High Performance Hybrid Information Service Architecture

Mehmet S. Aktas[1, 2], Geoffrey C. Fox[1, 2, 3], Marlon Pierce[1]

[1] Community Grids Laboratory, Indiana University
501 N. Morton Suite 224, Bloomington, IN 47404
{maktas, gcf, mpierce}@cs.indiana.edu
http://www.communitygrids.iu.edu/index.html
[2] Computer Science Department, School of Informatics, Indiana University
[3] Physics Department, College of Arts and Sciences, Indiana University

Abstract. We introduce a distributed high performance hybrid Information Service Architecture, which forms a metadata replica hosting system to manage both highly-dynamic, small-scale and relatively-large, static metadata associated to Grid/Web Services. We present an empirical evaluation of the proposed architecture and investigate its practical usefulness. The results demonstrate that the proposed system achieves high-performance and fault-tolerance with negligible processing overheads. The results also indicate that efficient decentralized Information Service Architectures can be built by utilizing publish-subscribe based messaging schemes.

1 Introduction

Information Services address the challenging problems of announcing and discovering resources in Grids. Existing implementations of Grid Information Services present several challenges. Firstly, most of the existing work have centralized components and do not address high performance and fault-tolerance issues [1, 2]. Secondly, previous work do not address distributed information management requirements of dynamic Grid/Web Service collections such as efficient request distribution and replica-content placement strategies [2]. Thirdly, none of the existing work adopts changes in client demands for performance optimization [1, 2]. Fourthly, previous work do not provide uniform interface for publishing and discovery of both dynamically generated and static information [1, 2]. We therefore see this as an important area of investigation.

To present the applicability of this investigation, we identify metadata management requirements of two application use domains: Global Multimedia Collaboration System (GlobalMMCS [3]) and Pattern Informatics Geographical Information System Grid (PI GIS-Grid [4]). GlobalMMCS is a peer to peer collaboration environment, where videoconferencing sessions, with any number of widely distributed services, can take place. GlobalMMCS requires persistent archival of session metadata to provide replay/playback and session failure recovery capabilities. The PI GIS-Grid is a workflow-style Grid application which requires storage of transitory metadata needed to correlate activities of participant entities. Both application domains require a decentralized, fault-tolerant metadata hosting environment which can support scalability of large-scale, read-mostly, quasi-static information and performance requirements of small-scale, highly-updated, dynamic information. Although much work has been done on information management in Grid Information Services, to our best knowledge,

none of the previous work addresses the metadata management requirements of both of types of application use domains.

In this paper, we propose a Grid Information Service that addresses aforementioned challenges of previous work and metadata management requirements of target application use domains. The main novelty of this paper is to describe the architecture, implementation and evaluation of Hybrid Grid Information Service (Hybrid Service) supporting both distributed and centralized paradigms and managing both dynamic, small-scale and quasi-static, large-scale metadata. The implications of this research are four-fold. First is to describe a Grid Information Service architecture, which responds to client demand changes. To our best knowledge, the proposed work is a pioneer approach that utilized multi-publisher, multicast communication to dynamic replication methodology in Grid Information Services without relying on information available on the Internet routers. Second is to describe the architecture of a fault tolerant and high performance Grid Information Service linking publish-subscribe based messaging schemes with associative shared memory platform for metadata management. Third is to describe the architecture of a Hybrid Service that integrates different Grid Information Services by using unification and federation concepts. Fourth is to identify and analyze the key factors that affect the performance of a metadata-system with multi-publisher, multicast communication strategies.

The organization of the rest of the paper is as follows. Section 2 reviews the background work. Section 3 discusses the Hybrid Grid Service Architecture. Sections 4-7 explain the design decision in fundamental issues of Hybrid Service Replica Hosting System: service discovery model, replica-content placement, consistency enforcement, access-request distribution. Section 8 analyzes the performance evolution of the prototype implementation. Section 9 concludes the paper with summary and future research directions.

2 Background

Existing service metadata discovery architectures can be broadly categorized as centralized and decentralized by the way they handle service information storage. In centralized approach, there is a central look-up mechanism where all services are dependent on one node. Mainstream service discovery architectures (such as JINI [5]), which have been developed to provide discovery of remote services residing in distributed nodes in a wired network, are based on a central registry for service registration and discovery. Most central service discovery solutions (such as UDDI [6] and its extensions [7]) are database-based solutions and require disk accesses to query/publish metadata. In decentralized approach, the research trend mainly focuses on decentralized search, where all the peers of the system actively participate the discovery process with no central database. Some of previous solutions (such as Chord [8]) with pure decentralized storage models focused on the concept of distributed hash tables (DHT), which assumes the possession of an identifier that identifies the service to be discovered. Some other decentralized approaches, such as Amazon Simple Storage Service (Amazon S3 [9]), focused on a peer-to-peer file distribution protocol called Bittorent, which is designed to distribute large amounts of widely distributed data onto peers, where each peer is capable of requesting and transferring data. Despite of their important features, previous metadata discovery solutions do not address the application requirements of aforementioned target application use domains. The centralized registry approach has good performance but presents limita-

tions such as single point of failure problem. Likewise, the decentralized registry approach also presents some limitations, as the resource placement at nodes is strictly enforced in structured peer-to-peer networks, which in turn cause a heavy overhead on the bootstrap of the network. The DHT approach is good on routing messages but limited to primitive query capabilities (key-based queries) [10]. The Bittorent approach has also challenges as its performance depends on the capacity of a centralized node (called tracker), which keeps track of peers in the network.

The proposed system differs from previous metadata discovery architectures as described below. Firstly, it supports both distributed and centralized paradigms in one hybrid architecture by linking publish-subscribe based messaging schemes with associative shared memory platform for metadata management. Secondly, apart from DHT based systems, it introduces a multi-publisher, multicast messaging infrastructure and communication protocol, which in turn enable advanced query capabilities to be routed between the network nodes. Thirdly, it explores a caching mechanism that would provide persistency and performance capabilities together. Existing work on Service Registries relies only on database solutions, which is good for persistency reasons; but requires I/O accesses every time when metadata access happens. Fourthly, it provides mutual exclusive access to the shared data, while none of the previous work on Service Registries support data-sharing. The proposed system design is discussed in Section 3 in details.

Another way of analyzing service discovery architectures could be based on the formation of the network and the way of handling discovery request distribution. In traditional wired networks, network formation is systematic since each node joining the system is assigned an identity by another device in the system [11]. Example wired network discovery architectures such as JINI [5] and Service Location Protocol [12] focus on discovering local area network services provided by devices like printer. In ad-hoc networks (unstructured peer-to-peer systems), there is no controlling entity and no constraint on the resource dissemination in the network. Existing solutions for service discovery for ad-hoc networks (e.g. pervasive computing environments) can be broadly categorized as broadcast-driven and advertisement-driven approaches [13]. In broadcast-driven approach, a service discovery request is broadcasted throughout the discovery network and each node satisfying the request sends a unicast response message. In advertisement-driven approach, each service advertises itself to all available peers with a “hello” message and all peers cache the advertisement of the broadcasting service. The WS-Discovery Specification [14] supports both broadcast-driven and advertisement-based approaches by using a hardware multicast mechanism. To minimize the consumption of network bandwidth, this specification supports the existence of registries and defines a multicast suppression behavior if a registry is available on the network. Existing solutions to service discovery architectures do not address the requirements of our target application domains. The traditional wired-network based architectures are limited, as they depend on a controlling entity, which assigns identifiers to participating entities. The ad-hoc networks have also some limitations. If the size of the network is too big, the broadcast-driven approach has a disadvantage, since it utilizes significant network bandwidth, which in turn creates a large load on the network. The advertisement-driven approach does not scale, as the network nodes may have limited storage and memory capability. The WS-Discovery approach is promising to handle metadata in peer-to-peer computing environment; however, it has the disadvantage of being dependent on hardware multicast for message dissemination.

The service discovery model of the proposed system differs from previous work as it does not have to rely on wired network discovery architecture or have any constraints on resource dissemination in the network. There are similarities between the proposed approach and the WS-Discovery approach, as they both support broadcast and advertisement-based approaches in their service discovery model. Different from WS-Discovery approach, the Hybrid Service utilizes a software-multicast model by utilizing its own multi-publisher, multicast publishes-subscribe based messaging scheme. The Hybrid Service discovery model and communication mechanism are discussed in Section 4 in details.

Replication is a well-known and commonly used technique to improve the quality of metadata hosting environments. Replication can be categorized as permanent-replication and server-initiated replication. Permanent-replication keeps the copies of a data permanently for fault-tolerance reasons, while the server-initiated replication creates the copies of a data temporarily to improve the responsiveness of the system. Sivasubramanian et al [15] give an extensive survey on designing and developing replica hosting environments, as does Robinovich in [16], paying particular attention to dynamic replication. These systems may be discussed under following design issues: a) distribution of client requests, b) selection of replica servers for replica placement, and c) consistency enforcement. Distribution of client requests is the problem of redirecting the request to the most appropriate replica server. Some of the existing solutions to this problem rely on the existence of a DNS-Server [16, 17]. These solutions utilize a redirector/proxy server that obtains physical location of a collection of data-systems hosting a replica of the requested data, and choose one to redirect client's request. Replica placement is another issue that deals with selecting data hosting environments for replica placement and deciding how many replicas to have in the system. Most of the solutions [16, 17], which apply to dynamic replication, assume all data-hosting servers to be ready and available for replica placement and ignore "dynamism" both in the network topology and in the data. In reality, data-systems can fail anytime and may present volatile behavior, while the data is being updated. The consistency enforcement issue is about ensuring all replicas of the same data to be the same. In [18] Tanenbaum gives a survey of different consistency enforcement approaches, implementations, and update protocols.

The fault-tolerance aspects of the proposed system's architecture differ from previous work on permanent replica-content placement, as some of its application use domains are highly dynamic. To this end, the Hybrid Service supports both permanent and dynamic replication. There are similarities between the Robinovich's dynamic replication methodology and our approach, since the proposed system adopts slightly modified versions of the replica selection and dynamic replication algorithms introduced by Rabinovich. Different from the Rabinovich's work, the Hybrid Service utilizes its own communication protocol, which does not rely on the information available on the Internet Routers to locate network nodes. There are also differences in target application use domains of the two systems: Rabinovich's approach is designed for web replica hosting systems, while the Hybrid Service is designed for distributed, service oriented architecture based Grid applications. The Hybrid Service system's design rationale in deciding the strategies for replica-content placement, consistency enforcement and request distribution are discussed in Sections 5-7 in the same order in details.

3 The Hybrid Grid Information Service

We designed and built a Hybrid Grid Information Service (Hybrid Service) to support handling and discovery of metadata associated with Grid/Web Services in Grid applications. The Hybrid Service is an add-on architecture that interacts with the local information systems and unifies them in a higher-level hybrid system. It provides a unifying architecture where one can assemble metadata instances of different information services. [19] describes the semantics of the Hybrid Service in details. In this paper, our main focus is to highlight the replica hosting environment aspects of the Hybrid Service and investigate various research issues related discovery, distribution and consistency.

Figure 1 illustrates general view of the Hybrid Service (See the picture on the left in Figure 1). The proposed hybrid system introduces uniform access and information resource management abstraction layers: First layer supports one-to-many communication protocols, while the latter manages one-to-many information service implementations. The prototype of the system unifies the two local information service implementations: WS-Context and Extended UDDI. It supports their communication protocols and utilizes publish-subscribe based messaging systems to enable communication in the information service network.

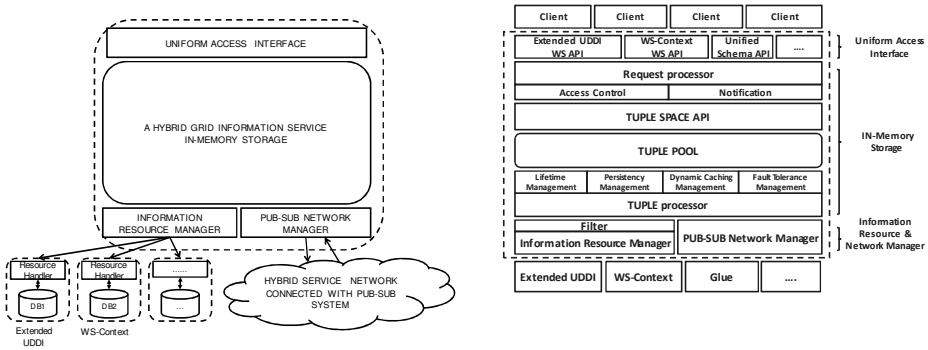

Figure 1 The picture on the left illustrates the general view of the Hybrid Service. The picture on the right illustrates the abstraction layers of the Hybrid Service from top-to-bottom. The dashed box indicates the Hybrid Service.

Figure 1 also illustrates a detailed view of the Hybrid Service (See the picture on the right in Figure 1). (1) Uniform Access Interface layer, which consists of multiple XML APIs, provides unified access to the system. (2) The Request-processing layer extracts incoming requests, notifies clients of the state changes in metadata and enforces controlled access to the in-memory storage. (3) TupleSpaces Access API provides access to a Tuple Pool (that is a generalized in-memory storage mechanism, implemented based on JavaSpaces Specification [20]) and supports all query/publish operations that can take place on the Tuple Pool. (4) The Tuple Processor layer processes metadata stored in the Tuple Pool and provides lifetime management, persistency management, fault-tolerant management and dynamic caching management of metadata. (5) The Filtering layer manages the filtering based on the user-defined mapping rules to provide transformations between the Unified Schema (i.e. a unified schema is an integration of XML schemas of different information services) instances and local schema instances. (6) The Information Resource Manager layer manages low-level information service

implementations and provides decoupling between the Hybrid Service and sub-systems. (7) The Pub-Sub Network layer manages communication between Hybrid Service instances by utilizing publisher and subscriber sub-components to provide communication among the service instances.

Figure 2 illustrates the distribution in Hybrid Service and shows N-node decentralized services from the perspective of a single service interacting with two clients. To achieve communication among the network nodes, the Hybrid Service utilizes a topic-based publish-subscribe software multicasting mechanism. This is a multi-publisher, multicast communication mechanism, which provides message-based communication. In the prototype implementation, we use an open source implementation of publish-subscribe paradigm (NaradaBrokering [21]) for message exchanges between peers. We identify following fundamental issues of designing the Hybrid Service Replica Hosting System: service discovery, replica-content placement, consistency enforcement, and request routing. These issues are discussed in the following sections in the same order.

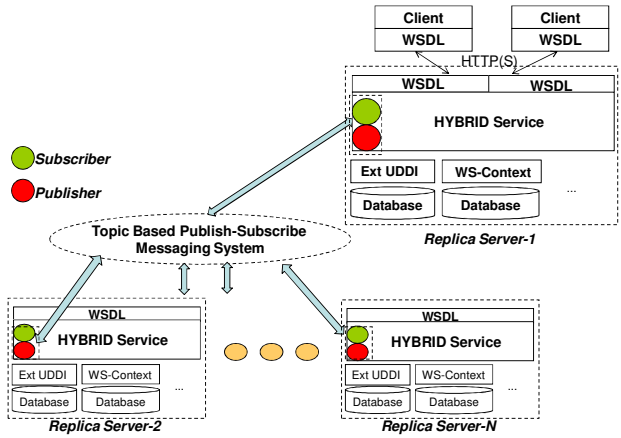

Figure 2 Distributed Hybrid Services.

4 Service Discovery

The Hybrid Service has a multicast discovery model to locate available services. In this model, the communication between network nodes happens via message exchanges. These messages are Server-Information Request and Response messages.

Server-Information Request and Response messages: A Hybrid Service node advertises its existence when it first joins the network with a message, the Server-Information Request. The purpose of this message is two-fold. First is to inform other servers about a newly joining server. Second is to refresh the replica-server-information data structure with the updated information (such as proximity and server load) every so often. This message is broadcasted through publicly known topic to every other available network nodes. The proximity between the initiator and the individual network nodes is calculated based on the elapse-time between

sending off the Server-Information Request and receiving the Server-Information Response message. The response message is sent back by unicast over a unique topic to the initiator and contains the server load information of the responding network node.

Service Discovery Model: Each Hybrid Service network node subscribes to the multicast channel (publicly known topic) to receive Server-Information Request messages. On receiving this request message, each node sends a response message, Server-Information Response message, via unicast directly to the newcomer Hybrid Service. This way, each node makes itself discoverable to other nodes in the system at the bootstrap. Each Hybrid Service node constructs a replica-server-information data structure about other available replica servers in the system. This data structure contains information about decision metrics such as server load and proximity. Each node keeps its replica-server-information data structure refreshed by sending out Server-Information Request messages periodically to obtain up-to-date information. This way each node keeps track of status information of the available network nodes.

5 Replica-content placement

The Hybrid service utilizes both permanent and dynamic (temporary) replication to enhance reliability and performance. The former replication technique is used as backup facility to enhance reliability, while the latter is used for performance reasons. Permanent copies are used to at least keep the minimum required number of replicas for the same data. Dynamic (temporary) copies are used to reduce access latency, as they are replicated onto servers in the proximity of demanding clients. The Hybrid service uses following two messages to provide replica-content placement: Context Storage Request and Response.

Context Storage Request and Response messages: A Hybrid Service node advertises the need for storage with a request message, the Context Storage Request. The purpose of this message is two-fold. First is to assign handling of the storage operation to those Hybrid Service nodes that are selected based on the replica-server selection policy. Second is to ask another Hybrid Service node to replicate or take over maintaining a context to enhance the overall system performance. With this message, the system is able to relocate/replicate contexts in the proximity of demanding clients. It is used in dynamic replication process and enables relocation/replication of contexts due to changing client demands. The Context Storage Request message is unicast over a unique topic to the selected replica server(s). By listening to its unique topic, each existing node receives a Context Storage Request message, which in turn includes the context under consideration. On receipt of a Context Storage Request message, a Hybrid Service node stores the context and sends a Context Storage Response message to the initiator. The Hybrid Service stores the context either as a permanent-copy or server-initiated (temporary) copy based on whether the context is being created for fault-tolerance reasons or performance reasons. The purpose of the response message is to inform the initiator that the answering node hosts the context under consideration. This message is also sent out by unicast directly to the initiator over a unique topic.

Decision metrics: The Hybrid Service takes both server load and proximity decision metrics into account when making replica-content placement decisions. The server load metric is represented with the following two factors: a) topical information (i.e. fraction of total unique

topics due to a given server on the network) and b) message rate (i.e. number of messages, issued by end-users, within a unit of time). Server load is periodically recorded and it reflects the average load of a Hybrid Service at a given time interval. The proximity metric is represented by the distance in network space between Hybrid Service instances. The proximity metric information is obtained periodically by sending ping requests (Server-Information Requests) to the available network nodes in the system through publish-subscribe system.

Permanent Replication: The Hybrid Service starts replica-content placement process (i.e. the distribution of copies of a context into replica hosting environment), when a new context is inserted to the Hybrid Service. This is needed to create certain number (predefined in the configurations file) of permanent replicas. On receipt of a client's publish request, an existing node checks whether or not to honor the request in the local storage by checking the load metric (see next paragraph for a discussion on how load is computed) with a maximum server load watermark. The maximum server load watermark determines the maximum load on the system. If a node is capable of storing, it performs the storage operation, and then starts the replica content placement process. Otherwise, it omits the storage operation and directly starts the replica-content placement process. The first step in replica content placement is to execute replica-servers selection (see following three paragraphs for further discussion on selection policy). Once the replica-server selection is completed, the initiator sends unicast message (Context Storage Request message) to the selected replica-servers. On receipt of a storage request, a replica server stores the context as a permanent-copy, followed by sending a response (Context Storage Response message) directly to the initiator (via unicast). The purpose of storing permanent-copy is for fault-tolerance. The number of permanent replicas is predefined with *minimum-fault-tolerance-watermark* in the configurations file and will remain the same for fault-tolerance reasons.

The load metric is recorded in a dynamic fashion periodically and reflects an indication of server load since the previous measurements. At each measurement interval, the new server load is calculated and recorded. The server load metric is represented by a vector reflecting multiple components. In the prototype implementation, we used two components to represent the load: topic information and message rate. This can easily be extended by including more components (such as computational load and storage utilization) into the server load vector. The value for the first component is estimated by finding the fraction of total unique topics due to the server under consideration on the whole network. This values is computed by dividing up the total number of local unique topics (i.e. # of unique topics created on the local storage) to the total number of global unique topics (i.e. # of unique topics created in the whole network). The value for the second component is computed by the number of messages issued by end-users within a unit of time. The server load metric is represented by a single value and computed by the product of these two server load component at each measurement interval.

The proximity metric is also recorded in a dynamic fashion and reflects the proximity since the previous measurements. The proximity is represented by the distance in network space between Hybrid Service instances and obtained periodically by sending ping requests to the available network nodes through publish-subscribe system. The proximity is measured by latency in the ping request, which gives the distance information between the two Hybrid Service instances. Each node keeps a proximity vector, i.e. distance vector, and refreshes it periodically.

To select replica servers for replica-content placement, we adopt a replica selection algorithm introduced by Rabinovic et al [16] and integrate it with our implementation. Based on the Rabinovich's approach, the replica server selection policy takes into account two decision metrics: server load and proximity. The algorithm begins by identifying a replica server with lowest server load and a replica server that is closest to the client. It then chooses the replica (among these two) by comparing the server load of the least-loaded replica with the server-load divided by two of the closest replica. This way, the algorithm chooses the closest replica server for permanent replication unless it is overloaded. The replica server selection process is repeated on target replica servers, until the initiator selects predefined number (*minimum-fault-tolerance-watermark*) of replica servers for replica-content placement. The initiator Hybrid Service chooses the best-ranked server among the selected replica-servers as the primary-copy to enforce consistency.

Dynamic replication: The Hybrid Service adopts a dynamic replication methodology introduced by Rabinovich et al [16] and integrates it with its communication infrastructure. Rabinovich's approach perform replica-content placement for two reasons: a) to reduce the load on a replica server, b) to adopt changes in the client demands. In our prototype implementation, we perform replication for the second reason, since our main interest is to create replicas, if it is only beneficial for client proximity. Each Hybrid Service *S* runs the dynamic replication algorithm with certain time intervals (*dynamic-replication-time-interval*) and re-evaluates the placement of the contexts that are locally stored. The dynamic replication algorithm begins by checking the local Hybrid Service if there are contexts that can be migrated or replicated onto other servers in the proximity of clients that presented high demand for these contexts. It does this by comparing the access request count for each context against some threshold values. If the total demand count for a replica *C* at a Hybrid Service *S* ($cnt_S(C)$) is below a ***deletion-threshold(S, C)*** and the replica is a temporary-copy, that replica will be deleted from local storage of Hybrid Service *S*. If, for some Hybrid Service *X*, a single access count registered for a replica *C* at a Hybrid Service *S* ($cnt_S(X, C)$) exceeds a ***migration-ratio***, that service (service *X*) is asked to host the replica *C* instead of service *S*. (Note that the migration-ratio is needed to prevent a context migrate back and forth between the nodes. In our investigation, we chose the *migration-ratio* value as % 60 based on the study introduced in [16]). This means service *S* wants to migrate replica *C* to service *X* which is in the proximity of clients that has issued enough access requests within the predefined time interval (*dynamic-replication-time-interval*). In this case, replica *C* will be migrated to service *X*. To achieve this, a Context Storage Request is sent directly to service *X* by service *S*. On receipt of a Context Storage Request, service *X* creates a permanent copy of the context, followed by sending a Context Storage Response message. If the total demand count for a replica *C* at service *S* ($cnt_S(C)$) is above a ***replication-threshold(S, C)***, then the system checks if there is a candidate Hybrid Service, which has requested replica *C*. If, for some Hybrid Service Y, a single access count registered for a replica *C* at service *S* ($cnt_S(Y, C)$) exceeds a ***replication-ratio***, that service (service *Y*) is asked to host a copy of replica *C*. (Note that, in order dynamic replication to ever take place, the replication-ratio is selected below the migration-ratio [16]. In our investigation, we chose the *replication-ratio* value as % 20.) This means service *S* wants to replicate replica *C* to service *Y* that is in the proximity of clients that has issued access requests for this context. In this case, replica *C* will be replicated to service *X*. To achieve this, a Context Storage Request is sent directly to service *X* by service *S*. On receipt of a Context Storage Request, service *X* creates a temporary copy of the context, followed by sending a Context Storage Response message.

6 Consistency enforcement

The Hybrid Service introduces two different models to address consistency requirements of aforementioned application use domains. The first model is for read-mostly applications. For these applications, the Hybrid Service allows clients to fetch copies of a context (permanent or temporary) freely from the metadata store, since read-only copies of a context are considered consistent. The second model is for consistency-sensitive applications with high update-ratio. For these applications, the Hybrid Service requires clients to subscribe to unique topics of contexts, so that they can be informed of the state changes immediately after an update occurs. To implement these consistency models, the Hybrid Service employs the primary-copy approach, i.e., updates are originated from a single site, to ensure all replicas of a data to be the same. In this section, we analyze the Hybrid Service implementation under three categories: "update distribution", "update propagation" and "primary-copy selection". The Hybrid Service performs these by using following messages: Primary-Copy Selection Request and Response, Primary-Copy Notification, and Context Update Request and Propagation.

Primary-Copy Selection Request and Response messages: To provide consistency across the copies of a context, updates are executed on the primary-copy host. If the primary-copy host of a context is down, the first Hybrid Service node, which does not receive any response from the primary-copy holding server, advertises the need for selection of primary-copy host with following message: Primary-Copy Selection Request. This message is sent out by multicast by the initiator Hybrid Service only to those servers holding the permanent-copy of the context under consideration. The Primary-Copy Selection Request message is disseminated over a unique topic corresponding to the metadata under consideration. We use the metadata key (UUID) as the topic, which all nodes, holding the permanent-copy of the metadata, within the system subscribe to. On receipt of a Primary-Copy Selection Request message, each node responds with the Primary-Copy Selection Response message directly to the initiator node. The purpose of this message is to inform the initiator about the permanent-copy of the context under consideration and give some information (such as hostname, transport protocols supported, communication ports) regarding how other nodes should communicate with the answering node. The response message is sent out by unicast over a unique topic. By listening to this topic, the initiator receives the response message from the answering node.

Primary-Copy Notification message: A Hybrid Service node uses a Primary-Copy Notification message to notify the newly selected primary-copy holder. This Notification message is disseminated by unicast directly to the newly selected node. By listening to its unique topic, each existing node may receive a primary-copy notification message, which in turn includes the assignment for being the primary-copy of the context under consideration. Each primary-copy holder of a given context subscribes to a unique topic (such as UUID/PrimaryCopy) to receive messages aimed to the primary-copy holder of that context.

Context Update Request and Propagation messages: A Context Update Request message is sent by a replica server to the primary-copy host to ask for handling the updates related with the context under consideration. This message is sent out via unicast by the initiator Hybrid Service directly to the primary-copy host over a unique topic. By listening to this topic, the primary-copy-host receives the context update request message. A Context Update Propagation message is sent by the primary-copy host only to those servers holding the context under

consideration. This message is sent via multicast to the unique topic of the metadata immediately after an update is carried out on the primary-copy to enforce consistency. By listening to this topic, each existing permanent-copy holder node receives a Context Propagation message, which in turn includes the updated version of the context under consideration.

Update distribution: The proposed system utilizes primary-copy approach to ensure all replicas of a data to be the same. The update distribution algorithm begins with by checking if the request contains a system-defined context key. If not, the system treats the request as if it is a new publication request. Otherwise, the system treats publication request as if it is an update request. An update operation is executed offline, i.e., just after an acknowledgement is sent to the client. If the primary-copy host is the initiator node itself, then the update is handled locally. If the primary-copy host is another node, then the update is forwarded to the primary-copy holder. The initiator service sends a message, Context Update Request, by unicast directly to the primary-copy-host for handing over the update operation. This message includes the updated version of the context under consideration. On receipt of a Context Update Request message, first, the primary-copy host extracts the updated version of the context from incoming message. Then, it updates the local context if the timestamp of the updated version is bigger than the timestamp of the primary-copy.

Update propagation: The proposed system utilizes push methodology for update propagation, multicast methodology for update dissemination and transmits the whole contents of updates. By using the push methodology, the Hybrid Service, holding the primary-copy of a context, propagates the updates immediately after an update occurs. By using the multicast methodology, the Hybrid Service propagates a context-update request to all available permanent-copy holding servers. The update propagation algorithm begins with by executing the update request at the primary-copy holding service. After the update process is completed, a Context Update Propagation message is sent to only those servers holding the permanent-copy of the context under investigation. The purpose of the Context Update Propagation is to reflect updates to the redundant copies immediately after the update occurs. On receipt of a Context Update Propagation message from the primary-copy, the initiator Hybrid Service node changes the status of the context under consideration from “updated” to “normal”. If there is no response received from primary-copy host within predefined time interval (*time-out_period*), the primary-copy host is decided to be down. In this case, the initiator node selects a new primary-copy host by using primary-copy selection algorithm. After a new primary-copy host is selected, the update distribution process is re-executed.

The system assigns a synchronized timestamp to each published context (newly written or updated). This is achieved by utilizing NaradaBrokering Network Time Protocol (NTP) protocol based timing facility. By utilizing this capability, the Hybrid Service gives sequence numbers to published data to ensure an order on concurrent write operations. This way, write/update requests are carried out on a data item x at primary-copy host s, in the order in which these requests are published into the distributed metadata store. However, this approach has also some practical limits, as the update rate is bounded by the timestamp accuracy of the synchronized timestamps.

Primary-copy selection: The primary-copy selection process is used to select a new primary-copy host, whenever the original primary-copy host is down. The primary-copy selection algorithm begins with by broadcasting a Primary-Copy Selection Request message to those servers holding the context to select the new primary-copy host. On receipt of a Primary-Copy Selection Request message, each replica-holding server issues a Primary-Copy Selection Response message. On receipt of the Response messages, the initiator obtains the information about nodes carrying the permanent copy of the context. Then the initiator selects the best replica server based on the aforementioned replica-server selection algorithm. After the selection is completed, a Primary-Copy Notification message is sent to the selected server naming it as the new primary-copy host. On receipt of a Primary-Copy Notification message, the permanent-copy holder becomes the new primary-copy holder and subscribes the unique address (/UUID/PrimaryCopy) corresponding to the primary-copy of the context.

7 Access request distribution

The Hybrid Service employs a broadcast based request distribution. Based on this scheme, if a query cannot be granted locally and requires external metadata, the request is broadcasted to those nodes hosting the requested metadata in the network at least to retrieve one response satisfying the request. This way the service is able to probe the network to look for a running server carrying the right information at the time of the query. The communication between network nodes for request access distribution happens via following messages: Context Access Request and Response.

Context Access Request and Response messages: A Hybrid Service node advertises the need for context access with the Context Access Request to the system. The purpose of the Context Access Request is to ask those servers, holding the context under demand, for query handling. This message is disseminated to only those nodes holding the context under consideration. This is done by multicasting the message through the unique topic corresponding to the metadata. (Note that we use UUID of the metadata as topic). By listening to this topic, each node, holding the context under consideration, receives a Context Access Request message, which in turn includes the context query under consideration. On receipt of a Context Access Request message, each Hybrid Service sends a Context Access Response message, which contains the context under demand, to the initiator. This message is sent out by unicast directly to the initiator over a unique topic. By listening to this topic, the initiator receives the response messages from nodes that answered the access request.

Request distribution: The Hybrid Service prototype implements a request distribution methodology, which is based on broadcast dissemination, where the requests are distributed to those servers holding the context under consideration. This approach does not require keeping track of locations of every single data located in the system. The request distribution algorithm begins with by issuing a Context Access Request message to the multicast group. This message contains minimum required information (such as context key) regarding the context in demand. On receipt of a Context Access Request message, a replica-holding Hybrid Service issues a Context Access Response message. Note that, each server keeps track of the count of access requests and the locations where access requests come from for each context. In turn, this enables the system to apply dynamic replication process and adapt to sudden bursts of client demands coming from a remote replica. This is why, if the access request is granted,

each server registers the incoming access request in the *access-demanding-server-information* data structure and increments the total *access-request-count* of the context under investigation. On receiving first Context Access Response message, the initiator Hybrid Service, obtains the context that can satisfy the query under consideration. Then a response message is sent back to inquiring client. The initiator only waits for responses that arrive within the predefined *timeout* value. If there is no available Hybrid Service node that can satisfy the context query within the *timeout* duration, the access process ends and a "not found" message is sent to the client.

8 Prototype Evaluation

An earlier evaluation study was conducted in [22] to investigate the performance of the prototype of the centralized version of the Hybrid Service. This study concluded that one can achieve noticeable performance improvements for standard inquiry/publish operations by simply employing an in-memory storage mechanism. In this research, we conduct an evaluation of the prototype of the Hybrid Service Replica Hosting System to understand its practical usefulness. To this end, the following research questions are being addressed:

- What is the effectiveness of the system in responding metadata access queries from the perspective of interacting clients?
- What is the cost of the access request distribution in terms of the time required to fetch a copy of a data (satisfying an access request) from a remote location?
- What is the effect of dynamic replication in the cost of the access request distribution in terms of the time required to fetch a copy of a data?
- What is the cost of the storage request distribution for fault-tolerance in terms of the time required to create replicas at remote locations?
- What is the cost of consistency enforcement in terms of the time required to carry out updates at the primary-copy holder?

Experimental setup environment: We explore the tradeoffs in choosing multi-publisher, multicast communication mechanism to implement a replica hosting environment. To do this, we conduct several experiments: effectiveness, distribution, dynamic replication, fault-tolerance and consistency enforcement. We leave out an extensive scalability experiment that would show the system work with high number of service nodes as a future study, as it is not the main focus of this evaluation. Thus, for the decentralized setting experiments, we have selected several nodes that are separated by significant network distances to facilitate the testing. The machines, used in these experiments, are summarized in Table 1.

Summary of Machine Configurations				
	Location	*Processor*	*RAM*	*OS*
gf6.ucs.indiana.edu	*Bloomington, IN, USA*	***Intel® Xeon™ CPU (2.40GHz)***	*2GB total*	*GNU/Linux (kernel release 2.4.22)*
complexity.ucs.indiana.edu	*Indianapolis, IN, USA*	***Sun-Fire-88, sun4u sparc SUNW***	*16GB total*	*SunOS 5.9*
lonestar.tacc.utexas.edu	*Austin, TX, USA*	***Intel(R) Xeon(TM) CPU 3.20GHz***	*4GB total*	*GNU/Linux (kernel release 2.6.9)*
tg-login.sdsc.teragrid.org	*San Diego, CA, USA*	***Genuine Intel IA-64, Itanium 2, 4 processors***	*8GB total*	*GNU/Linux*
vlab2.scs.fsu.edu	*Tallahassee, FL, USA*	***Dual Core AMD Opteron(tm) Processor 270***	*2GB total*	*GNU/Linux (kernel release 2.6.16)*

Table 1 Summary of the machines used in decentralized setting experiments

We wrote all our code in Java, using the Java 2 Standard Edition compiler with version 1.5. In the experiments, we used Tomcat Apache Server with version 5.5.8 and Axis software with version 2 as a container. The maximal heap size of the JVM was set to 1024MB by using the option –Xmx1024m. The Tomcat Apache Server uses multiple threads to handle concurrent requests. In the experiments, we increased the default value for maximum number of threads to 1000 to be able to test the system behavior for high number of concurrent clients. As back-end storage, we use MySQL database with version 4.1. We used the "nanoTime()" timing function that comes with Java 1.5 software.

Analyzing the results gathered from the experiments, we encountered some outliers (abnormal values). These outlier observations are numerically distant from the rest of observation data. The cause of the outliers is mainly the external effects, i.e., problems with network and server, as these outlier observations were not seen on the internal timing observations measuring only the system processing time. Due to outliers, the average may not be representative for the mean value of the observation times. This in turn may affect the results. For example, these outliers may increase the average execution time and the standard deviation. In order to avoid abnormalities in the results, we removed the outliers by utilizing the Z-filtering methodology. In Z-filtering, first, the average and standard deviation values are calculated. Then a simple test is applied. [abs(measurement_i-measurement_average)] / stdev > z_value_cutoff. This test discards the anomalies. After first filtering is over, the new average and standard deviation values are calculated with the remaining observation times. This process was recursively applied until no filtering occurred.

Simulation Parameters: Table 2 gives the simulation parameters for the fault-tolerance, distribution, dynamic replication and replica-content placement experiments. Note that, there are a number of tradeoffs involved in choosing simulation parameters. We repeated these experiments by varying values of parameters and explored these tradeoffs. We discuss some of the tradeoffs and our rationale in deciding these simulation parameters below. Here, we investigate an approximation of the optimal system performance. Thus, the results measured with the selected simulation parameters will be the optimal upper bound of the system performance.

metadata size and volume: We chose metadata size and volume from a real life application, i.e. Pattern Informatics, where the Hybrid Service is used. Thus, the metadata size and volume are 1.7 KB and 1000 respectively. An illustration of this metadata is given in Appendix A.

dynamic-replication-time-interval: In order to provide dynamic replication, metadata instances in a Hybrid Service are replicated in replica-hosting environment in a dynamic fashion within certain time intervals (*dynamic-replication-time-interval*). If the *dynamic-replication-time-interval* is chosen to be too small, then the system performance will be affected. If this time interval is too big, then the system will not adapt well to changes in client demands such as sudden bursts of request that come in from an unexpected location. Rabinovich et al introduced an extensive study on choosing values for the dynamic-replication tunable parameters. In our investigation, we chose the simulation parameters relying on their study in [16]. Thus, the value of *dynamic-replication-time-interval* is every 100 seconds.

minimum-fault-tolerance-watermark: To provide a certain level of fault-tolerance, we use a *minimum-fault-tolerance-watermark* indicating minimum required degree of replication. If the value is chosen to be high, then the time and system resources required completing replica-content placement and keeping these replicas up-to-date would be high. If the value is chosen to be too small, then the degree of replication (fault-tolerance level) will below. To facilitate testing of the system, we choose the minimum-fault-tolerance-watermark to be 3.

timeout-period*:* The tunable *timeout-period* value indicates the amount of time that a Hybrid Service node is willing to wait to receive response messages. If the *timeout-period* is too small, the initiator of a request will not wait enough for the context access responses coming from a multicast group. If the *timeout-period* is too big, then the query initiator may have to wait for a long time unnecessarily for some information that does not exist in the replica-hosting environment. To facilitate testing of the system, we choose the time-out value to be 10000 seconds.

deletion-threshold: If a temporary-copy (server-initiated) of a context is in low demand and its demand count is below *deletion-threshold*, then this temporary copy needs to be deleted. The *deletion-threshold* determines the rate for migration and replication occurring in the system. If a *deletion-threshold* is selected too low, the system will create more temporary copies, which will lead into high number of message exchanges in the system. If a deletion-threshold is too high, the system will keep low-demand temporary copies of a context unnecessarily. In our investigation, we chose the *deletion-threshold* value to be 0.03 request per second based on the study introduced in [16].

replication-threshold*:* If a context is in high demand and its demand count is above a *replication-threshold*, then the context is replicated as a temporary-copy. If the *replication-threshold* is selected to be too high, then the system will not adapt well to high number of client demands. If the *replication-threshold* is too low, the system will try to create temporary replicas at every remote replica where small number of requests comes in. This may cause unnecessary consumption of system resources. In our investigation, we chose the *replication-threshold* value to be 0.18 requests per second based on the study introduced in [16].

simulation parameters	Values
metadata-size	1.7 Kbytes
metadata-volume	1000
time-out value	10000 seconds
replication-threshold	0.18 requests per second
deletion-threshold	0.03 requests per second
minimum-fault-tolerance-watermark	3
dynamic-replication-time-interval	every 100 second

Table 2 Simulation parameters for the experiments

Effectiveness experiment: The effectiveness experiment is conducted to understand the performance and scalability of the prototype implementation for standard key-based query operations from a client's perspective. To conduct this experiment, two different test-phases are completed. In the first test-phase following cases are completed: a single client sends inquiry

requests to an echo service which receives a message and then sends it back to the client with no processing applied; a single client sends inquiry requests to a Hybrid Service which grants the request in in-memory storage. These test cases were repeated five times, each with 200 observations and we recorded the average response time. In the second testing phase, we investigated the following research question: How well does the Hybrid Service perform when the message rate per second is increased? To answer this question, we ramped-up the work load (number of messages sent per second) until the system performance degrades. To facilitate the testing, we use WS-Context Schema standard key-based query operations. This investigation is conducted using a Linux cluster with eight nodes located at the Community Grids Laboratory of Indiana University. Each node was equipped with Intel® Xeon™ CPU (2.40GHz), 2 GB RAM and ran Linux kernel 2.4.22. Both the Hybrid Service and testing client application were located in two different servers located in the same Linux cluster. The size of the metadata and size of the registry were 1.7 KB and 5000 metadata respectively. An illustration of this metadata is given in Appendix A.

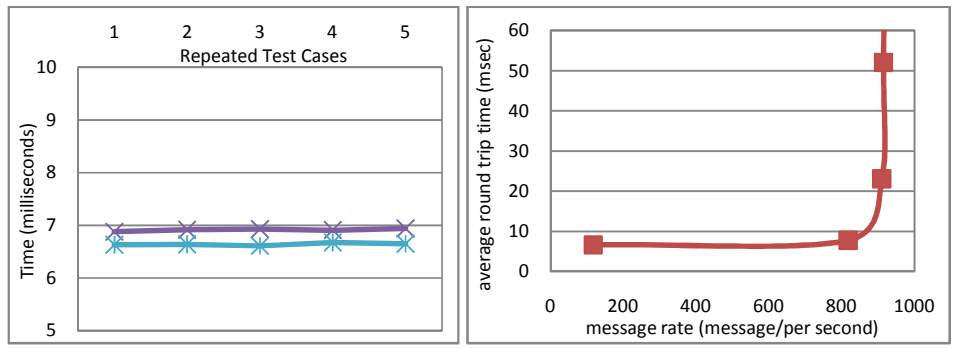

Figure 3 The figure on the left illustrates the round trip time chart for metadata inquiry requests. The figure on the right illustrates the metadata inquiry response times at various levels of message rates per second. The time units are in milliseconds.

Results of the effectiveness experiment: The results of the experiment were depicted in the figure above. Analyzing the results, we observe that the Hybrid Service achieves negligible processing overheads when responding client's queries by simply employing an in-memory storage mechanism. Analyzing the results, we also concluded that Hybrid Service performed well under increasing message rates. For inquiry request messages, we observe a threshold value after which the system performance starts decreasing due to high message rate. This threshold is mainly due to the limitations of Web Service container, as we observe the similar threshold when we test the system with an echo service that returns the input parameter passed to it with no message processing is applied.

Note that, this study proposes a system architecture that would address the two types of metadata domains: large-scale, static metadata and small-scale, dynamic metadata. For the first type, metadata can be replicated freely without concerning consistency, since the client accesses do not cause replica divergence. For the second type, metadata replication effectiveness is not a concern, as the state changes are propagated directly to application through a notification capability. Recall that, the Hybrid Service introduces a consistency model which requires consistency-sensitive applications to subscribe to unique-metadata-topics to receive updates.

Therefore, we leave out the investigation of effectiveness of the replica-content placement (from client's perspective). Instead, we explore the effectiveness our replication approach from server's perspective. To achieve this, we conduct the fault-tolerant experiment, which will be discussed later in the paper, to investigate the processing overheads involved in replica-content placement.

Distribution experiment: In this experiment, we conducted various testing cases to investigate the cost of distribution. We measured the cost of distributing access request into remote servers separated with significant network distances. In particular, we performed this experiment to answer following questions: a) what is the cost of access request distribution in terms of time required to fetch copies of a data (satisfying an access query) from remote locations?, b) how does the cost of distribution change when using multiple intermediary brokers for communication?, c) how does the performance of the distribution change for continuous, uninterrupted operations?

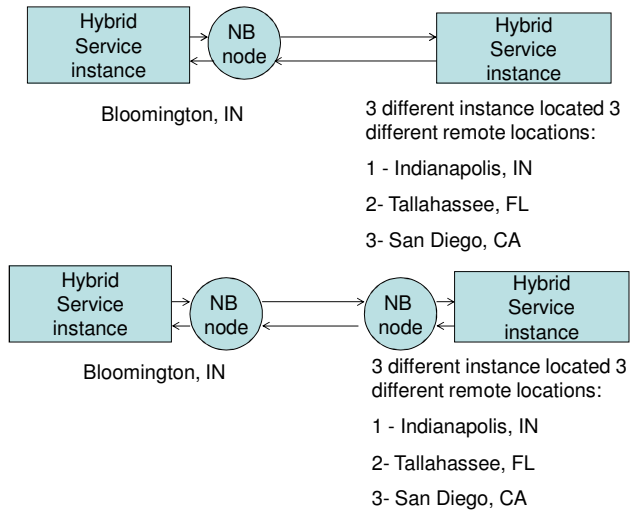

Figure 4 The design of the distribution experiment. The rounded shapes indicate NaradaBrokering nodes. The rectangle shapes indicate Hybrid Service instances located at different locations. The first test was conducted with one broker where the broker is located before the Hybrid Service instance in Bloomington, IN, while the second test was conducted with two broker nodes each sitting on the same machine before the Hybrid Service instance.

Results of the distribution experiment: We conducted distribution experiments for three different locations corresponding to three different network spaces. For each location, the system is tested for more than 25 thousand continuous operations and the time for each operation was recorded. By analyzing the results, we observed that the system shows stable performance for continuous, uninterrupted request distribution operations. To investigate the bottlenecks, we extracted the processing time involved for access request distribution. We depict the time spent in various sub-activities of distribution in Figure 5 and list the results in Table 3. The results indicated that regardless of how the Hybrid Service instances are distributed, the system showed the same stable and negligible performance, which is around 3.6 ms when using one intermediary broker. This time includes the Hybrid Service system processing over-

head and overhead of using an intermediary broker as part of publish-subscribe system. We observe that the overhead of access request distribution increases only by 1.2 ms when we use an additional intermediary broker.

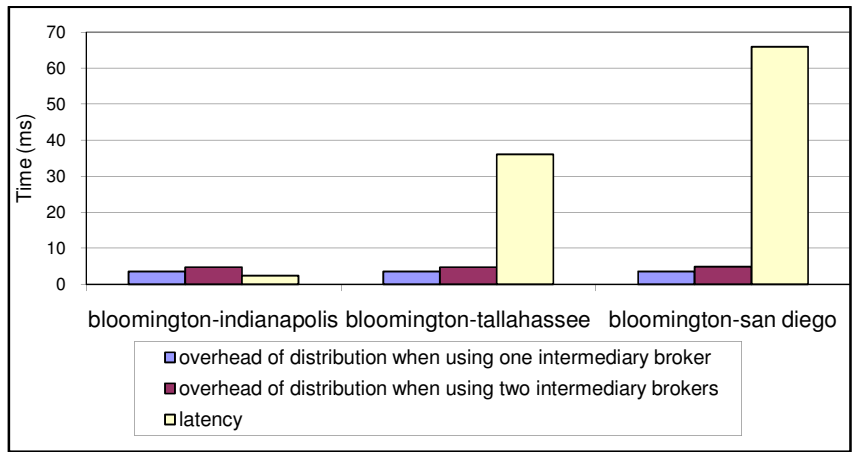

Figure 5 Time spent in various sub-activities of the request distribution scheme of the Hybrid Service

	one broker	two brokers	latency
bloomington-indianapolis	3.59	4.79	2.42
bloomington-tallahassee	3.55	4.78	36.05
bloomington-san diego	3.63	4.92	66

Table 3 Statistics for the figure above. Overhead of request distribution. Average timing is in milliseconds.

Dynamic replication experiment: In this experiment, we conducted a testing case to investigate the performance of dynamic replication. We used the dynamic replication for performance optimization to replicate temporary copies of contexts to where they wanted. In this experiment, we simulated a workload, where we have a thousand metadata in the Hybrid Service instance located at Indianapolis, IN. In this testing case, metadata from the Indianapolis instance was requested randomly by the Hybrid Service instance located at Bloomington. If the remote metadata is replicated to local site, the system simply obtains the data from local in-memory storage. We conducted two testing cases to answer the following questions: a) What is the cost of access distribution to fetch copies of a context from the remote location (Indianapolis), when the dynamic replication is disabled?, b) What is the cost of access distribution to fetch copies of a context from the remote location (Indianapolis), when dynamic replication is enabled?

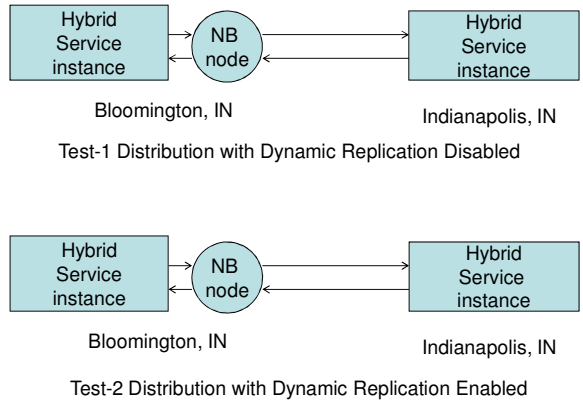

Figure 6 The design of the dynamic replication experiment. The rounded shapes indicate NaradaBrokering nodes. The rectangle shapes indicate Hybrid Service instances located at different locations. In the first testing case, dynamic replication capability is disabled. In the second testing case, dynamic replication capability is enabled.

Results of the dynamic replication experiment: Based on the results depicted in Figure 7, in this experiment, we observed that the dynamic replication methodology could actually move highly requested metadata to where they wanted. We observed that the system stabilized after around 16 minutes. Here, the system managed to move half of the metadata to the local site after around 8 minutes, where we observed the highest peak in the standard deviation values. This is simply because half of the access requests were granted locally, while the other half were granted at the remote location.

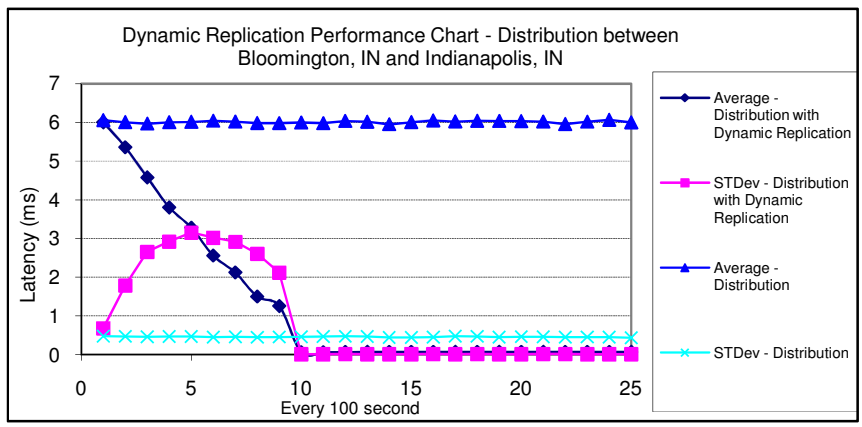

Figure 7 The results of the dynamic replication experiment. This figure depicts the metadata access latency and the standard deviation for two cases: a) first case is when dynamic replication option is disabled, b) second case is when dynamic replication option is enabled. The x-axis indicated the dynamic replica-content placement decision frequency, while the y-axis indicates the metadata access-latency.

Fault-tolerance experiment: In this experiment, we conducted various testing cases to investigate the cost of fault-tolerance when moving from centralized system to a decentralized replica hosting system. In particular, we performed our testing cases to answer following questions: a) What is the cost of replica-content placement for fault-tolerance in terms of the time required to create replicas at remote locations?, b) How does the system behavior change for continuous, uninterrupted replica-content placement operations?. To answer these questions, we conducted two testing cases: The first test was conducted with one broker when the broker was located before the Hybrid Service instance at Bloomington, IN. The second test was conducted with two brokers each sitting on the same machine before the Hybrid Service instances. In this experiment, we increased the fault tolerance level gradually and measured end-to-end latency for replica-content placement.

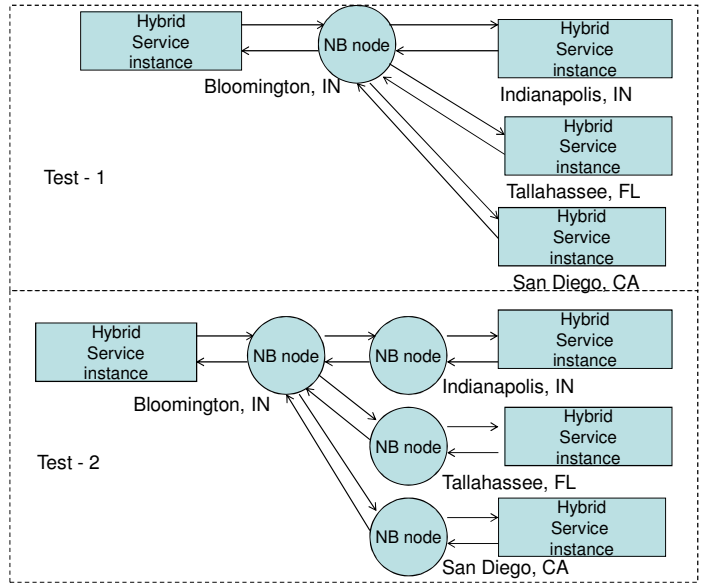

Figure 8 The design of the fault tolerance experiment. The rounded shapes indicate NaradaBrokering nodes. The rectangle shapes indicate Hybrid Service instances located at different locations. In the first testing case, we measure the end-to-end latency for varying number replica-content creation with only one broker. In the second case, we repeat the same test with two brokers.

Results of the fault-tolerant experiment: We conduct this testing case for one to three replica creations. For each testing case, the system is tested for more than thousand continuous operations and the time for each operation was recorded. By analyzing the results, we observed that the system performs stable for continuous, uninterrupted replica creation operations. To investigate the bottlenecks, we extract the processing time involved in replication creation. We depict the time spent in various sub-activities of replica creation in the figure below. The results indicate that the time required for one replica creation is only four milliseconds. The cost of replica creation time includes the Hybrid Service system processing overhead and overhead of using an intermediary broker as part of publish-subscribe system. We also observe that the time required for replica creation increases, as the number of replica copies increases. This is because; the system has to perform an additional unicast message for each additional replica creation. The time required for a unicast message is less than one milli-

second. The results also indicated that, the overhead of replica-content creation increases only by 1.2 ms, when we use an additional intermediary broker.

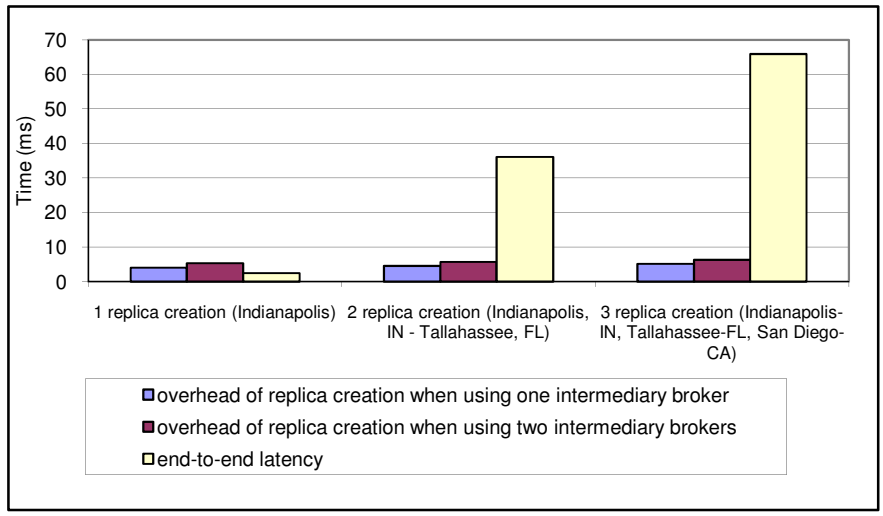

Figure 9 Time spent in various sub-activities of the replica-content creation scheme of the Hybrid Service.

	one broker	two brokers	end-to-end latency
1 replica (Indianapolis)	4.02	5.27	2.43
2 replicas (Indianapolis–Tallahassee)	4.54	5.67	36.05
3 replicas (Indianapolis–Tallahassee –San Diego)	5.13	6.24	65.90

Table 4 Statistics for the figure above. Overhead of replica-content creation. Average timing is in milliseconds.

Consistency enforcement experiment: The design of the consistency enforcement is similar to the distribution experiment depicted in Figure 4. In this experiment, our aim is to answer the following questions: a) What is the cost of consistency enforcement in terms of the time required to carry out updates at the primary-copy holder?, b) How does the system behavior change for continuous, uninterrupted update operations (for consistency enforcement)? To this end, we conducted two tests: The first test was conducted with one broker where the broker is located before the Hybrid Service instance in Bloomington, IN, while the second test was conducted with two broker nodes each sitting on the same machine before the Hybrid Service instances. In this experiment, we measured the time required to distribute an update request to the primary-copy holder of the context under consideration for consistency enforcement reasons.

Consistency enforcement experiment results: We conduct this testing case for three different locations. For each location, the system was tested for more than 25 thousand continuous operations. For each operation, time was recorded. By analyzing the results, we observe that

the system shows stable performance over time for continuous consistency enforcement operations. Based on the observations, we extract the processing time involved to provide consistency enforcement using publish-subscribe based messaging schemes. We depict the time spent in various sub-activities of distributing and carrying out the update request at the primary-copy holder in the figure below. The cost of consistency enforcement includes the Hybrid Service system processing overhead (for distributing update request to primary-copy holder) and overhead of using an intermediary broker as part of publish-subscribe system. We observe that the time required for consistency enforcement does not change regardless of how Hybrid System instances are distributed. Similar to our results in the previous two experiments, we observe that the overhead of consistency enforcement increases only by 1.2 ms when we use an additional intermediary broker.

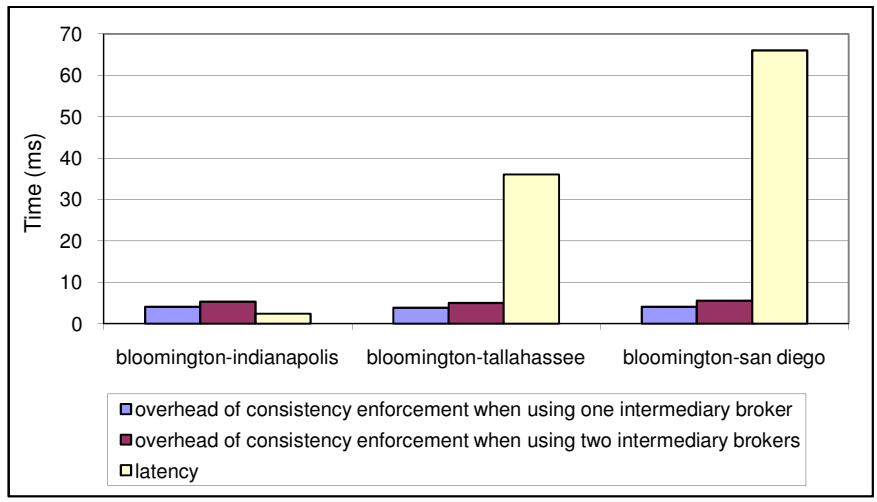

Figure 10 Time spent in various sub-activities of the Hybrid Service consistency enforcement scheme. The results analyze the overhead of distributing update requests to the primary-copy holder where the update requests take place for consistency enforcement reasons.

	one broker	two brokers	end-to-end latency
Bloomington – Indianapolis	4.05	5.32	2.42
Bloomington – Tallahassee	3.83	5.03	36.05
Bloomington – San Diego	4.07	5.49	66

Table 5 Statistics for the figure above. Statistics for overhead of update distribution. Average timing is in milliseconds.

9 Conclusions and Future Research Directions

This research presented a high performance, distributed Grid Information Service Architecture, Hybrid Grid Information Service, as a metadata replica hosting environment. To achieve distribution, the Hybrid Service uses publish-subscribe based messaging schemes to provide interaction among the distributed instances of the service. It utilizes a topic based publish-subscribe messaging communication to implement fundamental aspects of decentralized in-

formation systems such as fault-tolerance, access-request distribution, and consistency enforcement. To achieve high-performance in metadata access and improve the overall performance of the system, the Hybrid Service utilizes a performance optimization technique: dynamic migration/replication. This technique improves overall system performance by moving/replicating highly requested metadata to where they wanted.

The evaluation of the system prototype pointed out the following results. Firstly, it pointed out that the Hybrid Service is an effective solution with its negligible processing overheads and its high-performance under heavy workloads. Secondly, it pointed that the Hybrid Service presents stable behavior for access request distribution, replica creation and consistency enforcement over a high number continuous operations. Thirdly, it indicated that the cost of distribution, fault tolerance and consistency enforcement is in the order of milliseconds. These promising results shows that high-performance, distributed Grid Information Service Architectures can be built by utilizing publish-subscribe based messaging schemes. Fourthly, it pointed out that high-performance metadata access can be achieved by utilizing dynamic replication/migration technique. This technique also reduces the cost of repetitive access requests by moving temporary copies of contexts to where they wanted. Fifthly, it indicated the differences in the processing costs of different aspects of the distributed system. For example, the cost of fault tolerance is higher than the cost of distribution and consistency enforcement. This is because; there is an additional time required for performing additional unicast messages for higher fault-tolerance levels. Finally, it pointed out the trade-off between performance and fault-tolerance. The results indicated that the cost of replica-content creation increases, when the degree of fault-tolerance increased.

We applied the introduced system into different application domains such as geographical information system and sensor grids [4, 23-25], management of real-time streams in collaboration grids [26, 27]. We intend to investigate how good the system architecture is by applying it into wider range of application domains. We also plan on expanding the prototype evaluation by including an extensive scalability experiment, which would test the system with large communities of nodes, and an effectiveness experiment, which would test the replica-content placement efficiency from a client's perspective. An additional area of future investigation is information security. To complete the system, we intend to research an information security mechanism for the distributed replica hosting system. This effort should research the security concerns related to communication between network nodes and users, as well as security concerns related to authorization to deal with access control.

Acknowledgement: We thank Dr. Plale for stimulating discussions and her feedback on this research. This work is supported by the Advanced Information Systems Technology Program of NASA's Earth-Sun System Technology Office.

Appendix

A. Sample WS-Context Schema XML metadata

```
<?xml version="1.0" encoding="UTF-8"?>
<wscontext:context
        xmlns:wscontext="http://datatype.fthpis.cgl/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <contextKey>ABCCE800-AB35-11DA-A4FC-C80C5880CB18</contextKey>
        <serviceKey>ABCCE800-AB35-11DA-A4FC-C80C5880CB19</serviceKey>
        <sessionKey>ABCCE800-AB35-11DA-A4FC-C80C5880CB20</sessionKey>
        <name>context://GIS/PI/ABCCE544-CX35-11EA-BVFC-C34C7789CB33</name>
        <value>context:///GIS/VC/3ea29661-2d5e-11db-8c56-cf37cd202027/3ebd7162-2d5e-11db-8c56-
cf37cd202027/cost</value>
        <valueType>String</valueType>
        <lease>
                <timeout>1000</timeout>
                <isInfinite>false</isInfinite>
        </lease>
        <version>1</version>
</wscontext:context>
```

Bibliography

1. M. Gerndt, R.W., Z. Balaton, G. Gombás, P. Kacsuk, Zs. Németh, N. Podhorszki, H-L. Truong, T. Fahringer, M. Bubak, E. Laure, T. Margalef, *Performance Tools for the Grid: State of the Art and Future*. 2004, Shaker Verlag.
2. Zanikolas, S., Sakellariou, R., *A Taxonomy of Grid Monitoring Systems.* . Future Generation Computer Systems, 21(1), 2005: p. pp. 163--188.
3. Wu, W., et al., *Grid Service Architecture for Videoconferencing*, in *"Grid Computational Methods" Edited by M.P. Bekakos, G.A. Gravvanis and H.R. Arabnia*.
4. Aktas, M.S., et al., *iSERVO: Implementing the International Solid Earth Research Virtual Observatory by Integrating Computational Grid and Geographical Information Web Services.* PAGEOPH, 2004.
5. Ken Arnold, A.W., Byran O'Sullivan, Robert Scheifler, and Jim Waldo, *The JINI Specification*. 1999: Addison-Wesley, Reading, MA.
6. Bellwood, T., Clement, L., and von Riegen, C., *UDDI Version 3.0.1: UDDI Spec Technical Committee Specification* http://uddi.org/pubs/uddi-v3.0.1-20031014.htm. 2003.
7. *GRIMOIRES - UDDI compliant Web Service registry with metadata annotation extension, availble at* http://sourceforge.net/projects/grimoires.
8. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., Balakrishnan, H. *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications*. in *IEEE/ACM Trans. on Networking*. 2001.
9. M. Palankar, A.O., A. Iamnitchi, and M. Ripeanu, *Amazon S3 for Science Grids: a Viable Solution?* 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI'07), 2007.
10. Milojicic, D.S., et al. , *Peer-to-Peer Computing*, in *HP Labs Technical Report HPL-2002-57*. 2002, HP Labs.
11. S. Helal, N.D., and C. Lee. *Konark-A Service Discovery and Delivery Protocol for Ad-Hoc Networks*. in *In Third IEEE Conference on Wireless Communications Network (WCNC)*. March 2003. New Orleans, USA.
12. Guttman, E., Perkins, C., Veizades, J., *Service Location Protocol, RFC 2165, available at* http://rfc.net/rfc2165.html. 1997.
13. Tang, D., Chang, D., Tanaka, K., Baker, M., *Resource Discovery in Ad-Hoc Networks*, in *CSL-TR-98-769*. 1998, Stanford University.

14. Beatty, J., Kakivaya, G., Kemp, D., Kuehnel, T., Lovering, B., Roe, B., St. John, C., Schlimmer, J., Simonnet, G., Walter, D., Weast, J., Yarmosh, Y., and Yendluri, P. , *Web Services Dynamic Discovery (WS-Discovery) available from* http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-discovery.pdf. 2004.
15. Sivasubramanian, S., Szymaniak, M., Pierre, G., Steen, M., *Replication for Web Hosting Systems.* ACM Computing Surveys, 36(3):291--334, 2004.
16. Rabinovich, M., Rabinovich, I., Rajaraman, R., Aggarwal, A. *A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service*. in *Proc. 19th Int'l Conf. Distributed Computing Systems*. 1999.
17. Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B., *Globally distributed content delivery.* IEEE Internet Computing, 2002: p. pp 50-58.
18. Tanenbaum, A., Van Steen, M., *Distributed Systems Principles and Paradigms*. 2002. Cited in page 326.
19. Aktas, M.S., *Information federation in Grid Information Services*, in *Computer Science*. 2007, Indiana University: Bloomington, IN.
20. Sun_Microsystems, *JavaSpaces Specification Revision 1.0, 1999 available at* http://www.sun.com/jini/specs/js.ps.
21. Pallickara, S., Fox, G. *NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*. in *Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003*. 2003. Rio Janeiro, Brazil.
22. Mehmet S. Aktas, G.C.F., Marlon Pierce, *XML Metadata Services.* Concurr. Comput. : Pract. Exper., 2008.
23. Aydin, G., et al. *SERVOGrid Complexity Computational Environments (CCE) Integrated Performance Analysis*. in *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*. 2005: IEEE.
24. Aktas, M.S., et al. *Implementing Geographical Information System Grid Services to Support Computational Geophysics in a Service-Oriented Environment*. in *NASA Earth-Sun System Technology Conference* http://esto.nasa.gov/conferences/estc2005/index.html *University of Maryland, Adelphi, Maryland, June 28 - 30, 2005. All material is online for paper , presentation* http://www.esto.nasa.gov/conferences/estc2005/Presentations/a6p2.pdf *, and abstract* http://www.esto.nasa.gov/conferences/estc2005/author.html. 2005.
25. Aydin, G., Aktas, Mehmet S., Fox, Geoffrey C., Gadgil, Harshawardhan, Pierce, Marlon, Sayar, Ahmet, *SERVOGrid Complexity Computational Environments (CCE) Integrated Performance Analysis*. 2005.
26. Fox, G., et al. *Management of Real-Time Streaming Data Grid Services*. in *Invited talk at Fourth International Conference* http://kg.ict.ac.cn/GCC2005 *on Grid and Cooperative Computing (GCC2005), held in Beijing, China, Nov 30-Dec 3, 2005*. 2005.
27. Fox, G., et al. *Real Time Streaming Data Grid Applications*. in *Invited talk at TIWDC 2005 CNIT Tyrrhenian International Workshop* http://iwdc.cnit.it/ *on Digital Communications July 4-6 2005*. 2005.