Hybrid Grid Information Service

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 Grid 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 replica hosting system improves the quality of Grid Information Services in terms of high-performance and fault-tolerance with negligible processing overheads. The results also indicate that efficient decentralized Grid 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 information services present some challenges. For an example, most of the existing approaches to Information Services have centralized components and do not address high performance and fault-tolerance issues. Handling information requirements of dynamic Grid/Web Service collections requires high performance, decentralized, and fault tolerant information systems. Another example, existing Information Service mechanisms do not take into account demand changes when making decisions on metadata access and storage. However, replica hosting environments for information services should be able to relocate metadata to nearby locations of interested entities in order to provide efficient access, storage of the information. Therefore, we see a greater need for a system to facilitate resource discovery in peer-to-peer/grid service-oriented architecture based applications. In order to address these challenges, we designed, built and evaluated a Grid Information Service called Hybrid Grid Information Service (Hybrid Service) which forms a replica hosting system. This paper focuses on the experiences gained in designing and building the replica hosting system for the Hybrid Service.

The organization of the rest of the paper is as follows. Section 2 reviews the major solutions in state of art of the studies covered in this study. It gives an extensive survey on the previous metadata management solutions for replica hosting systems. The previous solutions are analyzed followed by discussions on the reasons why the previous solutions do not answer the research problem at hand. Section 2 also discusses various concepts and paradigms that are taken into account in designing a solution addressing the research problem. Section 3 gives a brief overview of the centralized version of the Hybrid Service. Section 4-8 presents the Hybrid Service as the replica hosting system. Section 4 discusses the decentralized architectural design details of the system. Section 5-8 discusses important aspects of the replica hosting system design. Section 9 analyzes the performance evolution of the prototype of the Hybrid Service replica hosting system. It presents benchmarking on distribution, fault-tolerance and consistency enforcement aspects of the system. Section 10 contains the conclusions and future research directions.

2 Background

Web Services may have complex characteristics and interact with one or a set of services. Service descriptions expressing these characteristics must be capable of accurately representing these services. We use the term "context" to define all available information associated with a Web Service. For the purposes of our research, context is a piece of information (metadata) describing behavior, environment and characteristics of a service. Context encapsulates not only activities that service is involved in but also the service itself as an entity. From this point forward, we will be using context and service metadata interchangeably, as they both refer to the information associated with a service.

Context has dynamic characteristics, as it indicates volatile behavior of Grid/Web Services. Moreover, context may have changing user demands over time and it needs to be reallocated based on changing user demands and locations. This requires a Grid Information Service that can meet with tight response-time requirements of dynamic Grid/Web Service collections and that can support optimization techniques in metadata access and move the highly requested metadata to where they wanted.

Locating resources of interest is a fundamental problem in resource intensive environments. An effective methodology to facilitate resource discovery is to provide and manage information about resources. Here, a resource corresponds to a service and information associated to it refers to metadata of a service. Thus, we see a greater need for metadata management solutions to make such metadata available in peer-to-peer/grid environments. We can analyze existing solutions for metadata hosting environments under several mainstream categories based on the ways they tackle with the research issues in sub-processes of metadata management: a) architectural design for storage handling b) formation of underlying networks.

Analysis of service metadata management research based on architectures: Existing service metadata discovery architectures can be broadly categorized as centralized and decentralized by the way they handle with 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 like JINI [1], Salutation [2], and Service Location Protocol [3] have been developed to provide discovery of remote services residing in distributed nodes in a wired network. Their architectures are based on a central registry for service registration and discovery. **Limitations:** The centralized registry approach presents a single point of failure and is limited to a certain storage capability. It does not scale up to high number of services that in turn creates a performance and scalability bottleneck for the system.

In decentralized approach, there is no central database. This research trend mainly focuses on decentralized search where all the peers of the system actively participate the discovery process. Previous solutions with pure decentralized storage models focused on the concept of distributed hash tables (DHT). This approach assumes the possession of an identifier such as hash table that identifies the service to be discovered. For instance, a DHT specifies a relation between a resource and a position in a distributed network. A good example of DHT is the Chord [4] project. Each entity of the network is hashed; therefore, the position of the entity in the network is determined through DHT. A message is routed to the closest entity to the final destination. Inspired from peer-to-peer discovery model, there has been work conducted on to develop peer-to-peer architectures [5, 6] for distributed information management. Another decentralized approach, Bittorent is a peer-to-peer file distribution protocol which is designed to distribute large amounts of widely distributed data. A Bittorent network consists of three entites: a tracker, a torent file and peers. A tracker is a server that keeps track of which peers (seeds, downloaders) are in the network. A torent is a metadata file that contains information about all the downloadable pieces of a data. A peer is software, which implements the Bittorent protocol. Each peer is capable of requesting, and transferring data across network. Peers are classified into two categories: seeds and downloaders. The former has the complete copy of the file and offers it for download. The latter has the parts of the file and downloads the file from other seeds or downloaders. An example peer-to-peer storage service, Amazon Simple Storage Service (Amazon S3) is a web-scale storage which supports use of the Bittorent protocol. It provides a simple web service interface used to provide storage and retrieval of any data across widely distributed area. **Limitations:** As the resource placement at nodes is strictly enforced in structured peer-to-peer networks, these systems suffer from a heavy overhead on the bootstrap of the network. Pure decentralized storage models have mainly focused on DHT approach. The DHT approach provides good performance on routing messages to corresponding nodes. However, it is limited to primitive query capabilities on the database operations [7]. Furthermore, the DHT approach does not take into account changes in the client demands and load balancing. The Bittorent approach and Amazon S3 storage service that utilizes the Bittorent protocol have the following limitations. First, the overhead involved in transferring small size data (e.g. in the order of kilobytes) is big. For example, the total required bandwidth for necessary protocol messages for downloading a small size data is high. Second, the tracker is a performance bottleneck and a single point of failure in the network. Thus, the performance of a Bittorent network depends on the capacity of the tracker. In addition, if the tracker fails, it is not possible for peers to locate each other.

Discussion: The centralized storage scales better in performance for limited storage capability compared to decentralized approach, whereas a decentralized approach can scale up to high amount of metadata where centralized approach fails. Pure decentralized storage models such as peer-to-peer service discovery architectures have focused on the concept of distributed hash tables (DHT). This method may provide better performance as the database operation messages are routed fast, however, it still does not provide the same performance to handle dynamic metadata as centralized database does. In this research, we take as a design requirement that the proposed system should support a) high performance by utilizing optimized request distribution techniques, b) fault-tolerance by increasing the availability of metadata, and c) peer-to-peer message distribution strategy by utilizing a classic middleware approach; publish-subscribe based messaging system.

Metadata management research based on formation of underlying networks: Another way of classifying service discovery architectures could be based on the formation of the network and the way of handling with 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 [8, 9]. Example wired network discovery architectures such as JINI [1] and Service Location Protocol [3] 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 there is no constraint on the resource dissemination in the network. Existing solutions [8, 10] for service discovery for ad-hoc networks (e.g. pervasive computing environments) can be broadly categorized as broadcast-driven and advertisement-driven approaches [11]. In broadcast-driven approach, a service discovery request is broadcasted throughout the discovery network. In this approach, if a node contains the service, it unicast with a response message. In advertisement-driven approach, services advertise themselves to all available nodes. In this case, each node interested discovering a service caches the advertisement of the service. The WS-Discovery Specification [12] supports both broadcast-driven and advertisement-based approaches. 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. **Limitations:** The traditional wired-network based architectures are limited, as they depend on a controlling entity, which assigns identifiers to participating entities. 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. **Discussion:** Metadata discovery solutions designed for ad-hoc networks are appropriate for Grid and peer-to-peer computing environments, as these solutions do not have any constraints on resource dissemination in the network. Among these solutions, the WS-Discovery approach is promising as it employs a pure peer-to-peer approach where the messages (advertisement/discovery) are broadcasted in the system. Inspired by WS-Discovery approach, we take as a requirement that the proposed system should employ a broadcast-based metadata discovery approach. Each message should include a unique identifier distinguishing the peer, which initiated the request. On receipt of a message, only the nodes that have the requested information should reply with a response message. Moreover, we also take as a requirement that the proposed system should employ an advertisement-driven approach for advertising the existence of network nodes. Apart from the WS-Discovery approach, the proposed system should use a software multicast based message dissemination for request distribution, metadata and network node advertisements.

Publish-subscribe paradigm: Most distributed systems rely on passing messages between processes. Thus, system entities communicate with each other by exchanging messages, which captures varying information such as search/storage requests, system conditions and so forth. These systems can be categorized based on their messaging infrastructures such as publish-subscribe systems, point-to-point communication systems, queuing systems, and peer-to-peer based systems [13]. Among them, publish-subscribe paradigm principles have gained importance in recent years, as recently released specifications such as Java Message Service [14] and WS-Eventing Specification [15] benefit from publish-subscribe system principles to standardize development of interoperable systems. The publish-subscribe paradigm uses an asynchronous messaging. In a publish-subscribe system, publishers can broadcast each message (e.g. through a topic), rather than addressing it to specific recipients. The messaging system then sends the message to all recipients that subscribed to a topic. **Advantages:** As it is asynchronous, a publish-subscribe system forms a loosely coupled architecture where the publishers do not know who the subscribers are. This messaging scheme is more scalable architecture than point-to-point solutions, since message senders only deal with creating the original message, and can leave the job of message distribution to the messaging infrastructure. **Limitations:** Messages are typically broadcasted over a network. This allows a more dynamic network topology. However, as the volume of messages increase, this may result in overloading of the network without appropriate pruning strategies. **Discussion:** In the architectural design of the proposed system, we take as a requirement that the system should support the publish-subscribe paradigm as a communication middleware for message exchanges between system entities. NaradaBrokering [16-20] is an open-source and distributed messaging infrastructure implementing the publish-subscribe paradigm. It establishes a hierarchy structure at the network, where a peer is part of a cluster that is a part of a super-cluster, which is in turn part of a super-super-cluster and so on. The organization scheme of this scenario forms a communication between peers that increases logarithmically with geometric increase in network size. The NaradaBrokering software is the most appropriate solution for our design decision, since its entities, i.e. brokers, specify constraints on the quality of service related delivery of events. It provides a substrate of Quality of Services (security, reliability, etc.). In turn, this enables various capabilities to the system such as order, duplicate elimination, reliable message delivery, security and so forth. Note that these capabilities are not inherently part of publish-subscribe paradigm.

Replication and consistency issues: Replication is a well-known and commonly used technique to improve the quality of metadata hosting environments. One approach to replication is to keep a copy of a data at every node of the network (full replication). The other approach is to keep a copy of a data only at a few number of replica servers (partial replication) [21, 22]. Replication can further be categorized as permanent-replication and server-initiated replication [22]. 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 for a period of time during which the data is in high demand.

Sivasubramanian et al [21] give an extensive survey on designing and developing replica hosting environments, as does Robinovich in [23], paying particular attention to dynamic replication. As the nature of some of the targeted metadata domains of this research is highly dynamic, we focus on replica hosting systems that are handling with dynamic data. These systems can 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 [23, 24]. 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. Some of the existing solutions that apply dynamic replication, monitor various properties of the system when making replica placement decisions [23, 25]. For instance, Radar [26] replicates/migrates dynamic content based on changing client demands. Spread [25] considers the path between the data-system and the client and makes decisions to replicate dynamic content on that path.

The consistency enforcement issue has to do with ensuring all replicas of the same data to be the same. A consistency enforcement model is a contract between a hosting environment and its clients [21]. Some classification approaches to categorize existing research for consistency enforcement are discussed in [21, 22]. Tanenbaum [22] differentiates consistency under two main classes: data-centric and client-centric. In the data-centric approach, all copies of a data are updated regardless of whether some client is aware of those updates. In the client-centric approach, consistency is ensured from a client's perspective. Client-centric consistency model allows copies of a data to be inconsistent with each other as long as the consistency is ensured from a single client's point of view. The implementations of the consistency models can be categorized as primary-based protocols (primary-copy approach) and replicated-write protocols [22]. In primary-copy approach, updates are carried out on a single server, while in the replicated-write approach; updates can be originated at multi servers. For an example, Radar [23] applies the primary-copy approach, which suggests a copy of a data item to be designated as primary-copy, to ensure consistency enforcement. Updates can be transferred in different ways. One approach, for example, is to transfer the whole content of a replica, while the other is to transfer the difference between the previous copy and the updated copy. Update propagation can be initiated in different ways. For example, data may be pulled from an up-to-date server (pull). Another example, an up-to-date server may keep track of the servers holding copies of a data and push the updates onto those servers (push). Some update propagation schemes combine pull and push methodologies. For instance, the Akamai project [24] introduces versioning where a version number is part of the data identifier, so that the client can only fetch the updated data (with a given identifier) from the corresponding data hosting system.

Discussion: The proposed architecture should differ from previous solutions for web replica hosting systems, as the intended use is not to be a web-scale hosting environment. Table 1 shows a summary of the useful strategies that we take as a design requirement for our implementation design. As for the request routing mechanism, we think that, broadcasting access requests would be the most appropriate request distribution solution considering the dynamic characteristics of the metadata domain. Some of the existing solutions to dynamic replication [23, 24] 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 can be highly updated. Thus, to capture such "dynamism", we take as a requirement that the proposed system should broadcast the requests to the nodes holding the data under question. For message dissemination, the system should employ a pure peer-to-peer approach, which is based on publish-subscribe based messaging schemes to achieve a multi-publisher multicast mechanism.

As for the replica placement methodology, we consider providing an architecture, which would allow both partial and full replication to take place with negligible system processing overheads. We also consider both permanent and server-initiated replication as appropriate strategies for the proposed system. The permanent-replication could provide a minimum required fault tolerance, while the server-initiated replication could improve responsiveness of the system.

To minimize the cost of consistency enforcement, we take as a requirement that the system should employ a client-centric consistency model, which suggests copies of a context can be inconsistent with each other; however, they should be consistent from a client's perspective.

Design Issue	**The design requirements of the proposed system**
Replica-content placement	copies of a context should be kept permanently for fault tolerant reasons (permanent replication)
	copies of a context should be kept temporarily for a time period during which the context is in demand to improve performance (server-initiated replication)
Request routing	client's request should be broadcasted to those nodes holding the context in question (broadcast-based request dissemination)
Consistency enforcement	updates should be carried out on a single server (primary-copy approach) - every update request should be assigned a synchronized timestamp, which can later be used for ordering among the updates
	copies of a context can be inconsistent with each other; however, they should be consistent from a client's perspective.
	whole content of a context should be broadcasted by the primary-copy to the redundant permanent-copy holders

Table 1 Summary of the replication and consistency enforcement strategies that we take as a requirement for the proposed system implementation.

As for the consistency enforcement protocol, the primary-copy approach is utilized. In the primary-copy approach, to perform an update operation just the primary-copy is locked. Since primary copies are distributed at various data-systems, a single site will not be overloaded with locking all its data for update operations. Thus, we take as a requirement that the system should support the primary-copy approach at the implementation stage of consistency enforcement.

As for the way an update is initiated, the push approach could be an appropriate solution. The push approach has a disadvantage since it requires the primary-copy host to store and keep track of the state of each replica server holding a copy of the replica. To overcome this limitation, we take as a requirement that the system should introduce an approach, which utilizes broadcast-based dissemination to send updates only to those nodes holding the redundant copies of a context. Based on this scheme, the primary-copy host could push the updates, when an update occurs. This multicast-based approach does not require the primary-copy host to keep the state of the partial replica set of a context.

3 Overview of the Hybrid Grid Information Service

We designed and built a Hybrid Grid Information Service (Hybrid Service) to support handling and discovery of metadata associated to 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.

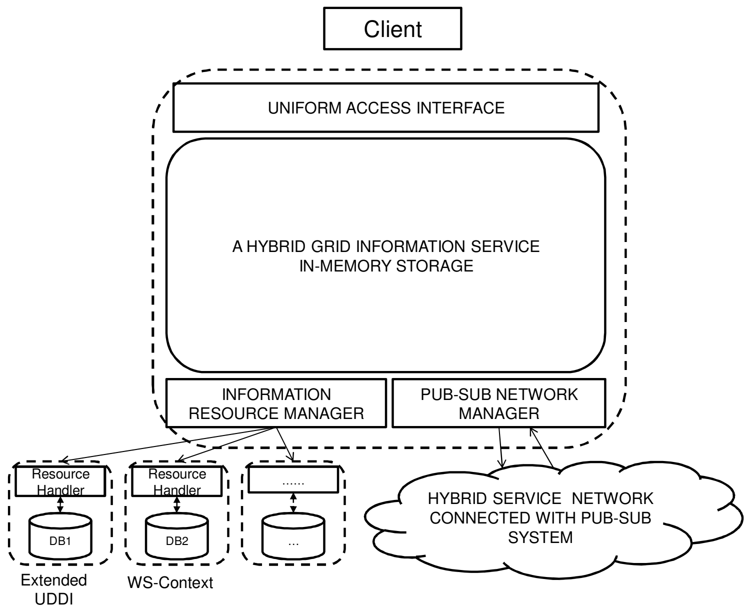

Figure 1 This figure illustrates the Hybrid Service interacting with a client. The dashed box indicates the Hybrid Service. It is an add-on architecture that runs one layer above information service implementations (such as the extended UDDI XML Metadata Service (our implementation of UDDI Specification) and WS-Context XML Metadata Service (our implementation of Context Manager component of the WS-Context Specification)) to handle metadata associated to services. It utilizes an Information Resource Management abstraction layer to interact with lower layer Information Services. The Hybrid Service forms a Replica Hosting System to enable discovery, request distribution, replication and consistency enforcement across the network. It utilizes a Pub-Sub Network Manager component to interact with the other network nodes.

The Hybrid Service provides a unifying architecture where one can assemble metadata instances of different information services. In order to achieve this, the Hybrid System Architecture introduces various abstraction layers for uniform access interface and information resource management. The uniform access abstraction layer is implemented to support one to many communication protocols. The information resource management abstraction layer is implemented to manage one to many information service implementations. In the prototype implementation, we have shown that the Hybrid Service is able to unify the two local information service implementations: WS-Context and Extended UDDI and support their communication protocols. The Hybrid Service Architecture is depicted in Figure 1. This figure illustrates a client interacting with the Hybrid Service, which is running as an add-on component above the Extended UDDI and WS-Context Information Services. The Hybrid Service utilizes a Pub-Sub Network Manager component to interact with the rest of the information service network. We discuss the Hybrid Service replica hosting system which enables discovery, request distribution, replication and consistency enforcement across the network nodes. The detailed discussion on the architectural design of the centralized Hybrid Service is the focus of another paper.

4 Hybrid Service Replica Hosting System Architecture

We identify three fundamental issues of designing a replica hosting system: replica-content placement, request routing and consistency enforcement. Replica content placement has to do with creating a set of duplicated data replicas across the nodes of a distributed system. Request routing has to do with redirecting a client request to the most appropriate replica server. Consistency enforcement deals with ensuring data coherency across replicas in the system. Figure 2 illustrates the decentralized version of the architecture.

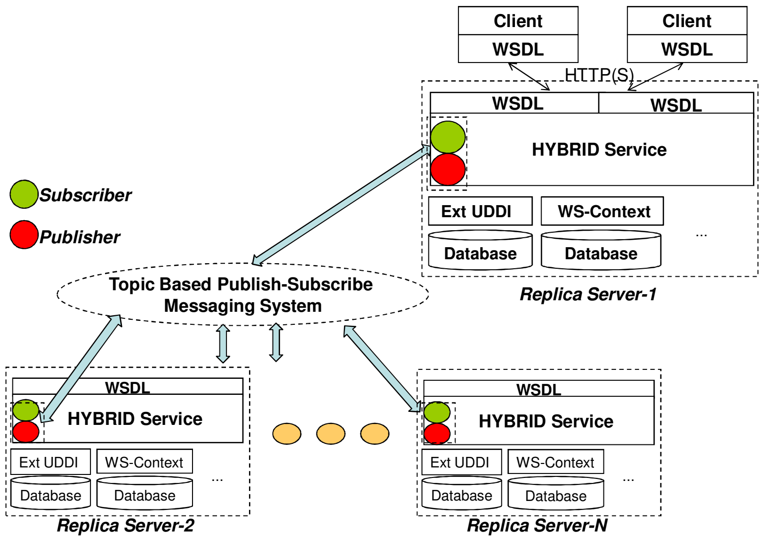

Figure 2 Distributed Hybrid Services. This figure illustrates N-node decentralized Hybrid Service from the perspective of a single Hybrid Service (Replica Server-1) interacting with two clients.

Replica placement issue consists of two sub-problems: replica server placement and replica content placement [21]. The former issue deals with the problem of finding suitable locations for replica servers, while the latter issue handles with selecting replica servers that should host a data. This study researched the latter problem, which concerns with the selection of replica servers that must hold the data under consideration. Replication can also be categorized by the manner in which replicas are created and managed. One strategy is permanent replication: replicas are manually created, managed and kept permanently. This strategy is mostly used for fault-tolerance reasons. Another strategy is server-initiated (dynamic) replication: replicas are created, managed and kept based on changing user behavior. This strategy is mostly used to enhance system performance. The proposed system utilizes both permanent replication and dynamic replication techniques. The permanent replication is used to provide fault-tolerance in terms of availability. Permanent copies are used to at least keep the minimum required number of replicas for the same data. The dynamic replication technique is used for performance optimization. It enhances performance by replicating data onto servers in the proximity of demanding clients that in turn reduces access latency.

Request routing redirects the client request to most appropriate replica holder. Request routing can be done via unicasting or broadcasting. In unicast communication, the request initiator needs to keep track of where each metadata is located. If the requested metadata is located in *N* locations, the initiator sends *N* separate messages, one to each server. This approach requires each server to keep and maintain the state of the metadata system which is an expensive operation. In broadcast communication, the request initiator sends only one message and the underlying broadcasting facility takes care of the delivery of the message. Note that broadcasting a message to *N* servers is no more expensive than unicasting *N* messages. The proposed system introduces efficient request distribution capabilities by employing a broadcast based request distribution. If a query cannot be granted locally and requires external metadata, the request is broadcasted to only 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.

Consistency enforcement is considered as a contract between the system and the users to ensure data coherency across replicas in the system. To enforce consistency, the proposed system 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. Tanenbaum classifies this approach as primary-based remote-write protocol [22]. This approach ensures that the primary-copy of a metadata holds up-to-date version of the context under consideration. All update operations are carried out on the primary-copy replica server and the updates are propagated to the permanent-copy holders by the primary-copy.

An important aspect of the proposed system is that it utilizes a software multicasting capability as a communication medium for sending out access and storage requests to the network nodes. This is a topic-based publish-subscribe software multicasting mechanism, and it is used to provide message-based communication. Any node can publish and subscribe to topics, which in turn create a multi-publisher multicast broker network. The replica hosting architectural design of the system is built on top such publish- subscribe based multicast broker network system as illustrated in Figure 2. To achieve a multi-publisher, multicast communication mechanism, in our prototype, we use an open source implementation of publish-subscribe paradigm (NaradaBrokering [27]) for message exchanges between peers. The NaradaBrokering [16, 27] software, an open-source, publish-subscribe based messaging infrastructure, to provide such communication. We discuss the identified design issues of the proposed architecture in the following sections in detail.

5 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 the Server-Information Request message is two-fold. First purpose is to inform other servers about a newly joining server. Second purpose is to refresh the replica-server-information data structure with the updated information (such as proximity and server load information) 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 Service-Information Response message is sent back by unicast over a unique topic to the initiator. This message also 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. This is done by sending out Server-Information Request messages periodically to obtain up-to-date information. This model enables the system to keep track of proximity and server load information of the available network nodes. This is required for decision-making process of fundamental aspects of the decentralized system architecture such as replica-content placement and consistency enforcement.

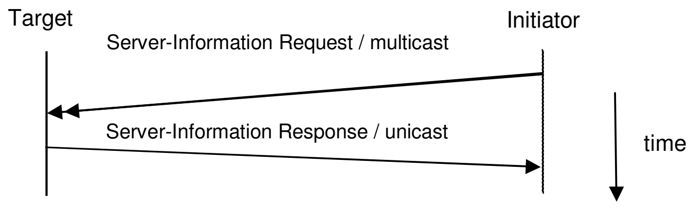
Target
Initiator
Server-Information Request / multicast
Server-Information Response / unicast
time

Figure 3 - Message exchanges for Hybrid Service Discovery Model. Each newcomer node sends out a multicast probe message to locate available services in the network. Each target node responds with a unicast message to make themselves discoverable. This figure illustrates the interaction between the initiator server and the target network nodes for service discovery model.

6 Replica-content placement

In a distributed system, data is replicated to enhance reliability and performance. Replica content placement is a replication methodology that deals with replicating newly inserted data onto other servers, which are capable for storage. After replication, there may only be two types of copies of a context in the system: permanent and server-initiated (temporary). A permanent copy of a context is used as a backup facility to enhance reliability. A server-initiated copy is created temporarily and used to enhance system performance. For the permanent-copy of a context, the Hybrid Service subscribes to a unique topic to receive access/update request concerning the context under consideration. For the server-initiated copy of a context, the Hybrid Service does not subscribe to a topic to minimize the number of messages exchanged for request distribution. The server-initiated copies are only used to enhance the system performance. The Hybrid service uses messages to provide replica-content placement. These messages are Context Storage Request and Response messages.

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 the Context Storage Request message is two-fold. First purpose is to assign handling of the storage operation to those Hybrid Service nodes that are selected based on the replica-server selection policy. The second purpose is to ask another Hybrid Service node to replicate or take over maintaining a context to enhance the overall system performance. Note that 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. By listening to this topic, the initiator receives response messages from the nodes that handled the storage request.

Decision metrics: The Hybrid Service uses some measurements to decide on replica-content placements. It takes both server load and proximity decision metrics into account when making replica-content placement decisions. The server load metric is a decision metric, which may be represented with multiple factors. We used the following two factors: a) topical information (i.e. number of unique topics, which the Hybrid Service subscribe to) and b) message rate (i.e. number of messages, issued by end-users, within a unit of time). If the number of topics, which a Hybrid Service subscribes to, is high, it is likely that the Hybrid Service will receive high number of access/update messages. If the message rate on a given Hybrid Service is increased, its performance will start dropping down. Therefore, we take into consideration the topical and message rate information as server load metrics. Each node can estimate its own server load based on these two factors. Server load is periodically recorded and it reflects the average load of a Hybrid Service at a given time interval. Note that, each nodes keeps decision metrics information about other nodes in the system. The server load information is obtained periodically by sending a Server-Information Request message to other available network nodes in the system. The proximity metric is the decision metric, which is used to indicate 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 topics. The latency in the ping request gives the proximity information between the two Hybrid Service instances.

Permanent Replication: When there is a newly inserted context into Hybrid Service, it starts the replica-content placement process (i.e. the distribution of copies of a context into replica hosting environment). This is needed to create certain number (predefined in the configurations file) of permanent replicas. We must note that, on receipt of a client's publish request, an existing node checks if it can handle the request under consideration. Each existing node decides if it is able to store the context by checking the server *instantaneous-server-load* against the *maximum-server-load-watermark*. Those replica-servers, which are capable of handling the request, perform the operation. However, if the node is overloaded, then this operation is forwarded to the best possible server. Figure 4 depicts message exchanges between an initiator Hybrid Service node and a target Hybrid Service node for replica content placement.

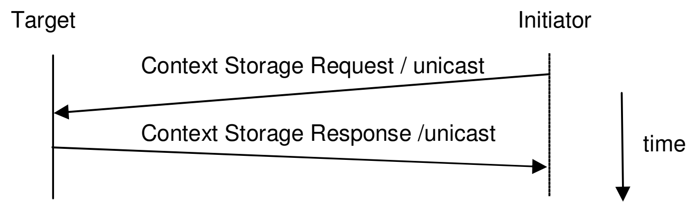
Target
Initiator
Context Storage Request / unicast
Context Storage Response /unicast
time

Figure 4 - Message exchanges for Storage (Replica content placement). This figure illustrates the interaction between the initiator server and the target network nodes to complete replica-content placement.

Our replica server selection policy takes into account two decision metrics: server load and proximity. To enforce our selection policy and select replica servers for replica-content placement, we adopt the replica selection algorithm introduced by Rabinovic et al [23] and integrate it with our implementation. 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.

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 (acknowledgement) message directly to the initiator (via unicast). The newly-selected primary-copy holder receives its Context Storage Request message with a flag indicating that it is the primary-copy holder of that context. Note that, 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. We also utilize the dynamic replication methodology, which is discussed in the next section. This is a performance optimization technique that may move/replicate permanent-copies of a replica onto servers if it is only beneficial for client proximity. This way, the system improves its responsiveness in terms of minimizing the access latency, as the copies of a replica are moved onto servers where the requests are originated.

Dynamic replication: In order to take into consideration sudden changes in client demands, we use dynamic replication as a performance optimization technique. Dynamic replication deals with the problem of dynamically placing temporary replicas in regions where requests are coming from. This is a push-based replication methodology where a dynamically generated replica is pushed (replicated/migrated) onto a replica server. Such replicas are also referred as push caches [28]. Dynamic replication decisions are made autonomously at each node without any knowledge of other copies of the same data. In our implementation, we adopt the dynamic replication methodology introduced by Rabinovich et al [23]. This methodology introduces an algorithm, which is used for the Web Hosting Systems, which maintain widely distributed, high-volume, rarely updated and static information. The dynamic replication algorithm by Rabinovich et al considers two issues: a) a replication can take place to reduce the load on a replica server and b) a replication can take place due to changes in the client demands. Our main interest is to provide an optimized performance by replicating temporary-copies of contexts to replica servers in the proximity of demanding clients. To this end, we only focus on the second issue, which concerns with creating replicas if it is only beneficial for client proximity.

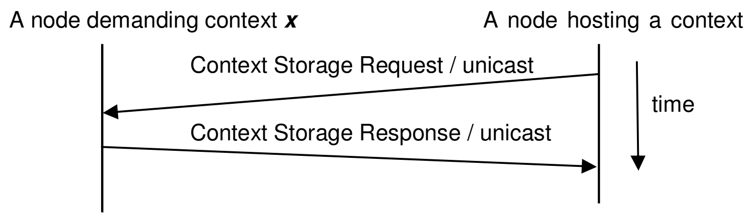
A node demanding context ***x***
A node hosting a context
Context Storage Request / unicast
time
Context Storage Response / unicast

Figure 5 - Message exchanges for Dynamic Replication/Migration. The dynamic replication/migration process replicates/migrates data if the demand exceeds certain thresholds. This figure illustrates the interaction between a hosting server and demanding server to complete replica placement/migration for context x.

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. It checks 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 [23]). 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 [23]. 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.

7 Consistency enforcement

At any given snapshot of the Hybrid Service network, the system may contain temporary and permanent of copies of a context. On one hand, temporary copies are kept for performance reasons. On the other hand, permanent-copies are kept for fault-tolerance reasons. Each Hybrid Service assigns/creates unique topics for each individual permanent-copy (to receive access and update requests), while it creates no topics for the temporary copies (to avoid flooding the network with access messages). This creates an environment where the system may have different versions of the context, as the temporary copies are not updated. To achieve consistency from the target applications perspective, the Hybrid Service introduces different models to address consistency requirements of different applications. The first model is mainly for read-mostly applications. For these applications, different copies of the context are considered to be consistent and the Hybrid Service allows clients to fetch any copies of the context (permanent or temporary). The second model is for the applications where the update-ratio is high and the consistency enforcement is important. In this case, the Hybrid Service requires the applications to subscribe unique topics of the metadata that they are interested. This way, these applications will be informed of the state changes happening in the metadata immediately after an update occurs. In this model, the primary-copy holder broadcasts the updates through the unique topic corresponding to the metadata under consideration.

We divide the implementation of consistency enforcement into two categories: “update distribution” and “update propagation”. The “update distribution” deals with how the Hybrid Service implements an update operation that take place on the distributed metadata store. The “update propagation” deals with how the Hybrid Service implements the methodology for propagation of updates. To achieve consistency enforcement the Hybrid Service uses messages. These messages are Primary-Copy Selection Request and Response messages, Primary-Copy Notification message, and Context Update Request and Propagation messages.

Primary-Copy Selection Request and Response messages: In order 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, a Hybrid Service node advertises the need for selection of primary-copy host of the context with following message: Primary-Copy Selection Request. This message is sent out by multicast by the initiator Hybrid Service node only to those servers holding the permanent-copy of the context under consideration. The purpose of the Primary-Copy Selection Request message is used to select a new primary-copy host if the original is considered to be down. 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. By listening to this topic, each existing node receives this message. 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 directly to the initiator 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 node 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: On receiving client publication requests, a Hybrid Service node first checks 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.

The system assigns a synchronized timestamp to each published context (newly written or updated). This is achieved by utilizing NaradaBrokering NTP protocol based timing facility. By utilizing this capability, we give sequence numbers to published data to ensure an order is imposed on the concurrent write operation that take place in the distributed data store. Based on this strategy, a write operation could take place on a data item, only if the timestamp of the updated context was bigger than the version number of the most recent write. This ensures that 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.

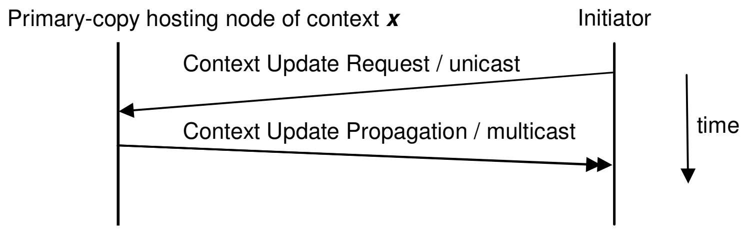
Primary-copy hosting node of context *x*
Initiator
Context Update Request / unicast
Context Update Propagation / multicast
time

Figure 6 - Message exchanges for update operation of a context. This figure illustrates the interaction between the initiator server and the primary-copy host node of context *x*.

An update operation is executed offline, i.e., just after an acknowledgement is sent to the client. The update distribution process is executed to perform updates on the primary-copy holder of a context. 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 handling of a context. The Context Update Request message means that the initiator node is interested in updating the primary-copy replica. This message is sent via unicast offline of the publication request. 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. 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 (*timeout_period*) in response to Context Update Request, the primary-copy host is decided to be down. In this case, the initiator node should select a new primary-copy host. After a new primary-copy host is selected, the aforementioned update distribution process is re-executed.

We utilize synchronized timestamps to label published metadata. This allows us to impose an order on the actions that take place in the distributed metadata store. In our implementation, we combine the synchronized timestamps with the primary-based consistency protocol approach. Based on this strategy, each published context is given a synchronized timestamp. An update operation could take place on a data item, only if the timestamp of the newly published update is bigger than the version number of the most recent update. This way, all write operations can be carried out on the primary-copy host, in the same order they were published in to the system. However, this approach has also some practical limits, as the update rate is bounded by the timestamp accuracy of the synchronized timestamps. To achieve ordering among the distributed updates, we use NTP protocol based synchronized timestamps provided by the NaradaBrokering software timing libraries [29].

Update propagation: In a distributed data-system, an update propagation process can either be initiated by the server which is in need for the up-to-date copy and wants the pull updates from primary-copy host (pull methodology) or by the server that holds the update and wants to push to other replica servers (push methodology) [30]. In our prototype implementation, we utilized push methodology for update propagation and multicast technique for dissemination of updates. Based on this methodology, whenever an update occurs the primary-copy immediately reflects the changes to the redundant copies in order to keep them up-to-date. Updates can be distributed in two ways: unicast and multicast [22]. In unicast update propagation, the primary copy server sends its updates to replica holders by sending separate messages. In multicast update propagation, it sends its updates using an underlying multicasting facility, which in turn takes care of sending messages to the network. For dissemination of updates, we use the multicast approach and publish the update to the unique topic corresponding to the metadata. This way, the system is able to send the updates only to those permanent-copy holding servers.

Primary-copy selection: The primary-copy selection process is used to select a new primary-copy host for consistency enforcement reasons, if the original primary-copy host is down at the moment. A primary-copy host of a context is considered down, if no answer is received in response to a message (such as Context Update Request message) that is directed to it. When the primary-copy host of a context is considered down, the primary-copy selection process is executed step-by-step as depicted in Figure 7 and explained as in the following.

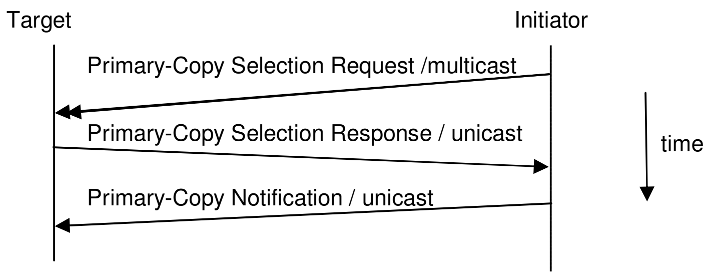
Target
Initiator
Primary-Copy Selection Request /multicast
Primary-Copy Selection Response / unicast
Primary-Copy Notification / unicast
time

Figure 7 - Message exchanges for Primary-Copy Selection process. This figure illustrates the interaction between the initiator server and the target network nodes to complete the primary-copy selection process. Time arrow is down.

Say, a Hybrid Service node finds out that a primary-copy of a context is down. In this case, the initiator broadcasts a Primary-Copy Selection Request message to only those servers holding the context to select the primary-copy host. On receipt of a Primary-Copy Selection Request message, each replica-holding server that maintains a "permanent" copy of the context under consideration, issues a Primary-Copy Selection Response message. Here, the purpose of a Primary-Copy Selection Response message is to inform the initiator that the answering node contains a permanent copy of the context under investigation. On receipt of the Primary-Copy Selection 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 a replica server selection process, which is described earlier, as the primary-copy server. In this case, A Primary-Copy Notification message is sent to the selected server indicating that it is selected as the new primary-copy host for the context under investigation. On receipt of a Primary-Copy Notification message, the permanent-copy holder becomes the primary-copy holder and subscribe the unique address (/UUID/PrimaryCopy) corresponding to the primary-copy of the context under consideration.

8 Access request distribution

On receipt of a client's inquiry request, a Hybrid Service node looks up for the requested context within local storage. If the context exists in local storage, then the inquiry is satisfied and a response message is sent back to the client. If the inquiry asks for external metadata, the system performs the request distribution (access) process, which is discussed in the next section in length. The communication between network nodes for request access distribution happens via message exchanges. These messages are Context Access Request and Response messages.

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 message 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. (Note that we use IP address of the initiator as topic to send responses via unicast back to the initiator). By listening to this topic, the initiator receives the response messages from nodes that answered the access request.

Request distribution: The prototype implements a request distribution methodology, which is based on broadcast dissemination where the requests are distributed to only those servers holding the context under consideration. This approach does not require keeping track of locations of every single data located in the system. It makes use of copies of a data that are not frequently accessed and kept only for fault-tolerant reasons. In turn, this improves the responsiveness of the system. In this scenario, the initiator node issues a Context Access Request message to the multicast group, if the client's access request is not satisfied in the local storage. This message contains minimum required information (such as context key) regarding the context in demand. The Context Access Request means that the initiator node is interested in discovering the qualified replica servers that may contain the requested context and answer with a response.

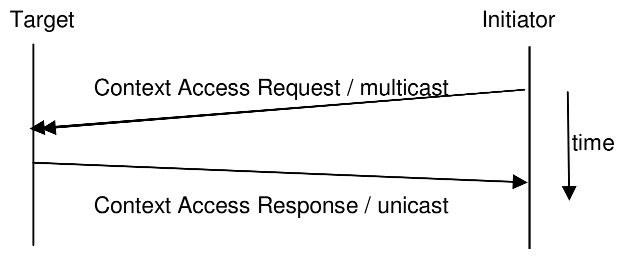
Target
Initiator
Context Access Request / multicast
time
Context Access Response / unicast

Figure 8 - Message exchanges for context access. This figure illustrates the interaction between the initiator and a target node hosting the context for request distribution. Time arrow is down.

On receipt of a Context Access Request message, a replica-holding Hybrid Service issues a Context Access Response message. The purpose of a Context Access Response message is to send a response with the context satisfying the query. (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 node, 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.

9 Prototype Evaluation

This section presents an evaluation of the prototype implementation of the Hybrid System Replica Hosting Architecture and investigates its practical usefulness. In this section, the following research questions are being addressed:

- 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: For the decentralized setting experiments (such as distribution, fault-tolerance and consistency enforcement), we have selected nodes that are separated by significant network distances. The machines, used in these experiments, are summarized in Table 2.

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 2 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 backend 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). 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: In order to investigate the research questions related with provide replica-content placement, access distribution, dynamic replication and consistency enforcement, the focus of the simulation experiments was on key-based publish (save operation) and inquiry (retrieve operation) capabilities. In the experiments, we used the following simulation parameters.

metadata size and volume: We chose average values for the size and volume of the metadata which were used in the simulation from a real life application Pattern Informatics in which the Hybrid Service is used. To this end, this metadata sample has a fixed size of 1.7Kbyte and the volume of the metadata is a thousand.

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*). The trade-off in choosing the value for *dynamic-replication-time-interval* is similar to the one for *backup-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 [23].)

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. The trade-off in choosing the value for *minimum-fault-tolerance-watermark* is the following. 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.

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. The trade-off in choosing this number is the following. 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. For example, if there are two replica servers, one in U.S. and the other in Australia, the query initiator located in U.S. may miss the result coming from the node located in Australia with a small *timeout-period*. 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.

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 based on the study introduced in [23].

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. (Rabinovich et al [23] discusses the dependency between replication and deletion thresholds that in turn indicates that the value of *replication-threshold* must be selected above *deletion-threshold.* In our investigation, we chose the *replication-threshold* value based on the study introduced in [23].)

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 3 Simulation parameters for the experiments

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.

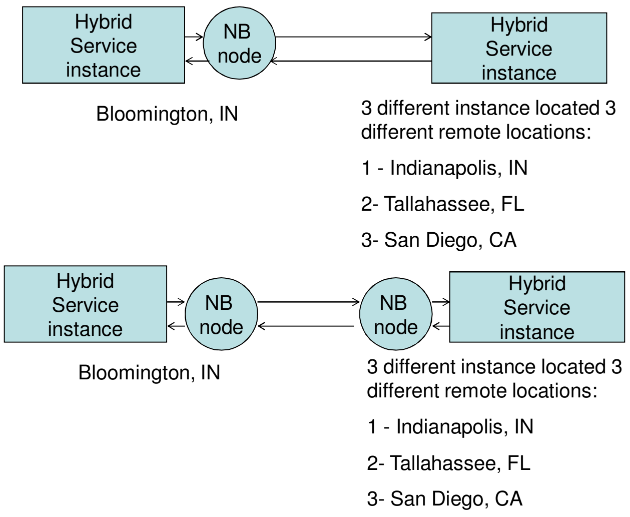

Figure 9 The design of the distribution experiment. The rounded shapes indicate NaradaBrokering nodes. The rectangle shapes indicate Hybrid Service instances located at different locations. The first test was conducted with one broker where the broker is located before the Hybrid Service instance in Bloomington, IN, while the second test was conducted with two broker nodes each sitting on the same machine before the Hybrid Service instance.

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?

Results of the distribution experiment: We conduct distribution experiments for three different locations corresponding to three different network spaces.

Figure 10 illustrates the experiment between Bloomington/IN and Indianapolis/IN. We repeat this test for two other locations as well. We extract the processing time involved for access request distribution. We depict the time spent in various sub-activities of distribution in

Figure **11** and list the results in
Table **4**. By analyzing the results, we observe that regardless of how the Hybrid Service instances are distributed, the system showed the same stable performance, which is around 3.6 ms when using one intermediary broker. This time includes the Hybrid Service system processing overhead 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. The results also indicated that the system performs well for continuous, uninterrupted request distribution operations.

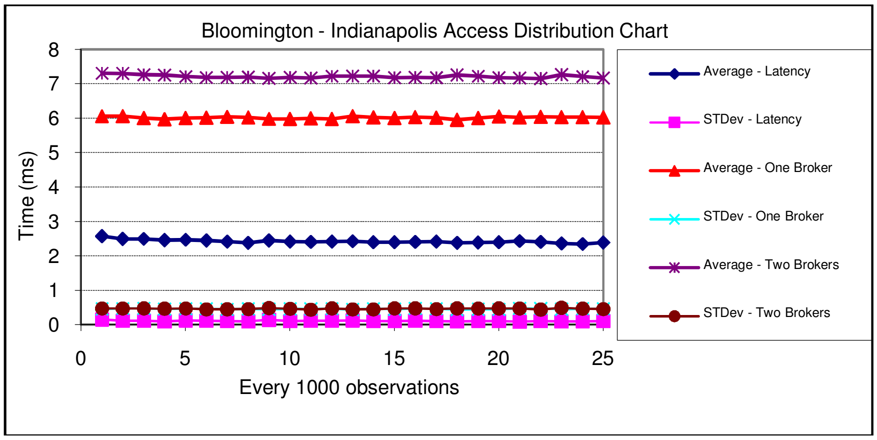

Figure 10 The Distribution Experiment Results between Bloomington and Indianapolis - Each point in the graph corresponds to average of 1000 observations.

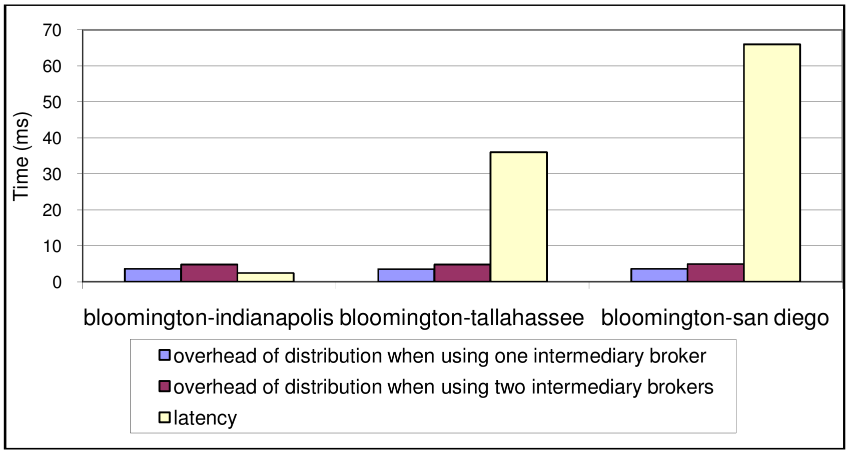

Figure 11 Time spent in various sub-activities of the request distribution scheme of the Hybrid Service

	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 4 Statistics for Figure 11. Overhead of request distribution. Average timings 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.

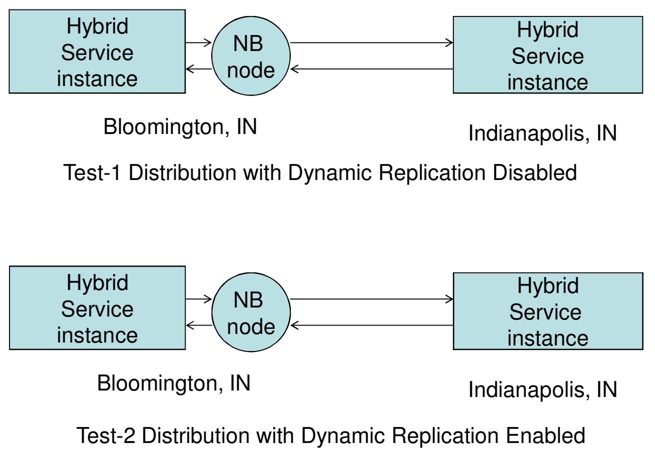

Figure 12 The design of the dynamic replication experiment. The rounded shapes indicate NaradaBrokering nodes. The rectangle shapes indicate Hybrid Service instances located at different locations. In the first testing case, dynamic replication capability is disabled. In the second testing case, dynamic replication capability is enabled.

Results of the dynamic replication experiment: Based on the results depicted in Figure 13, 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.

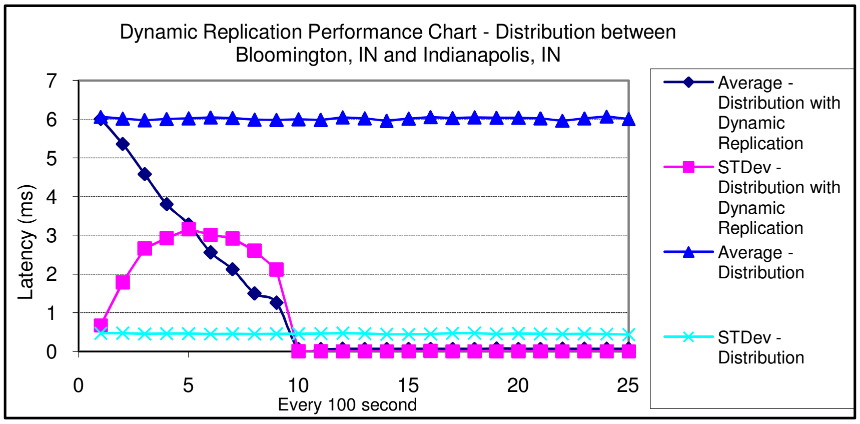

Figure 13 The results of the dynamic replication experiment.

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.

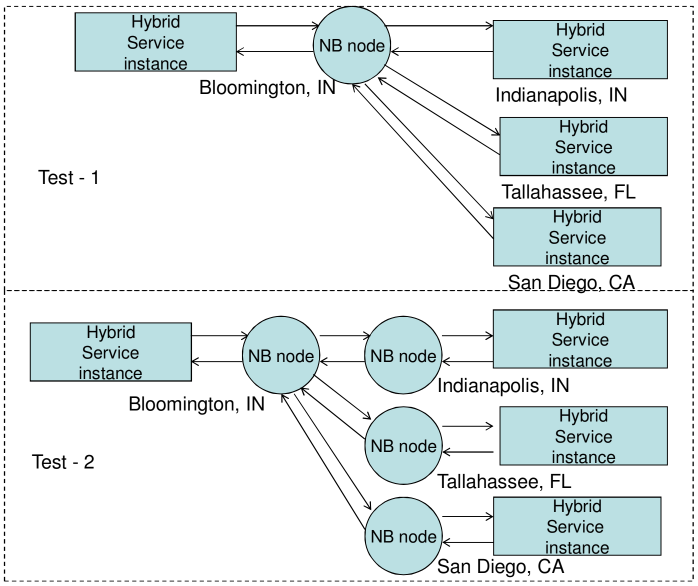

Figure 14 The design of the fault tolerance experiment. The rounded shapes indicate NaradaBrokering nodes. The rectangle shapes indicate Hybrid Service instances located at different locations. In the first testing case, we measure the end-to-end latency for varying number replica-content creation with 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. Figure 15 illustrates the results from one replica creation test. Based on the testing results, we extract the processing time involved to provide fault-tolerance by utilizing publish-subscribe based messaging schemes. We depict the time spent in various sub-activities of replica creation in Figure 16 and list in Table 5. By analyzing the results, we observe that the system presents a stable performance over time for replica creation. We observe 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 millisecond. The results also indicated that, the overhead of replica-content creation increases only by 1.2 ms, when we use an additional intermediary broker.

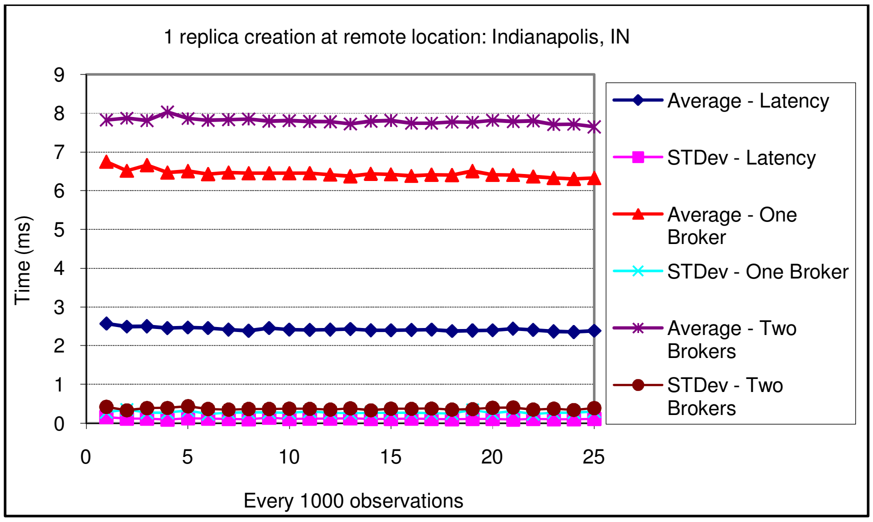

Figure 15 Fault Tolerance Experiment results when one replica is created at Indianapolis, IN. Each point in the graph corresponds to average of 1000 observations**.**

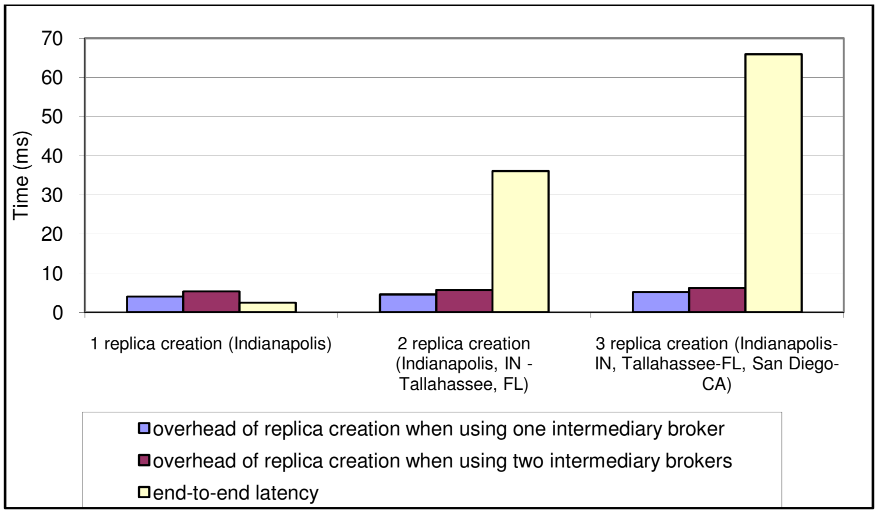

Figure 16 Time spent in various sub-activities of the replica-content creation scheme of the Hybrid Service.

	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 5 Statistics for Figure 16. Overhead of replica-content creation. Average timings in milliseconds.

Consistency enforcement experiment: The design of the consistency enforcement is similar to the distribution experiment depicted in Figure 9. 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. Figure 17 depicts the results from Bloomington-Indianapolis testing case. Based on the results, 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 Figure 18 and list in Table 6. This 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. By analyzing the results, we also observe that the system presents a stable performance over time for continuous consistency enforcement operations.

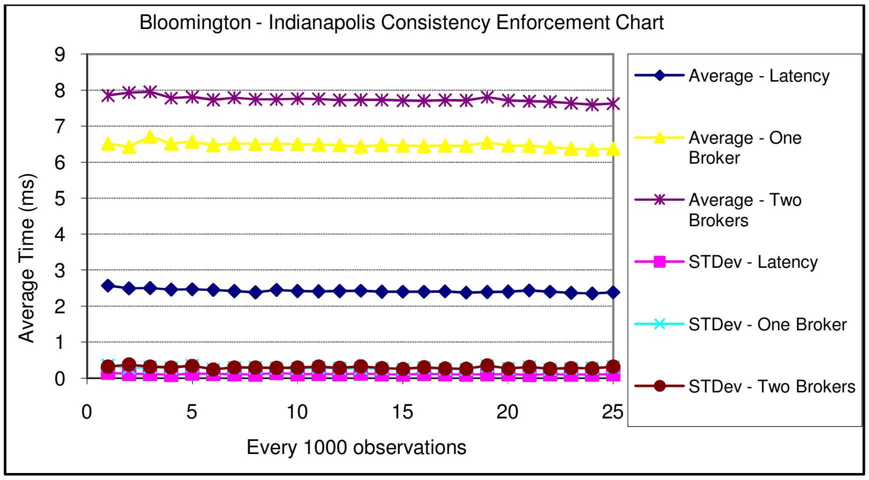

Figure 17 Consistency Enforcement Experiment Results when an update request (originated from Bloomington, IN) is carried out on the primary-copy holder located in Indianapolis, IN. Each point in the graph corresponds to average of 1000 observations.

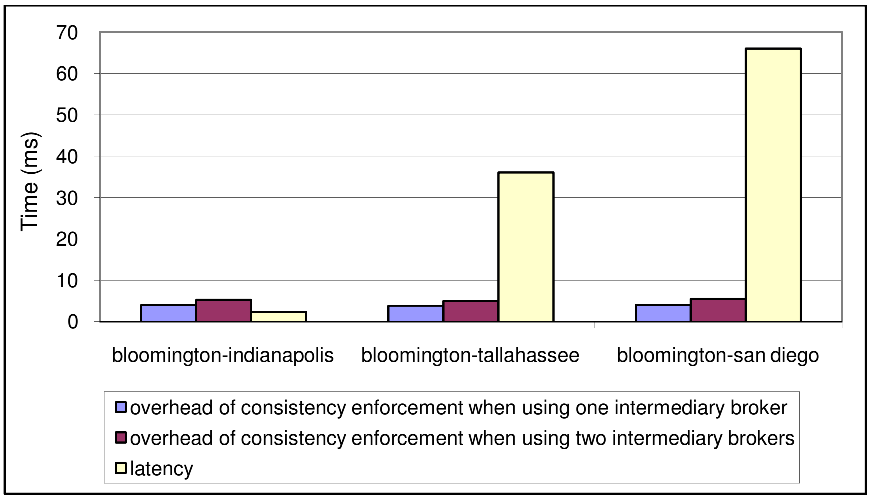

Figure 18 Time spent in various sub-activities of the Hybrid Service consistency enforcement scheme. The results analyze the overhead of distributing update requests to the primary-copy holder where the update requests take place for consistency enforcement reasons.

	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 6 Statistics for Figure 18. Statistics for overhead of update distribution. Average timings in milliseconds.

10 Conclusions and Future Research Directions

This research presented the Hybrid Grid Service architecture as a replica hosting environment and evaluated its performance. To achieve decentralization, we utilized publish-subscribe based messaging schemes to provide interaction among the distributed instances of the Hybrid System. We utilized a topic based publish-subscribe messaging communication to implement fundamental aspects of decentralized information systems such as fault-tolerance, distribution, and consistency enforcement. To improve the overall performance of the system, we have also used performance optimization techniques such as dynamic migration/replication, which improves overall system performance by moving/replicating highly requested metadata to where they wanted.

The evaluation of the system pointed that Hybrid Service presents stable behavior for access request distribution, replica creation and consistency enforcement over a high number continuous operations. The results indicated that, with our solution, the cost of achieving distribution, fault tolerance and consistency enforcement is in the order of milliseconds. We also observed that the cost of fault tolerance is higher than both the cost of distribution and the cost of consistency enforcement. This is because; there is an additional time required for performing additional unicast messages for higher fault-tolerance levels. The results also pointed out that we can achieve performance optimization by employing dynamic replication technique in decentralized metadata management. The results indicated that the cost of repetitive access requests could be reduced by moving temporary copies of contexts to where they wanted. Finally, this study pointed out the trade-off between performance and fault-tolerance. Here the fault-tolerance is considered in terms of availability (i.e. degree of replication). The results indicated that the cost of replica-content creation increases, when the degree of fault-tolerance increased.

To complete the system, we intend to research an information security mechanism for the distributed replica hosting system. This research should investigate 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.

Bibliography

1. Ken Arnold, A.W., Byran O'Sullivan, Robert Scheifler, and Jim Waldo, The JINI Specification. 1999: Addison-Wesley, Reading, MA.
2. The_Salutation_Consortium_Inc., Salutation architecture specification (part 1), version 2.1 edition available at http://www.salutation.org. 1999.
3. Guttman, E., Perkins, C., Veizades, J., Service Location Protocol, RFC 2165, available at http://rfc.net/rfc2165.html. 1997.
4. 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.
5. Ripeanu, M., Foster, I. Mapping the Gnutella Network: Macroscopic Properties of Large Scale Peer-to-Peer Systems. in In 1st International Workshop on Peer to Peer Systems. 2002.
6. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S. Search and Replication in Unstructured Peer to Peer Networks. in In 16th ACM International Concerence on SuperComputing. 2002. New York, USA.
7. Milojicic, D.S., et al. , Peer-to-Peer Computing, in HP Labs Technical Report HPL-2002-57. 2002, HP Labs.
8. 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.
9. Marin-Perianu, R., Hartel, P., Scholten, J. A Classification of Service Discovery Protocols. in Technical Report TR-CTIT-05-25 Centre for Telematics and Information Technology, University of Twente, Enschede. ISSN 1381-3625 2005.
10. R. Hermann, D.H., M. Moser, M. Nidd, C. Rohner, A. Schade, DEAPspace--Transient ad hoc networking of pervasive devices. Computer Networks 2001. Volume 35 p. pp 411-428.
11. Tang, D., Chang, D., Tanaka, K., Baker, M., Resource Discovery in Ad-Hoc Networks, in CSL-TR-98-769. 1998, Stanford University.
12. 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.
13. Pallickara, S., H. Gadgil, and G. Fox. On the Discovery of Topics in Distributed Publish/Subscribe systems. in Proceedings of the IEEE/ACM GRID 2005 Workshop, http://pat.jpl.nasa.gov/public/grid2005/ pp 25-32. Seattle, WA. 2005.
14. Happner, M., Burridge, R., Sharma, R., Java Message Service Specification available at http://java.sun.com/products/jms. 2000, Sun Microsystems.
15. Box, D., Cabrera, L., Crithchley, C., Curbera, F., Ferguson, D., Geller, A., Graham, S., Hull, D., Kakivaya, G., Lewis, A., Lovering, B., Mihic, M., Nib-

lett, P., Orchard, D., Saiyed, J., Samdarshi, S., Schlimmer, J., Sedukhin, I., Shewchunk, J., Smith, B., Weerawarana, S., Wortendyke, D. , Web Service Eventing available at http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf. 2004, Microsoft, IBM & BEA.

16. Pallickara, S. and G. Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. in Lecture Notes in Computer Science. 2003: Springer-Verlag.
17. Pallickara, S., et al., A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems. 2005.
18. Pallickara, S., et al., Support for High Performance Real-time Collaboration within the NaradaBrokering Substrate. 2005.
19. Fox, G., S. Pallickara, and X. Rao. A scaleable event infrastructure for peer to peer grids. in JGI '02: Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande. 2002: ACM.
20. Fox, G. and S. Pallickara, Deploying the NaradaBrokering substrate in aiding efficient web and grid service interactions. Proceedings of the IEEE, 2005. 93(3): p. 564-577.
21. Sivasubramanian, S., Szymaniak, M., Pierre, G., Steen, M., Replication for Web Hosting Systems. ACM Computing Surveys, 36(3):291--334, 2004.
22. Tanenbaum, A., Van Steen, M., Distributed Systems Principles and Paradigms. 2002. Cited in page 326.
23. 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.
24. 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.
25. Rodriguez, P., Sibal, S. , SPREAD: Scalable Platform for Reliable and Efficient Automated Distribution. Computer Networks, 2000. vol. 33, nos. 1-6: p. pp. 33-49.
26. Rabinovich, M., Aggarwal, A. RaDaR: A Scalable Architecture for a Global Web Hosting Service. in WWW8. May 1999.
27. 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.
28. Gwertzman, J.a.S., M. The Case for Geographical Push-Caching. in Proc. Fifth Workshop Hot Topics in Operating Systems, IEEE, 1996. pp. 51-55. Cited on page 327. 1996.
29. Bulut, H., S. Pallickara, and G. Fox. Implementing a NTP-Based Time Service within a Distributed Brokering System. in ACM International Conference on the Principles and Practice of Programming in Java, June 16-18, 2004 Las Vegas, NV. 2004.
30. Rabinovich, M. Issues in Web Content Replication. in Bulleting of the IEEE Computer Society Technical Committee on Data Engineering. 1998.