalable storage system. Hence, the domain based DSSM architecture inherently facilitates data access locality. In addition, each domain selects a domain agent from the domain members to cooperate with other domains according to a domain formation algorithm. As only the domain agents who form a virtual agent domain participate in the inter-domain communication, system wide information dissemination can be done far more efficiently than flat flooding [25].

The DSSM can be regarded as a virtual two-tiered hierarchy, but it is different to the traditional hierarchical architecture. It replaces a few root nodes of the traditional hierarchy with many GOS agents which can be expanded to a large-scale, thus avoiding the single point of failure and the potential performance bottleneck. The DSSM can achieve near-infinite dynamic scalability due to the two-tiered architecture. The intra-domain scalability is obtained by expanding the system storage capacity incrementally with additional GOS devices along with associated network interfaces that expand the data transmission rate proportionally. Additional domain can join the DSSM to achieve the inter-domain scalability.

2.3. Survivability and security of DSSM

Survivability denotes the ability of a system to maintain or restore an acceptable level of performance during system component failures by applying various restoration techniques. Survivability is very important for the DSSM since the data in GOS device is invaluable. P2P architecture has strong survivability because it is a fully distributed system with no central coordination, and peers are autonomous, unreliable, and not trustworthy. The DSSM adopts a P2P manner to organize the stable and trusted GOS devices into a two-tiered domain architecture, it naturally inherits the strong survivability of P2P.

Many data redundancy mechanisms have been devised to protect the data loss in cluster storage systems [26]. Because the failure of GOS devices can directly incur data loss, we employ data redundancy to guarantee the survivability. In the event of one GOS device in a physical domain failing or leaving the domain, the information will be updated automatically and the workload will be redirected to other GOS nodes where the redundant data resides. If a domain agent fails, another agent candidate in the same domain will take over the role automatically. The DSSM achieves high survivability due to the virtual two-tiered domain architecture.

The DSSM is based on the existing Grid environment (GT4) which consists of Grid service providers, MDSs and a CA centre. The security of DSSM is within the Grid Security Infrastructure (GSI) [27] framework of GT4. Each GOS device has to request and receive a host certificate from the CA, thus allowing joining the DSSM. Non-trusted nodes are not permitted to join the DSSM. In order to access the storage resources of DSSM, a user has to request and receive a user certificate from the CA as well.

3. Domain formation algorithm of DSSM

The DSSM architecture consists of two levels. The bottom level adopts multicast to achieve dynamic, scalable, and self-organized physical domains. The upper level is a virtual domain that consists of geographically distributed and dynamic GOS agents selected from all domains.

The objective of the domain formation algorithm is to manage the geographically distributed GOS devices across the Internet effectively. A good domain formation algorithm should not change the domain configuration too drastically when a few nodes are leaving or joining. Otherwise, the domain agents cannot control their domains efficiently and thus lose their roles as local coordinators. The algorithm consists of knowledge of the neighbors of each node in a domain to organize the domain members and select a domain agent from all candidate members. Several distributed domain formation algorithms have been devised over the years. One is the lowest-ID algorithm [28]. The other is the highest-connectivity (degree) algorithm [29]. Gerla and Tsai [30] simulated the two algorithms. In this section, we will detail the domain formation algorithm of the DSSM at two levels, respectively.

3.1. The formation algorithm of physical domain at bottom level

All GOS devices in the physical domain at the bottom level are organized as a scalable storage system and assigned with a multicast IP address to provide a single entry point to the storage space. In order to avoid unnecessary network traffic to those GOS devices and switches where the request is not intended due to the multicast, any two GOS devices are at most one hop away. The formation algorithm of physical domain allows GOS devices to join or leave the running GOS network automatically. All GOS devices in the domain are self-discovery and self-organizing as an autonomy system.

Each GOS device maintains a Neighbor Information Table (NIT) which keeps a list of all active GOS devices with their related resource information. Each entry of the NIT represents the information of one GOS device. The entry consists of IP address, remainder storage capacity, and processing power. A data structure is adopted to describe the entry of NIT. The data structures should be maintained in memory so that they can be accessed with little overhead. Therefore the storage capacity occupied by the data structures is important and should be kept as small as possible. The data structure takes 24 bytes per entry. For a physical domain which has 500 GOS devices, it takes only $500 * 24 = 12{,}000$ bytes to track the whole physical domain. Today's servers are normally equipped with more than 1 GB main memory. Compared with the size of main memory, the storage capacity of NIT is negligible. For the common workloads, searching a link list consisting of 500 nodes is unlikely to result in a system bottleneck due to the involved processing power. All GOS devices are equal in functions except the GOS agents. Each GOS device has a coarse grain metadata description of the overall resource information of the physical domain in terms of NIT, which is very helpful for a candidate GOS device to take over the role of the GOS agent when the agent leaves its domain.

The NIT is updated when it is necessary. Each device can discover its neighbors by a process of probing. When a new GOS device wants to join the domain, the device sends a one hop probing multicast "JOIN" request of its coming and the corresponding local resource information such as the processing power and

the storage capacity, and waits for the response of other online GOS devices. The online GOS devices which receive the "JOIN" request add the oncoming device's information to their NITs and send an "ACCEPT" acknowledgment back to the probing GOS device using unicast. The oncoming device constructs its own NIT in terms of the acknowledgment messages. All GOS devices in the domain are self-organized into an overlay GOS network according to the NITs.

Since the DSSM is based on stable and trusted GOS devices, the leaving of a GOS device is normally caused by maintenance, upgrade and other reasons. The GOS device which wants to leave a domain sends a multicast "LEAVE" message in the domain. Once the online GOS devices receive the message, the devices delete the leaving device's information and update their NITs immediately. Because short messages are sent at regular intervals between GOS devices, if a message is not received from a particular device, then the device is assumed to have failed and other remainder GOS devices delete the device from their NITs. The periodic intra-domain heartbeat diffusion with a reasonable low frequency is feasible due to the stable and trusted GOS devices. On the other hand, excessive false detections will increase maintenance cost significantly and unnecessarily. By tuning the diffusion period, the time to recover from a failure can be balanced against the bandwidth overhead of repeated transmissions.

3.2. The formation algorithm of virtual agent domain at upper level

Multiple physical domains can be formed simultaneously, and the domain formation procedure may continue iteratively. Eventually, all GOS devices will become affiliated with domains. Although the domains can be self-organized and self-administered independently, multiple domains may choose to operate cooperatively. The domain agents are selected to play this role.

As discussed in Section 3.1, during the process of the physical domain formation at bottom level, a processing power list of all GOS devices in the domain is formed and recorded on the NIT. Because a domain agent has to communicate with other domain agents to coordinate and manage all GOS devices within the domain, it consumes more computing resources than the ordinary domain members. The GOS device which has the highest processing power within a domain is selected as a domain agent. If the GOS agent fails or leaves the domain, the second GOS device on the processing power list automatically takes over the role. The isolated GOS devices are identified as default domain agents even if there is only one device in a domain.

The selection procedure of a GOS agent is straightforward. The first GOS device of a domain is regarded as an agent by default. The second GOS device which wants to join the domain compares its processing power against the first one. The GOS device which has higher processing power will be reselected as an agent. If the second GOS device has the same processing power as that of the first one, the agent will be kept unchanged. By analogy, we can have a unique GOS agent in a physical domain eventually. Once the domain agents are determined, now the problem is how to take advantage of the agents to coordinate multiple physical domains through GOS network efficiently.

The domain agents do not use flat flooding or multicast to probe other agents due to the communication traffic and security reasons. A GOS service which maintains a list of all domain agents within the DSSM is running on the Internet (see Fig. 1). If a new domain agent wants to join the DSSM, firstly, the agent sends a request to contact the running GOS service which has a public address. Secondly, according to the agent list, the GOS service redirects the request to the running domain agents which are close to the requester. Thirdly, the running GOS agents who received the request update their Agent Neighbor Tables (ANTs) and send an acknowledgement to the new comer. Finally, the new comer constructs its ANT in terms of the acknowledgment messages from the online neighbor domain agents.

The ANT is similar to the NIT, but much simpler. Each entry of the ANT is an IP address which represents a neighbor domain agent. According to the ANT, periodic heartbeat messages are used to confirm the abnormal leaving of the domain agents. All domain agents who are equal in functions communicate with each other to construct a virtual agent domain in terms of the ANT.

Following the domain formation algorithm, all GOS devices are organized and coordinated by the physical domains at bottom level and the virtual agent domain at upper level.

4. Wrapping storage resources into Grid service

Building a large-scale system by composing of hundreds or even thousands of geographically distributed resources poses challenges to manage the resources efficiently. Service is becoming a basic application pattern of Grid because the service can be considered as a basic building block of an infinite resource pool which provides good scalability through its inherent parallelism, and facilitates simple incremental resource expansion (to add resources, one just adds services). Grid users can stack simple modular service piece by piece as demand grows.

A Grid service is a stateful Web Service with an associated lifetime which provides a set of interfaces through which Grid users may interact. The latest Open Grid Service Architecture (OGSA) [31] defines a set of protocols and standards for creating, naming, and discovering persistent and transient Grid service instances. Grid service must provide their users with the ability to access and manipulate state. As the basis of OGSA, Web Services Resource Framework (WSRF) [32] is a family of six Web Services specifications that build on SOAP [33], WSDL [34], and WS-Addressing [35] to define the WS-Resource approach for modelling and managing state in a Web service context. The WS-Resource consisted of a resource document and a corresponding Web service to implement a stateful service. Because Web service is stateless, the resource document employs XML [36] schema to capture state information for a WS-Resource due to the portability and ease in machine processing. The Web service can check and alter states contained in the resource document.

The DSSM divides GOS devices into multiple domains which are self-organized in a P2P manner according to the geographical location. All domains cooperate with each other through their domain agents. Based on the WSRF, each domain agent in the DSSM calculates the available storage resources in the domain in terms of its NIT, and wraps the storage resources as a Grid service. If one domain agent cannot provide satisfied resources, multiple domain agents can cooperate to provide storage resources (e.g. one domain can borrow storage resources from another domain through the communication of the domain agents). This approach introduces another level of flexibility on the resource utilization while taking into account of the overall efficiency.

The storage resource oriented Grid service consists of four parts that are implemented by four classes: A storage service factory, a storage service instance, a storage resource home, and the storage resources (see Fig. 2). When dealing with multiple resources, the WSRF specifications recommend employing the factory/instance pattern that is a well-known design pattern in software design, and especially in object oriented languages. The factory/instance pattern adopts one service in charge of creating the resources (the service factory) and another one to actually access the information contained in the resources (the service instance). Because the GOS devices offer resources for hundreds of thousands of Grid users with different requirements, the factory/instance pattern is employed in the storage service. The storage resource home is responsible for registering and updating the MDS when it is necessary (e.g. when a user subscribes to some storage resources, the information of the remained storage resources in the SSP has to be updated in MDS.). The reader is referred to [37] for a comprehensive understanding of the storage resource oriented Grid service.

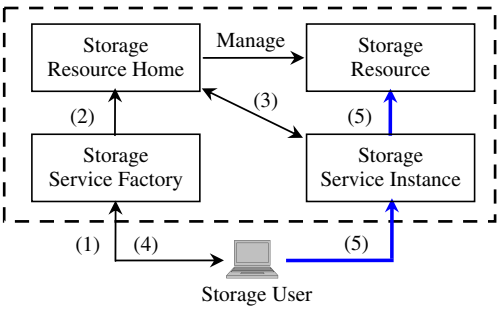

Fig. 2. Components of a storage resource oriented Grid service.

The WS-Resource is adopted in the storage oriented Grid service to describe and access any state of the storage resources. This method can solve the multi attribute range queries. The WSRF lifetime management is employed to describe the resources that are destroyable via web services interfaces. Each time a new storage service instance is created, two properties are assigned, namely storage capacity and lifecycle. The users set the initial values when they submit a request for storage resources with their requirements. The states of these two properties are changed frequently due to the user's operation on the storage resource (e.g. when a user transfers some data to the subscribed storage resource, the state of the storage capacity has to be changed correspondingly).

A newly created WS-Resource is uniquely identified by a WS-Addressing EPR [38] that is used to interact with the resource and distinguish between different storage service instances for different users. The storage users can employ the EPR to identify the corresponding storage resources and perform methods to change the state of the properties.

5. Grid service based storage resource discovery

With an understanding of GOS domain and storage oriented Grid service, we now illustrate how multiple domains might be used in a global Grid environment.

The resources in Grid are often characterized by the dynamic, heterogeneous and distributed features, therefore the description, discovery, and monitoring of resources are challenging problems. Efficient resource discovery is a crucial problem in the large-scale and geographically distributed Grid systems. The main requirements include: (1) low performance overhead; (2) fast response to query; (3) ability to locate the service provider with good Quality of Service (QoS); (4) self-organizing capability to deal with dynamic joining and leaving of resources without centralized control [18].

MDS is a key component to provide basic mechanisms for resource discovery and monitoring in a typical Grid environment. Traditional approaches maintain a centralized MDS or a set of hierarchically organized MDSs to index resource information in Grid. However, three reasons limit the efficiency of the traditional approach. The first, the centralized approach and the root node of the hierarchical method have the inherent drawback of a single point of failure. The second, the approaches may result in system bottlenecks in a highly dynamic environment where many resources join, leave, and can change characteristics at any time. The third, the approaches cannot scale well to a large-scale and geographically distributed system across the Internet.

Many research efforts have investigated technologies for a P2P model based MDS to overcome the above shortcomings [17,39]. The unstructured approaches (e.g. Gnutella-style flooding) propagate the query messages to all devices with a Time-To-Live (TTL) value to control the scale of search. But the approach wastes network bandwidth and does not scale well because of the large volume of query messages generated by flooding [39,40]. In structured P2P systems, Distributed Hash Tables (DHT) are used to map contents into network addresses which can be located easily [41]. This method is very efficient in locating contents by one specific name or attribute, but not supporting multi attribute range queries [42].

The resource discovery in DSSM is based on the unstructured approach but different with that. Fig. 3 depicts the resource discovery process in DSSM. The whole system consists of three GOS domains, three

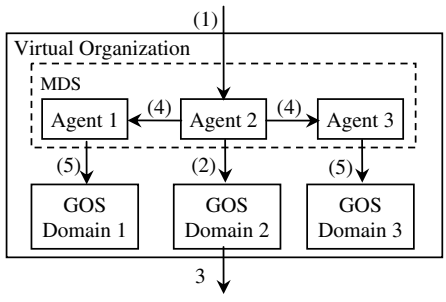

Fig. 3. Resource discovery of DSSM in GT4 environment.

domain agents are cooperating with each other to provide index service (in the dashed rectangle). We only focus on the location phase after we gain an entry point into the GOS service community. The entry point can be obtained by out-of-band method or other approaches.

The major steps of resource discovery are labelled with the sequence number as defined in the following descriptions. (1) A request initiated by a user is redirected from the GOS service published on the Internet to the domain agent 2. (2) Once the request is received, the domain agent 2 searches its NIT since the NIT has the resources information of the whole physical GOS domain 2. (3) If satisfied resources are found in the domain, the domain agent 2 will notify the corresponding GOS devices which have the required resources to communicate with the user directly (e.g. call GridFTP to transfer data) (The domain agent has the ability to pool resources in the domain to match the user's resource requirements). (4) Otherwise, the agent 2 forwards the request to its neighbor agent 1 and agent 3 using unicast according to its ANT. (5) The agent 1 and agent 3 will simultaneously repeat the operations as agent 2 did in step (2) and step (3) to achieve inter-domain parallelism.

For the traditional unstructured P2P searching method, the corresponding device sends a query message to its neighbors, which in turn forward it to their own neighbors. If a device possesses the requested resource, it sends a query hit message that will follow the same path back to the requesting device [43]. The method used in DSSM is more efficient and scalable than the traditional one, because it avoids flooding, eliminates centralized or hierarchically architecture, and the GOS device which has satisfied resources can communicate with users directly to avoid expensive store-and-forward data copying between GOS devices, GOS agents and users.

6. Prototype system and experimental evaluation

We constructed a proof-of-concept prototype with three GOS devices, one Wide Area Network (WAN) emulator [44], and several client machines. All components were connected through a 1000/100M adaptive switch. Table 1 shows the system configurations of the prototype. The WAN emulator forwards packet at line rate and has user-settable delay and drop probability. In order to simulate a distributed WAN environment, all traffic among domains passed through the WAN emulator.

6.1. Dynamicity and reliability evaluation

The dynamicity and reliability of the DSSM architecture were measured at two stages. At the first stage, we set the IP address of three GOS devices within one network segment to denote a single physical domain. To illustrate the dynamic scalability and reliability of the architecture, we first configured one GOS device in one domain, and then the other two GOS devices joined the domain respectively, finally, the three GOS devices continuously left and joined the domain. We repeated the above process for more than 50 times, it did not cause any problems in our experiment. Because all GOS devices in the domain have the same processing power, the first GOS device of the domain is selected as an agent by default even it has no other GOS agents to cooperate with.

At the second stage, to simulate two physical domains, we configured two network segments by setting the IP address of the three GOS devices. The first domain consisted of two GOS devices, and the second domain had only one GOS device. The communication traffic between two domains passed through a WAN emulator. The DSSM architecture was formed successfully by the following steps: (1) The first GOS device in domain 1

Table 1
System configurations of the prototype

	GOS Device	WAN Emulator and Clients
CPU	Intel Xeon 2.8GHZ	Intel Pentium 2.66 GHZ
Memory	2G	512M
NIC	Two Broadcom 100/1000M	Intel(R) PRO/100M
Disks	Six IBM FRU 32P0730	Seagate ST340014A
OS	Red Hat(Kernel 2.4.21)	Red Hat (Kernel 2.4.21)

was started and regarded as an agent by default. (2) The second GOS device joined the domain 1. The GOS device which has higher processing power should be selected as an agent. In our test, the GOS devices had the same processing power. Therefore, the first GOS device played the role of an agent as discussed in Section 3.2. (3) Two GOS devices in the domain 1 created their NITs in terms of multicast communications in the domain. (4) The GOS agent of domain 1 wrapped the storage resources of the two GOS devices into Grid services, and then put its address information on the list of the GOS service which has a public address. (5) The single GOS device in domain 2 acted as a domain agent, wrapped its local storage resources into Grid service, and registered its address in the public GOS service. (6) The agents of domain 1 and domain 2 constructed their ANTs in terms of the request and acknowledgement, respectively. According to the ANT, a virtual agent domain consisting of two agents will be constructed by the communication among the domain agents. The formation and decomposition of DSSM were repeated more than 50 times continuously without incurring any problems.

As discussed in Section 3.1, the DSSM is based on stable and trusted GOS devices that are less dynamic than pure P2P nodes. The leaving or joining of GOS devices or GOS agents are normally caused by maintenance, upgrade and other reasons. It is unlikely to happen frequently. We do believe the above test can validate the dynamicity and reliability.

6.2. Resource discovery evaluation

The interaction between users and the DSSM is mediated through a GOS service published on the Internet. When a user wants to access the storage resources of DSSM, firstly, it sends a request to the GOS service with its requirements. Secondly, the GOS service redirects the request to a domain agent which is close to the user. Thirdly, a P2P model is employed by the agent to locate the storage resources which are wrapped into Grid services. Fourthly, the user chooses a suitable Grid service from a list of available Grid services located by the request. The storage resources behind the Grid service can be distributed, and is transparent to the user through the user interface. Finally, the user can access the storage resources by creating file folder, uploading, downloading and deleting files.

Response time seen by the Grid users is an important metric in evaluating the performance of resource discovery of the DSSM. The measured response time is the time between when a user submits a request to the DSSM and when the last byte of the response is delivered to the user. In our experiment, each client machine can simulate many different users with different connections to the DSSM. Each user submitted a query request to the DSSM and waited for the response. After receiving the response, it immediately submitted another request. We employed average response time to measure the performance. The data traffic between the client machines and DSSM passed through the WAN emulator. The Round Trip Time (RTT) of the emulator in our experiment was set to 20 ms, 40 ms and 60 ms, respectively. Because the speed of light in fibre is roughly 66% of the speed of light in vacuum, the transfer distance simulated by the WAN emulator were $300 \times 10^6 \times 0.66 \times 10 \times 10^{-3} \approx 2000$ km, 4000 km and 6000 km, respectively.

Fig. 4 illustrates the average response time of the resource discovery of DSSM. The X axis is linear, whereas the Y axis is in logarithmic scale. It shows that the average response time increases linearly with the growth of the number of users. The reason is that the average response time is limited by the processing time of the domain agents. With the increase in number of users, the longer queue time results in high response time.

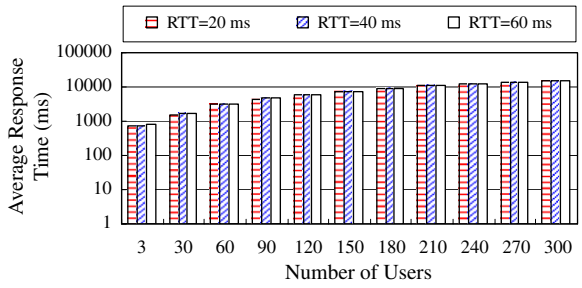

Fig. 4. Average response time of the resource discovery of DSSM.

Before we started our experiment, we expected to see a significant impact of the RTT on the average response time. For three users, we did observe a growth of the average response time when the RTT was increased. However, Fig. 4 shows a negligible performance impact of RTT on the average response time, especially when a large number of users access the DSSM simultaneously. We do believe that the introduced delay by RTT has an impact on slowing down the request rate which goes to the DSSM. But the RTT is relatively small to the query time. The reason is that the query uses SOAP messages which involve a lot of overhead incurred by XML. Mani and Nagarajan [45] reported that XML's way of representing data takes more than 400% overhead compared with the way adopted by binary. This is a hot topic in the Web service community. A lot of methods have been devised to solve the problem (e.g. compressing XML or a suitable cache mechanism). We just focus on the performance evaluation of the resource discovery of DSSM.

6.3. The impact of dynamicity on the resource discovery

In this measurement, we investigate the impact of dynamicity on the resource discovery of DSSM. The RTT of the WAN emulator was set to 20 ms. The basic test environment and conditions were the same as that of Section 6.2, but we configured the GOS devices and GOS agents to leave and join the DSSM every 60 s to simulate a dynamic environment as discussed in Section 6.1.

We expect that the dynamicity of DSSM has a negligible performance impact on the resource discovery, because when a GOS agent leaves a domain, another candidate GOS device should immediately take over the role. Fig. 5 shows the average response time of a static and a dynamic DSSM, respectively. It illustrates that the dynamicity of DSSM does have performance impact on the resource discovery. The reason is that there is only one GOS device which plays the role of a GOS agent in the domain 2. If this GOS agent leaves DSSM, no candidate in the domain can take over the query traffic which goes to the agent, which results in a heavy traffic to the GOS agent of domain 1. With the increase in number of users, the situation becomes worse, which incurs a long queue time and leads to a high response time. It validates that the query traffic can be shared by multiple GOS agents, and the traffic which goes to each GOS agent decreases with the growth in number of GOS agents.

6.4. Bandwidth evaluation

Please note that this paper focuses on a dynamic and scalable storage management architecture which may involve hundreds or even thousands of distributed GOS devices. The goal of this work is to support long-distance and bulk data access of large-scale and complex Grid applications. Grid service combines the Web service and WSRF to provide a service based Grid environment that enables heterogeneous environments to be integrated and reconciled to accomplish complex tasks. The core of Web service is XML which offers portability and ease of machine processing. However, the XML's way of representing data takes more than 400% overhead compared with the way adopted by binary [45]. It is not a good idea to transfer all data in XML especially for the critical real-time applications. We take advantage of the Grid service to discover and locate the resources. Whereas the data transfer is still using binary code rather than XML to maintain the performance [46]. Because GridFTP [47] provides a secure, robust, fast and efficient transfer of (especially bulk)

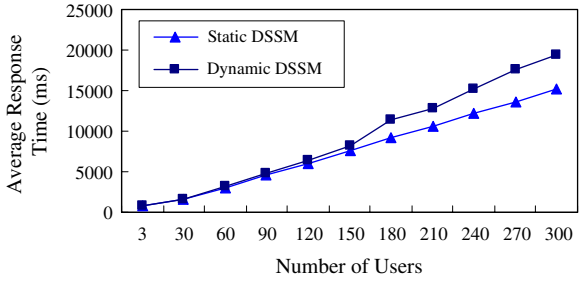

Fig. 5. Average response time of the static and dynamic DSSM.

data, it was adopted by the DSSM to transfer data between the GOS devices and Grid users. A Grid user contacts the public GOS service to find the available storage services and chooses one service with 10GB capacity, and then employs GridFTP to transfer data between the storage service and its local storage space.

The performance of the implemented prototype was measured by system bandwidth that is the amount of data divided by the data transfer time. The data transfer time was defined as the difference between the end time and start time of a data transfer excluding the process of resource discovery. A set of files ranging from 32 MB to 512 MB were transferred over the emulated WAN to measure the bandwidth. The RTT of the emulator in our experiment was set to 120 ms which denotes 12,000 km. The file with specified size is called workload in the following discussion. All the performances reported in this paper are based on the average of 300 measurements. The parallel transfer of multiple streams of the prototype was investigated since this is a key feature of GridFTP.

Fig. 6 shows that as the size of the workloads increases, the bandwidth grows at the same time. The system bandwidth can be improved by increasing the number of parallel streams of GridFTP. In our experiment, for the workload of 512 MB, the bandwidth boosted from 3.6 Mbits/s to 26 Mbits/s when the stream number reached 80. In general, the performance curves with different workloads are very similar. The bandwidth increases with the growth of parallel stream number, but there are some exceptions. It is interesting to observe that when the stream number reaches 5, the bandwidth of 32 MB workload reaches the maximal value and then starts to decrease slowly, and the bandwidth of 64 MB workload reaches the point of saturation. The reason will be explained in the following paragraph.

Parallel data transfer requires that the data is partitioned across multiple parallel streams at sender side (scatter) and the striped data is combined at receiver side (gather). The data scatter/gather incurs some overhead penalty. To justify the penalty, the performance gain achieved by parallel transfer has to be greater than the overhead involved in data scatter/gather. For a certain size of workloads, with the increase of the parallel stream number, the overhead produced by scatter/gather grows. There are a fixed number of parallel streams which can strike a balance between the overhead involved in data scatter/gather and the benefit gained by parallel transfer. For example, the number is 5 for the 32 MB workload. For a fixed parallel stream number, with the growth of workload size, compared with the performance gain obtained by parallel transfer, the overhead of scatter/gather is relatively reduced. We conclude that parallel stream data transfer works better on large files than small files.

In order to investigate the impact of the background query requests on the system bandwidth, we adopted two client machines to submit query requests continuously to simulate a real working environment, while another client machine employed GridFTP to transfer data. We observed negligible bandwidth variation. The reason is that the query requests are processed by the GOS agents, while the data is transferred between the GOS device and the user directly. The traffic does not interfere with each other if the network is not a bottleneck.

In summary, the prototype and experimental evaluation demonstrate that the DSSM architecture proposed in this paper can organize the distributed GOS devices into a dynamic, scalable, and reliable storage pool to satisfy the storage capacity and bandwidth requirements posed by complex Grid applications. The system could be expanded to a large-scale with an evaluation of a large-scale experiment.

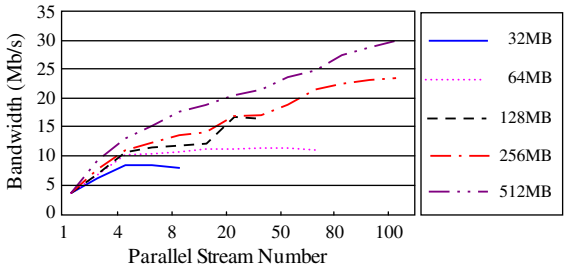

Fig. 6. Bandwidth measured with different stream number and different workloads.

7. Conclusions and discussion

In this paper, based on an existing Grid environment, we proposed and designed a DSSM architecture which organizes GOS devices into different domains to enhance the data access locality in terms of geographical area. P2P model is employed to endow the architecture with dynamic scalability and reliability. Storage resources are wrapped into Grid services which are employed as basic building blocks of an infinite storage pool. Storage expansion is achieved by simply adding services. The DSSM architecture avoids the hierarchical or centralized approaches of traditional Grid architecture, eliminates the flat flooding of unstructured P2P system, and provides a dynamic storage pool in Grid environment. The proof-of-concept prototype gives useful insights into the architecture behavior of DSSM architecture.

The DSSM can be regarded as a virtual two-tiered hierarchy, but it replaces a few root nodes of the traditional hierarchy with many GOS agents which can be expanded to a large-scale, thus avoiding the single point of failure and the potential performance bottleneck. The architecture does incur some additional overhead which results in performance penalty on the GOS agents in three scenarios. The first, if hundreds of thousands query requests go to a particular domain agent simultaneously, the agent could become a potential system bottleneck. The reason is that all query traffic going to a domain has to pass through the domain agent. The second, an intelligent domain agent is able to pool the storage resources and keep the load balance among the GOS devices in the domain. It takes some additional computing overhead to do this job. The third one is the worst scenario. When a GOS agent is serving data access to users, a large number of query requests going to the agent will heavily overload the agent. This scenario should be avoided or alleviated. However, the I/O traffic taken over by each agent will be decreased with the increase in number of domain agents. The probability that most of the I/O traffic goes to a particular domain can be decreased, if the hot data is arranged properly. A dynamic and transparent data replication mechanism which automatically places the data replicas across domains where they are needed is able to alleviate the performance impact on the GOS agents, it is also crucial to the overall performance. The first, it improves the aggregate bandwidth by providing simultaneous data access to multiple data copies. The second, it reduces the latency between the data provider and data consumer by copying the data near the data consumer, because the latency is incurred not only by the protocol stack of Grid service, but also by the distance of the communication [46]. We are currently working on the intelligent data replication mechanism of DSSM. The reader is referred to [46] for a more comprehensive discussion of the service based storage Grid.

Grid Oriented Storage is an ongoing project supported by EPSRC/DTI of United Kingdom, which focuses on the next generation data storage system architecture for the Grid computing era. We are implementing the DSSM in an existing Grid environment and may work together with a particular Grid application.

Acknowledgements

The authors would like to thank the anonymous reviewers for helping us refine this paper. Their constructive comments and suggestions are very helpful. In addition, the authors are grateful to Prof. Denis Trystram and Prof. Erhard Rahm for giving us the opportunity to clarify our thoughts about the DSSM.

References

[1] Wong Han Min, Wang Yonghong Wilson, Yeo Heng Ngi, et al., Dynamic storage resource management framework for the grid, in: Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies, 2005, pp. 286–293.
[2] Ian Foster, Carl Kesselman, Steven Tuecke, The anatomy of the grid: enabling scalable virtual organizations, International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.
[3] Storage Resource Broker, <http://www.sdsc.edu/srb/index.php/Main_Page>.
[4] A. Rajasekar, M. Wan, R. Moore, G. Kremenek, T. Guptil, Data grids, collections, and grid bricks, in: Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), 2003, pp. 2–9.
[5] A.Rajasekar, M. Wan, R. Moore, MySRB & SRB – components of a data grid, in: Proceedings of the 11th International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 24–26, 2002, pp. 301–311.
[6] Distributed Parallel Storage System Project, <http://wwwdidc.lbl.gov/DPSS/>.
[7] High Performance Storage System, <http://www.hpsscollaboration.org/hpss/index.jsp>.

[8] Atsushi Manabe,Kohki Ishikawa, Yoshihiko Itoh, et al., A data grid testbed environment in Gigabit WAN with HPSS, Computing in High Energy and Nuclear Physics (CHEP03), 2003.
[9] DataCutter, <http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm>.
[10] M. Beynon, R. Ferreira, T. Kurc, A. Sussman, J. Saltz, DataCutter: Middleware for filtering very large scientific datasets on archival storage systems, in: Proceedings of the 17th IEEE/8th Goddard Conference on Mass Storage Systems and Technologies, USA, 2000, pp. 119–133.
[11] Monitoring and Discovery System, <http://www.globus.org/toolkit/mds/>.
[12] F. Wang, N. Helian, S. Wu, Y. Deng, K. Zhou, et al., Grid-oriented storage, IEEE Distributed Systems Online 6 (9) (2005) 1–4.
[13] D. Talia, P. Trunfio, Towards a synergy between P2P and grids, IEEE Internet Computing 7 (4) (2003) 94–96.
[14] A. Rowstron, P. Druschel, Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01), 2001, pp. 188–201.
[15] John Kubiatowicz, David Bindel, Yan Chen, et al., OceanStore: an architecture for global-scale persistent storage, in: Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), 2000, pp. 190–201.
[16] Free Heaven Project, http://freehaven.net/.
[17] S. Basu, S. Banerjee, P. Sharma, Sung-Ju Lee, NodeWiz: Peer-to-peer resource discovery for grids, in: Proceedings of the IEEE International Symposium on Domain Computing and the Grid (CCGrid 2005), vol. 1, 2005, pp. 213–220.
[18] Cheng Zhu, Zhong Liu, Wei Ming Zhang, Weidong Xiao, Dongsheng Yang, Analysis on greedy-search based service location in P2P service grid, in: Proceedings of the Peer-to-Peer Computing, 2003, pp. 110–117.
[19] Frederik W. Jansen, Erik Reinhard, Data locality in parallel rendering, in: Proceedings of the 2nd Eurographics Workshop on Parallel Graphics and Visualisation, 1998, pp. 1–15.
[20] Aart J.C. Bik, Reshaping access patterns for improving data locality, in: Proceedings of the 6th Workshop on Compilers for Parallel Computers, 1996, pp. 229–310.
[21] D. Bernholdt, S. Bharathi, D. Brown, et al., The earth system grid: supporting the next generation of climate modeling research, Proceedings of the IEEE 93 (3) (2005) 485–495.
[22] Wolfgang von Rueden, Rosy Mondardini. The Large Hadron Collider (LHC) Data Challenge. <http://www.ieeetcsc.org/newsletters/2003-01/mondardini.html>.
[23] Mar'ia E. G'omez, Vicente Santonja, Characterizing temporal locality in I/O workload, in: Proceedings of the 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, 2002.
[24] Jean-Baptiste Ernst-Desmulier, J. Bourgeois, F. Spies, J. Verbeke, Adding new features in a peer-to-peer distributed computing framework, in: Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2005), 2005, pp. 34–41.
[25] B. Yang, H. Garcia-Molina, Improving search in peer-to-peer networks, in: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, July 2002, pp. 5–14.
[26] Kai Hwang, Hai Jin, Roy S.C. Ho, Orthogonal striping and mirroring in distributed RAID for I/O-centric cluster computing, IEEE Transanctions on Parallel Distribution System 13 (1) (2002) 26–44.
[27] V. Welch, Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective. 2004. http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf.
[28] A. Ephremides, J.E. Wieselthier, D.J. Baker, A design concept for reliable mobile radio networks with frequency hopping signaling, in: Proceedings of IEEE, vol. 75, 1987, pp. 56–73.
[29] A.K. Parekh, Selecting routers in ad-hoc wireless networks, in: Proceedings of the SBT/IEEE International Telecommunications Symposium (ITS1994), 1994, pp. 420–424.
[30] Mario Gerla, Jack Tzu-Chieh Tsai, Multidomain, mobile, multimedia radio network, Journal of Wireless Networks 1 (3) (1995) 255–265.
[31] The Open Grid Services Architecture, <http://www.globus.org/ogsa/>.
[32] Web Services Resource Framework, <http://www.globus.org/wsrf/>.
[33] Simple Object Access Protocol, <http://www.w3.org/TR/soap/>.
[34] Web Services Description Language, <http://www.w3.org/TR/wsdl>.
[35] WS-Addressing, <http://msdn2.microsoft.com/enus/library/ms951225.aspx>.
[36] Extensible Markup Language, <http://www.w3.org/XML/>.
[37] Yuhui Deng, Frank Wang, A heterogeneous storage grid enabled by Grid service. ACM SIGOPS operating systems review, Special Issue: File and Storage Systems 41 (1) (2007) 7–13.
[38] WS-Addressing, <http://msdn2.microsoft.com/enus/library/ms951225.aspx>.
[39] A. Iamnitchi, I. Foster, D. Nurmi, A peer-to-peer approach to resource discovery in grid environments, in: Proceedings of the 11th Symposium on High Performance Distributed Computing, Edinburgh, UK, August 2002, pp. 419–434.
[40] L. Qin, C. Pei, C. Edith, et al., Search and Replication in unstructured peer-to-peer networks, in: Proceedings of the 16th ACM International Conference on Supercomputing (ICS'02), New York, 2002, pp. 84–95.
[41] R. Sylvia, S. Scott, S. Ion, Routing algorithms for DHTs: some open questions, in: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Springer-Verlag, Heidelberg, 2002, pp. 45–52.
[42] Min Cai, Martin Frank, Jinbo Chen, Pedro Szekely, MAAN: a multi-attribute addressable network for grid information services, in: Proceedings of the 4th International Workshop on Grid Computing, 2003, pp. 184–192.

[43] Carlo Mastroianni, Domenico Talia, Oreste Verta, A P2P approach for membership management and resource discovery in grids, in: Proceedings of the Internationa Conference on Information Technology (ITCC05), 2005, pp. 168–174.
[44] Wide Area Network Emulator, <http://cs.ecs.baylor.edu/~donahoo/tools/nistnet/>.
[45] A. Mani, A. Nagarajan, Understanding Quality of Service for Web services, January. <http://www-106.ibm.com/developerworks/library/ws-quality.html>, 2002.
[46] Yuhui Deng, Frank Wang, Challenges and opportunities of service enabled storage grid, ACM SIGOPS Operating Systems Review 41 (4) (2007) 79–82.
[47] GridFTP, <http://www.globus.org/grid_software/data/gridftp.php>.