A Scalable Autonomous Replica Management Framework for Grids

Vladimir Vlassov, Dong Li
Dept of Electronic, Computer and Software Systems, School for Information and Communication Technology, Royal Institute of Technology (KTH) Stockholm, Sweden
vladv @kth.se, dongl@ kth.se

Konstantin Popov
Swedish Institute of Computer Science (SICS) Kista, Sweden
kost@sics.se

Seif Haridi
Dept of Electronic, Computer and Software Systems, School for Information and Communication Technology, Royal Institute of Technology (KTH) Stockholm, Sweden
seif@kth.se

Abstract

Data replication can reduce access time and improve fault tolerance and load balancing. Typical requirements for a replica management system include an upper bound on replica Round Trip Time, scalability, reliability, self-management and self-organization, and ability to maintain consistency of mutable replicas. This article presents the design and a prototype implementation of a scalable, autonomous, service-oriented replica management framework for Globus Toolkit Version 4 using DKS. DKS is a structured peer-to-peer middleware. Grid nodes are integrated into a P2P network. The framework uses the ant metaphor and techniques of multi-agent systems for collaborative replica selection. We propose also a complimentary "background" service that collects access statistics and optimizes replica placement based on access pattern and replica lifetimes statistics. We have tested and profiled the prototype.

1. Introduction

The Grid aims at secure integration of heterogeneous computing resources in a standard and uniform way. Sharing of computing facilities is provided by *Computational Grids*, and *Data Grids* support applications addressing large amount of data that can be distributed and replicated over the Globe. Managing storage facilities and data replicas in large scale and volatile Data Grids is a challenging task.

A replica management system determines locations of replicas and keeps track of them, and maintains replica consistency. It should be scalable and optimize replica placement in order to reduce access time and communication costs, and maintain a required level of consistency of mutable replicas [1]. Typical replica management systems (e.g. [2][3]) consist of:

- *Information services* such as the replica location service (RLS), user access history, and a metadata catalog. Some services, in particular RLS, must provide good scalability and fault-tolerance.
- *Data transfer service* provides for file transfer among collaborating Grid nodes.
- *Security infrastructure*, including authentication and authorization for remote users, and secure communication and data transfer.
- *Data consistency management service* can have different requirements depending on the application and data access patterns. In many scientific applications the datasets are read-only and therefore replicas are consistent automatically. Applications with mutable data should be allowed to choose an optimum level of replica consistency.

Our research aims at building a scalable higher-level replica management system that provides optimized replica selection and placement. Our system prototype is implemented using the Globus Toolkit 4 (GT4) [4]. Our framework utilizes the DKS (Distributed K-ary System) Peer-to-Peer middleware [5][6] that allows to organize Grid nodes into self-organizing P2P overlay networks. The GT4 GridFTP [7] is used for data transfer. Our framework contains two specially developed components: the DKS Replica Location Service (DKSRLS) built on the DKS P2P network, and the Node Location Component (NLC) based on GT4 WS Monitor and Discovery Systems (MDS) aggregator framework. The framework also provides for a data consistency mechanism.

The dynamicity of the Grid environment presents challenges for replica management. In order o simplify the work of Grid users and administrators on replica management, consistency maintenance, and to improve reliability, a replica management system must be

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

autonomously self-managing and self-organizing wrt high-level user-specified objectives and policies [8].

We aim at providing the aforementioned self-* properties in our framework. First, we rely on self-organization of the DKS P2P middleware. Second, we try to utilize the self-organization properties of *ant colonies*. Ants are simple autonomous computing units, or agents. We employ ants in the replica placement service of our framework. Ant colonies are *complex adaptive systems* (CAS, e.g. [9]). CAS methods are used to study and engineer the behavior of large-scale physical, biological, social, and computer systems.

The remainder of the paper is structured as follows. Section 2 presents background information on GT4, P2P systems, the DKS P2P middleware, and also related work. Section 3 describes the architecture and the prototype of our framework. Section 4 presents results of the preliminary evaluation of the prototype that we have done in order to validate the approach and to find potential bottlenecks. A detailed evaluation of different design options is our future work. Finally, our conclusions and future work are given in Section 5.

2. Background and Related Work

2.1. Self-Organizing P2P systems

Peer-to-Peer (P2P) is a distributed computing paradigm in which peers are acting as both clients and servers. A P2P application builds a self-organizing overlay network over a network such as the Internet.

Both P2P systems and the Grid address the issue of collaboration among users with a diverse set of resources to share. As Grid systems scale up and the P2P techniques begin to capture shared use of more specialized resources, we witness a convergence between these two paradigms [10] [11] [12].

There are *unstructured* and *structured* P2P systems [10]. In unstructured P2P systems such as Gnutella and KaZaA, each peer is randomly connected to a set of other peers (neighbors) and therefore communication and search operations in such systems are based on flooding, random walks [13], or identifier search using the Bloom filters [14]. In structured P2P systems such as Chord [15], Tapestry [16], Pastry [17], SkipNet [18], and DKS [5][6], peers are organized in a structured network with deterministic routing that allows to reach each peer in a bounded number of hops, typically logarithmic in the size of the network. Most of recent structured P2P systems also provide the distributed hash table (DHT) functionality.

Distributed K-ary System (DKS) [5][6] is a structured P2P middleware. DKS organizes peers in a circular identifier space and maintains routing tables of logarithmic size. Each node is responsible for an interval on the identifier space. A routing table in each DKS node consists of $\log_k(N)$ levels, where N is number of nodes in the network and k is a configuration parameter. Each level contains k entries corresponding to k intervals with pointers to the first peer encountered in (responsible for) the interval. This kind of structure looks like a spanning tree. If $k = 2$, DKS becomes similar to Chord [15]. The lookup operation follows a path of the spanning tree, ensuring the logarithmic cost in the worst case.

DKS self-organizes itself as nodes join, leave and fail. The join and leave operations are locally atomic. DKS implements the efficient correction-on-change and correction-on-use mechanisms for maintenance of routing tables [19]. Symmetric replication [20] distributes replicas evenly among DKS nodes and enables concurrent requests improving efficiency and fault-tolerance. DKS provides also efficient broadcast and multicast [21] [22] [23].

2.2. Replica Management in Globus Toolkit 4

Our replica management framework has been developed for Grids built using the Globus Toolkit 4 (GT4) [4]. GT4 implements the Web Service Resource Framework (WSRF) [24] and uses it to implement the Open Grid Services Architecture [25] components.

GT4 GridFTP and Replica Location Service (RLS) are the basic data management services. RLS provides access to information about the location of replicated data. Higher-level services such as Reliable File Transfer (RFT) [26] and Data Replication Service (DRS) [27] are based on GridFTP and RLS. Replica management in GT4 is based on the Giggle framework [28] that provides a hierarchical distributed index of replicas in which each replica is identified by a pair of a logical file name (LFN) and a physical file name (PFN). Local Replica Catalogs (LRC) store the PFN-LFN mappings; and Replica Location Indices (RLIs) collect the mapping information from LRC and are organized into a hierarchical distributed index. Information in RLIs is periodically refreshed by LRCs according to a soft-state-update protocol.

2.3. Related Work

The Giggle[28]-based replica location framework in GT4 has limitations: LRCs and RLIs are deployed statically and cannot be automatically reconfigured. It is also not easy to recover from RLI failures.

P2P solutions [11][12] have been proposed to improve reliability and to simplify maintenance of the replica management in GT4. In [11] the authors

organize all RLIs into a structured P2P overlay network based on Chord with successor replication. In [12] the authors propose to use the Kademlia [29] P2P overlay as the replica location service.

Our replica management framework is also built on a P2P overlay network. However, unlike the Chord used in [11], the DKS uses symmetric replication for better load balancing and reliability. Furthermore, for replica location and placement our framework introduces a distributed algorithm based on the ant metaphor. It also includes a proactive statistics service that can optimize replica placement (e.g. to remove unused replicas and/or to place/redistribute popular replicas) based on observed access patterns and other criteria. In the current implementation, replica placement is based on RTT requirements and access patterns. In our future work, we intend to extend the list of parameters that replica placement and selection take into account, e.g. the node load. Our framework also supports a weak replica consistency mechanism.

Replica Management Service (RMS) in DataGrid [30] is more autonomic than Data Replication Service (DRS) of GT4. RMS is employed to locate "best" replicas of the files needed to execute a job allocated to a requesting node. RMS uses an auction-based replica location mechanism in which auction bids are weighted mostly according to estimated replica transfer times.

In contrast to RMS that creates new replicas only if it is expected to be beneficial in the long term, our replica location service is more aggressive and creates replicas on demand at any place that can reduce the file access time. Optimized replica (re)distribution and removals are left to the Statistics Service. In our future work, we intend to take into account more factors in optimized replica selection and placement.

3. Replica Management Framework

The replica management system is intended to place and manage replicas according to client-specific QoS requirements and observed data access patterns. We assume that a client of the replica management system is a computing element of a job execution site as shown in Figure 1 [31]. Assume a client needs file access with specific QoS requirements which are based on the Round Trip Time (RTT). On a client request, the replica management system should locate and/or place replica on a suitable Grid node. Note that from the point of view of QoS requirements alone, the best place to put a replica would be the user's own storage, but it can be impossible due to e.g. insufficient storage.

Each job execution site (see Figure 1) has its own instance of a Replica Manager that includes replica management components, and a storage element (file server) to store replicas. Here, the word "component" refers to a functional unit that can provide one or more services. The word "service" refers to a web service.

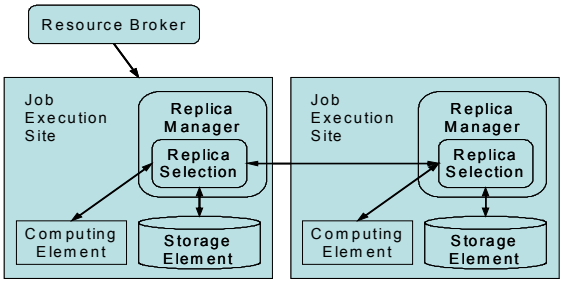

Figure 1 A typical DataGrid architecture [31]

Replica managers are connected in a structured P2P network built using the DKS middleware as depicted in Figure 2. Even though the DKS P2P overlay network is used only by the replica management system, the network can be also used by other services of the Data Grid. Figure 2 illustrates main components and services of our replica management system.

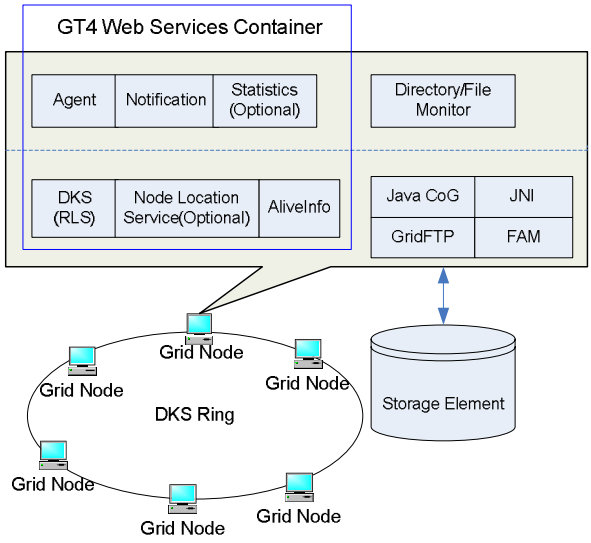

Figure 2 Main components of replica management system

The system is composed of three reusable lower-level components: *Location Information*, *Data Consistency* and *Data Transfer*, and two higher-level components *Replica Selection (RS)* and *Statistics*, which act upon the lower-level ones.

3.1.1. Location Information Component provides information on replica locations and all Grid nodes in the DKS P2P network. This component consists of two sub-components: the *Replica Location Service (RLS)* and the *Node Location Component (NLC)*. The latter serves to locate and observe state of live system nodes.

RLS is a distributed service that comprises all nodes in the system. Each node is present in a DKS P2P overlay network maintained by the RLS. The overlay network hosts a DHT with the replica index. DHT maps file names (key) onto lists of replica positions (value). RLS API includes the *add* (a replica position), *query* (positions of all replicas for a given file name), and *delete* operations. Note that every node on the RLS overlay network has a local access point to RLS.

NLC includes the *Node Location Service (NLS)* and the "*aliveinfo*" service. NLC is based on the GT4 WS MDS Aggregator Framework with the Query Aggregator Source, where the NLS acts as an aggregator sink and the "aliveinfo" is a WSRF service that registers its resource property to a service group in the NLS. NLC allows clients to query NLS for a list of nodes participating in DKS network, and to monitor nodes' resources. In addition it provides an entry point for nodes joining the DKS network. In the current version of the prototype, "aliveinfo" resource properties are the file list in the storage element (see Figure 2) and the node address. We intend to extend the service so it can provide more useful information on the status of nodes, such as load and available storage size. The NLS is based on the GT4 Index Service and manages node addresses of all registered live Grid nodes using a soft-state protocol. NLS can be queried with the help from the aggregator framework. When a node joins the system, its "aliveinfo" service registers to the NLS. NLS is optional for each site, but at least one instance must exist in the network.

3.1.2. Data Consistency Component (DCC) serves to maintain replica consistency. DCC monitors specified replicas and propagates changes using the epidemic approach [32]. It also updates RLS. DCC utilizes the File Alteration Monitor (FAM) [33] and an upper-level "Data Consistency" application that maintains data consistency. Each Grid node has an instance of FAM that monitors the local storage. Although we tested FAM under Linux only, we expect its latest version (2.7.0) to be portable. FAM is developed in C and requires Java Native Interface (JNI) [33] to be used to call its routines. Data Consistency applications interact with FAM asynchronously through a log.

Our approach is flexible and independent of any particular data access patterns in applications. However it can only guarantee weak consistency among replicas and may not work well for time-critical data. Yet in many cases, scientific data files are used mostly for analysis and do not change frequently. We can also set a suitable update propagation latency.

3.1.3. Data Transfer Component is created on each node. It consists of a GridFTP client and one GridFTP server. Note that GridFTP supports third party transfer.

3.1.4. Replica Selection (RS) Component selects and/or places replicas on client requests. RS checks whether there is at least one replica that fulfills the RTT requirement; otherwise it finds a node where a new replica can be placed to fulfill the requirement.

The RS component includes two services, an obligatory *agent* and an optional *notification*. The responsibility of the agent service is threefold. First, it is an entry point to replica selection, i.e. it can work as a "delegate agent" to receive a task from clients and conduct selection work on behalf of the requesting client. Second, it indirectly interacts with other agents in the DKS network according to the "ant" algorithm and senses the network environment to help a delegate agent to find an appropriate replica. Replica selection using agent services and ants is described in detail in Section 3.2. Third, the agent service is also responsible for measuring the RTT between the local node and a requesting user node: the agent issues a configured number of ping messages sent over a TCP connection.

The agent service provides the following major operations:

- *receiveRequest* is used to submit a client replica location request containing a requested file name, required RTT and the client's available storage size. The agent obtains replica positions from the RLS and computes RTTs from those positions. If no replica fulfills the requirements, the agent calls the *motivate* operation on nodes that have the replicas;
- *motivate* initiates the distributed ant algorithm by invoking the *antContainer* operation on neighboring nodes referenced by longest "fingers" in the node's routing table. In other words, the *motivate* operation send ants into the DKS network to find replica locations that fulfill the client RTT requirement;
- *antContainer* implements the ant algorithm as described in Section 3.2.1. This operation is called an ant arrives to the node.
- *getRTT* measures RTT to a remote node;
- *getFileList* and *getFileSize* operations return all files names from the local file system and the size of a specified file, respecively.

The RS notification service is used to inform a requesting client of the selected replica position. Before a user delegates a request to an RS agent, it creates a resource in a notification service. The resource properties are the values associated with replica selection, such as file name, RTT requirements and, eventually, the selected replica position and its RTT value. When the delegate agent gets the final selection results, it updates the resource and enables an asynchronous notification to the user. For each replica selection, the delegation agent and the notification service could be in different Grid nodes.

3.1.5. Statistics Component is used to collect statistics on file accesses and replica lookup activities. This statistics can be used to improve replica placement decisions and/or job scheduling. In the current prototype, this component is implemented as a centralized service provided by one of the Grid nodes. It receives file change records from each Data Consistency component and replica selection results from agent services. Using the statistics, the Statistics Service can decide to remove a replica if it is not used frequently enough, or, conversely, proactively create a new replica if the replica's file becomes popular.

3.2. Replica Selection Method

3.2.1. The Autonomous ant algorithm simulates the behavior of an ant colony consisting of a large number of simple ants. An ant colony can exhibit the elements of *emergent behavior*, i.e. although the actions and interactions among ants are simple, together they can generate more complex and richer patterns than those produced by individual ants alone. Resnick [34] describes an artificial ant following three simple rules: (i) wander around randomly, until it encounters an object; (ii) if it was carrying an object, it drops the object and continues to wander randomly; and (iii) if it was not carrying an object, it picks the object up and continues to wander. Although the rules are simple, the ants conforming to them are able to collect small pieces and group them into larger ones. Ants can also use *stigmergy* [35] which is an agent communication method by modifying their local environment. For example, ants can leave pheromones along their trails, indirectly guiding other ants.

The ant algorithm is self-organizing and adaptive. It works in large scale volatile environments. Although the algorithm seems simple, it does not require any rules specific to variations in the initial conditions and the environment. The ant algorithm has already been used in the Data Grid for load balancing [36] [37].

In our framework the task of the ant is to find a replica location that fulfills the RTT and memory requirements. Ants walk from one ant container to another for a specified number of steps (hops), and return to the delegate agent afterwards. In the current prototype ants travel around the DKS ring, but in the future work we intend to investigate how efficiency of the replica location service (service response time) and quality of replica placement depend on different options in the ant algorithm, such as the number of ants, the starting points, the walk path, etc. Note that here the structured P2P overlay networks can have advantage over the unstructured ones because we can better control the ant paths and therefore reduce the number of nodes redundantly visited by multiple ants.

Ants are implemented as Java objects that encapsulate the ant state. Currently, the state includes a name of the file, the number of completed hops, the current minimum RTT that ant has measured and the corresponding position, the delegate agent position, and the RTT requirement and the user position.

3.2.2. Replica selection using the ant mechanism is illustrated on Figure 3. First, the user issues a replica selection request by creating a resource in a notification service (1) as described in Section 3.1.3. The user also subscribes to the resource properties of replica selection results in the notification service so that when the search process is complete, the user is notified. Next, the user submits the replica selection parameters (the file name, RTT and other QoS metrics, and user's available storage space size) to agent service (2), which will be its delegate agent. Note that the user can also choose a remote agent service in the DKS network located using the NLS. Figure 3 shows that the chosen delegate agent is on the Grid node D. After the delegate agent receives the client request, it queries the RLS to find existing replica positions. From one of these positions (Grid node A in Figure 3), the agent gets the file size and decides whether the file could be placed in the user node. This step is not shown in Figure 3. If the file is too large, the delegate agent contacts the agents at the existing replica positions in order to decide whether any of those replicas satisfies the RTT requirement (3) (4). If none of them does, the delegate agent motivates the agents at the existing replicas (5) to start searching for replica location using the ant mechanism. Each motivated agent sends out several ants (6) to the nodes in the first level (level 0) of DKS routing table. In Figure 3 ants are sent to nodes A and B. The ant container can provide ants with information left by ants that have already passed the container. If information about a better position(s) was left behind by other ant(s), the ant chooses that position as its next destination. Our preliminary performance evaluation experiments indicate that this behavior shortens ant exploration time. After walking a number steps (three steps in Figure 3), all ants return to the delegate agent where the best position is chosen (7). The ant can also be instructed to return to the delegate agent as soon as it finds the *first* position that fulfills the requirements. If the delegate agent discovers an appropriate position, it makes a new replica in this position (8) by a third party transfer, registers it in RLS, reports this selection result to the Statistics service, and changes the resource properties associated with selection results in the notification service to inform the user (9). The file is copied to the new position from the existing replica that has the shortest RTT to the new position.

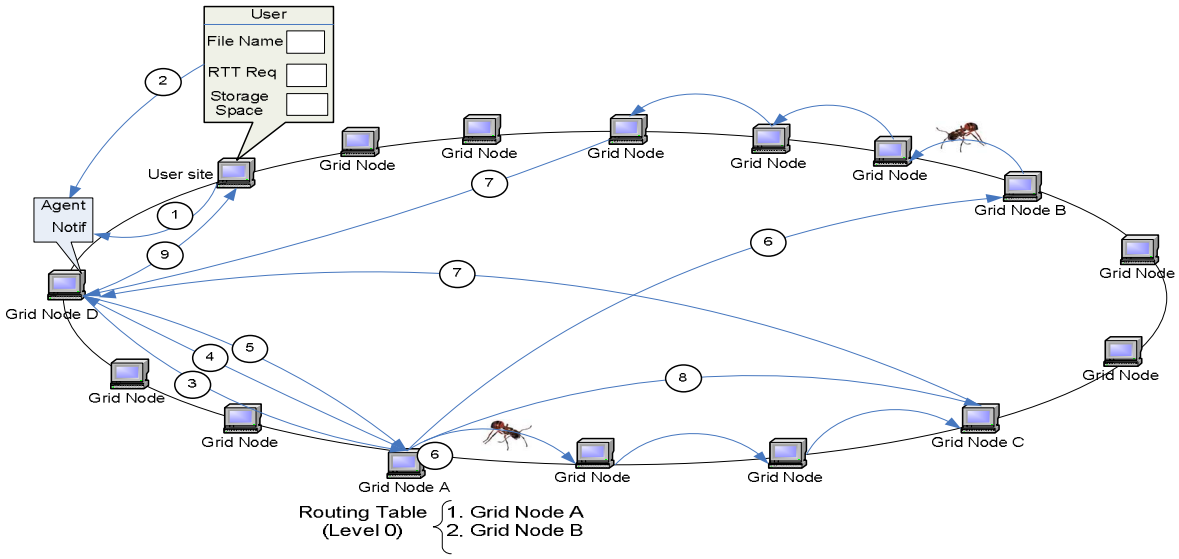

Figure 3 Replica selection using ants

4. Preliminary Performance Evaluation

In this paper, we report results of preliminary performance evaluation of the prototype of Replica Management Framework, leaving more detailed evaluation to our future work. We profiled the prototype in order to see how different parts of the framework contribute to the total service time.

In our evaluation experiments, a client submits replica selection requests to our Replica Management component that includes Replica Location Service (RLS), Notification service and the three lower-level components deployed in a GT4 container. The client computer is a PC with Intel Pentium processor 1.6 GHz, 512MB RAM and Intel 802.11b wireless card. The client requests to place a replica of a specified file with the RTT requirement of 100 milliseconds indicating that the local storage space the client can provide is zero, i.e. that a replica is not allowed to be placed at the client site. The experimental Grid includes several nodes connected by the DKS middleware in the DKS ring. There is at least one replica of the requested file in the DKS ring and positions of all existing replicas are recorded in RLS. The RTT values are randomly generated according to the uniform distribution between 1 and 1000 ms.

We divide the replica selection process into the following phases and estimate duration of each phase.

1) *Submit a replica placement request* is the phase when the client creates a resource in the Notification service and subscribes to its properties for result notification. The resource contains client requirements for replica placement.
2) *Query RLS* is the phase when the delegate agent queries the RLS for positions of replicas of the requested file.
3) *Judge the existing replica site* phase includes two steps: (1) to get the file size from one of the replica positions and to check whether the client site has enough storage space to store a replica of the file; and (2) to decide whether the existing replica site can satisfy the RTT requirement.
4) *Seeking appropriate positions by motivating ants* is the phase to use ants walking around the DKS ring to find replica position(s) that fulfill the RTT requirement.
5) *Register the new replica* is the phase to copy the replica to the new site and register its position in RLS. In our profiling experiments, the file copy time that is takes to create a new replica, was not counted in the service time as it depends on actual file size and GridFTP configuration which may vary greatly between different sites. In our future work we plan to include an optional mode in which the replica placement service can compute (predict) the file copy time and interact with the client to check whether it is worth copying the replica to a new location. It might be inefficient to copy the replica to a new position because the file can be too large or/and the link might be too slow.
6) *Inform the client* is the phase to update the resource properties in the Notification service with the results and, in this way, trigger the Notification service to notify the client.

To enforce the Replica Management to find a new position for a replica, all Grid sites with existing replicas generate RTTs longer than the required RTT so that the sites do not meet the RTT requirement in the 3-rd phase (*Judge the existing replica site*). In our future work, we plan to assess how the service time depends on the number of ants, the size of the DKS ring, its population and a distribution of placements (sites) in the ring that fulfill the RTT requirement. Assuming that the *k* parameter in the DKS is 2, the requested site motivates two ants that walk a specified number (5 in our experiments) of steps. The client submits the request several times in order to obtain the average performance shown in Table 1.

We observe the large variance of duration of the phase 4 in which the Replica Management seeks appropriate positions by motivating ants. This is due to the fact that the RTT value is generated randomly and the ant could encounter different values of RTT as it walks around the DKS ring. In real-world WAN environments where network traffic may change greatly, the variance of time in this phase can be large.

Table 1: Time anatomy of replica placement

phase	time (ms)	standard deviation	total service time, %
1. Submit request	426	32	13,2%
2. Query RLS	144	15	4,5%
3. Judge existing replicas	618	24	19,1%
4. Seek positions	1155	957	35,8%
5. Register the new replica	271	19	8,4%
6. Inform the client	615	20	19%

Table 1 also shows the distribution of time consumed by different phrases. As expected, the phrases 3 and 4 rank the top 2, as they consume 19,1% and 35,8% of the total profiled service time, respectively. Both phases include many inter-node communications that is time-consuming. Therefore, in order to decrease the replica selection time, we need to find a way to reduce the need to communicate with remote services in phases 3 and 4 by, for example, recording (caching with some TTL) and using results of previously resolved requests in order to predict replica positions without assessing existing replicas and seeking for new positions by motivating ants. The phases 1 and 6 also significantly contribute to the service time. They are both related to the operations on the WS resources, which may account for the large time consumption.

5. Conclusions and Future Work

We have presented our approach to building a replica management service for Grids constructed with GT4. In order to achieve good scalability, high reliability and high quality (i.e. short service response time and optimized placement of replicas according to client requirements) of the replica management, we use the DKS P2P middleware to build a structured self-organizing overlay network of Grid nodes with DHT functionality. The DHT is used as an index service with multiple access points to locate replicas given a file name; whereas the DKS routing table is used to direct ant agents to find positions for new replicas. High reliability of DHT is achieved by automatic and reliable data replication and locally-atomic node join and leave operations. We use a distributed ant algorithm for replica placement that fulfills client requirements. Using several ants allows to reduce service time. Ants can be instructed to find a first suitable position or currently the best position. In order to reduce service time, the service can place replicas at the first found position that fulfills the client requirements but might be not the best one. Client may request to find either the best or any position for a replica. In order to improve QoS we propose to run in background a dedicated distributed process that can optimize replica placement and control replica lifetime using the Statistic service for monitoring replica placement and replica access patterns. In particular, it can replicate “popular” and remove unpopular replicas.

In order to evaluate our approach we have implemented a system prototype using the DKS middleware and GT4. We have profiled the replica management prototype in order to indicate potential performance bottlenecks and to identify ways for future enhancements of the framework and the prototype. In our future work, we intend to evaluate how different design options such as the number of ants, ant walking algorithm and the stigmergy behavior affect performance and quality of the service.

In the current design and implementation, we use the RTT requirement specified by client when making replica selection or placement decisions and ignore some other factors that may affect the replica access time in such a way that the required access time indirectly specified by the RTT requirement is not met even though the RTT estimates made by the Replica Management fulfill the requirement. This may reduce QoS of the Replica Management framework that can be estimated as follows: how many “good” (in term of access time) replicas it can locate/place and how “good” they are. To improve the quality of the Replica Management framework, we develop a special Statistics component that observes client access patters and replica placement and, based on these observations, proactively optimizes replica placement by removing redundant replicas, creating new and redistributing existing replicas of files which have been accessed most often. In our future work we plan to take

more factors into consideration when making placement decisions and controlling replicas, such as storage response time, replica transfer time, file access pattern, load of the storage elements.

References

[1] D. Dullmann, W. Hoschek, J. Jaen-Martinez, B. Segal, "Models for replica synchronization and consistency in a data Grid", in *Proc. of the 10th IEEE Int Symposium on High Performance Distributed Computing (HPDC'01)*, 2001.

[2] L. Guy, P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger, "Replica Management in Data Grids", GGF5 Working Draft, July 2002.

[3] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, B. Moe, "Wide Area Data Replication for Scientific Collaborations", in *Proc. of the 6th IEEE/ACM Int. Workshop on Grid Computing (Grid'05)*, 2005.

[4] Globus Toolkit 4. URL: http://www.globus.org/toolkit/

[5] Distributed K-ary System (DKS), http://dks.sics.se/

[6] L. Alima, S. El-Ansary, P. Brand and S. Haridi, "DKS(N, k, f) A family of Low-Communication, Scalable and Fault-tolerant Infrastructures for P2P applications", in *3rd Int. Workshop on Global and P2P Computing on Large Scale Dist. Systems (CCGRID'03)*, Tokyo, Japan, May 2003.

[7] GridFTP Protocol Specification (Global Grid Forum Recommendation GFD.20). March 2003.

[8] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology", IBM, Oct 15, 2001.

[9] John H. Holland, *Hidden Order: How Adaptation Builds Complexity*, Addison Wesley, 1996.

[10] J. Crowcroft, T. Moreton, I. Pratt, A. Twigg, Chapter "Peer-to-peer Technologies" in *The Grid: Blueprint for a New Computing Infrastructur*, Morgan Kaufmann, 2004.

[11] M. Cai, A. Chervenak, and M. Frank, "A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table", in *Proc. of the SC2004 Conference*, Nov 2004.

[12] A. Chazapis, A. Zissimos, N. Koziris. A peer-to-peer replica management service for high-throughput Grids. In *Proc. of the Int. Conf. on Parallel Processing*, IEEE 2005.

[13] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proc. of the Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 177-190. ACM, 2002.

[14] S.C. Rhea and J. Kubiatowicz, "Probabilistic location and routing", In *Proc. of the 21st Annual Joint Conf. of the IEEE Computer and Communications Societies*, June 2002.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", in *Proceedings of the ACM SIGCOMM' 01 Conference*, August 2001.

[16] B.Y.Zhao, J.D. Kubiatowicz, and A.D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing", UCB//CSD-01-1141, U.C.Berkeley, April 2001.

[17] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", In *IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware)*, pages 329-350, Nov. 2001.

[18] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties", in *Proc of the 4th USENIX Symp. on Internet Technologies and Systems*, March, 2003.

[19] A. Ghodsi, L. Alima, and S. Haridi, "Low-Bandwidth Topology Maintenance for Robustness in Structured Overlay Networks", In *38th Int. HICSS Conference*, Springer 2005.

[20] A. Ghodsi, L. Alima, and S. Haridi, "Symmetric Replication for Structured Peer-to-Peer Systems", in *The 3rd Int Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Trondheim, Norway, August 2005.

[21] L. Alima, A. Ghodsi, P. Brand, and S. Haridi, "Multicast in DKS(N, k, f) Overlay Networks", in *Proc. of the 7th Int. Conf. on Principles of Distributed Systems*, Dec 10-13 2003, La Martinique, France, Springer, 2004.

[22] A. Ghodsi, L. Alima, S. el-Ansary, P. Brand, and S. Haridi, "Self-Correcting Broadcast in Distributed Hash Tables", in *Parallel and Distributed Computing and Systems*, ACTA Press, 2003.

[23] S. El-Ansary, L. Alima, P. Brand, S. Haridi, "Efficient Broadcast in Structured P2P Networks", in *Proc. Of the 2nd Int. Workshop On Peer-To-Peer Systems*, Springer 2003.

[24] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems", *IFIP Int. Conference on Network and Parallel Comp.*, LNCS 3779, pp 2-13, Springer, 2005.

[25] Globus: OGSA. http://www.globus.org/ogsa

[26] W. Allcock, I. Foster, R. Madduri, "Reliable Data Transport: A Critical Service for the Grid", *Building Service Based Grids Workshop, Global Grid Forum 11*, June 2004.

[27] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe, "Wide Area Data Replication for Scientific Collaborations", in *Proc. of 6th IEEE/ACM International Workshop on Grid Computing (Grid2005)*, 2005.

[28] A. Chervenak, E. Deelman, I. Foster, *et al..*, "Giggle: A Framework for Constructing Scalable Replica Location Servcies", In *Proc. of Supercomputing'2002*, 2002.

[29] P. Maymounkov and D. Mazieres, "Kademlia. A peer-to-peer information system based on the XOR metric", in *Proc. 1st Int. Workshop on Peer-to-Peer Systems,* Cambridge, USA, March 2002, Springer 2002.

[30] P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger, "File-based Replica Management", *Future Generation Computer Systems*, 22(1):115-123, Elsevier 2005.

[31] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini, "Design of a Replica Optimisation Framework", *DataGrid-02-TED-021215, CERN,* Dec 2002.

[32] H. Stockinger, "Distributed Database Management Systems and the Data Grid", In *Proc. Of the 18th IEEE Symposium on Mass Storage Systems*, IEEE 2001.

[33] FAM. http://oss.sgi.com/projects/fam

[34] M. Resnick, *Turtles, Termites, and Traffic Jams*, MIT Press, 1994.

[35] G. Theraulaz and E. Bonabeau, "A Brief History of Stigmergy", *Artificial Life,* 22(5): 97-116, MIT Press 1999.

[36] J. Cao, "Self-Organizing Agents for Grid Load Balancing", in *Proc. of 5th IEEE/ACM Int. Workshop on Grid Computing (GRID'04)*, pp. 388-395, 2004.

[37] A. Montresor, H. Meling, and Ö. Babaoglu, "Messor: Load-Balancing through a Swarm of Autonomous Agents", in *Proc. of the 1st Int. Workshop on Agents and Peer-to-Peer Computing*, Bologna, Italy, pp. 81-89, July 2002.