Comet: A Scalable Coordination Space for Decentralized Distributed Environments *

Zhen Li and Manish Parashar
The Applied Software Systems Laboratory
Dept. of Electrical and Computer Engineering, Rutgers University, Piscataway NJ,08854, USA
Email: {zhljenny, parashar}@caip.rutgers.edu

Abstract

The scale, heterogeneity, and dynamism of emerging distributed and decentralized environments make coordination a significant and challenging problem. In this paper we present Comet, a scalable peer-to-peer content-based coordination space. Comet provides a global virtual shared-space that can be associatively accessed by all peer nodes in the system, and access is independent of the physical location of the tuples or identifiers of the host. Dynamically constructed, transient coordination spaces are also supported to enable context locality to be explicitly exploited for improving system performance. The design, implementation, and experimental evaluation of Comet using the PlanetLab platform are presented.

1 Introduction

Coordination can be defined as *managing the runtime dependencies and interactions among the elements in the system*. With the emergence of wide-area distributed and decentralized environments, such as pervasive information systems, peer-to-peer systems, and computational Grid infrastructures, these dependencies and interactions can be complex and various (e.g. peer-to-peer, client-server, producer-consumer, collaborative, at-most/at-least/exactly, etc.), and both coordinated entities and the nature of the relationships and interactions between them can be ad hoc and opportunistic. As a result, realizing these coordination behaviors using low-level protocols becomes extremely difficult.

Clearly, designing and developing a coordination middleware that meets these requirements are non-trivial. A key issue is the choice of the underlying coordination model. Models based on direct communication, such as Remote Procedure Call, imply a strict coupling in time, place, and name between the interacting entities. Consequently, such models are not suitable for the large decentralized systems where maintaining common knowledge about the names/identifiers, addresses of an end-point as well as the syntax and semantics of the interfaces is infeasible. In contrast, shared-space based coordination model has been adopted to address the challenges of incomplete knowledge, system dynamism, and heterogeneity. This approach has been popularly adopted for supporting the coordination among system entities, e.g., JavaSpaces [6], TSpaces [8], Lime [10]. However, these systems are not appropriate in wide-area P2P environments, which require rich data expressibility, flexible matching, and scalable performance.

In this paper, we present the design, implementation, and evaluation of Comet, a scalable content-based coordination space for wide-area P2P environments. Comet provides a global virtual shared-space that can be associatively accessed by all peers in the system, and access is independent of the physical location of the tuples or identifiers of the host. It builds on a distributed hash table (DHT) substrate, which has emerged as an efficient abstraction for developing data lookup systems in wide-area P2P networks, e.g., CAN [12] and Chord [14]. These systems provide scalable routing performance and support dynamic join and departure of peer nodes, and can route around failures. However, these DHT-based systems maintain rigid overlay structures and use consistent hashing for achieving extreme scalability, which results in two disadvantages. First, the unique key based data distribution can not support tuple searches with partial keywords or wildcards in a scalable manner. Second, the content and context localities are not preserved. These are important factors for improving the performance of coordination primitives, since content locality enables textual similar tuples to be placed on the same node or a set of close nodes, and context locality enables the communication traffic only within the set of nodes within a physical organization or geographical area.

The Comet coordination model is based on a global vir-

*The research presented in this paper is supported in part by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826.

tual shared-space constructed from a semantic information space. The information space is deterministically mapped, using a locality preserving mapping, to a dynamic set of peer nodes in the system. The resulting peer-to-peer information lookup system maintains content locality and guarantees that content-based information queries, using flexible content descriptors in the form of keywords, partial keywords and wildcards, are delivered with bounded cost. Using this substrate, the space can be associatively accessed by all system peers without requiring the location information of tuples and host identifiers. The Comet provides transient spaces that enable the applications can explicitly exploit context locality. The current prototype of Comet builds on the JXTA [1] peer-to-peer framework and is deployed on PlanetLab [2], a wide-area heterogeneous distributed environment.

The rest of this paper is structured as follows. Section 2 introduces the background and related work. Section 3 describes the architecture and implementation of Comet. An experimental evaluation is demonstrated in section 4, and section 5 presents the conclusion and outlines current research directions.

2 Background and Related Work

2.1 Shared-space Based Coordination and Linda

The shared-space based coordination model was made popular by Linda [7], which defines a centralized tuple space that provides a shared message repository to exploit generative communication [7]. The key attributes of Linda include: (i). Asynchronous communication that decouples senders and receivers in space and time. An inserted tuple will exist independently in the tuple space until it is explicitly removed by a receiver, and tuples are equally accessible to all receivers but bound to none. (ii). An associative multicast medium through which multiple receivers can read a tuple written by a single sender using a pattern-matching mechanism instead of the name and location. (iii). A small set of operators (write, read, and remove) providing a simple and uniform interface to the tuple space.

In Linda, a tuple is an ordered sequence of typed fields and a single tuple space is a multi-set of tuples that can be accessed concurrently by several processes using simple primitives. Tuples are put to the space by executing the *out(t)* operation, extracted using the destructive primitive *in($\bar{t}$)*, and read using the non-destructive primitive *rd($\bar{t}$)*, where t is the tuple and $\bar{t}$ is a template that matches the tuple. Both *in* and *rd* are blocking operations. If multiple tuples match a template, one tuple will be nondeterministically returned. The template $\bar{t}$ may contain wildcards that are matched against actual values in a tuple during the pattern-matching process. For example, a tuple ("task", 12) is matched by a template ("task", ?Int).

2.2 Related Work

Several research projects and commercial products have successfully adopted and enhanced the shared-space model to construct robust coordination platforms. Examples include JavaSpaces [6], TSpaces [8], and XMLSpaces [15]. While these systems extend the original Linda model with improved language expressiveness and control flexibility, they maintain the centralized tuple space implementations which seriously limits the scalability of these systems.

More recent coordination systems have focused on fully decentralized architecture to address scalability in distributed systems, e.g., Lime [10], PeerWare [5], PeerSpace [4], etc. The two most related to this research, Lime and PeerWare, which build on the concept of Global Virtual Data Structures (GVDS), are described below. Lime [10] realizes a global space that is transiently shared and dynamically built upon the local data spaces of a set of hosts and the operations performed locally can have global effects. The primary objective of Lime is to provide Linda-like coordination in mobile environments. It exploits a flat data structure and extends the Linda interfaces with a location parameter λ, expressed in terms of agents or host identifiers. PeerWare [5] realizes a forest of trees, composed of nodes and documents, which are contributed by each peer. Its basic access primitive *execute* requires the $\mathcal{F}_D$ and $\mathcal{F}_N$ functions to filter the operated set of documents and nodes. Both these systems implicitly employ the *context-aware* programming style where information about the location of system components (e.g., nodes, hosts, or agents) is required by the coordination primitives.

However, maintaining such a global knowledge about location in large and highly dynamic distributed systems is infeasible. In contrast, Comet uses the *context-transparent* approach and realizes the GVDS as a distributed hash table, where all operations only use tuple content and are independent of the current state of the system and the mapping of content to these peers.

3 Design and Implementation of the Comet

Comet is constructed from a semantic multi-dimensional information space defined by the coordinated entities. This information space is deterministically mapped onto a dynamic set of peer nodes in the system using a locality preserving mapping. Comet is composed of layered abstractions prompted by a fundamental separation of communication and coordination concerns. A schematic overview of the system architecture is shown in Figure 1. The communication layer provides scalable content-based messaging and manages system heterogeneity and dynamism. The coordination layer provides Linda-like associative primitives and supports a shared-space coordination model. Dynamically

constructed transient spaces are also supported in Comet to allow the applications explicitly exploit context locality for improving system performance.

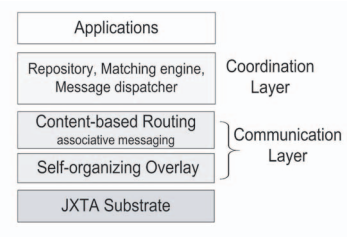

Figure 1. A schematic overview of the Comet system architecture.

The current prototype of Comet has been implemented on Project JXTA [1], a platform independent peer-to-peer framework, where peers can self-organize into peergroups, discover peer resources, and communicate with each other. The Comet coordination space is provided as a JXTA peergroup service that can be concurrently exploited by multiple applications. The peergroup provides a secure environment where only member peers can access the service instances running on peers of the group. If any one peer fails, the collective peergroup service is not affected and the service is still available from other peer members. Further, the transient spaces are implemented based on the peergroup concept. Peers can belong to several spaces and an application can dynamically switch between coordination services associated with these spaces. To destroy the transient space, each peer node in the peergroup stops the service and deletes its local space instance.

3.1 Tuples and Tuple Distribution

In Comet, a tuple is a simple XML string, where the first element is the tuple's tag and is followed by an ordered list of elements containing the tuple's fields. Each field has a name followed by its value. The field name may act as the type function. The tag, field name, and value must be actual data for a tuple and can contain wildcard ("*") for a template tuple. This lightweight format is flexible enough to represent the information for all kinds of applications and has rich matching relationships [15]. It is suitable for efficient information exchange in distributed heterogeneous environments.

A tuple can be retrieved if it exactly or approximately matches a template tuple. Exact matching requires the tag and field names of the template tuple to be specified without any wildcard, as in Linda. However, this strict matching pattern must be relaxed in highly dynamic environments, since applications (e.g., service discovery) may not know the exact tuple structures. Comet supports tuple retrievals with incomplete structure information using approximate matching, which only requires the tag of the template tuple to be specified using a keyword or a partial keyword. Examples are shown in Figure 2. In this figure, tuple (a) tagged "contact" has fields "name, phone, email, dep" with values "Smith, 7324451000, smith@gmail.com, ece" can be retrieved using (b) or (c).

```
<contact>                          <contact>                          <con*>
   <name> Smith </name>               <name> Smith </name>               <na*> Smith </na*>
   <phone> 7324451000 </phone>        <phone> 7324451000 </phone>        <*>
   <email> smith@gmail.com </email>   <email>* </email>                 <*>
   <dep> ece </dep>                   <dep> * </dep>                     <dep> ece </dep>
</contact>                         </contact>                         </>
        (a)                                (b)                                (c)
```

Figure 2. Examples of tuples in Comet.

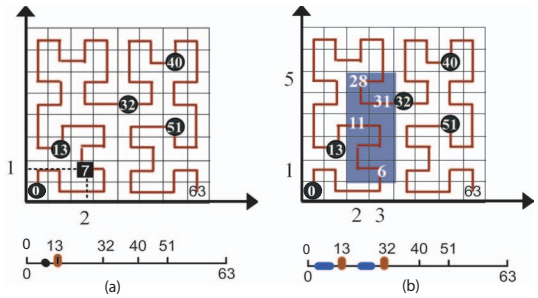

Figure 3. Examples of mapping tuples from 2D information space to 1D index space.

Comet employs the Hilbert Space-Filling Curve (SFC) [9] to map tuples from a semantic information space, consisting of based-10 numbers and English words, to the linear node index. Each tuple is associated with k keywords selected from its tag and names. They are defined as the **keys** of the tuple in the k-dimensional (kD) information space. For example, the keys of (a) in Figure 2 can be "contact, name" in a 2D information space. Tuples are local in the information space if their keys are lexicographically close or have common keywords. The dimensionality of the information space and the selection of keys can be specified by the application.

A Hilbert curve is a locality preserving continuous mapping from a kD space to a 1D space. It is locality preserving in that points that are close on the curve are mapped from close points in the kD space. The Hilbert curve readily extends to any number of dimensions. The locality preserving property of the Hilbert curve enables the tuple space to maintain content locality in the index space. In Comet, the Hilbert SFC is used to index overlay peers and map tuples from the information space to the linear peer index space. If the keys of a tuple only include complete keywords, the tuple is mapped as a point in the information space and located on at most one node. If its keys consist of partial key-

words, wildcards, or ranges, the tuple identifies a region in the information space. This region is mapped to a collection of segments on the SFC and corresponds to a set of points in the index space. Each node stores the keys that map to the segment of the curve between itself and the predecessor node. For example, as shown in Figure 3, five nodes (with id in solid circle) are indexed using SFC from 0 to 63, the tuple defined as the point (2,1) is mapped to index 7 on the SFC and corresponds to node 13, and the tuple defined as the region (2-3,1-5) is mapped to 2 segments on the SFC and corresponds to nodes 13 and 32.

3.2 The Communication Layer

The communication layer provides an associative communication service and guarantees that content-based messages, specified using flexible content descriptors, are served with bounded cost. This layer essentially maps the virtual information space in a deterministic way to the dynamic set of currently available peer nodes in the system while maintaining content locality.

3.2.1 Abstraction

This layer provides a single operator: **deliver** ($\mathcal{M}$). The message $\mathcal{M}$ includes (1) a semantic selector that is flexibly defined using keywords, partial keywords, or wildcards from the information space, and specifies a region in this space, and (2) a payload consisting of the operation to be performed at the destination and the data. This operator forwards the message $\mathcal{M}$ to all destination nodes containing content that lies in the region defined by the semantic selector, i.e., that matches the selector. The resolution of this operator depends on the current information existing in the system as well as the current peer nodes. It is unlike low-level messaging protocols that send/receive messages to/from specific destinations, the destination(s) in this case are dynamically determined based on the state of the system. Coordination operations are directly built on this operator, where the semantic selector is defined by the elements of the tuple and the payload includes the tuple data and action, e.g. "store" for *out* and "read" for *rd*. Note that this operator can return one, some, or all of the matched tuples.

3.2.2 Implementation

This layer provides scalable content-based routing and data delivery operations. It includes a content-based routing engine and a structured self-organizing overlay.

The routing engine provides a decentralized information discovery and associative messaging service. It implements the Hilbert SFC mapping to effectively map a multi-dimensional information space to a peer index space and to the current system peer nodes, which form a structured overlay. The resulting peer-to-peer information system supports flexible content-based routing and complex queries containing partial keywords, wildcards, or ranges, and guarantees that all existing data elements that match a query will be found.

The engine implements the abstraction and has a single operator: post(keys, data), where keys form the semantic selector and data is the message payload provided by the above layer. If the keys only include complete keywords, the engine routes the message using the overlay lookup mechanism. If the keys contain partial keywords or wildcards, the message identifies a region in the information space. The region is defined as one cluster [9] if the SFC enters and exits it once. A cluster might be mapped to one or more adjacent overlay nodes and one node can store multiple clusters. It is not scalable to send a message to each cluster since the number of nodes can be very high. In Comet, the Squid optimization scheme [13] is adopted to improve the scalability, which embeds the query tree into the overlay network and distributes the cluster refinement at each node.

The overlay is composed of peer nodes, which may be any node in the system (e.g., gateways, access points, message relay nodes, servers, or end-user computers). The peer nodes can join or leave the network at any time. While the Comet architecture is based on a structured overlay, it is not tied to any specific overlay topology. In the current implementation of Comet, we use Chord [14], which has a ring topology, primarily due to its guaranteed performance, efficient adaptation as nodes join and leave the system, and the simplicity of its implementation. In principle, this could be replaced by other structured overlays.

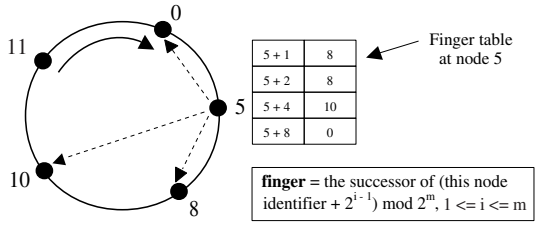

Figure 4. Example of the Chord overlay.

The overlay layer provides a simple abstraction to the layers above: lookup(identifier). Given an identifier, this operation locates the node that is responsible for it, i.e, the node with an identifier that is the closest identifier greater than or equal to the queried identifier. Every node in Chord is assigned a unique identifier and maintains a *finger table* for routing. An example of a Chord overlay network with 5 nodes is shown in Figure 4. Each node constructs its finger table when it joins the overlay and finger tables are updated any time a node joins or leaves the system. The lookup algorithm in Chord enables the efficient data routing with

O(Log N) cost [14], where N is the number of nodes in the system.

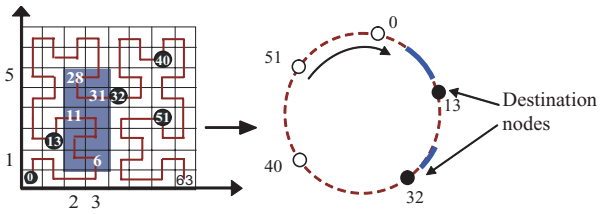

Figure 5. Example of content-based routing in Comet.

Content-based routing in Comet is achieved as follows. SFCs are used to generate a 1D index space from the multi-dimensional keyword space. A simple query composed of only keywords is mapped to a point on the SFC. A complex query containing partial keywords or wildcards is mapped to regions in the keyword space and to corresponding clusters (segments of the curve) on the SFC. The 1D index space generated from the entire information space is mapped onto the 1D identifier space used by the overlay network formed by the peer nodes. As a result, using the SFC mapping any query request can be located. For example, the tuple in Figure 3(a) defined as a point (2,1) in a 2D space is mapped to index 7 on SFC and routed on Chord (an overlay with 5 nodes and an identifier space from 0 to 2^6-1) to node 13, the successor of the index 7. Similarly, the tuple in Figure 3(b) defined as a region (2-3,1-5) in the 2D space is mapped to 2 segments on the SFC, and routed to node 13 and node 32 on Chord, shown in Figure 5.

3.3 The Coordination Layer

The coordination layer provides tuple operation primitives to support shared-space based coordination model and can be enhanced with notifications [6] and reactivities [11].

3.3.1 Abstraction

This layer defines the following coordination primitives, which retain the Linda semantics, i.e., if multiple matching tuples are found, one of them is arbitrarily returned (and removed). If there is no matching tuple, the operation waits for one to appear.

- Out(ts, t): an non-blocking operation that inserts the tuple t into space ts.
- In(ts, $\overline{t}$): a blocking operation that removes a tuple t matching template $\overline{t}$ from the space ts and returns it.
- Rd(ts, $\overline{t}$): a blocking operation that returns a tuple t matching template $\overline{t}$ from the space ts. The tuple is not removed from the space.

3.3.2 Implementation

The main components of this layer include a data repository for storing, pending requests, and retrieving tuples, a flexible matching engine, and a message dispatcher that interfaces with the communication layer to convert the coordination primitives to messaging operations and vice versa. Tuples are represented as simple XML strings as they provide small-sized flexible formats that are suitable for efficient information exchange in distributed heterogeneous environments. The data repository stores tuples as DOM level 2 objects [3]. It employs a hash structure to perform pattern matching at a constant time in memory.

In Comet, it is assumed that all peer nodes agree on the structure and dimension of the information space. The *Out*, *Rd* and *In* operations are implemented using the post operation of communication layer. Using the associated keys of a tuple, each tuple is routed to an overlay peer node and a template tuple may be routed to a set of nodes. Since the keys of a template for exact matching only contain complete keywords, the template will be mapped to one point in the index space, and thus routed to the node that may store matched tuples using the overlay lookup protocol. The tuple distribution and exact retrieval processes using *Out* and *In/Rd* operators are illustrated in Figure 6 and 7 respectively.

The process consists of the following steps: (1) Keywords are extracted from the tuple and used to create the keys for the post operation. The payload of the message includes the tuple data and the coordination operation. (2) The query engine uses the SFC mapping to identify the indices corresponding to the keys and the corresponding peer id(s). (3) The overlay lookup operation is used to route the tuple to the appropriate peer nodes. This operator maps the logical peer identifer to the JxtaID of the node, and sends the tuple using the JXTA Resolver Protocol. The *Out* operation only returns after receiving the Resolver Query Response from the destination to guarantee tuple delivery. In the case of *Rd* and *In* operations, templates are routed to the peer nodes in a similar manner. The *In* and *Rd* operations block until a matched tuple is returned by the destination in a peer-to-peer manner.

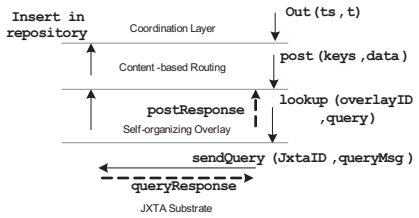

Figure 6. Tuple distribution using the Out operator.

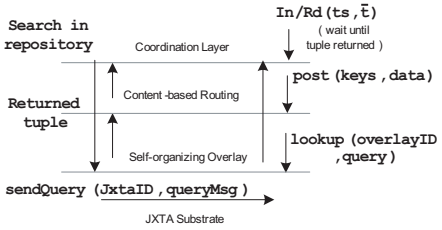

Figure 7. Exact tuple retrieval using the In/Rd operator.

The approximate retrieval process is similar. A retrieval request may be sent to multiple nodes and each of them may return a matched tuple. The *In* and *Rd* are separately implemented. In case of *Rd*, the first searched tuple is returned to achieve efficiency, the following requests are ignored by the query node if a matched tuple is found or removed when time out. For an *In* operation, the matched tuple must be deleted only after getting the confirmation from the query node. When a matched tuple is found locally, the node sends a copy to the query node and waits for a delete confirmation. After getting a matched tuple, the query node sends a confirmation to the node that first returns the tuple and informs other nodes to remove the corresponding pending requests by a complex query.

3.4 Transient Spaces in Comet

Based on the description presented above, Comet is naturally suitable for context-transparent applications. However, certain applications, e.g., mobile applications, require that context locality be maintained in addition to content locality, i.e., they impose requirements for context-awareness. The uniform operators provided by Comet do not distinguish between local and remote components of a space. While this is a convenient abstraction, it does not maintain context locality and may have a detrimental effect on system efficiency for these applications.

To address this issue, Comet supports dynamically constructed transient spaces that have a specific scope definition (e.g., within the same geographical region or the same physical subnet). The global space is accessible to all peer nodes and acts as the default coordination platform. Membership and authentication mechanisms are adopted to restrict access to the transient spaces. The structure of the transient space is exactly the same as the global space. An application can switch between spaces at runtime and can simultaneously use multiple spaces.

4 Comet Operation and Evaluation

4.1 Overall Operation

The overall operation of the Comet includes two phases: *bootstrap* and *running*. During the *bootstrap* phase peer nodes join the group and exchange messages with the rest of the group. During this phase, the joining peer attempts to discover an existing peer in the system and to construct its routing table. It sends discovery messages to the group. If the message is unanswered after a pre-defined time interval (in the order of seconds), the peer assumes that it is the first in the system. If a peer responds to the message, the joining peer queries this bootstrapping peer according to the join protocol of the underlying overlay, and updates routing tables in the overlay to reflect the join.

The *running* phase consists of stabilization and user modes. In the stabilization mode, a peer node responds to queries issued by other peers in the system. The purpose of the stabilization mode is to ensure that routing tables are up-to-date, and to verify that other peer nodes in the system have not failed or left the system. In the user mode, each peer node interacts as part of the coordination space to provide coordination services.

4.2 Experimental Evaluation

This section presents the evaluation of the performance of Comet prototype deployed on PlanetLab [2], a large scale heterogeneous distributed environments composed of interconnected sites with various resources. Note that while Comet is currently deployed on more than 50 sites on PlanetLab, the highly dynamic and uncertain availability of PlanetLab prevented us from evaluating the system on all these nodes. An evaluation of the average performance of Comet using up to 32 nodes is demonstrated below. We are working on extending these results to the other accessible PlanetLab nodes. In the experiments, each machine ran an instance of Comet, serving as a peer node in the overlay.

Since PlanetLab nodes are distributed across over the globe, communication latencies can vary significantly over time and node locations. As a result, we adopted the following methods in the experiments: (1) In each experiment, at least one node was selected from each continent, including Asian, Europe, Australia, and North America. For example, some of the Internet domains involved include `snu.ac.kr`, `huji.ac.il`, `xeno.cl.cam.ac.uk`, `cs.uit.no`, `upc.es`, `cs.vu.nl`, `uwaterloo.ca`, `stanford.edu`, etc. (2) Nodes randomly joined the Comet system during bootstrap phase, resulting in a different physical construction of the ring overlay in each run. (3) The experiments were conducted at different time of the day during a 4-week

period, and each experiment ran continuously for about 3 hours.

A ping-pong like process is used in the experiments in which an application process inserted a tuple into the space using the *Out* operator, read the same tuple using the *Rd* operator, and deleted it using the *In* operator. The contents of tuples are randomly generated strings with a fixed length of 110 bytes. To simulate the different application behaviors, the tuples were generated through a Poisson process with inter-arrival mean time 1s, 5s, and 10s. The experiments measured the average run time for each of the primitives and abstractions provided by Comet, including the tuple insertion operation *Out*, exact retrieval operation *Rd/In*, transient spaces, and approximate retrieval operation *Rd/In*.

4.2.1 Tuple Insertion and Exact Retrieval

In these experiments, the *Out* and exact matching *Rd/In* operations are evaluated using a 3D information space. For an *Out* operation, the measured time corresponds to the time interval between when the tuple is posted into the space and when the response from the destination is received, i.e., the time between Post and PostResponse in Figure 6. For a *Rd/In* operation, the measured time is the time interval between when the template is posted into the space and when the matched tuple is returned to the application, assuming that a matching tuple exists in the space, i.e., the time between Post and receiving the tuple in Figure 7. This time includes the time for template routing, repository matching, and returning the matched tuple. Note that the in-memory template matching time is very small comparing to the communication time for the ping-pong test.

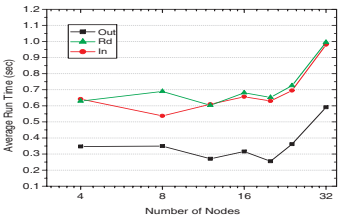

Figure 8. Average run time for Out and exact In/Rd operations.

The average run time of the operations for increasing number of nodes is plotted in Figure 8. The X-axis is the logarithm with base 2 of the number of running nodes. The Y-axis is the average run time in seconds. As seen in Figure 8, the operation time increases by a factor of about 2 when the system size grows by a factor of 8. This is expected as the complexity of the underlying Chord lookup protocol is O(Log N), where N is the number of nodes in the system. Note that the costs plotted in the figure have the same order as the cost of the Chord lookup protocol presented in [14]. This protocol has been shown to scale to 10^4 nodes using simulations in [14]. As a result, we believe that Comet will exhibit similar scalability. Each value plotted is averaged over 3 experiments. Figure 9 plots the histogram of *Out* operation run time on 32 nodes in one experiment. This plot shows that 95% samples are less than 0.65 second, one standard deviation away from the mean.

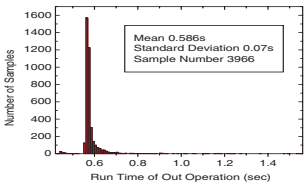

Figure 9. Histogram of Out operation run time on 32 nodes.

4.2.2 Approximate Tuple Retrieval

In these experiments, the performance of approximate *In/Rd* operations are evaluated using 2D and 3D information spaces. The templates used in the experiments are: (Case 1) one keyword and at least one partial keyword, e.g., (contact, na*), (contact, na*, *), and (Case 2) one keyword or partial keyword, e.g.,(con*, *), (contact, *, *). For a *Rd/In* operation, the measured time is the time interval between when the template is posted into the space and when the matched tuple is returned. The average time over about 500 operations is shown in Figure 10. The figure shows that while the operation time increases with the number of nodes and the dimensions of the information space, and the rate of increase is typically much smaller than the rate of increase of the system size. Further, the number of query processing nodes in Squid has been shown, using simulations, to be a small fraction of the total nodes [13], i.e., below 8% in 2D and 20% in 3D for (Case 2) with system size increasing from 1000 nodes to 5000 nodes. Also, this fraction decreases as the system size increases. As a result, we can conclude that the approximate retrieval operation in Comet will also scale to large systems.

4.2.3 Evaluation of the Comet Transient Spaces

The time for the creation and service initialization of a transient space is about 100 seconds per node for the stabilization. This time includes the time required to initialize a JXTA peergroup and execute the join protocol. The insertion and exact retrieval performance for a 4-node space scenario varies with the geographical location of the nodes, as shown in Figure 11. The average run time for the case where the nodes are located within a LAN at Rutgers University is

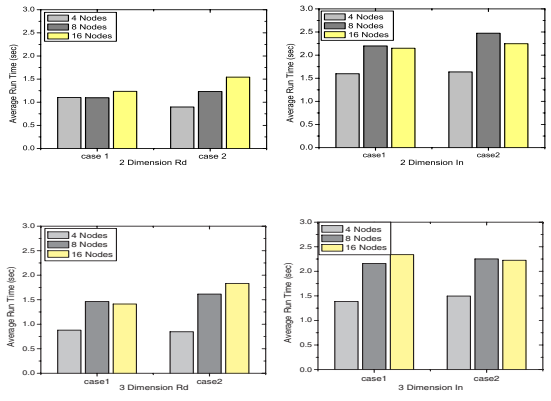

Figure 10. Average run time for approximate In/Rd operations.

about 4 times smaller than the case where these nodes are within America, and 6 times smaller than the case where the nodes are distributed across 4 continents. From these results, it can be concluded that initialization is an one-time cost and setting up the transient space can improve system performance in terms of operation latencies.

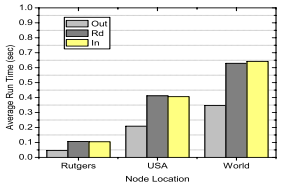

Figure 11. The system performance with different geographical locations.

5 Conclusion and Ongoing Work

In this paper we presented Comet, a scalable P2P coordination space with rich expressiveness. Comet provides a global virtual shared-space that can be associatively accessed by all system peers without requiring the host identifiers and location information of tuples. The virtual space builds on a distributed hash table where the index space is directly generated from the semantic information space used by the coordinating entities. Using the DHT enables Comet to guarantee both information lookup and performance, which are essential requirements for a coordination middleware in decentralized distributed environments. Dynamically constructed, transient coordination spaces are also supported to enable context locality to be explicitly exploited for improving system performance. The design, implementation, and evaluation of Comet were discussed in detail. Initial experimental results on PlanetLab demonstrate that both the scalability and performance of the system are promising.

Current research efforts include improving the efficiency of the approximate *In* operation, extending the system with *RdAll* and *InAll* primitives, and providing mechanisms at the coordination layer to address fault tolerance and load balance.

References

[1] Project JXTA. http://www.jxta.org.

[2] Project PlanetLab. http://www.planet-lab.org.

[3] Document Object Model (DOM) Level 2 Core Specification.w3c recommendation. http://www.w3.org/TR/DOM-Level-2-Core, 2000.

[4] N. Busi, C. Manfredini, A. Montresor, and G. Zavattaro. PeerSpaces: Data-driven Coordination in Peer-to-Peer Networks. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 380–386. ACM Press, 2003.

[5] G. Cugola and G. Picco. PeerWare: Core Middleware Support for Peer-To-Peer and Mobile Systems. Technical report, Politecnico di Milano, 2001.

[6] J. et al. JavaSpace Specification 1.0. Technical report, Sun Microsystems, 1998.

[7] D. Gelernter. Generative Communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[8] T. J. Lehman, S. W. McLaughry, and P. Wycko. TSpaces: The Next Wave. In *Proceedings of Hawaii International Conference on System Sciences*, 1999.

[9] B. Moon, H. v. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[10] A. Murphy, G. Picco, and G.-C. Roman. Lime: A Middleware for Physical and Logical Mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 524–536, 2001.

[11] A. Omicini and F. Zambonelli. Coordination for Internet Application Development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, 1999.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-addressable Network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[13] C. Schmidt and M. Parashar. Enabling Flexible Queries with Guarantees in P2P Systems. *IEEE Network Computing, Special issue on Information Dissemination on the Web*, (3):19–26, June 2004.

[14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[15] R. Tolksdorf and D. Glaubitz. Coordinating Web-based Systems with Documents in XMLSpaces. In *Proceedings of the Sixth IFCIS International Conference on Cooperative Information Systems*, 2001.