DataSpace - querying and monitoring deeply networked collections in physical space Part I - Concepts and Architecture

Tomasz Imieliński, Samir Goel
Department of Computer Science,
Rutgers University
{imielins@cs, gsamir@paul}.rutgers.edu

October 14, 1999

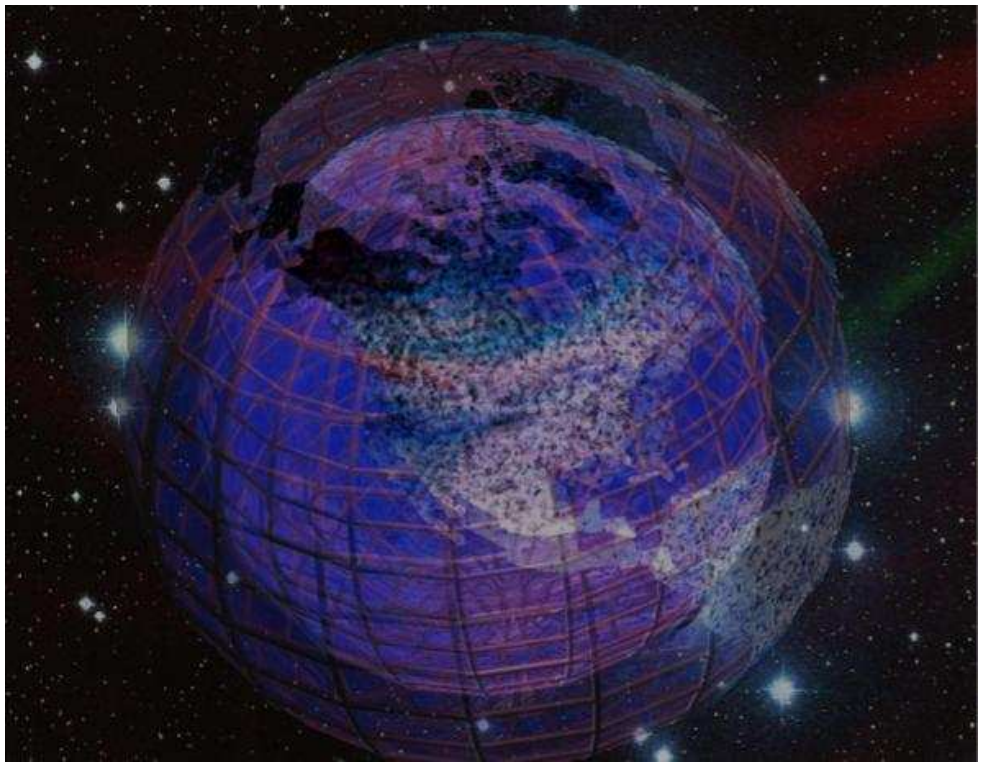

Figure 1: DataSpace is three dimensional physical space stretching from 10 kilometers below the surface of earth to 100 kilometers above the surface of earth, that is accessible to the network. DataSpace enables one to issue queries within every cubic millimeter block of physical space

Abstract

The DataSpace is a three dimensional physical space 100 kilometers above and 10 kilometers below the surface of earth that is accessible to the network. It is addressed geographically as opposed to the current "logical" addressing scheme of the Internet.

With the enormous 128 bit addressing space of IP version 6, one can individually address every cubic centimeter of the physical space in DataSpace with approximately 90 bits of area code. This would include every street, building, room, basement or even drawer of a desk. The DataSpace would thus serve as the host for the entire part of the physical world that is connected to the network. The billions of objects populating the DataSpace, each aware of its own geographic location[1] will form "dataflocks", mobile object classes which can be selectively queried, monitored and controlled. To support the DataSpace, we propose a version of the multicast protocol called "spacecast". Here, the network plays the role of a *Database machine*, handling queries through spacecast which "illuminate" selected datacubes and gather multiple responses from the objects that respond to the query. The user will interact with the DataSpace through spacecast in the same way that a person interacts with his physical environment through the information in reflected light waves.

1 Introduction

The physical space around us is illuminated with different types of radiation that selected sets of objects reflect: most of our environment reflects visible light, bones reflect X-ray radiation, tissues resonate in magnetic fields. Similarly, one can imagine the physical space illuminated by its response to a network request placed by a user. A request placed to the object may ask for retrieval of data or the completion of a task. Through such a network-mediated illumination of the physical world, one can experience and control billions of objects in one's immediate or remote environment. We call such a system the *DataSpace* and define it simply as space containing data. It can be a 3-D real physical space encompassing earth (figure 2) or it can be a 3-D real physical space encompassing our galaxy (figure 3). Of more relevance perhaps is the 3-D real physical space starting from 10 kilometers below the surface of earth and extending upto 100 kilometers above the surface of earth (figure 4). We use this definition of DataSpace in this report.

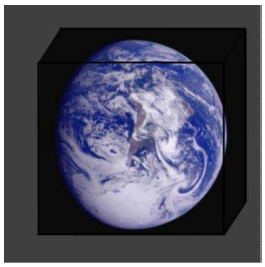

Figure 2: 3-D physical space encompassing planet earth

The DataSpace is populated by a massive number of objects that produce and locally store data about themselves. In the DataSpace, physical objects are no longer characterized just by shape, size, and color. They are also characterized by processor type, the amount of memory and the network connection.

[1] This process is already well under way with many cars equipped with the GPS antennas. We predict that location awareness will be as universal as time awareness is today on every computer or electronic device, in a very near future

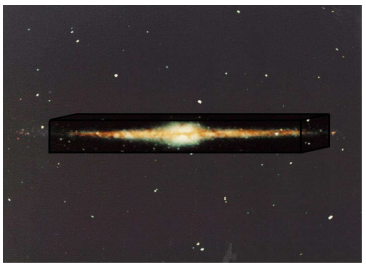

Figure 3: 3-D physical space encompassing our galaxy

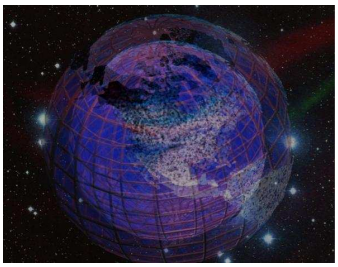

Figure 4: 3-D physical space stretching from 10 kilometers below the surface of earth to 100 kilometers above the surface of earth

The DataSpace is spatial by definition, built from three dimensional administrative or geometric datacubes. Administrative datacubes come in many sizes and can encapsulate a city, a street, a building, a basement, a room and even a drawer or a shelf; they can also include the interior of an engine (one of the cylinders for example) or a left hemisphere of someone's brain; geometric datacubes can range from a cubic mile around the World Trade Center to a cubic millimeter in the retina of an eye. Spatial coordinates are the basic points of reference to navigate and query the DataSpace, making the DataSpace very much like *real space* which we see and move through every day. However moving through the DataSpace, a user acquires an enhanced awareness of his surroundings as well as of even the most remote areas including information that would be previously invisible or out of his access. The DataSpace gives the user a sixth sense through which to perceive the world.

The objects in a DataSpace must be organized in a way that they can be usefully accessed and manipulated. This is achieved by grouping objects into classes called *dataflocks*. Dataflocks are often mobile class of objects that move through the physical world while still maintaining (some level of) connectivity to the network. Dataflocks are accessed through the datacubes they are located in and through their own class properties.

For new objects, becoming part of a datacube is a simple "plug and play" mechanism. Rather than requiring a complex prior protocol, this event involves little processing other than registering of a physical object in the scope of our vision.

Plug and Play is emphasized in JINI [13], which is the most serious attempt today to build "smart spaces" of devices such as home appliances, printers, xerox machines. These devices are connected to the local area network and can be digitally controlled. However, JINI is still oriented through the logical structure of the Internet where locality and distance are defined not by

geographic space but rather by the Internet's logical structure of subnets. In contrast, the DataSpace is spatial and embedded in the physical reality which all of us are used to. Here, the "local area network" is replaced by a room, a street, or the top of a mountain, depending on where the user is located. Also, the DataSpace is a global concept, not restricted only to the immediately surrounding space but allowing querying, monitoring and controlling of remote spaces as well.

Finally, the concept of a DataSpace has to be contrasted with that of a *database*. Databases store information locally about remote physical objects. In this scheme, the physical objects become merely the artifacts of their corresponding entry in a database. In the DataSpace, the data is stored locally in objects, and becomes a property when queried (or "illuminated").

The organization of the DataSpace is geographic as opposed to the current logical structure of the Internet. In this way, DataSpace is structured analogously to the real physical space which surrounds us. In this work we build on earlier work with Navas where we introduced GeoCast for geographic messaging [11] and have implemented it within DARPA's GLOMO program.

1.1 Basic features of DataSpace

The following are the basic functionalities of DataSpace :

- DataSpace is a collection of datacubes which are either administrative or geometric.
- DataSpace is populated by Dataflocks, classes of objects which can be static or mobile. Each object in a dataflock is connected to the network and has the ability to locally store data it collects or produces which can subsequently be queried or monitored remotely.
- DataFlocks can be queried, monitored or controlled from any location in the world. Queries and monitors are spatial and geographic in nature; *Space* is the main querying category in DataSpaces: all queries have to be spatially constrained.

1.2 Examples

Each floor of the World Trade Center in New York City is an example of an administrative datacube, with its administrative parent as the World Trade Center datacube. Example of dataflocks which can populate such datacubes are: tourists, employees of different companies that rent space in the World Trade Center, elevators, door locks, surveillance cameras, shock sensors, etc.

Figure 5: Large Datacube which includes Manhattan and a smaller one which includes several floors of the World Trade Center

Elevators form a good example of a mobile dataflock. Elevators may have the following properties: *range* (the floors that they service); and the *current floor* they are currently on. Notice that

the latter one reflects a rapid mobility of the Elevator dataflock, moving within minutes through different datacubes of the World Trade Center.

One kilometer high physical space which covers 10000 square kilometers covering Manhattan with surroundings and its immediate airspace is an example of a geographic datacube. Such datacube may be populated by dataflocks of helicopters with their current 3-dimensional position, flying direction, amount of fuel, etc. as properties; other dataflocks may have properties such as cellular channels and their current status (availability etc), direction and strength of wind, temperature as measured by various sensors, etc.

1.3 Querying and Monitoring

Querying and monitoring of object collections will be spatially driven. First, the users will identify the datacube that they want to query. Each datacube will have its DS (datacube directory service) which will list all dataflocks *registered* at that datacube. Only dataflocks registered in the given datacube can be subject of queries in that datacube. In general, very local dataflocks (such as door-locks in a building) will only be registered in small datacubes immediately encompassing them. Large datacubes will contain, broad, general dataflocks; answering queries in larger datacubes will invariably involve aggregation. For example, a query about locations of all taxi cabs in Manhattan may be answered by first providing distribution of their total count over two dimensional spatial grid (say, blocks); the exact locations can be obtained eventually through the process of *query zooming* to the smaller datacubes.

We view querying by analogy to illuminating the physical space. First, a datacube is illuminated with a message carrying a query then only the objects which satisfy the query, "reflect" the light. In larger datacubes, instead of individual objects, one may have sub-datacubes responding with aggregates. This has a clear "visual" analogy — details are usually not visible from far distances; one can zoom in if details are required.

In the subsequent sections, we provide more detailed descriptions of DataSpace query, and DataSpace monitors, and describe the architecture of the entire system. We will start by justifying our basic decision to implement DataSpace largely on the network level using our proposed modification of multicasting, called spacecast.

Before we proceed, we enumerate the main challenges of DataSpace design and the key principles that guided our design of DataSpace protocols:

1.4 Challenges and Principles of DataSpace Design

The main challenges and the key principles of our DataSpace design are as follows:

Challenges

- *Scalability*: DataSpace consists of billions of datacubes, and encloses a very large number of objects. Consequently, huge number of queries are possible in DataSpace. Therefore, scalability becomes a major concern in the design of DataSpace.
- *Different scales of space*: The size of a datacube in DataSpace can range from being very large (e.g., one enclosing a city) to very small (e.g., one enclosing retina of someone's eye).
- *Different mobility characteristics*: The objects in DataSpace can have vastly different mobility characteristics — from immobile objects (a refrigerator, a desk) to mobile (a car, plane, human) objects.

Principles

- DataSpace is geographically (spatially) organized just like real, physical space is, rather than "logically" like today's Internet
- No plugging costs if there is no play — objects which are not queried should not have to incur any prior cost due to registration, having to send out messages, etc
- Only frequent queries should have infrastructure support, no need to invest in infrastructure on infrequent queries

We identify two quality of service parameters for characterizing any DataSpace design: *observability* and *awareness*. *Observability* characterizes the quality of answer to a query that one can expect to get in the system. For e.g., the answer may be "90% complete", or may reflect a state that is "10 mins. delayed", etc. *Awareness* characterizes the quality of monitoring (of the answer to a query) possible in the system. For e.g., the knowledge base of a monitoring object may be "10 mins. behind" the actual state of the system, or may be "90% accurate", etc.

2 Solutions — application layer versus network layer

There are several possible implementations of the DataSpace that one may consider. We will concentrate on two alternatives: the *application layer solution* and *network layer solution* through the use of multicast. Let us first consider the application layer solution. Such solutions already exist in the form of search engines today, on which distributed servers store information about objects. Queries are processed by the servers (say, the nearest one) and objects always inform servers about changes of their state — new objects send registration messages and objects that "go out of service" can send deletion notifications. Direct querying of objects is practically impossible — it would be equivalent to "web crawling" for each query; unless multicasting is implemented on the application layer using some form of tunneling. One such solution is Intentional Naming System [1]. As mentioned in [1], the major drawback of such schemes is they are simply not scalable to the potential scale of a DataSpace system.

2.1 Network as a DataSpace machine

Another alternative is to implement the basic functionalities of the DataSpace on the network level. First of all, it is very natural to implement broadcast to a specific datacube (what we call datacube illumination) via a multicast mechanism. This can be done using shared multicast trees [3, 4], where multicast group membership for a given datacube is calculated on the basis of the location of a given object[2]. Thus, the network can take care of making sure that a given message "illuminates" only specific datacubes. Moreover, the network can also support indexes on selected attributes of dataflocks through multicasting. More precisely, each pair q = (Attribute, value) for such indexed attribute has a corresponding multicast address and all objects which satisfy q belong to that multicast group. This is analogous to a list of objects associated with an index entry (through so called "inverted index") in databases.

[2]Later we will show how objects find out about their locations, either through their own GPS [14] or through the protocol called GeoARP

In general, querying will be implemented using the combination of network indexes together with the application level filtering; just like database querying typically combines using one index for the first approximation, followed with sequential checking of each record for satisfaction of other conditions of the query.

In general we believe that implementation of a number of primitives of DataSpaces on the network level has several advantages over the implementation of DataSpaces on the application layer:

- It is much more network efficient. Multicasting avoids opening multiple unicast connections which result in waste of bandwidth and uneconomical use of the network
- It delegates the basic primitive(s) to the lower level of the abstraction hierarchy — always a good idea to have lower level operations implemented this way. It allows the network to deal with the network specific issues such as disconnection, router failure etc; rather than having to re-implement what the network does, on the application layer.

  It is closer to "plug and play" philosophy of seamless network presence for devices and objects — implementing join/drop/response operations on the lower levels of the network hierarchy makes them easier to standardize and frees an application from unnecessary burden.
- It utilizes very large addressing space allocated through IPv6 — thus it takes advantage of the new infrastructure which will soon be widely deployed. It may actually be a "killer application" for it.
- It saves energy on the client's level by performing operations on the network layer with the CPU in the doze mode. Using multicast addresses to perform filtering on the network level is much more energy efficient than performing the same filtering on the application level.

3 Terminology

In this report, we frequently use the following terms:

- *Dataflock*: A group of objects in DataSpace having some common property.
- *Space handle*: It is an identifier for a datacube. It is obtained from the GPS co-ordinate of the datacube. These preserve the *prefix property*.
- *Network index*: An attribute/method for which every pair (method, value) has a corresponding multicast address and all objects which satisfy the predicate (method = value) are members of the corresponding multicast group.
- *Subject handle*: Pairs q = (method, value) such that the method is indexed are called *subject handles*.
- *Datacube Directory Service (DS)*: It stores the state information of a datacube.
- *Brokers*: Broker of a query caches the answer to the query. Objects interested in finding answer to a query q can request the answer directly from the broker of q.

- *Shared tree*: We assume a shared tree multicast routing protocol. Shared tree refers to the tree built by the routing protocol for delivering multicast packets for a particular multicast group.
- *Core router*: We assume a shared tree multicast routing protocol. Core router for a multicast group is the router on which its corresponding shared tree is rooted.
- *Multicast addresses in DataSpace*: There are two types of multicast addresses in DataSpace, each with its own unique prefix. They are:
  - *Passive* multicast address: a multicast address used for querying. These multicast addresses have the symbolic prefix PASSIVE-MCAST-ADDR.
  - *Active* multicast address: a multicast address used for monitoring. These multicast addresses have the symbolic prefix ACTIVE-MCAST-ADDR.
- *Representative datacube*: DataSpace is sparsely populated, not every datacube has an associated DS. Datacubes without DS are represented by the smallest datacube that encloses them and has a DS. So, for any datacube, its representative datacube is the smallest datacube that encloses it and has a DS.
- *DS-discovery*: Given any datacube, the mechanism of finding DS of the representative datacube is called DS-discovery mechanism.

In the next section we define the basic concepts of DataSpaces more formally.

4 DataSpace Architecture

DataSpace is the physical space which is connected to the network. In this report we assume three dimensional space[3].

DataSpace is a collection of datacubes which are either geometric or administrative. *Geometric cubes* are defined as three dimensional cubes ranging from tens of cubic kilometers to the smallest ones possibly even of the size of cubic centimeter (although the smallest datacubes which we propose will be of the order of cubic meters). *Administrative datacubes* include countries, states, towns, streets, buildings, rooms, regions of a body or machine internals. Each datacube has a corresponding *space handle* that encodes its location in 3-dimensional physical space and and its size.

Datacubes are populated by *dataflocks* — which are static or mobile classes of objects. Each object (sensor, machine, piece of software, physical object) is characterized by a set of methods. Dataflocks will be subject of querying or monitoring. Some of the dataflock methods will be supported by *network indexes*. By the network index, we mean an attribute/method for which every pair (method, value) has a corresponding multicast address and all objects which satisfy the predicate (method = value) are members of the corresponding multicast group. Pairs q = (method, value) such that the method is indexed are called *subject handles*. Network indexes are used in processing of the DataSpace queries in a similar way as database indexes are used in processing database queries.

[3]In principle, DataSpace can even be 4-dimensional with time as the fourth dimension, to support querying the object histories as well

Each datacube has its own local directory service (DS) which contains entries for the dataflocks that are registered in that datacube. Each dataflock can be registered in many datacubes. The DS will contain the network indexes for such dataflock. Mobile dataflocks will have to re-register in the new datacubes that they enter.

Each subject handle q, in addition to defining the group of objects which satisfy it, may also have another group of objects associated with it. That group, denoted by Interested(q), monitors all changes to the membership of q. In other words every time a new object joins q or an existing object drops out of q it sends out a message to the Interested(q) group. Members of the group Interested(q) can themselves be viewed as a dataflock registered either locally, in the same datacube as the members of q, or in another, possibly remote, datacube. Thus, there are in principle two multicast addresses associated with the subject handle, q: the *passive* address which allows reaching objects satisfying q and the *active address* which allows objects satisfying q to actively update their status for the group Interested(q).

The members of Interested(q) monitor updates to the membership of query q, and hence know the answer to query q. A few of these members may offer this knowledge base to anybody who is interested in finding the answer to query q. Such members are called *Brokers* of query q. They are similar in concept to a network cache. Thus, an object interested in finding the answer to query q, may request the Broker of q for the answer instead of directly querying the objects. Having brokers tends to be more efficient especially for queries whose answer do not change rapidly. For these queries, it is more resource efficient to cache the answer to the query and use it to service the request of the objects, instead of querying the objects directly.

4.1 Datacube Directory Service (DS)

DS for a given datacube will store four types of entries:

- Mapping between the descriptors q and the corresponding subject codes
- Mappings between the descriptors Interested(q) and the corresponding subject codes: these multicast addresses will be used to disseminate updates to q (additions and deletions) among the interested domestic clients *within* the same datacube.
- address of the Core router for the datacube. This acts as core router for all the multicast groups that are local to this datacube.
- address of the broker for the datacube. This acts as broker for all the queries that are local to this datacube.

Notice that subject code corresponding to Interested(q) will be combined with area codes corresponding to that of q, i.e., of the datacube where the monitored objects are present.

4.2 Addressing

IPv6 [6] has 128 bit long addresses with 112 bits available for multicast addressing space. There are enough bits in multicast address space to address all datacubes in the world down to the cubic centimeter for a space which is 100 kilometers high and 10 kilometers deep around the world. We won't need that level of precision and will concentrate here on addressing down to the cubic meters rather than cubic centimeters.

DataSpace addresses will occupy a part of the multicast addressing space of IPv6. DataSpace addresses will be built from two parts: *area codes* which encode datacubes and *subject codes* which denote individual descriptors.

4.2.1 DataSpace Addressing

The proposed design of the DataSpace addressing space should be treated as a proof of concept rather than cast in stone structure. Below, we show that with sixty four bits we can successfully encode all datacubes down to cubic meter for all physical space which is currently used by any living creatures.

Here is a "back of the envelope" calculation which shows how the 64 bits can be used to encode the area codes. Let our DataSpace be 100km deep space which extends 10km down the ocean and 90km up over 500 million square kilometers — this is 50 billion cubic kilometers (5×10^{19} cubic meters). To address every cubic meter in the DataSpace we need about 64 bits; to address every cubic centimeter, 83 bits, if we add administrative datacubes and higher level, larger datacubes, we may just simply double the total number of datacubes — increasing the required area code size by just one bit[4].

The remaining 48 (of 112 bit IPv6 multicast addressing space) bits will constitute the *subject code* of the DataSpace address. The subject codes will correspond to elementary descriptors. Multiple descriptors can be mapped to the same subject code in different area codes, just as the same phone number can be assigned to different people in different area codes. Subject codes will be used for q, and Interested(q) groups as well.

We feel that 48 bits provide more than sufficient addressing space for possible subject handles[5]. Note, that only a subset of methods for a given dataflock is network indexed.

In the next section we show that, in fact, we may need much fewer bits to encode the DataSpace.

Local Addressing Schemes

We predict that DataSpace queries will be characterized by much more locality than, for example, telephone calls. In other words a vast majority of DataSpace queries will actually query the dataflocks which are resident in the same datacube as the querying client. In a bit less restrictive scenario, the querying clients will be located "close" to the queried datacube, that is not in the same datacube but in a small distance from it. If such locality is present one could significantly reduce the number of bits necessary for the area code; since the same area code may be used to denote multiple datacubes.

This is analogous to a hypothetical situation in the telephony (which would not be very realistic in that setting) that callers would only call people who are located within the same area code or within the area code which is located close to the caller (for example in the neighboring state or, say, on the same coast). Then, for callers who are located in the same area code one may multiply the total number of available numbers by allowing the area code digits to be used for individual telephone numbers. Similarly, by restricting calls to the groups of 9 states one may only use one digit area code and utilize the rest of the digits for the number itself. This intuition when carried over to the DataSpaces can reduce the number of digits necessary for the area code depending on the degree of locality of the user querying.

[4] In any multilevel hierarchy, the total number of nodes in it is always less than twice the number of leaves

[5] 2^{48} is about 10^{15} — plenty of space to encode significant subject handles for even a vast number of dataflocks which populate a given datacube

With k bits reserved for the area codes one may "reuse" the area code in a similar manner as frequencies are reused in cellular architectures. Having k bits in the area code would allow local querying within 2^k datacubes.

In general, we believe that the majority of dataflocks will be very local, only a small fraction of the dataflocks will have a more global character.

We envision possible use of several addressing schemes, with a small number of bits in the address space (3-4) used to distinguish among such different addressing schemes.

5 DataSpace Protocol: Overview

The DataSpace protocol involves supports two operations: querying and monitoring.

5.1 Querying

A DataSpace query is an expression of the following form:

Q = Select
From <Dataflocks>
Where <Condition>
In <Datacube>

The DataSpace query returns the network identifiers (can be unicast addresses for example) of the dataflock objects which satisfy the query Q.

Condition in a query is a conjunction of elementary descriptors of the form (method OP value) where OP can be an arithmetic operator and Datacube is either a geometric or administrative datacube.

Two parameters that are critical in processing a DataSpace query are : *space handle* and *subject handle*. The subject handle is selected from the group of subject handles that not only appear in the condition of the query, but are also one of the network indexes for the queried <Dataflock>. Space handle is the area code corresponding to the <Datacube> mentioned in query Q.

Query Q is processed by first sending it as unicast message to the broker of Q in the <Datacube> specified in Q. If the broker is out of date then the query is sent as multicast message within the <Datacube>. This part of the querying protocol involves two stages *illuminating* a datacube, and *selective reflection*. Both these terms usually describe the behavior of light waves and are used here on purpose to further emphasize the analogy of DataSpace to the real space. DataSpace illumination involves multicast message directed at entire datacube. Such message is routed entirely on the basis of the area code ignoring the subject code of the multicast address. Selective reflection is implemented through the network layer filtering based on the subject code of the multicast address of the query with the application level filtering corresponding to the rest of the query.

Within a datacube, a query is processed in DataSpace in a way very similar to the way querying is performed in a database system. As an example, consider the processing of query for finding all the tube-lights in CoRE building that are currently not working:

Q_1 *= Select*
From <Electric-Fixtures>
Where <Type=TUBE-LIGHTS AND status=NOT-WORKING>
In <CoRE-Building>

Assuming that there is a database that maintains information about all the electrical fixtures of CoRE building and their status, and assuming that the database has an index on attribute "Type", the query will be processed by first using the index to select all the records that satisfy the condition: *Type=TUBE-LIGHTS*, and then sequentially checking the selected records for the remaining part of the condition: *status=NOT-WORKING*.

In DataSpace, all tube-light's fixtures within CoRE building will subscribe to the *passive* multicast address given by a combination of space handle corresponding to CoRE building and subject handle corresponding to Type=TUBE-LIGHTS (assuming that "Type" is one of the network indexes of dataflock "Electric-Fixtures"). The above query will be processed by sending a mulitcast message to the datacube enclosing CoRE building at the above multicast address. The remaining part (*status=NOT-WORKING*) of the *Conditions* in the query is placed in the body of the message. This multicast message will reach all the tube-lights in CoRE building. Each of them will respond with their identifier if they also satisfy the other conditions present in the body of the message (*status=NOT-WORKING*). Thus the first level of filtering is performed at the network layer while the second level of filtering is performed at the application layer. In effect, the multicast mechanism serves as a (network) index for the massively distributed information repository, i.e., the DataSpace.

5.2 Monitoring

In DataSpace an object may monitor a query of the form:
Q = Select
From <Dataflocks>
Where <Condition>
In <Datacube>
Monitors send updates to the extension of the query Q to the clients who are interested in receiving such reports.

A client that is interested in monitoring the query Q will have to join the multicast group corresponding to Interested(q) and space handle, S. Here, q is one of the subject handles mentioned in <Condition>, and the space handle, S, corresponds to that of <Datacube> specified in Q.

5.3 Comparison with database model

Figure 6 compares the DataSpace model with the traditional database model. In database model, a database contains entries about real physical objects. Here the physical objects become merely the artifacts of their corresponding entry in the database. Users of this database issue operations and receive the results. In contrast, in DataSpace model, data and the physical object is a single entity — data resides with its source. All entities (data source as well as clients) are connected to the network. Here the network serves as a database engine and allows users to issue queries and perform monitoring operations. The network takes care of routing the query to the objects that satisfy it, and routing the updates to the objects that are interested in it.

6 Observability and Awareness

We identify two quality of service parameters for characterizing any DataSpace design: *observability* and *awareness*.

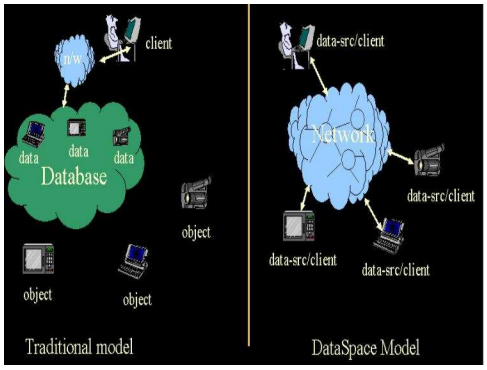

Figure 6: Comparison of DataSpace model with database model

The query q is R-observable at an object o, if o can be provided with the answer to q such that the answer satisfies restriction R. The restriction R specifies certain assurances on the quality of the answer. For example, in a DataSpace of sensors sensing temperature in all parts of a building, the query *"all sensors reading more than 200F"* is said to be 60second-observable if the querier receives response from *all* sensors satisfying this query within 60 seconds. This is analogous to the 5, 10, 15 minute quotes from the stock market.

An object o is R-aware of a query q if at any time it knows the current answer to q, where "current" is defined by restriction R. The restriction R specifies certain assurances on the quality of the answer known at o. For example, in a DataSpace of sensors sensing temperature in all parts of a building, the building administrator is said to be `Tunits`-aware of the query *"is there fire in any part of the building ?"*, if he knows within the T units of time *exactly* which portions of the building are on fire.

Thus, R-observability and R-awareness refer to different "approximated" views of the answer to a query and the ability to monitor its answer, respectively. R can be a time defined predicate, such as "within 10 minutes" or it can be based on the properties of the query answer (say 90% accurate). R-observability and R-awareness reflect the need to define approximate query answering and query monitoring in a volatile networked environments with varying reachability, disconnections, etc.

7 Multicasting and SpaceCasting

Multicast protocols available today are not a perfect fit for the DataSpace applications for the following reasons :

Multicasting today	Spacecasting
• small number of groups	• very large number of groups
• long messages	• short messages
• one way	• two-way, generates many responses
• groups are usually short term and dynamic	• most groups are long term

Spacecast protocol is used to route multicast messages in DataSpace. As we described earlier, a destination multicast address (spacecast address) is made up of two parts — an *area code* and a *subject code*. Each spacecast address defines a spacecast group. Membership of an object in spacecast group is therefore determined by location of this object and whether it satisfies the corresponding subject handle.

Location of the object is used to determine the area codes of all datacubes inside which the given object currently resides. Similarly, network indexed attributes of object's dataflock are used to provide all the subject handles associated with that object. Summarizing; a given object belongs to all spacecast groups formed by any combination of one of the object's area codes with one of the object's subject codes. Notice that spacasting can be viewed as extension of *paging* — we "page" specific datacubes and have the message filtered within that datacube on the basis of the subject code.

In the absence of a GPS device, an object finds its location using an algorithm similar to GeoARP [11]. It does this by broadcasting GeoARP message with a small TTL to find out the geographic location of the neighbors of that object.

The infrastructure which supports spacecast consists of "fat" shared multicast trees. Fat shared trees are described in detail in section 8.1.1.

Brokers provide solution to the problem of frequent aggregation of multiple query answers — reducing the number of times when dataflock objects have to be queried directly.

Spacecasting is truly "plug and play" since a new object only needs to join the appropriate spacecast groups based on the knowledge of its location and queries that the object satisfies, both of which can be obtained without requiring any prior manual configuration.

7.1 Networking Infrastructure

We envision the networking infrastructure which is like today's Internet — a combination of heterogeneous wired and wireless networks such as satellite, cellular, wireless LAN, infrared, broadband fiber-optics etc. Ultimately, the networking infrastructure for the DataSpace will have a multi-hop character with the satellite and fiber and backbone and a number of ad hoc smaller networks connecting smaller and smaller datacubes. We predict a proliferation of the GPS and other location providing devices which will make location awareness as widespread as time awareness is now[6]. The combination of ubiquitous network with the universality of location awareness among the devices which are connected to the network form the basic requirement for the DataSpace. This process is already well under way.

8 Challenges

In this section, we discuss the key technical challenges that need to be addressed before the proposed DataSpace architecture can be realized.

8.1 Huge number of multicast groups

In DataSpace, the potential number of possible queries runs in billions. Following our approach of network as a DataSpace engine, we map each of these queries to a spacecast (multicast) address.

[6] Just as each computer today is equipped with a clock, the computer of tomorrow will have a location providing utility

Thus, we have potentially billions of multicast groups in the system. Presence of such humongous number of multicast groups raises the following serious scaling issues:

- *Multicast forwarding table size*

  The first and foremost issue concerns the size of multicast forwarding table. With so many multicast groups, even with a shared tree scheme wherein a router stores just one entry per multicast group, it is very likely that some routers have unmanageably large multicast forwarding tables.

- *Overhead of maintaing the trees*

  The second issue relates to the cumulative overhead of maitaining shared trees for so many multicast groups. In particular, the cumulative overhead of the *Join* and *Prune* messages may become a significant fraction of the data traffic if it is not handled properly. This is very likely because of the presence of potentially billions of in the DataSpace.

- *Discovering Core-router for a multicast group*

  The third issue concerns the Core-router discovery mechanism. We need an efficient mechanism for discovering Core-router for any multicast group. Also, given the relatively limited memory available at the routers, the mechanism should not require the routers to store the mapping between queries and their corresponding core routers, for all possible queries. Core-router discovery mechanisms that exist today [5] does not scale very well to the scaleof a DataSpace system.

- *Discovering the broker for a particular query*

  Given the potentially billions of queries in DataSpace, the mapping between query and its corresponding broker forms a huge information base. We make the following observations: firstly, it is *not* always resource efficient to cache the answer of a query. This is because a cached answer requires receiving constant updates in order to keep the answer current. These updates represent an overhead which may far exceed the savings gained by not querying the objects directly. This typically happens in queries whose answer changes frequently but are not queried frequently. Thus, not all queries are worth caching. Secondly, the set of queries that are worth caching changes over time. These two observations suggest a need for a system that can evaluate if a query is worth caching. Also, given the scale of the system, it is important that load distribution on brokers is not skewed.

8.1.1 Fat Shared Trees

We propose the concept of fat shared trees in order to address the first and second challenge mentioned above. A "fat" shared tree represents multiplexing of shared trees with common prefix (area code), just as shared multicast trees represent multiplexing of source-specific multicast trees. They reduce the total number of shared trees and hence, the number of multicast forwarding table entries at the routers. We propose to use "fat" shared trees both in order to reach the datacubes, and within a datacube.

In Fat shared trees, the reduction in forwarding table size comes at a cost. Forcing multiple multicast groups on the same "fat" shared tree results in some unnecessary traffic — a packet for any component multicast group is received by all the hosts connected to the "fat" shared tree. The objective is to strike a balance between unnecessary traffic and forwarding table size. At

points where unnecessary traffic exceeds a certain threshold, we need to cut the fat shared tree into multiple thinner fat trees such that unnecessary traffic volume is within acceptable limits. Also, at points where routers are on the verge of experiencing overflow in their forwarding table, they can coalesce a few fat shared trees to get a fatter shared tree.

We intend to achieve this global objective by performing splitting and coalescing operations locally at the routers. In order to perform these operation dynamically at a router, we need a mechanism for detecting when and how to spilt a fat tree, and when and how to coalesce a few fat shared trees. These four components define the fat shared tree protocol.

Note that the characteristic of a shared tree (fat or thin) is determined by the type of its corresponding entry in the forwarding table of a router. A shared tree might be part of a fat shared tree at a router, but might not be multiplexed with any other shared trees at another router. Thus, the characteristic of a shared tree is a view specific to a router. When we talk about coalescing or splitting operations, we are referring to local operations at a router to change its view of the shared tree — it can aggregate a few forwarding table entries into a single entry thereby creating a local view of a fat shared tree, or it can decide to split a shared tree from a fat shared tree creating a local view that the shared tree is not multiplexed with other shared trees.

The proposed protocol may be seen as a modification to CBT [3] (with the difference that core router discovery mechanism is implemeted at application layer [12]). It is described in detail in [7].

8.1.2 Discovering Core-router and Broker

We address the third and fourth challenges by designating a core router and a broker for every datacube. They act as core router and broker respectively for all the queries local to the datacube. The DS of a datacube maintains the address of these two entities for the datacube. The broker does not necessarily cache the answer to all the queries local to the datacube. Using the approach similar to Simple Multicast [12] proposal, we differentiate between the mechanism of finding the core router for a multicast address and the mechanism of routing multicast messages. The former is done at the application layer. We believe that the traditional approach of requiring multicast routers to be able to discover core router corresponding to any multicast address is not feasible because of the sheer scale of the system. It is important to note that we do not rule out the possibility of having more than one core routers and brokers in a datacube, if the load increases.

Since DataSpace is sparsely populated, not every datacube has an associated DS. Datacubes without DS are represented by the smallest datacube that encloses them and has a DS. These datacubes can be viewed as representatives of the all those datacubes inside them that do not have a DS. The mechanism for discovering DS of the representative datacube (DS-discovery mechanism) is described in detail in [7].

8.2 Monitoring operation: challenges

Supporting monitoring operation also requires addressing following challenges:

- The primary concern is how to keep the overhead associated with sending updates to a minimum. This goal may be broken into two sub-goals:
  - How to make sure that objects satisfying a query q send updates only when the Interested(q) set is non-empty.

  - When should an update be generated — should it be triggered by a change in state of an object, or should it be periodic. An update may be triggered when an object becomes a member so that it can be added in the answer set of the query, and it can be triggered when it is no longer a member so that it can be deleted from the answer set of the query.

- How to make sure that the objects monitoring a query have most up-to-date answer to the query in face of failures of objects and loss of updates in the network. A related issue is, should the entries in the answer be aged out if they are not refreshed.

It is useful to have an object generate an update periodically for three reasons: firstly, it takes care of loss of updates in the network, and secondly, it help members of Interested(q) to catch up with missed updates. Lastly, it allows the entries of the answer at the monitors to be aged out if they are not refreshed. This is an effective way for dealing with cases when objects fail and go out of service without generating an update indicating that they are no longer a member of the answer set of a query. Against these advantages the tradeoff is more number of updates and hence more associated overhead. A related issue that needs to be sorted out is how frequent should the updates be sent.

8.3 Brokers: challenges

Presence of brokers in our system give rise to the following related challenges:

- How does a broker decide which queries are worth caching and which are not. The problem is compounded by the fact that the set of queries that are worth caching changes over time.

  In our querying protocol, an object first direct its query to a broker. Only when the broker does not know the answer to the query, it queries the objects directly. We believe that worth of a query is a function of two parameters: popularity of a query (i.e., the rate at which the query is asked), and cost of monitoring the query (i.e., the overhead due to updates). Based on the rate at which a broker is receiving different queries, it can determine their relative popularity. However, determining the cost of monitoring a query may be nontrivial, especially because it will change over time.

- It is desirable to give the ability to a broker to spawn another broker at a candidate node when the load increases. The two issues here are: how should a node declare itself as candidate node for such an operation, and how should a broker find the set of candidate nodes and what's the heuristic for selecting a node from possibly a set of candidate nodes.

- It is desirable to give the ability to a broker to move itself to a candidate node that is closer to the querying objects. The issues here are: how should a broker decide whether it is resource efficient to move to some other node, and how should it decide which node to move to.

- *Query Zooming*

  Queries over a larger datacube should involve aggregation. For e.g., a query about locations of all taxi cabs in Manhattan may be answered by first providing distribution of their total count over two dimensional spatial grid (say, blocks). The user may subsequently issue the same query in a smaller and smaller datacube to get the more and more exact answer. This process is called query zooming.

Intuitively, query zooming mechanism may be implemented by aggregating the answer to the query over smaller sub-datacubes and storing them with their brokers. These aggregates will be returned when an object issues a query over a large datacube. Following issues need to be addressed in order to realize query zooming:

- Different queries require different aggregation function. How does a broker know which aggregation function to use for a particular query.
- In a smaller datacube, how can a broker determine which queries should be aggregated and stored.

8.4 Supporting Observability and Awareness

In TCP/IP protocol suite, the network layer offers best effort connectionless service. Supporting *observability* and *awareness* with some guarantees (other than none) is an open issue. Even in networks with QoS support, it is not clear how the QoS at the network layer translates to guarantees on answer to a particular query, or guarantees on the knowledge of answer to a particular query when performing monitoring operation.

8.5 Other Issues

- *Response implosion*

  A DataSpace comprises of huge number of objects. It is very likely that a querying object is flooded by responses, as it will not be uncommon for some queries to be satisfied by a large number of objects. We call this the response implosion problem. We have already described two mechanisms for dealing with this: brokers and *query zooming*. Other possible solutions are: *samplecast* and *gathercast*.

  In samplecast, an object that satisfies query q responds with a probability p. The querying client looks at the number of responses received and can estimate the total number of responses, thus avoiding the response implosion problem. It may then proceed by increasing p to some higher value, $p\prime$ and obtaining a bigger sample of the answer.

  Gathercast [2] is an efficient programmable solution to the problem of applying network wide aggregation and filtering. The aggregation of packets in the network can be carried out by establishing transformation functions at various points in the network. Packets routed through these points can be subjected to transformations that have been programmed previously. Examples of transformations include combining small packets, replacing a set of packets with an aggregate packets, removing redundancy, and replacing a series of packets with a summary packet etc.

- *Nearcast*

  If an object is querying in DataSpace in order to discover services (for e.g., discovering a broker for a query), it would certainly like to contact the nearest available one. This suggest a need to be able to nearcast. The issue is how to implement nearcast mechanism in a way that is scalable to the size of DataSpace.

9 Conclusions

We have defined a new concept of the three dimensional DataSpace which is the physical space enhanced with the connection to the network. The DataSpace is a collection of datacubes which are populated by often mobile classes of objects called dataflocks. Object in dataflocks produce and store their own data, such objects can be queried and monitored on the basis of their properties. The DataSpace is the next generation of the world wide web, with two major differences: it is embedded in the physical reality — organized in a geographic/spatial manner rather than logical as WWW is today; and it supports mobility of massive number of objects which produce and store their own data while moving through the DataSpace. While browsing is the main navigation mechanism for the web, querying and monitoring are the main such mechanisms for the DataSpace.

10 Acknowledgment

We are grateful for discussions and constant encouragement from B.R Badrinath. Earlier discussions with Arup Acharya are gratefully acknowledged although the full responsibility for any of the report's shortcomings is solely with the authors.

References

[1] W. Adjie-Winoto, E. Schwartz, and H. Balakrishnan. An architecture for intentional name resolution and application-level routing. Technical Report MIT-LCS-TR-775, MIT, February 1999.

[2] B. R. Badrinath and Pradeep Sudame. Gathercast: An efficient multi-point to point aggregation mechanism in IP networks. Technical Report DCS-TR-362, Department of Computer Science, Rutgers University, July 1998.

[3] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT): An architecture for scalable inter-domain multicast routing. In *Proceedings of the SIGCOMM*, 1993.

[4] S. Deering et al. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, April 1996.

[5] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. RFC 2362: Protocol independent multicast-sparse mode (pim-sm): Protocol specification, June 1998. Status: EXPERIMENTAL.

[6] R. Hinden and S. Deering. RFC2373: IP version 6 addressing architecture, July 1998. Status: INFORMATIONAL. ftp://ftp.isi.edu/in-notes/rfc2373.txt.

[7] Tomasz Imieliński and Samir Goel. Dataspace - querying and monitoring deeply networked collections in physical space, Part II - protocol details. Technical Report DCS-TR-400, Rutgers University, July 1999.

[8] Tomasz Imieliński and Samir Goel. Dataspace - querying and monitoring deeply networked collections of physical objects. To appear in Proceedings of of the DIMACS Workshop on Mobile Networks and Computing, March 1999.

[9] Tomasz Imieliński and Samir Goel. Dataspace - querying and monitoring deeply networked collections of physical objects. In *Proceedings of the International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'99)*, Seattle, WA, August 1999.

[10] Tomasz Imieliński and Samir Goel. Dataspace - querying and monitoring deeply networked collections of physical objects. Position paper for the NSF/DARPA/NIST Workshop on Smart Environments, July 1999.

[11] Julio C. Navas and Tomasz Imielinski. Geographic addressing and routing. In *Proc. of the Third ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'97)*, Budapest, Hungary, September 1997.

[12] R. Perlman et al. Simple Multicast: A design for simple, low-overhead multicast, Internet-Draft, February 1999. http://search.ietf.org/internet-drafts/draft-perlman-simple-multicast-02.txt. Work in Progress.

[13] Sun Microsystems Inc. *Jini Architecture Specification*, January 1999. Technical Specifications. http://www.sun.com/jini/specs/jini-spec.ps.

[14] *GPS SPS Signal Specification*, second edition, June 1995. http://www.navcen.uscg.mil/gps/geninfo/gpsdocuments/sigspec/default.htm.