RC22101 (W0106-031) June 26, 2001
Computer Science

IBM Research Report

Locating Application Data Across Service Discovery Domains

Paul Castro, Benjamin Greenstein, Richard Muntz
University of California, Los Angeles
Los Angeles, CA 90095-1596

Chatschik Bisdikian, Parviz Kermani
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532

Maria Papadopouli
Columbia University
1214 Amsterdam Ave
Mail Code: 0401
New York, NY 10027-7003

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Locating Application Data Across Service Discovery Domains

Paul Castro*, Benjamin Greenstein, Richard Muntz
University of California, Los Angeles
Los Angeles, CA 90095-1596
{castrop,ben, muntz}@cs.ucla.edu

Chatschik Bisdikian, Parviz Kermani
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
{bisdik,parviz}@us.ibm.com

Maria Papadopouli*
Columbia University
1214 Amsterdam Ave
Mail Code: 0401
New York, NY 10027-7003
maria@cs.columbia.edu

ABSTRACT

The bulk of proposed pervasive computing devices such as PDAs and cellular telephones operate as thin clients within a larger infrastructure. To access services within their local environment, these devices participate in a service discovery protocol which involves a master directory that registers all services available in the local environment. These directories typically are isolated from each other. Devices that move across service discovery domains have no access to information outside their current local domain. In this paper we propose an application-level protocol called VIA that enables data sharing among discovery domains. Each directory maintains a table of active links to other directories that share related information. A set of linked directories forms a data cluster that can be queried by devices for information. The data cluster is distributed, self-organizing, responsive to data mobility, and robust to failures. Using application-defined data schemas, clusters organize themselves into a hierarchy for efficient querying and network resource usage. Through analysis and simulation we describe the behavior of VIA under different workloads and show that the protocol overhead for both maintaining a cluster and handling failures grows slowly with the number of gateways.

1. INTRODUCTION

Researchers in industry and academia commonly propose to manage pervasive computing devices such as Personal Digital Assistants (PDA), cellular telephones, hand-held computers, and embedded sensing devices with *service discovery architectures*. Examples of such architectures include SLP [11], Jini [13], Salutation [21], and Universal Plug and Play (UPnP) [22]. These architectures rely on service discovery protocols that have similar features. The main component of most protocols is a "master directory" or "master channel" located at a well-known network address. Devices register their services by contacting the master directory and clients can "discover" all the registered services by consulting the same directory. For example, a PDA that enters a new environment can use the master directory to locate a printer that is capable of printing a color photograph.

Master directories limit their scope to devices within a local *service discovery domain*. The boundaries of a service discovery domain can be administratively defined such as an IP subnet, or they can be the result of a physical property such as the range of a wireless network. For example, an office equipped with a Bluetooth [3] network can be its own service discovery domain. There can be many such domains for a single building. While this provides applications with a flexible, responsive, and robust means to find and use local services, many applications require a more global context for their operation. Consider a PDA outfitted with a camera that can take high-resolution photographs. Rather than store the pictures in its limited memory, the PDA off loads the pictures to storage elements it has previously discovered. As the PDA moves from domain to domain, it discovers and uses different storage elements to store its pictures. Later, we would like to retrieve all the pictures taken by the PDA. If we assume that the PDA keeps track of where it's been and what storage elements it has used, then it is relatively straightforward to retrieve the data. However, relying on a client participation model is unsatisfactory since a client is forced to maintain a history of its actions. This history is easily invalidated if a storage element moves from one domain to another after the PDA stores a picture on it. Also, users who do not have access to the history cannot retrieve the data. For example, suppose a user "discovers" a picture taken by the PDA on a local storage element and would like to see other pictures taken by the PDA. It would be very difficult to find these pictures without access to the history. Given the limitations of the client participation model we prefer to rely on an *infrastructure approach* to keep track of data.

A naive solution is to use a centralized index of all data on the network. Clearly, this solution does not scale well since the index is a bottleneck and a single point of failure. We can improve on this by employing a distributed index. Researchers have looked at using

The work reported in this paper was partially supported by NSF grants 0086116, 0085773, and 9817773 as well as an IBM Faculty Partnership Award.
*Part of this work was done while the authors were visiting IBM Research.

distributed indices in the context of *wide-area service discovery*. The goal of wide-area service discovery is to globally disseminate service information in master directories across service discovery domains. Rosenberg, *et al* developed the WASRV extensions to SLP that use multicast to disseminate information across discovery domains in a two-tier hierarchy [19]. Czerwinski, *et al* propose the Service Discovery Service (SDS), a secure descendant of SLP where service information is partitioned by semantics into a multi-tier hierarchy of distributed servers [6]. Multiple hierarchies can be maintained in SDS though the construction of these hierarchies is currently left to administrative policy. While not specifically for WAN applications, Adjie-Winoto, *et al* developed the Intentional Naming System (INS) that replicates a tree index among multiple server nodes to share service information [2].

As computers become more pervasive, data will be fragmented and distributed over many devices on the network. Service discovery domains enforce a cellular topology where data is inaccessible between cells. Finding the right data across domains will be of fundamental importance for many applications. While distributed index approaches such as those proposed for wide area service discovery can be employed, there are many issues that must be addressed. In general, replication does not scale well in the wide area. Both disseminating replicas of an index and keeping all the replicas of that index synchronized is problematic. While hierarchical schemes scale better, the reliance on policy to construct these indices places too much burden on administrators to centrally manage and coordinate the construction of the hierarchy. It is also unclear what an optimal hierarchy should be. Ideally, we would like a scheme that scales well, imposes low-overhead on the network, is self-organizing, and is robust to failures.

We have developed an application-level protocol for Verified Information Access (VIA[1]) to address some of these issues. We are focused on supporting data sharing across service discovery domains for specific applications. We are interested in applications that need to recover specific data items that could be located anywhere in the network such as the previously described PDA camera example. We are also interested in supporting applications that need to share data to a wide audience such as a distributed sensor system. Our approach is also applicable to other data sharing applications such as the increasingly popular Internet peer-to-peer file sharing architectures.

The design of VIA was guided by the following observations about sharing data across service discovery domains:

- An index should be tailored to the pattern of queries and the content of the underlying data sources. Instead of creating a general index to handle all queries we can create a smaller index optimized for a few applications. Application specific indices may be more appropriate and have the advantage of being practical to implement. For example, we can construct a distributed sensor system that detects congestion points on the highway. If applications are only interested in the "trouble spots" on the highway we can create a specific data-centric structure that allows applications to find that data quickly.

- Rather than break apart and distribute the index by content as is done in hierarchical server schemes, we can distribute the index so that each server only manages "local" information. For example, a Carnegie Hall server is only responsible for tracking data left on devices within the Carnegie Hall discovery domain. While this distributes semantically related parts of the index over many servers, changes to the index outside of Carnegie Hall have minimal impact on the local server. To reconstruct the fragmented index, servers are connected by "links" to each other in a peer-to-peer overlay network.

The remainder of this paper is as follows. In Section 2 we provide an overview of VIA and describe our self-organizing, data-centric, hierarchical indexing scheme. Section 3 provides details about specific operations in the VIA protocol. We develop equations in Section 4 which predict the overhead of VIA under certain conditions and discuss optimality considerations. In Section 5 we report measurements from a PARSEC [17] simulation of two operations in VIA as well as workload measurements for a VIA testbed under different distributions of queries and data. In Section 6 we discuss related work. Finally, we summarize this paper and describe planned work in Section 7.

2. VIA OVERVIEW

In this section we provide an overview of VIA. Our goal is to design an overlay network maintained by an application-level protocol based on the principles of robustness, self-organization, scalability, simplicity, and local autonomy.

- Robustness - pervasive computing environments are populated by a dynamic array of information devices. These devices can be mobile and are subject to various types of failure. Services built on these devices will exhibit similar properties. VIA should be resilient to these effects while remaining efficient.

- Self-organization - to minimize administration overhead we would like our distributed index to be self-organizing. Discovery domains can add and remove themselves from the index as the data changes. A self-organizing approach can also adapt to dynamic conditions.

- Scalability - while it is difficult to achieve a complete solution for wide-area data dissemination, VIA should maximize scalability while bounding the resources needed to implement it. Ideally, VIA should support a large number of discovery domains and devices as well as a large number of data disseminating applications.

- Local autonomy - participation in a wide-area scheme necessarily relies on entities outside local control. Discovery domains should have the ability to control the amount of resources they are willing to spend for participating in VIA.

We provide an overview of VIA in this section. Details of the VIA protocol follow in Section 3.0.

1. Also Latin for "way."

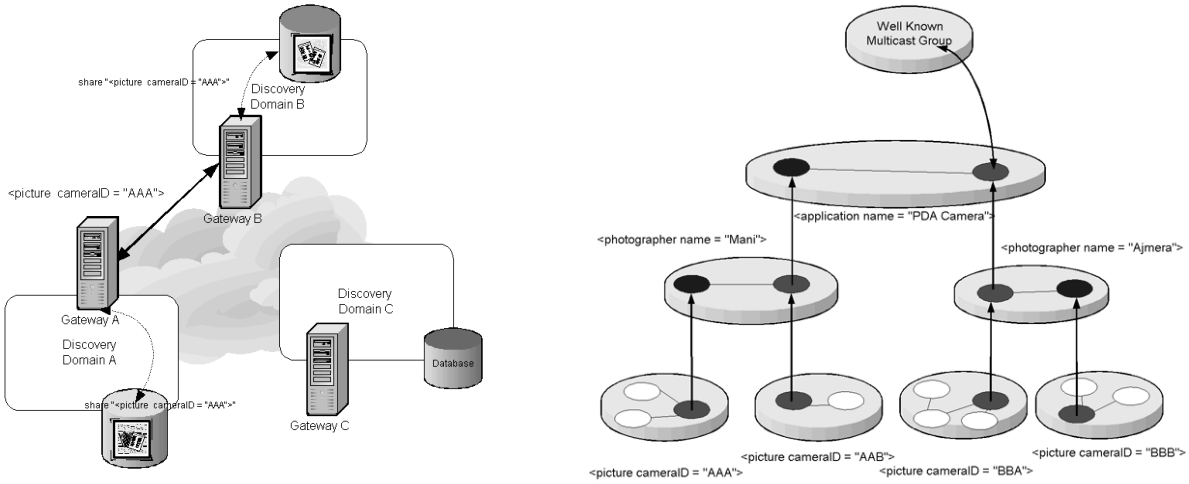

Figure 1: a) Topology of discovery domains. b) Hierarchy of clusters.

2.1 Gateways

Figure 1a shows the expected topology of discovery domains on the network. In the figure, we assume that the master directory service resides on a "gateway" that connects a discovery domain to a "main channel" allowing communication between gateways. Gateways act as mediators between devices in a discovery domain and resources available in other domains. The main channel acts as a broadcast mechanism between gateways and can be a implemented in many different ways, e.g. an IP-multicast channel or a set of cooperating servers. We assume that the main channel provides Internet-like connectivity between gateways.

Applications describe data they produce through *metadata tags*. A metadata tag can function as a description of data or it could be the data itself. In VIA, we assume that a metadata tag is an ordered list of attributes with a finite set of values. Applications can initiate data sharing by sending a metadata tag to the local gateway in a special share message. A query is also a list of attributes with a finite set of values. An attribute in a query can be "wildcarded" indicating "don't care" for an attribute.

Clients direct all queries for data to the main channel. In the absence of any other mechanism, all gateways listening on the main channel receive the query, process it against the set of metadata tags they have recorded, and only return non-null results to the client. This is analogous to popular peer-to-peer file sharing mechanisms that rely on controlled flooding to query and disseminate data [9]. In general, flooding is not a good solution since it can saturate the capacity of the main channel. Flooding also maximizes the number of queries each gateway must process since a gateway is forced to listen to all queries on the main channel.

Ideally, only the set of gateways that have non-null responses to a query should receive it. We can approach this ideal in two ways: 1) modify the main channel to more selectively deliver queries to the relevant gateways; 2) organize the gateways to cooperatively filter queries to minimize the number of queries each must process and reduce the requirements on the main channel. Both these approaches are complementary. VIA focuses on the second approach by creating an application-level overlay network. While an application-level protocol may be less efficient than a network layer approach, it does not require modifications to existing network protocols or hardware.

Researchers have investigated application-level peer-to-peer overlay networks to address perceived flaws in IP multicast. Yoid [8] and Narada [5] propose end system application-level multicast alternatives in which end systems self-organize into broadcast groups without any router support beyond supporting IP. Scattercast is similar in concept except it relies on special proxies in the network to support the formation of groups [18].

Figure 1b depicts an application-level overlay network of gateways with hierarchical semantics for the PDA camera scenario. In the figure, only one gateway listens to the main channel while other gateways have elected to "get behind" selected gateways that cooperatively filter queries for them. The top-level gateway (on the main channel) performs the least-restrictive filtering by passing on all queries that are relevant to the PDA camera application. At the next level, gateways filter on the name of the photographer that took the picture. The final level of filtering is done at the bottom layer where gateways check the ID of the camera that took the picture. Gateways benefit from this hierarchical scheme since it reduces the number of irrelevant queries they receive. The main channel benefits from reduced capacity requirements. As in any hierarchical scheme, workload distribution is not equal as top-level nodes act as "martyrs" for the benefit of nodes deeper in the hierarchy. Relieving these top-level nodes is outside our current consideration though this can be accomplished by load balancing schemes or changes to the main channel (for example, cooperating multicast channels [20]).

VIA's contribution is the ability for gateways to self-organize into a hierarchy based on the types of queries in the main channel as well as the actual data items being shared on the network. This is accomplished through "linking" operations in which a gateway joins a *cluster* of other gateways that have similar data. These linking operations can be repeated to form clusters of clusters. Clusters

aggregate as needed to create a cluster hierarchy. Only gateways that require filtering join a cluster hierarchy while other gateways remain on the main channel. Of course, the benefits of moving a gateway off the main channel are offset somewhat by the cost of placing the gateway into a hierarchy. VIA attempts to create an advantageous balance between the two (see Section 4).

2.2 Self-organization

Gateways operate autonomously from other gateways and only have partial knowledge about the state of other gateways on the network. Gateways on the main channel may receive many irrelevant queries; it is advantageous for them to join a cluster to minimize the number of irrelevant queries they must process. While there is some cost associated with being a member of a cluster, this cost can be considerably lower than the cost of processing many irrelevant queries.

In VIA, a gateway must determine if it is doing too much wasted work. This can be done by monitoring the number of relevant and irrelevant queries it receives. If it is processing too many irrelevant queries, the gateway attempts to join a cluster on the network. Note that there can be several clusters on the network and a gateway must determine the "best" cluster to join based on its limited knowledge about the state of the system. The "best" cluster is one one that will forward the least number of irrelevant queries to the gateway. To join a cluster, a gateway locates the members of the cluster and links to the "closest" member.

The number of queries that are irrelevant to a gateway are a function of the metadata tags on the gateway and the query distribution. In VIA, the metadata tags encode information about the expected query distribution through the ordering of the attributes. From this ordering, a gateway can find the best cluster to join for a particular metadata tag purely through *generalizing* that tag. Generalization is an iterative process where a metadata tag is transformed into an increasingly less restrictive filter. Ideally, this transformation should consider all other metadata tags on the network. However, a gateway generally cannot know the state of the other gateways so generalization is a greedy transformation based on local information. This approach is somewhat of a gamble since a gateway does not know if a cluster for the generalized metadata tag exists. Though not discussed in this paper, a better but more complex strategy would employ a less greedy transformation that considered the clusters already on the network; we are investigating this as part of our research.

Aggregates of clusters form a cluster hierarchy. A cluster hierarchy represents a *logical filtering hierarchy* where nodes higher in the hierarchy reduce the workload of nodes lower in the hierarchy by filtering out irrelevant queries. The filtering power of a hierarchy is determined by the metadata tags on each gateway. In VIA, metadata tags can be employed as filters to process queries. We say a metadata tag *M** subsumes another metadata tag *M* (written as *M* > M*) if $M \subseteq M^*$. We can also say that two metadata tags are equal (*M*=M*) if they describe the same set of data.

In general, a gateway G with metadata tag *M* can get behind any gateway with metadata tag *M** if *M*>M*. Given a set of possible gateways to get behind, G would like one that results in the least number or irrelevant queries. Ideally, G would like to be behind a gateway G', with filter *M** such that M*>M and M^*/M is minimal. The generalization process in VIA attempts to find the minimal filter *M** for a given metadata tag *M*.

Consider the following situation depicted in Figure 2. Three discovery domains are on the main channel via gateways G1, G2, and G3. Devices in each domain produce data described by the metadata tags *F1*, *F2*, and *F3*. We commonly think of these tags as descriptions of the content of data in the discovery domain, but they could also be the actual data (for example, sensor readings). A device sends a tag to a gateway in order to share this data outside of the local domain. Initially, G1, G2, and G3 receive every query that clients send to the main channel.

Suppose all queries are directed for data managed by G3. G1 and G2 must process these queries even though they cannot respond to them. To correct this egregious situation, G1 attempts to get behind another gateway by generalizing its current metadata tag *F1* to *F1** and then joining a cluster with a matching filter to *F1**. At the first iteration *F1 = F1** and G1 uses the main channel to find another gateway with a matching metadata tag.

Unfortunately for G1, there are no other gateways it can get behind. G1 assigns itself as the "root" of a cluster described by *F1** and will perform filtering for any gateway that joins its cluster. It should be noted that G1 does not do any more work than it did before it generalized. While it does not benefit from its actions, G2 does. G2 also realizes that it is doing too much work and generalizes its tag *F2* to *F2**. When G2 goes to the main channel it discovers that *F1** and *F2** match. G2 leaves the main channel and sets up a communications link to G1. G1 will now forward all queries that match *F1** to G2. G1 and G2 form a cluster described by *F1**. This process can be repeated as long as the metadata tags can be generalized. For example, G1 can make a second attempt to join a cluster by generalizing *F1** to *F1***. G1 goes to the main channel and discovers from G3 that *F1**= F3*. G1 can now leave the main channel and get behind G3.

The reverse can also happen. Suppose the content of the queries change and now most of them are directed at G2. In the final topol-

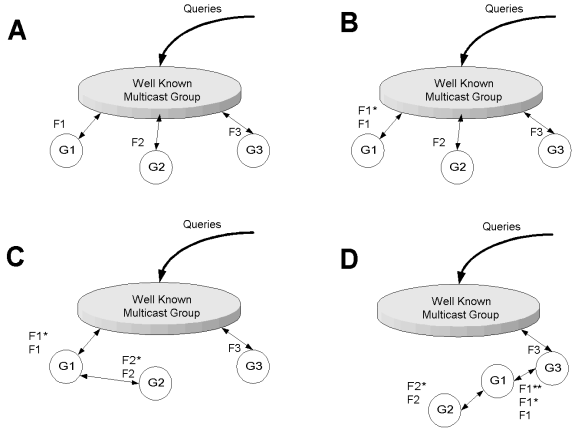

Figure 2: Filtering.

ogy of our previous example, queries are forwarded along the path G3 - G1 - G2. This indirection increases the query response latency perceived at the client and causes extra work for G1. If this situation occurs, G2 can break out of the current cluster hierarchy and return to the main channel.

VIA forms hierarchies by allowing clusters to cluster together under a common root. If a gateway G is a cluster root then it can repeat the cluster process with other roots. For example, assume a cluster rooted by G1 has metadata tag *M** and the cluster rooted by G2 has metadata tag *F**. G1 and G2 generalize their metadata tags to *M*** and *F***, respectively. If M**=F** then G1 can link to with G2 to form a new level on the cluster hierarchy with only G1 as the root of the new cluster. Although this higher level cluster does not filter out all irrelevant queries, the G2 will have some filtering done for it by G1.

Gateways create a cluster hierarchy in a bottom-up fashion. Metadata tags provide the semantics for clustering and aggregation. A cluster hierarchy is a distributed index of these tags that can efficiently process some queries. This can be restrictive since the index must be based only on the data contained within each discovery domain. This may result in an index that performs poorly for the queries in the main channel. A top-down approach would partition the data based on the content of the queries. We consider VIA to be a hybrid approach. Metadata tags describe a data schema for an application that can be organized for an expected query workload. This data schema acts like "DNA," guiding the bottom-up formation of cluster hierarchies. Thus, VIA can be used to create many different hierarchies by just specifying different data schemas for each application. More details can be found in Section 3.

VIA creates a filtering hierarchy which behaves differently than other hierarchical schemes. In a more conventional hierarchy, parent nodes have knowledge about the content of their children and will not forward irrelevant queries. Filtering is weaker in VIA since parent nodes do not know the contents of their children and will pass on any query that makes it through its filter to all its children. However, what we lose in query processing efficiency we gain in protocol simplicity since there is no need to keep data consistent between parents and children. This is ideal in a service discovery architecture where data is dynamic and update costs need to be minimized.

3. VIA PROTOCOL DESCRIPTION

VIA consists of operations broadly classified into two categories. At its most basic, VIA allows gateways to create, discover, and maintain clusters of themselves on the network through *spanning tree operations*. In addition, VIA aggregates clusters into cluster hierarchies for efficient query processing through *aggregation operations*.

Gateways locally decide whether or not they should aggregate. Since gateways only have partial knowledge about other parts of the network, these local decisions are based partly on heuristics which result in globally beneficial behaviors for aggregation.

We simulated VIA using the Parallel Simulation Environment for Complex Systems (PARSEC), a C-based discrete event simulation language [17]. We have also implemented prototype VIA gateways in our laboratory as part of our on-going research in pervasive computing infrastructures. Details about our simulations and workload experiments are presented in Section 5.

3.1 XML Data Schema

In our implementation, we rely on the eXtensible Markup Language (XML) to specify metadata tags. XML metadata tags are necessarily hierarchical and are used to guide the aggregation process. All photographs in our PDA camera example would have a metadata tag such as:

```
<application name=”PDA color camera”>
    <photographer name = “Mani”>
        <picture time=”12 Aug 2000,1220 GMT”
cameraID=”AAA” fileLoc=”$/12”/>
        <picture time=”12 Aug 2000,1223 GMT”
cameraID=”AAA” fileLoc=”$/22”/>
        <picture time=”12 Aug 2000,1245 GMT”
cameraID=”AAA” fileLoc=”$/145”/>
    </photographer>
</application>
```

From this data schema we could have a data cluster for "photographer name =’Mani’" and a cluster for "photographer name =’Ajmera’." We could also have a cluster for the more general "photographer" type with the "name" attribute set to a wildcard. This cluster subsumes the previous clusters where photographer name is specified.

Queries are partially instantiated versions of the XML metatags. For example, a query for all pictures taken by Mani would be:

```
<application name=”PDA color camera”>
    <photographer name=”Mani”>
        </picture time=”*” cameraID=”*”>
    </photographer>
</application>
```

In the query, the picture attributes for *time* and *cameraID* use the "*" wildcard to indicate "don’t care" on time and any cameraID. Note that the query above represents a Boolean "AND" of the attributes in the query. We can achieve Boolean "OR" semantics by chaining together queries like the one above separated by the special delimiter "||".

The design of VIA supports the construction of application specific indices. All attribute names in VIA are local to a single application. To construct an index over a set of applications, we would need to define an ontology that maps attribute names from one application to another. However, for this paper we restrict our discussion to supporting a single, application specific index.

A client sends all queries to the main channel. We model the main channel as an IP-multicast group. These queries are received by root gateways of other clusters. These clusters return their non-null responses to the client. Though beyond the scope of this paper, we

can adopt an incremental response scheme prevent message implosion at the client-side.

Gateways generalize metadata tags by adding "don't care" conditions to values in the tag. This process is application dependant and should be done in a manner that optimizes the partitioning of the data for an expected query workload. In our current implementation, generalization occurs by systematically placing wildcards on all values at the deepest level in the schema that is not already wildcarded.

For example, a generalized version of the metadata tag for the PDA camera example would be:

```
<application name="PDA color camera">
    <photographer name = "Mani">
       <picture time="12 Aug 2000,1220 GMT"
cameraID="*" fileLoc="*">
       <picture time="12 Aug 2000,1223 GMT"
cameraID="*" fileLoc="*">
       <picture time="12 Aug 2000,1245 GMT"
cameraID="*" fileLoc="*">
    </photographer>
</application>
```

This generalized metadata tag represents all pictures taken by Mani regardless of the cameraID or file location but distinguished by a timestamp.

3.2 VIA Spanning Tree Operations

VIA has four distinct spanning tree operations: *growth*, *non-root failure*, *root failure*, and *merging*. Growth and non-root failure are the basic operations. Root failure and merging are variants of these operations

3.2.1 Basic Operations: Growth

Gateways participate in VIA by maintaining a cluster table that specifies information about the cluster and links to other gateways in the cluster. Each individual gateway can limit their participation in VIA by specifying the number of links they are willing to process in their cluster table. If a gateway is willing to accept *b* links ($b \geq 2$) and the gateway currently has L links then we define the *edge set* of a cluster to be any gateway where ($L < b$). New membership to the cluster is only accepted by the edge set of the cluster. We also define the *rootside link* of a gateway as the link that connects the gateway to the partition of the cluster that contains the root node; this is normally the first link a gateway forms when it joins the cluster.

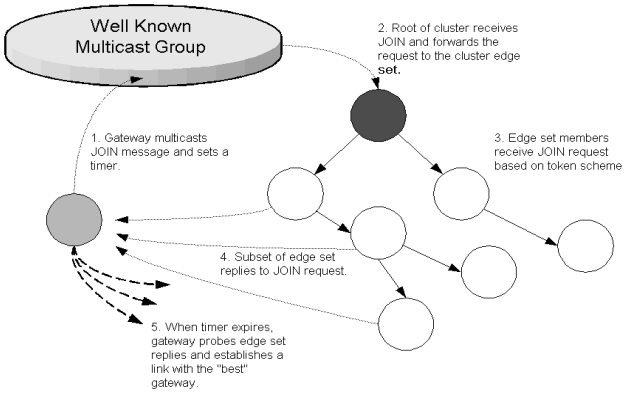

Figure 3: Growth state diagram.

All clusters have a unique ID created when the cluster first forms. In our current implementation, the ID is a 128-bit randomly generated ID which is assumed to be unique across the network. This is the same scheme adopted by Jini service identifiers [7]. A combination of the ID, the doctype, and a metadata tag should uniquely identify every cluster on the network.

The following algorithm is used where G is a gateway seeking to join a cluster:

1. G multicasts a JOIN message containing the metadata tag of the data it is sharing and a token count. G sets an "edge timer" and waits for responses. If G does not receive any responses before the edge timer expires then G assigns itself as the root of a new cluster and waits for JOIN, LINK, QUERY, or FAIL messages.

2. If a matching cluster exists, G will receive EDGE RESPONSE messages from the edge set of the cluster. The number of responses is limited by a token count scheme (see below). EDGE RESPONSE messages contain the network address of the responding gateway, the DTD identifier, the metadata tag of the cluster for which it received the JOIN request, and a 128-bit ID for this cluster. When the edge timer expires, G probes the gateways from which it received the edge responses. Probing consists of "pinging" the gateway and measuring the round trip delay. In the current implementation, G will link to the gateway that has the lowest round trip delay for the ping. Ties are broken arbitrarily.

3. To set up a link with another gateway, G sends a LINK message to the gateway. If the gateway returns an ACCEPT message, then G adds an entry into its link table for the gateway and sends out a REFRESH message for the link. The REFRESH message is the heartbeat message that maintains the soft state between the two gateways. Currently, links must be refreshed at least every 60 seconds. The link is now established and heartbeat messages should be exchanged between the gateways for that particular cluster. Heartbeat messages are unicast using UDP.

4. After the link is set up, G will send a CHANGE ID message to all other links it has for this cluster. This only happens if G is recovering from a failure.

The edge set for an arbitrary cluster could be quite large and we need to be sure that a gateway does not get overwhelmed by responses to a JOIN message. When a gateway multicasts its JOIN message, it includes a *token count* that represents the maximum number of responses it would like to receive. Gateways use this

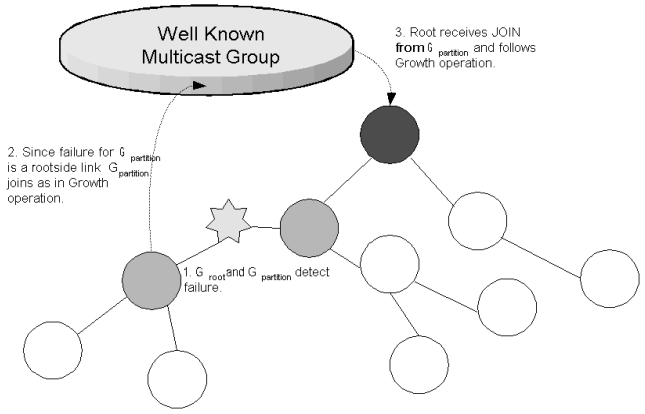

Figure 4: Non-root failure state diagram

token count to limit the number of responses to a JOIN message and also as a means to keep the topology of the cluster roughly balanced.

When the root of a responding cluster receives the request, it checks its link table to see if it has an open entry. If it does, it consumes one of the tokens in the token count, sends an EDGE message to the requester, and probabilistically distributes the remaining tokens to its peers. If it does not have an open link, it just probabilistically distributes all the tokens to its peers. Subsequent nodes repeat this process. In this scheme it is more likely that a node closer to the root will respond to the JOIN. VIA relies on randomization to add new members to the cluster in a more balanced manner. Ideally, we would like to minimize the network diameter of the cluster since this is related to the length of the path a query must travel.

3.2.2 Non-root Failure

A non-root failure occurs when a gateway fails or a link is broken between gateways. Links can fail because of network partitions, congestion, gateway failures, etc. Failures create multiple partitions of a cluster and it is up to VIA to restore cluster connectivity. In general, at least two gateways will detect a failure (except in the case where a "leaf" gateway fails). Only one of these gateways will be in the partition that contains the root. We use this fact to coordinate the repair process; in VIA, gateways only need to repair rootside failures. Consider two gateways that detect a link failure. G_{root} is the gateway in the root partition while $G_{partition}$ is the root in the non-root partition. For G_{root}, the failure is in a non-rootside link. The opposite is true for $G_{partition}$.

1. When G_{root} detects a failure on a non-rootside link, it updates the entry in its link table and does nothing.

2. When $G_{partition}$ detects the failure, it attempts to join the partition that contains G_{root} since the failure is on a rootside link. It does so in the manner specified by the Growth operation (see Section 3.2.1)

Note that this procedure will work for an arbitrary number of failures. For any number of failures, there will be several partitions but only one will contain the root. The gateways in the remaining partitions that detect the failure will attempt to rejoin the root partition.

3.2.3 Root Failure

Root failures are handled similarly to non-root failures. The main difference is that a root failure will create multiple non-root partitions. To combat this, each peer of the root records that it is a root peer in the cluster entry in its link table. If it detects a failure in the root, then it assigns itself as root. Each gateway generates a new ID and sends a CHANGEID message to its peers. The new ID is propagated to all members of its partition. After this, the new roots attempt to merge with the other trees as specified in the next section.

3.2.4 Merge

Multiple clusters that share the same data types but are not connected can be formed due to failures or delays in the network. For example, a gateway could set a short edge timer and assign itself as the root of a new cluster prematurely. To handle this situation root nodes periodically multicast EXIST messages to other roots. EXIST messages contain an identifier for the document type of the

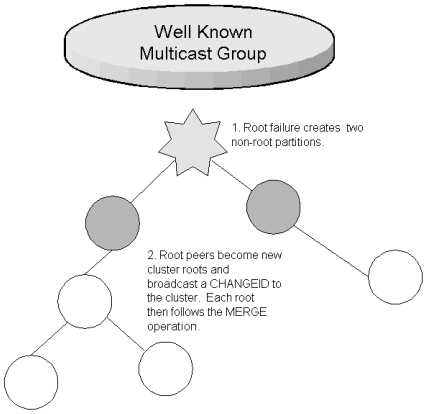

Figure 5: Root failure state diagram

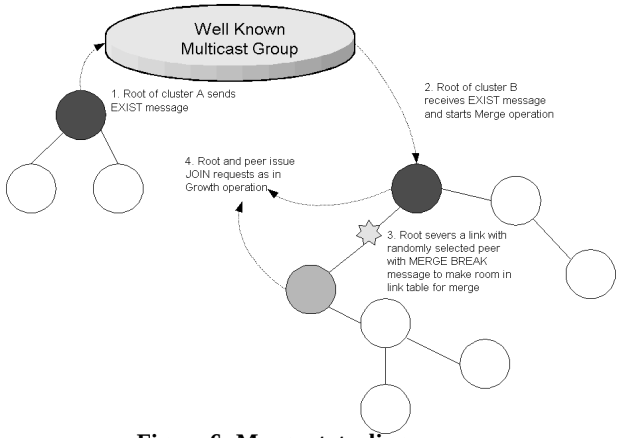

Figure 6: Merge state diagram

cluster, the metatag, and the cluster ID. If a root node G_1 discovers the existence of another cluster (rooted by G_2) that has a matching document type and metatag but a different cluster ID, then it does the following:

1. G_1 uses a backoff protocol to merge with the other tree. G_1 sets a random timer and waits for a MERGE message from G_2. If a MERGE message arrives, then G_1 does nothing. Otherwise, G_1 sends a MERGE message to G_2.

If G_1 has open links for the cluster, then G_1 joins as specified by the growth operation. If G_1 does not have an open link, then it randomly selects a link and breaks it by sending a MERGE BREAK message. The resultant clusters now join as specified in the growth operation.

3.3 Aggregation Operations

The aggregation operations enhance the spanning tree operations to allow the formation of cluster hierarchies. VIA has two distinct aggregation operations: *aggregate* and *split*. An analysis of VIA (see Section 4) indicates that there is a break point in network cost between VIA and flooding. With complete knowledge about the system, gateways could determine perfectly when they should aggregate. However gateways operate on partial knowledge and VIA relies on a *hit ratio* and *filtering coefficient* based heuristics to determine when a gateway should join and leave a cluster.

The hit ratio H of a cluster is defined as: Q_{res}/Q where Q_{res}= # non-null responses for that cluster and Q= # queries received for that cluster. A gateway maintains a hit ratio for every cluster it participates in. Aggregation is triggered when $H < T_1$ where T_1 is a user defined threshold.

The filtering coefficient F of a cluster is defined as Q_c/Q_{root} where Q_c= # queries received for that cluster and Q_{root}= # queries received by the root on the main channel. F represents the amount of filtering done by a cluster hierarchy for a given gateway. Splitting is triggered when $H > T_1$ and $F > T_2$, where T_1 and T_2 are user defined thresholds.

H is a measure of how much irrelevant work a gateway is doing for a particular cluster. A gateway is motivated to reduce the irrelevant workload. F is a measure of the benefit of being in a cluster. Gateways receive the most benefit from a cluster if it filters out many irrelevant queries. Under this scheme, if there are many relevant queries on the main channel, the gateway prefers to be on the main channel. If the opposite is true, the gateway prefers to join a cluster hierarchy to filter out irrelevant queries.

The aggregation operation is almost the same as the growth operation. To form an aggregate cluster, the root of the cluster monitors its hit ratio for that cluster. If the hit ratio is too low, the root creates an additional metadata tag that is a generalized version of the original cluster metadata tag. It then follows the same steps to join a cluster as in the growth operation only it uses the generalized metadata tag.

The split operation is similar. A gateway monitors its filtering coefficient for a given cluster. If the filtering coefficient is above a threshold, then the gateway can break out of that cluster and return to the main channel.

4. ANALYSIS OF VIA

Aggregation provides us with potentially a b^h savings over flooding where b is the average number of links per gateway and h is the height of the hierarchy. If the number of gateways in a cluster is N, cluster hierarchies have potentially an Nb^h reduction in message overhead on the main channel compared to flooding schemes. Clearly, this is not the realized savings since there is an implicit tension between maximizing the Nb^h factor and minimizing the overhead of maintaining VIA clusters. The content of the data managed by clusters and the content of the queries have a tremendous effect on the performance of the protocol. In this section we present a "back of the envelope" overhead analysis of VIA.

4.1 Overhead Analysis

We first characterize the overhead as a function of these parameters. Define:

Q_r = queries/second
R = # roots subscribed to multicast
U_r = updates/second
S_r = heartbeats/second
N = average # gateways in cluster
C = average # clusters
K = edge message overhead due to joins
Ω = # gateways in the network
ω = # gateways sharing data
α = # query selectivity coefficient ($0 \le \alpha \le 1$)
β = # cluster selectivity coefficient ($0 \le \beta \le 1$)
b^h = aggregation savings (data schema dependant)

The selectivity coefficients α, β summarize the effect of different queries and cluster hierarchies. The more general a query the more likely it is to intersect with the data managed by the clusters. Along the same lines, the more general the cluster types, the more likely they are to intersect with queries asking for data. Both these factors are dependent on the actual data captured by the clusters as well as the actual requests from clients. We say that more general queries and cluster types generate more overhead.

The overall overhead for VIA is the sum of the multicast (flooding on the main channel) traffic and unicast (cluster hierarchy) traffic. Multicast traffic is generated by queries and JOIN messages. We are mainly concerned with queries since this has a larger impact on the network resources consumed by VIA. Gateways send JOIN messages when the cluster topology needs to be changed. We assume that every "update" requires a change in the cluster topology and that unicast traffic is dominated by heartbeat messages. In this case, the overall overhead V is:

$$V = Q_r R + U_r(R + K) + S_r NC + Q_r \beta NC + Q_r \alpha NC \quad (1)$$

The first term $Q_r R$ is the multicast traffic generated by queries. The second term, U_r (R+K), is the multicast traffic generated by updates. The third term $S_r NC$ is the unicast traffic generated by REFRESH messages. The fourth and fifth terms represent the overhead of responding to queries based on the selectivity of the query and the application data schema.We can compare this to a flooding approach to finding data. In this case, the total overhead is just the multicast overhead to broadcast a query and the cost of some gateways responding to the query

$$M = Q_r \omega + Q_r \beta NC \quad (2)$$

We now have a description of the overhead of VIA in terms of the query workload, update workload, and the percentage of gateways that participate some cluster. If we assume $R << NC << \Omega$ then VIA offers advantages over flooding as the query rate increases. Clearly, if R/Ω, U_r/Ω and NC/Ω is small then $V < M$. This makes sense intuitively. If a client needs to find data that is only managed by a small number of gateways compared to the total number of gateways then it is better to have VIA. If the client needs data that is managed by a large number of gateways compared to the total number of gateways then flooding can be more efficient since it incurs no protocol overhead for updates and REFRESH messages.

From the equations above, we can find the benefit *M-V* of VIA over flooding on the main channel (multicast). Ideally, the soft-state rate should be set equal to the update rate in order to provide minimum response time for cluster topology changes. In the case where $S_r = U_r$ we have:

$$M - V = Q_r(R(Nb^h - 1) - \alpha NC) - U_r(R + K + NC) \quad (3)$$

The intent is to maximize *M-V*. The multiplicand of Q_r is the "win" we get for filtering at the multicast level. The multiplicand of U_r is the "loss" we incur for having to maintain the clusters. Note that the selectivity of the queries is not a factor here since the number of gateways that have to respond to a given query is the same in either the multicast scheme or VIA. From the equation we see that the VIA protocol overhead can be dominated by the REFRESH message rate. We must chose S_r judiciously to achieve a reasonable balance between detecting topology changes and the cost of detecting these changes.

4.2 Towards Optimal Hierarchies

It is unlikely that VIA, which relies on a greedy generalization function and imprecise measures of network distance, will form an optimal hierarchy. However, we are able to characterize the features of an optimal topology. For example, flooding is an upper bound on cost since we should not do any worse than that. An ideal lower bound on the cost for a given query is the number of gateways that have a non-null response to the query since we must at least send a query to those gateways.

If we assume that the cost of maintaining a VIA hierarchy is minimal (e.g. we set the refresh rate to be very low), then VIA mainly needs to optimize on the cost of processing queries. Consider the case when all gateways are on the main channel. The cost associated with processing a single query is related to the number of gateways that receive it. For flooding, all gateways receive the query, which represents the maximum cost. VIA reduces the overall cost of processing a query by partitioning gateways into clusters. Since the root of a cluster can filter out irrelevant queries, the overall query processing cost is less. To find the minimal cost, we can search over all possible partitions of the gateways and measure the cost for a given query workload. Clearly, the query workload and the data on the gateways are strong determinants of the actual cost.

The total number of possible partitions of gateways can be very large. Searching this space is prohibitive and not possible in practice; gateways generally do not have knowledge about other gateways on the network. To approach an "optimal" configuration, VIA relies on the tuning of the heuristics used by the aggregation operations so that gateways will cluster until it is no longer advan-

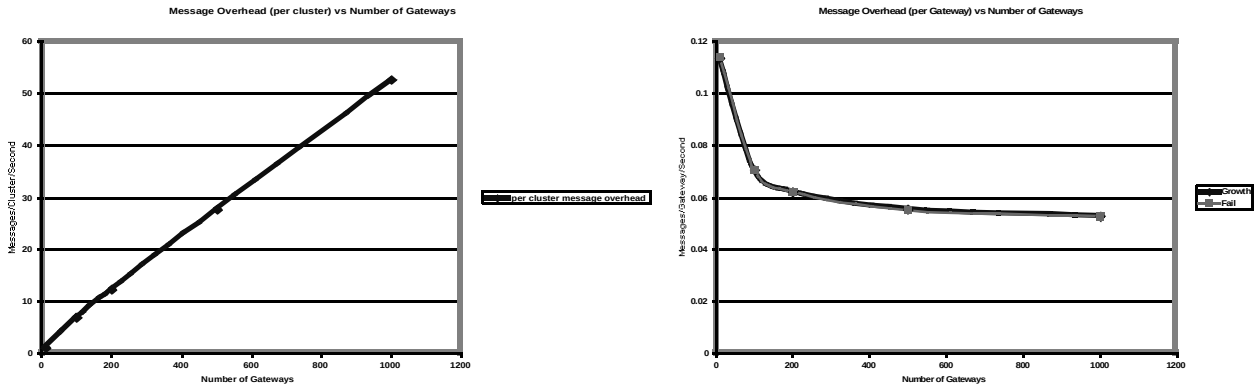

Figure 7: a) Message overhead per cluster vs. number of member gateways for the growth operation. b) Message overhead per gateway vs. number of gateways for the growth and fail operations. The decrease in the overhead is due to averaging of the overhead for idle nodes and active nodes. In general, as the cluster size increases, the number of idle nodes grows faster than the number of active nodes.

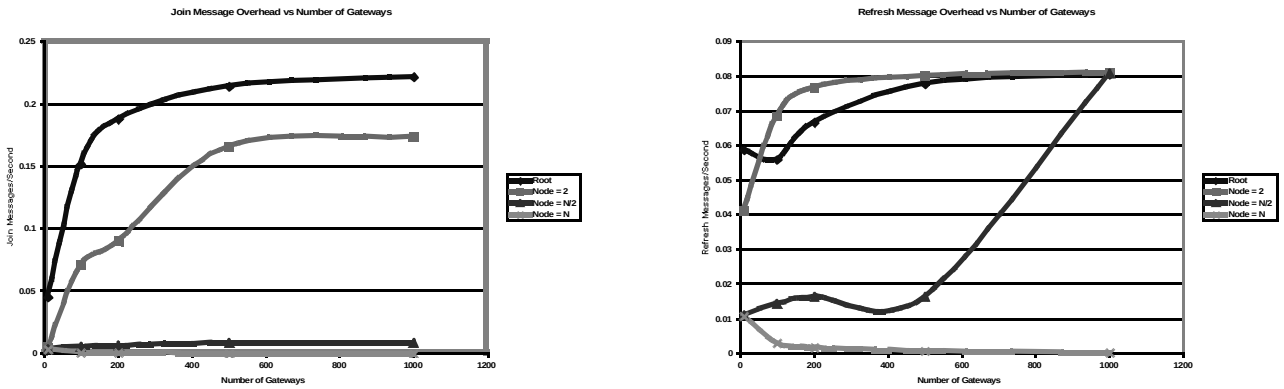

Figure 8: a) Join message overhead vs. number of gateways. b) Refresh message overhead vs. number of gateways.

tageous. Gateways make decisions based on local state information that potentially result in a good "global" behavior for the system.

5. MEASUREMENTS

In this section we describe our simulation experiments and measurements of our prototype implementation. The simulation is used to explore scalability issues and other scenarios that are difficult to control in the prototype.

5.1 PARSEC Simulation

We simulated the growth and fail operations in PARSEC. We chose these operations since other operations are based on the growth and fail operations. In our simulations we grew a single logical cluster of sizes 10, 100, 200, 500, and 1000 gateways. Gateways joined the cluster sequentially every 8 seconds until the cluster membership was complete. We set the REFRESH message rate to be constant at once per minute and gateways detected failures if they did not receive a REFRESH message after two minutes. We did not simulate query or update workloads since we are currently interested in a baseline overhead for the VIA protocol. However, we did perform workload experiments with our prototype gateway testbed.

We simulated the fail operation for fully grown clusters. In our experiments up to 3 gateways could fail simultaneously or in sequence. We measured the message overhead for failures and also verified that the partitions caused by failures would eventually rejoin to create a single cluster.

In this simulation, we "grew" clusters of size 10, 100, 200, 500, and 1000. It should be noted that the run times were greater for larger clusters. As will be shown later, one of the major culprits in increased message traffic are the soft-state REFRESH messages sent between gateways.

Figure 7a shows the message overhead per cluster. The message overhead is the bandwidth consumed by a cluster. We can see that the overall bandwidth consumed by the cluster is linear with the size of the cluster. It should be noted that the bandwidth consumption is somewhat amortized over the network and that the overhead is the maximum overhead any one link might see. As can be seen from Figure 7b, the overhead of each gateway is relatively independent of the size of the cluster. We also simulated a failure in a gateway close to the root after the entire cluster grew. Failure recovery does not generate much additional traffic and the overhead is roughly equivalent to growth with no failure.

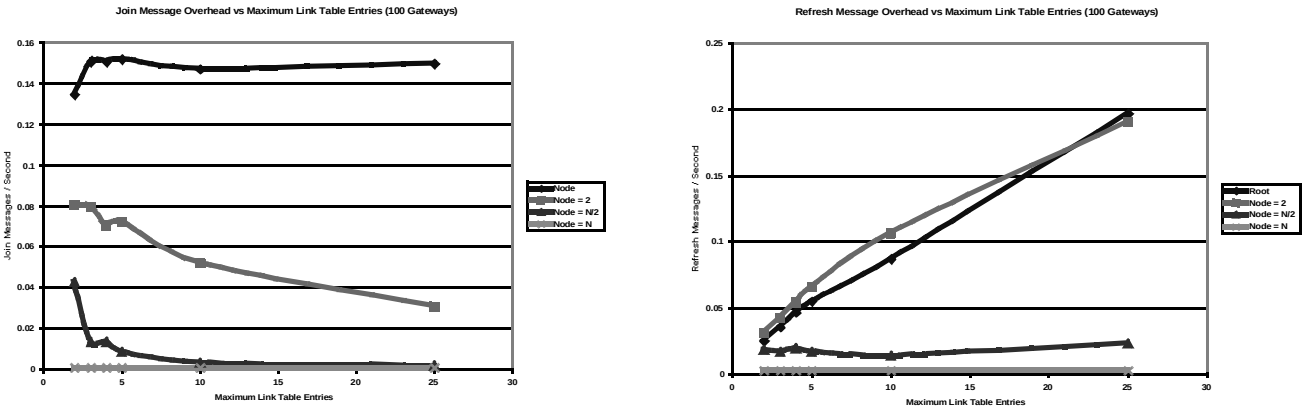

Figure 9: a) Join message overhead vs. maximum number of link table entries for a 100 gateway cluster. b) Refresh message overhead vs. maximum number of link table entries for a 100 gateway cluster.

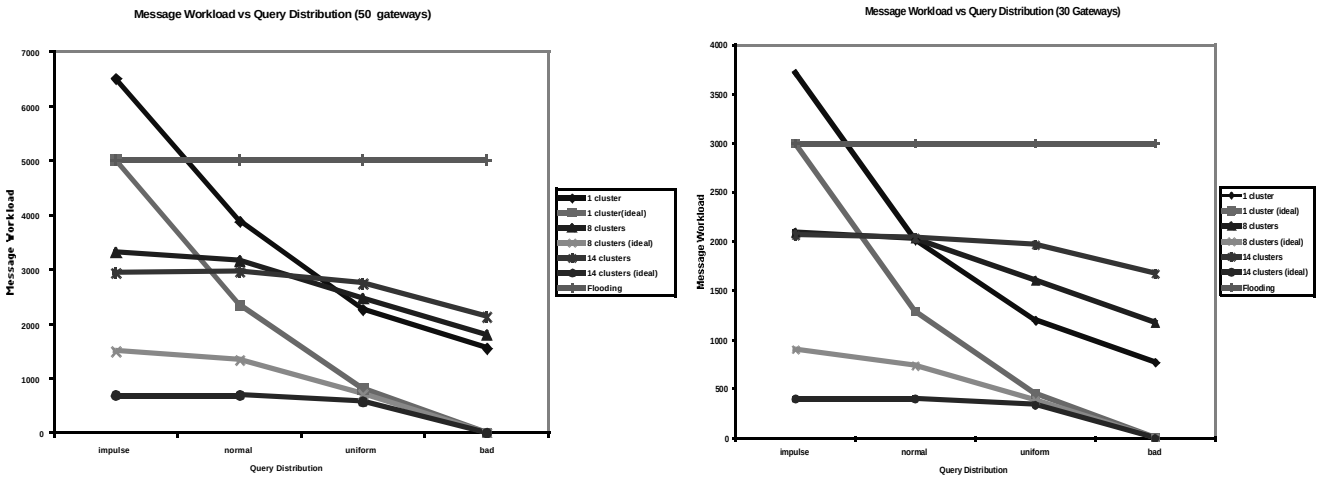

Figure 10: Message Workload vs. Query Distribution for a) 50 gateways. b) 30 gateways.

The graph in Figure 7b shows a counter-intuitive result in that overhead seems to decrease as the size of the network increases. This occurs because the overhead measure is an average measure taken from idle gateways and active gateways. In general, only a few gateways will be active in VIA at any one time, and the most active gateways are closer to the root. Most other gateways are only responsible for sending out REFRESH messages. As the cluster size grows, the ratio of idle gateways to active gateways grows and this causes the average overhead to decrease.

To illustrate the heterogeneous activity of gateways at different distances from the root, we measured the message traffic for the root gateway, a peer gateway, a "middle gateway," and a leaf gateway. We selected the peer gateway to be the third gateway that joined the collective. The middle gateway is the N/2 gateway to join where N is the eventual cluster size. The leaf gateway is the last gateway to join.

Figure 8a and Figure 8b are measures of the JOIN and REFRESH message overhead for gateways at different distances from the root. In these cases the message overhead approaches a constant dependant on the number of links a gateway has. When a gateway has allocated all its link table entries for a cluster it just forwards JOIN requests to its peers. REFRESH messages are also dependant on the number of links a gateway has set up. This is a bit more erratic as you increase the distance from the root but approaches a constant.

Clearly, there is a time component also; the later you join the cluster, the less messages you will send out. However, over a long period of time, the number of messages a gateway must send is more dependant on the number of links it has with other gateways. We simulated a 100 gateway cluster and measured the message overhead as a function of the maximum number of links a gateway allocates in its link table per cluster. For our runs each gateway was set to accept 2,3, 4, 5, 10, and 25 entries per cluster in their link tables. As in the previous simulations we measured the overhead for nodes at different distances from the root. Figure 9a and Figure 9b show the results for JOIN and REFRESH messages, respectively. From the figures, it is clear that as the probability of having an open link decreases JOIN and REFRESH messages approach a constant.

In all cases nodes closer to the root have a higher overhead then nodes farther away. The total overhead approaches a constant dependent on the number of links a gateway allocates to participate in a cluster. If gateways in a cluster allocate a high number of link entries per cluster, then the overall number of messages generated by the cluster decreases.

5.2 CLUSTER TESTBED

One way to measure the performance of a cluster hierarchy is the level of *stress* on each gateway and link in the hierarchy for a given workload. If we assume that the protocol has low overhead then the workload is driven mainly by queries. The stress on links arises from the number of queries that are distributed through the hierarchy and the network distance these queries must travel. The stress on a gateway arises from the number of queries it must process. Ideally, the cluster hierarchy is a minimal spanning tree and distributes a minimal number of irrelevant queries down each link.

There are three distributions that affect the overall performance of a cluster hierarchy and form the basis of a general workload model:

- Query distribution - workload is driven by the queries. If many gateways have a non-null response to a query, then the cluster hierarchy will do more work. If most gateways have null responses, then the cluster hierarchy should do less work. In VIA, we assume that the query distribution is known, and the topology of the cluster hierarchy is defined by an application data schema.

- Data content distribution - related to query distribution is the distribution of the content of the data on the network. Cluster hierarchies form based on similarity measures defined by the query distribution. Most of the cost reduction is achieved by "good" partitions of data. A cluster hierarchy (and an index in

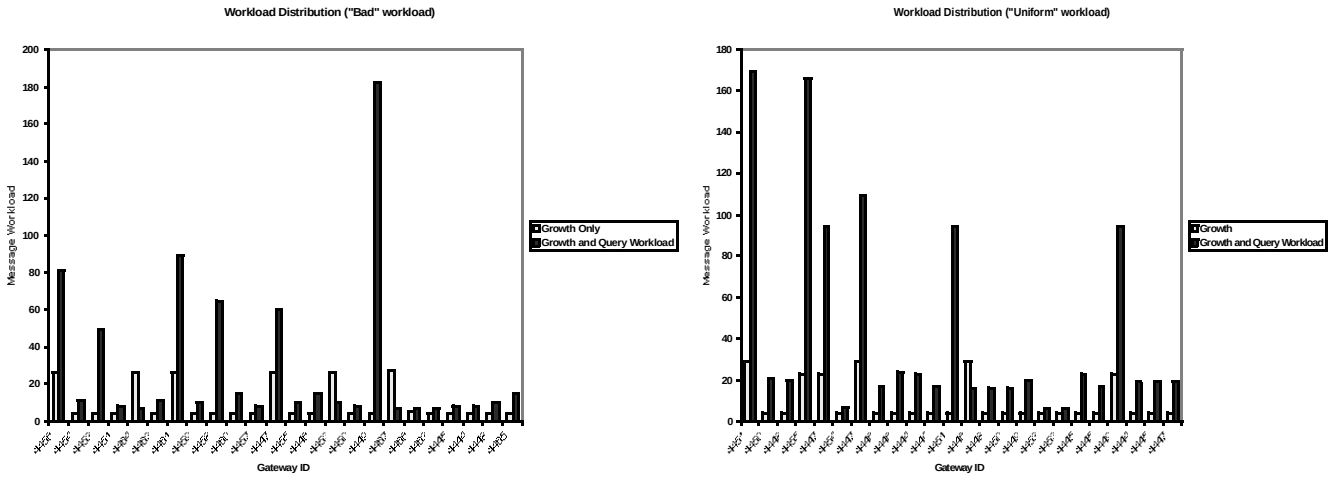

Figure 11: Workload distribution for 24 gateways, 6 cluster system under a) "bad" workload b) "uniform" workload

general) performs best when these partitions exist since we can easily filter out irrelevant queries. If these partitions do not exist then the cluster hierarchy will be less effective.

- Data object distribution - clearly, the topology of the spanning tree is dependent on how data is physically distributed on gateways in the network. Gateways that form a cluster and are "close" to each other will induce less stress on the network than gateways that form a cluster and are "far" from each other. Data distribution has a direct effect on the network distances of the links in the cluster hierarchy.

We expect VIA to form good hierarchies when the data object distribution, query distribution, and data content distribution are skewed to create obvious partitions. For example, an ideal case is when similar data is clustered close together on the network and queries are directed for specific clusters. As these skews become more uniform, the cluster hierarchy performs worse because, like all indices, it is optimized for specialized situations rather than general ones.

We have implemented prototype VIA gateways to run preliminary workload tests. We intend to employ VIA as a component of our on-going research in pervasive computing infrastructures. In our infrastructure work, sensor data is collected and deposited on distributed data repositories that exist in different discovery domains. The goal is to locate the data of interest by querying the repositories running VIA. To generate sample distributions, we have created a *data item generator* that seeds gateways with metadata tags and a *workload generator* that issues queries to gateways under different content distribution assumptions.

We have run experiments to investigate the performance of VIA for representative query distributions and three different cluster hierarchies representing increasing levels of data fragmentation in the data content distribution. We do not consider data object distribution in these experiments. In our experiments, we seeded gateways using our data item generator with sensor metadata tags and grew different numbers of clusters with 30 and 50 gateways. We then used our workload generator to create query workloads under four different distributions assumptions. In the "impulse" distribution, all queries were directed at a single cluster. In the "normal" distribution, all queries were directed at three different clusters under Gaussian assumptions. We also performed experiments for "uniform" distribution of queries where queries were directed uniformly against all clusters, and a "bad" (also called "negative") distribution where the queries were not directed at any cluster. Each of these distributions: "impulse," "normal," "uniform," and "bad" should generate decreasing amounts of work for the system since the queries become less relevant. Figure 10a and Figure 10b show the results of our experiments for different levels of data fragmentation. In our experiments we measured the estimated workload of each gateway by counting the number of incoming messages it had to process. The sum of all these counts is the system workload. We compare the experimental cases with "ideal" cases. In the ideal case, only gateways that have a non-null response to a query receive the query. We also compared the experimental cases with flooding.

As can be seen in the ideal cases, the overall workload of the system should decrease as the workload distribution approaches the "bad" distribution. VIA curves match this decrease in the ideal curves closely though they are "shifted" up due to the overhead of VIA. Flooding performs poorly since the workload is constant even as the queries become completely irrelevant. In Figure 10a, VIA performed worse than flooding for the impulse distribution for a single large cluster in our simulation because we grew the cluster first and then simulated the workload. In a real network, this overhead will not be as severe since cluster growth and query processing will occur simultaneously. Overall, VIA performs much better in all other cases. As presented in our analysis section, at its worst, VIA approaches flooding but performs much better overall.

Work is distributed unevenly in a cluster hierarchy. Figure 11a and Figure 11b show the distribution of workload for 24 gateways aggregated into 6 different clusters under "bad" and "uniform" workloads. Both figures show the message workload before the query workload (growth only) and the message workload after the query workload (growth and query workload). Under the "bad" workload, all clusters eventually aggregated into a 3 level hierarchy and one root did most of the work. Under the uniform distribu-

tion, gateways did not aggregate completely and more work was done by the roots of each cluster. It should be noted that most gateways are well below the amount of work they would have done in flooding.

A general workload model for VIA is very complex and we did not consider the complete set of parameters in our experiments. We are currently implementing a more complete workload model and simulation as part of our research.

6. RELATED WORK

VIA provides a flexible means to locate application specific data across service discovery domains. Our primary mission was to develop a robust, scalable, and simple means to share data among multiple discovery domains connected by a network. VIA is data centric. VIA constructs clusters in a bottom up fashion and uses clear semantics for self-organizing peer-to-peer overlay networks and hierarchies of these networks. These hierarchical clusters can adapt their configuration based on query demand and network resources.

Past extensions to service discovery architectures have attempted to solve the problem of sharing service information. For example, the WASRV extensions to SLP use advertising agents to multicast local service information to the wide area. This information is filtered by broker agents who in turn share this information with their local discovery domain. Jini lookup services can be chained together so clients can "crawl" from one lookup service to another lookup service to find information [7]. Additionally, lookup services can share service information by setting up tunnels between them. These tunnels can act like the links used in VIA but each tunnel must be configured manually.

INS and SDS are more general architectures for sharing service information beyond the local domain [2][6]. The basic scheme of INS is to replicate a "name tree" among a spanning tree of Intentional Name Resolvers (INRs) which manage their own local service information. Both VIA and INS spanning trees are maintained with a soft state mechanism. VIA clusters rely on global multicast to bootstrap the cluster creation process while INS relies on a well-known entity called the Domain Service Resolver that has information about all INRs on the network. In INS, the name tree acts as a global index for services in each local discovery domain. The INR spanning tree is similar to a cluster except that the index is replicated and shared by all nodes. Propagating name tree updates through the network and processing the name tree can be difficult. Lilley extends INS with the concept of "virtual spaces" to reduce the work any one INR must perform to manage the name tree [14]. VIA spanning trees differ from INR trees in that they directly correspond to the contents of the gateways. There is no notion of a globally replicated and shared index. While this potentially leads to higher latency when querying the network, it avoids the need to propagate update information to each node. Adjie-Winoto develops a distributed algorithm so the INR trees will eventually converge to a minimum spanning tree [1]. We do not attempt to create a minimum spanning tree in VIA but rather rely on randomization to create adequate topologies.

SDS is part of the UC Berkeley NINJA project [6][10]. SDS uses a hierarchical arrangement of special SDS servers to distribute service information and includes facilities for security and privacy. At the "bottom" of the hierarchy are SLP discovery domains. On top of these discovery domains, multiple hierarchies of SDS servers can exist. In the current implementation, SDS relies on administrative policy for both specifying which hierarchies should exist and also how SDS servers should be configured within the hierarchies. This differs from VIA, which uses a data centric methodology to automatically create clusters and hierarchies.

Distributing information over the wide area is not a new problem. The Domain Name System (DNS) is an example of a highly successful global data dissemination scheme [16]. However DNS is not directly applicable to wide area data sharing since it achieves its scalability from the constrained semantics of the IP hierarchical addressing scheme. Other schemes for web caching are more about data replication and movement to reduce data retrieval latency. For example, Michel, *et al* proposes a self-organizing scheme of adaptive web caches that replicate and migrate data closer to the requesters of that information [15]. However, it is assumed that the client already knows where the source of the data is.

Chandra, *et al* developed the Gryphon publish/subscribe system as a communications medium for applications in a distributed environment [4]. Clients can subscribe to messages based on content, and these messages are delivered using cooperating "brokers" that share routing information about subscriptions. Gryphon focuses on the timely delivery of data and the cost of computation at each node; a client expresses an interest in data (by content) and Gryphon provides a means to disseminate this data. VIA sets up cluster hierarchies before clients express interest. In some sense, VIA is a dual of Gryphon.

Researchers in *ad hoc* sensor networks have looked at *ad hoc* routing as a means to move data from sources to sinks. Like web caching, these applications are more concerned about data movement. Often, some type of controlled flooding constrained by the range of on-board wireless communication is used. *Ad hoc* routing approaches are flexible and self-organizing but operate under much more severe conditions than what we assume. For example, Intanagonwiwat, *et al* proposes a "data diffusion" approach where information sinks can diffuse their interests in a wireless sensor network [12]. These interests create "gradients" that information sources can use to forward their information to the sinks. It is commonly assumed that nodes in the sensor network can only see their neighbors, and that power and processing resources are severely constrained. In VIA, we assume Internet-like connectivity between gateways.

Sturtevant, *et al* propose a novel self-organizing Information Discovery Graph (IDG) to locate multimedia information on the wide area [20]. Using a set of topic managers, multicast groups are set up on the fly to manage specific types of information. The multicast groups arrange themselves in a hierarchical fashion. The semantics of the hierarchy are determined by administrative policy. IDG uses a top down approach to create a self-organizing topic hierarchy. VIA uses a data centric bottom up approach to create

self-organizing application data hierarchies. The two approaches are complementary since IDG can "pick up" where VIA leaves off.

Application level overlay networks are of growing interest in the Internet community [5][8][18]. Gnutella is a peer-to-peer file sharing network of GnutellaNet clients [9]. A GnutellaNet client can act as both a client and a server. To bootstrap into the Gnutella network, a GnutellaNet must first know the IP address of another GnutellaNet client already on the Gnutella network. When a client queries the network (using keywords) for shared files, this query is broadcast to all other GnutellaNet clients in the querying servers "horizon." A horizon is used to scope the query to only a portion of the global Gnutella network to limit the amount of traffic. However, since all queries must be broadcast the Gnutella scheme can consume a lot of bandwidth. VIA limits the need to broadcast queries through the use of forwarding links and self-organizing hierarchies.

7. CONCLUSIONS

We have proposed VIA as a means to locate application specific data across service discovery domains. Under certain assumptions about the cellular structure of discovery domains connected by a wider area network VIA is a robust, scalable, and simple means to locate application specific data. We do not try to solve the problem of locating general data since it may not be necessary. We also do not claim that VIA can be directly applied to the wide area service discovery problem. It is our goal to support specific applications that require some amount of autonomy within a discovery domain but need a global context to locate and access data. One compelling application is a global sensor system such as a smart highway. In such a scheme, local authorities can set up heterogeneous sensor infrastructures. VIA can be used to tie together the data collecting activities of the local sensor infrastructures so that global queries can be issued. For example, we could query the system to find all the traffic accidents in Los Angeles.

VIA has both multicast overhead and inter-gateway communications overhead. There is a strong relationship between these two overheads. For example, if each gateway was the root and only member of its own cluster, then VIA is just a global multicast scheme. If each gateway is part of a single super collective, then VIA is simply a peer-to-peer network overlay scheme. The question is then, what is the proper balance between multicast and inter-gateway communications overhead? Clearly, this is dependent on the topology of the clusters as well as the query demand for certain types of data. If there is a high query demand for a certain type of information then it may be more efficient to rely on a specific cluster to manage this information. If the query demand for the information is low, it may be more efficient to use global multicast since the overhead in maintaining the cluster for a certain period of time exceeds a single broadcast to all gateways. We intend to look at these issues more closely.

In our current implementation, we assume that queries are simply multicast and individual gateways respond to the client. This scheme does not scale to large size clusters since the client can easily be flooded by too many responses. We are investigating a method in which the probabilistic token consumption scheme used in the growth operation can be extended to control the flow of responses to the client. For example, the client can incrementally request information from the cluster based on the distance of the gateway from the cluster root. This mechanism would require each gateway to maintain the state of the query for an individual client.

As in any wide area scheme, scalability is an important concern. In this paper we assume that individual metadata tags can be treated independently from others. For a large number of metadata tags this could result in a high density of links at a single gateway. To address this issue, we could aggregate metadata tags to reduce the number of links. However, there is an implicit tension between aggregating metadata tags to improve the physical topology of the cluster (i.e. minimize the communication links between gateways) and having the flexibility to form an efficient filtering hierarchy. We plan to investigate the optimal partitioning of metadata tags as well as lossy aggregation methods to address scalability in VIA.

8. ACKNOWLEDGMENTS

The authors would like to thank Murali Mani and Akash Nanavati for useful discussions regarding filtering trees. We would also like to thank Ted Kremenek for listening to our arguments. Finally, we thank the anonymous reviewers for helpful comments regarding this paper.

9. REFERENCES

[1] W. Adjie-Winoto. A Self-configuring resolver architecture for resource discovery and routing in device networks. Master's thesis, Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May, 2000.

[2] W Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. *Operating Systems Review*, 35(5):186-201, December, 1999.

[3] C. Bisdikian, A. Edlund, and B. Miller. *Bluetooth Revealed: the insider's guide to an open specification for global wireless communications*. Prentice Hall, 2000.

[4] T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, D. Sturman, and G. Banavar. An efficient multicast protocol for content-based publish subscribe systems. *International Conference on Distributed Computing*, 1999.

[5] Y. Chu, S. Rao, H. Zhang. A Case for End System Multicast. *Performance Evaluation Review*, 28(1):1-12, ACM SIGMETRICS 2000, June 2000.

[6] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An architecture for a secure service discovery service. *Proceedings of the fifth Annual ACM/IEEE International Conference on mobile computing and networking*:24-35, 1999.

[7] W. Keith Edwards. *Core Jini*. Prentice Hall, 1999.

[8] P. Francis. Yoid: extending the multicast internet architecture. http://www.aciri.org/yoid/, September 1999.

[9] Gnutella. http://gnutella.wego.com.

[10] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, Z.M. Mao, S. Ross, and B. Zhao. The Ninja architecture for robust internet-scale systems and services. *Special Issue of Computer Networks on Pervasive Computing, to appear.*

[11] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608.

[12] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A scalable and robust communications paradigm for sensor networks. *Computer Networks and ISDN Systems*, August, 2000.

[13] JINI(tm) Connection Technology. http://www.sun.com/jini.

[14] J. Lilley. *Scalability in an Intentional Naming System*. Master's thesis, Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May, 2000.

[15] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web Caching: Towards a new global caching architecture. *Computer Networks and ISDN Systems*, 30:2169–2177, November, 1998.

[16] P.V. Mockapetris and K. Dunlap. Development of the Domain Name System. *Proceedings of SIGCOMM'88*, August, 1988.

[17] UCLA Parallel Computing Laboratory and R. Meyer. Parsec User's Manual 1.0. February, 1998. http://may.cs.ucla.edu/projects/parsec.

[18] S. Ratnasamy, Y. Chawathe, and S. McCanne. A Delivery based Model for Multicast Protocols using Scattercast. Unpublished, UC Berkeley, July 1999.

[19] J. Rosenberg, E. Gutman, R. Moats, and H. Schulzrinne. WASRV Architectural Principles. Internet Draft, Internet Engineering Task Force, Feb. 1998. Work in progress.

[20] N. Sturtevant, N. Tang, and L. Zhang. The Information Discovery Graph: Towards a Scalable Multimedia Resource Directory. *Proceedings of the IEEE workshop on internet applications (WIAPP)'99*, July, 1999.

[21] The Salutation consortium. http://www.salutation.org.

[22] Universal plug and play. http://www.upnp.org.