A Framework for Classifying Peer-to-Peer Technologies

Krishna Kant, Ravi Iyer
Server Performance Architecture
Enterprise Architecture Lab
Intel Corporation, OR

Vijay Tewari
Distributed Systems Architecture
Distributed Systems Lab
Intel Corporation, OR

Abstract— **Popularized by Napster and Gnutella file sharing solutions, peer-to-peer (P2P) computing has suddenly emerged at the forefront of Internet computing. The basic notion of cooperative computing and resource sharing has been around for quite some time, although these new applications have opened up possibilities of very flexible web-based information sharing. This article provides a framework for classifying current and future P2P technologies. The main motivation for the classification is to identify basic characteristics of P2P applications so that the infrastructure to support P2P computing can concentrate on these basic characteristics.**

Keywords: peer-to-peer computing, distributed systems, network-of-workstations, taxonomy, file sharing, cycle sharing.

I. Introduction

Last one year or so has seen an explosive growth in the use of file-sharing software in order to exchange digital audio, video and other types of files. The trend was started by Napster (`www.napster.com`), which allows sharing of MP3 music files among an arbitrary set of users. Napster quickly led to numerous variants of file-sharing software including Wrapster (a slight generalization of Napster) and Morpheus (which provides general file sharing with optimized download algorithms using multiple copies based on the FastTrack protocol). Concurrently, distributed versions of file-sharing have also been developed, including Gnutella (`gnutella.wego.com`) and Freenet (`freenet.sourceforge.net`).

These early developments sparked a new interest and flurry of activity in the so-called peer-to-peer (P2P) computing space, including its application to improve the current web infrastructure. Although a lot of issues relevant for P2P computing have been examined in the traditional parallel/distributed computing areas, the availability of vast resources via the Internet requires a re-examination of many issues to address problems of scalability to millions of nodes and coping with restricted addressability and intermittent connectivity. In this article, we provide an introduction to the major recent developments that have brought P2P computing to the forefront of Internet computing and propose a framework for classifying P2P computing services. Such a classification reduces the current and developing P2P applications into a set of basic characteristics and thus is expected to simplify the job of supporting these applications. The classification is also expected to reveal "holes" in both the application space and the infrastructure needed.

We note here that because of the relative novelty of the P2P computing field, the classification framework proposed here is to be considered only as tentative; as the P2P computing area matures, the classification itself may need to be revisited. Also, *the classification proposed here is purely from the application environment perspective*; it is surely possible to examine other aspects such as social issues, pricing, business-model and legal issues, which are not addressed here. *The classification is also not intended for product evaluations.* For example, by virtue of their implementation methods certain products may provide better security, local autonomy, fault-tolerance, etc. than other products covering the same space, but the dimensions chosen in the classification are not designed to reflect these differences.

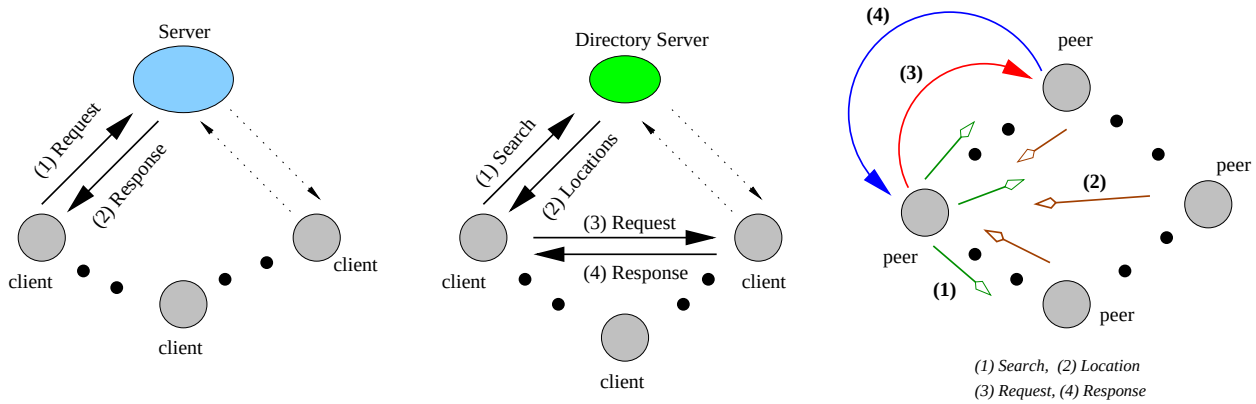
II. Emerging Peer-to-Peer Usage Models and Services

In this section, we provide a brief overview of many instances of P2P computing usage scenarios.

A. Content Sharing

The Napster solution enables MP3 file sharing under the control of a centralized directory server which maintains basic addressability and availability information about the user nodes and the meta-information about the shared files. The centralization allows a quick search for the requested file(s) and assists in identifying the most suitable location to download the files. The actual file transfer still happens over a direct TCP connection between the requester and owner nodes.

Unlike Napster, the Gnutella community is based on a fully distributed approach and can be thought of as an ad-hoc network set up among a set of peers. Figure 1 shows a pictorial representation of traditional client-server, Napster-style, and Gnutella-style methods of satisfying a query. A peer A initially needs to know the IP address of at least one member, say B, of an existing Gnutella community. Once A connects to B, A obtains information on all the nodes that B is aware of and thus can establish direct TCP/IP connections to those that it finds most interesting. Each Gnutella node specifies a set of local folders as *shared* which can be searched based on partial or full match on the desired file names. The search starts towards all connected



(a) Traditional Client−Server Computing (b) Napster−style Computing (c) Gnutella/Freenet−style Computing

Fig. 1. File Sharing Approaches: An illustration

neighbors of the originating node and progresses recursively up to some predetermined number of hops. Each search message is stamped with a globally unique ID to prevent duplicate responses for the same request. Based on the responses received, the user selects appropriate file(s) for downloading and initiates an HTTP-like connection with each owner node.

Like Gnutella, Freenet uses a fully distributed model but introduces a number of innovations (anonymity, caching, etc) that merit some discussion. A Freenet node shares storage (rather than files or objects) by designating a local "shared" directory where the files can be inserted by any of the peers. Each file is identified (not necessarily uniquely) via a key that reflects the file content (and, in general, could include other information such as access rights). Each node retains locally stored files in an LRU file cache and maintains the metadata of all local and some remote files in another LRU cache. The remote file information accumulates because when a local file is deleted, the corresponding metadata is still retained along with likely location of the file. When a node receives a search request for such a file, the metadata can be used to efficiently direct the search to the node that potentially holds the file. If the node receiving a search request cannot find any match on the local metadata, it directs the request to some number of nodes whose stored key is "close" to the request key [5]. This process is repeated only up to a certain hop-count; a failure is indicated if no match is found within the specified hop-count.

When a match is found, the requested object is returned backwards along the request path (different from Gnutella, where the object retrieval requires another explicit request based on search results.) In Freenet, each node in the path caches the response object in order to satisfy future requests more quickly. The object insertion follows a similar procedure, in that a local insertion at a node results in automatic propagation of the object to neighboring nodes up to a given hop-count.

Although these applications and their derivatives concentrate only on the file-sharing aspect of P2P computing, the same approach can also be used for the purposes of "web crawling", distributed auctioning and e-library applications. One of the fundamental problems in these early peer-to-peer content sharing applications is the lack of efficient and scalable content addressability. Some recent studies have addressed this issues by designing scalable data location protocols [19], [9], [18], [15]. For example, the emphasis in Chord [19] is to map keys onto nodes using consistent hashing techniques. The intention of this work is that content sharing applications can be built on top of Chord by associating a key with each data item and storing the key/data item pair at the node to which the key maps. Similar attempts have been made in Past[10], OceanStore [15] and CAN [18].

B. Hardware Resource Sharing

One of the early drivers for P2P computing was the realization that a typical home PC is mostly idle and thus could be harnessed for solving complex computational problems. The idea is to decompose the problem into large chunks that can run concurrently with very little interaction, referred by some as "embarrassingly parallel" computation [21]. Perhaps the best known (but simplest) examples of this "CPU cycle sharing" usage are SETI@HOME (`setiathome.ssl.berkeley.edu`) and its variants (`www.entropia.com, www.ud.com`). A pictorial comparison of traditional compute servers versus the SETI@HOME approach is shown in Figure 2. Recently the SETi@HOME model has been successfully used by Oxford University team in reducing the possible anthrax treatment compounds from 3.5 billion to a more manageable 300,000. The data set was searched in a relatively short period (4 weeks) as opposed to years (`www.newscientist.com/news/news.jsp?id=ns99991953`). Similar ideas have been tried with other complex problems such as finite-element analysis, climate change models, graphical rendering of complex scenes, etc. Examples involving sharing of other hardware resources (e.g., memory or disk) also abound. For example, peer-to-peer extension of the popular i-drive service (`www.idrive.com`) provides

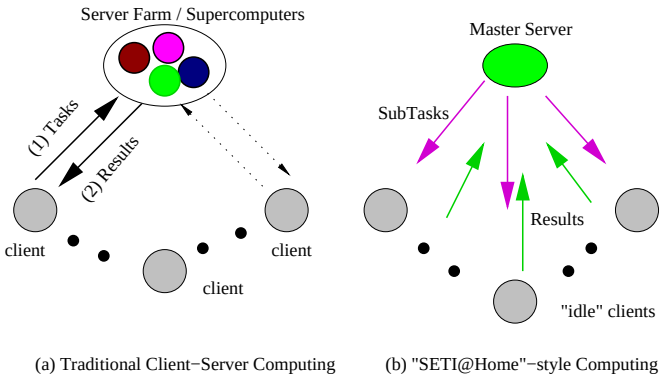


Fig. 2. Resource Sharing Approach: An illustration

the potential to store huge amounts of data (perhaps in a replicated manner) without corresponding investment in storage equipment.

C. Collaborative Computing and Communications

Collaborative computing refers to a usage model of distributed computing which primarily enables users to work together in arriving at a solution to problem. The geographic dispersion of organizations has led to software solutions which enable people to collaborate together. Socially collaboration is person to person activity and the P2P computing model closely mimics this activity. Groove (`www.groove.net`) is one such decentralized collaborative platform which enables small group interaction. The Groove communication framework is essentially P2P with server intervention in the case of network discontinuities like firewalls, proxies and NAT's. Additionally the need to communicate amongst people has also led to significant advances in instant messaging, which is a quintessential P2P application.

III. P2P and Distributed Computing

In the past several years, a number of projects have attempted to provide flexible environments to support arbitrary parallel computing in environments ranging from tightly coupled clusters to a wide-area heterogeneous environments. Somewhere in between lies the notion propagated by projects such as network of workstations (NOW) where the environment is more controlled than the heterogeneity of the Internet but the coupling is not as tight as in the case of clusters. Several prototype systems have been built (including O/S support, languages and tools) over the last 10-15 years. One such early system is Amoeba [17] that supports dynamic migration of programs. Other notable projects are Berkeley NOW [1] and Iowa's Batrun system [20]. References to a number of key papers on the subject may be found in [13], which describes a system for distributing and executing a parallel program on a distributed system. Sharing of other resources such as disk space and main memory has also been explored extensively. For example, reference [8] proposes the use of client main-memory for network file-caching.

The emergence of P2P computing and Internet requires re-examining these applications on a much wider-scale and in heterogeneous environments. The basic idea of these is to provide an abstraction of a parallel machine. The ultimate vision of many of these projects is to essentially provide a single virtual parallel machine by hiding (to the extent desired by the programmer) all the complexities associated with vastly different machines, local operating systems, communication protocols, local resource management, access control and security policies, wide variations in machine and network speeds and loading levels, failures, etc. A few such attempts are the Legion at Univ. of Virginia (now owned by Avaki Corp; `www.avaki.com`), Globe `www.cs.vu.nl/~steen/globe/`) and Globus (`www.globus.org`). Reference [12] provides an overview of large-scale resource sharing using Legion. These projects use a distributed object model, where each object provides a specific service and encodes information about how to work with the local environment where it is deployed (local O/S, access control and security policies) and how to deal with failures, exceptions, delays, etc. Selection of appropriate objects, their behavior and their location is handled by some sort of object request broker service much like CORBA `www.corba.org`) or DCOM (`www.microsoft.com/com/tech/DCOM.asp`). With an all-encompassing WAN O/S, the Internet can be thought of as a "service grid" much like an electrical power-grid, i.e., the desired service can be located and supplied in an uninterrupted way irrespective of any local difficulties such as failures, delays, access problems, etc.

The basic consideration in designing grid O/S's is a complete site autonomy. The designers of Legion (`www.cs.virginia.edu/~legion`) used the following basic design objectives namely, easy to use, seamless computational environment, local autonomy, high performance via parallelism, persistent namespace, security for both users and providers of resources, manage and exploit resource heterogeneity and minimum impact on resource owner's local computation. To ensure security, Legion uses sophisticated access control mechanisms and encryption. Granularity is at the "method level" for access control, thereby giving a fair amount of flexibility for enforcing varying security policies. The Legion system is responsible for finding the appropriate resources, coordinating and executing required processes, and returning results. It abstracts the complexities associated with a distributed system by providing transparent scheduling, data management, fault tolerance, site autonomy, and various security options.

The Globus toolkit (`www.globus.org`) attempts to solve a class of problems called the Grid problem which is defined as "flexible, secure coordinated resource sharing among dynamic collections of individuals and resources". The Globus organization has articulated a clear vision about the architecture which would provide a solution for problem statement. In addition, the Globus toolkit is an open source implementation of the architectural vision. Its componentized architecture enables using the

appropriate blocks for the problem at hand. The latest blueprint for the Globus toolkit, referred to as the Open Grid Services Architecture (OGSA) [22], reflects the convergence of the Grid computing model and the "Web-services" distributed computing model. The Bayanihan project (`www.cag.lcs.mit.edu/bayanihan`) is another attempt to explore and develop the idea of volunteer computing and uses web services as the underlying technology to make it happen.

Recently, a number of companies have announced products to support complex P2P applications in their respective application environments. Some specific examples include MAGI (`www.endeavors.com`), Groove (`www.groove.net`), Sun's JXTA (`www.jxta.org`), and Microsoft's .NET (`www.microsoft.com/net`. MAGI (Micro-Apache Generic Interface) is an extension of Apache project and uses WebDAV (IETF RFC 2518) and SWAP (simple workflow access protocol) to provide sophisticated services for e-business automation. Groove provides a small-scale collaborative computing environment in Microsoft Windows environment. .NET supports XML based web services via SOAP (`www.w3.org/TR/SOAP`) protocol. JXTA provides low-level mechanisms (but not policies) for peers to interact with one another.

IV. A Proposed Classification for Peer-to-Peer Computing

Before delving into a classification of P2P services, we first need to define what "P2P computing" really means. Unfortunately, there is currently no widely accepted definition of the concept, except for the general notion that the processing is spread over a large number of "agents" (or participating nodes) with minimal central control. We can describe P2P computing as a collective computing environment where an "agent" is not only able to act as both a "client" and a "server", but can also interact with other agents in more complex ways to accomplish the task at hand. However, not every agent can be required to have all these capabilities. Some agents (e.g., laptops or hand-held mobile devices) may be unsuitable as service providers, whereas others (e.g., fixed functionality servers) may not need to act as clients. In any case, one important aspect of P2P networks their ad-hoc nature: it should be possible for agents to join and leave P2P communities in a very dynamic fashion.

With P2P computing seen as a generalization of the client-server paradigm, the taxonomy presented here is identified from the perspective of major characteristics of the applications that traditionally have been viewed as "client-server" type, but could be cast to varying degrees in P2P paradigm. The taxonomy is not concerned with the details of the object model used, or with the overall distributed system that supports the application. In this respect, our focus is different from the one in [16], which presents a comprehensive hierarchy for classifying entire distributed computing systems. Also, as stated earlier, the classification proposed here is to be considered tentative and may need to evolve with P2P computing itself. It would be noted that not all applications encompassed by the taxonomy are peer-to-peer; this is unavoidable and actually desirable; the purpose of the taxonomy is to not only classify legitimate P2P applications but also to show how they fit in the larger scheme of things.

A. Classifying Dimensions

In approaching the classification, we identify the following dimensions of the problem, and in each case indicate certain extreme values. *Note that the extreme values are only illustrative; real implementations will typically fall somewhere in-between these extreme values.*

1. Resource (or data) storage: organized or scattered.[1]
2. Resource control: organized or scattered.[2]
3. Resource usage: isolated or collaborative.
4. Global state control: loose or tight.
5. QoS constraints: loose (e.g., non real-time), moderate (e.g., online query/response), or tight (e.g., streaming media).

Here we have used the term "resource" for the entity of interest in order to cover the entire spectrum from hardware resources (e.g., CPU cycles, main memory, disk space, etc.) all the way up to arbitrary objects that are designed to provide some complex service (e.g. an object that can interact with a local database system). We also informally refer to resources as "data", mainly to distinguish them from "control" or "meta-data", which refers to the information needed for locating and accessing the resources.

Resource Storage: The first dimension refers to the way resources are stored. The two extremes here are (a) "organized" storage, where the resources are located in one or more globally known locations (or nodes), and (b) "scattered" storage, where the resources are stored under the control of the requesting agents themselves. Note that by saying "under the control of requesting agents", we allow for the data storage either by the agent nodes themselves or at some auxiliary nodes known only to the agents. The "organized" storage can also be viewed as the traditional server-based storage as well, where the servers are the "globally known locations". Note that the centralized storage of all the data (as in traditional web-servers) is merely a special case of organized storage with only one globally known location containing all the data.

Each resource needs a "handle" so that it can be accessed. If the resources are distributed among several nodes, there is the problem of associating handles with addresses (e.g., node address, subsystem address, etc.) in order to access the resource. In general, the requester may not specify the handle directly, but instead identify the resources of interest by means of some other information (name, properties, etc.), which eventually gets translated into one or more resource instances. In any case, one or more levels of mapping (e.g., mapping resource name into

[1] We avoided the use of more familiar terms like "centralized" and "distributed" since we want to emphasize the control aspect (service-provider controlled vs. user controlled data) as opposed to the number of locations storing the data.

[2] See previous footnote.

handle, and handle into its current address) may be needed in order to locate the resource of interest. This mapping can be considered as "metadata" or "control information", which also needs to be located and resolved. In the current web-context, the URL (uniform resource locator) typically acts both as a resource identifier and the address, which makes the metadata rather trivial to deal with.

Resource Control: In a P2P environment, a well thought out scheme for organizing, maintaining and accessing metadata is crucial for locating all desired resources and retrieving them efficiently. The second dimension in our taxonomy relates to this metadata, and we again identify the two extremes as "organized" and "scattered". As with data, an "organized" storage relates to the metadata being available in one or more globally known locations (or nodes), whereas "scattered" storage refers to metadata also under the control of the agents and hence not known globally. Many systems are organized hierarchically from the control perspective. For example, individual agents or nodes might be grouped into a set of "domains", and these domains further grouped into higher level domains, and so on. In such a structure, every domain at each level must have a "manager" node which is responsible for access to the domain members and for communicating with its parent domain manager. This leads to a tree structure for storage of control information. We consider this an admissible form of "organized" control. Reference [6] defines a wide-area video-conferencing implementation that uses this form of control.

Resource Usage: Once the required resources are located, we need to identify how they are used. In the current web context, each resource is accessed individually using a request-response paradigm. This is, of course, inadequate for implementing sophisticated services. In general, several resources (perhaps located at different nodes) may need to work concurrently in order to accomplish the task at hand. It is expected that before the work "session" starts, the invoking application will provide some parameters or data to set up the context for resource usage, and at the end of the work session, collect the results together. However, during the work session itself, simple applications may use each resource in isolation with no message exchanges or interdependencies. We call this the "isolated" usage mode. The current request-response type of web usage can be considered as a special case of this isolated usage mode. On the other extreme, the agents providing the resources of interest may collaborate via arbitrary multi-party interactions. We call this extreme as "collaborative" usage and it includes multicasting, multi-party synchronization, remote procedure calls, call backs, etc.

Global State Control: The next dimension refers to the global state control requirements of the P2P application. In the popular file-sharing applications like Gnutella/Freenet, there is little need for trying to maintain a global view of the P2P network or to address related issues of consistency, synchronization and local autonomy. In these applications, not only the peers but also the data (files, metadata) they contribute can change willy-nilly and thus there is little in the way of global state to speak of. Other applications, such as hosting of web-pages or collaborative computing would typically require a more coordinated approach. For example, every new content type or copies of existing content type may have to "registered" before being used.[3] As an extreme case, a database hosted in parts over a large ad-hoc network of nodes can be considered as a P2P application, but would need to do an elaborate global state management in order to ensure ACID properties.

An important determinant of state management requirements is the consistency model for the hosted data and metadata. Restrictive models such as one-copy serializability will require considerable bookkeeping and coordination and may be impractical. The current web-infrastructure already shows that weaker consistency constraints may be quite acceptable in return for better performance. The understanding of P2P nature of the service on the part of the users may allow for weaker consistency constraints for many applications than those in traditional client-server implementations. For example, instead of ensuring serializability, it may be enough to ensure that a later transaction from the same user does not access a staler information than what was found before (of course, subject to a "eventual progress" type of constraint). Such weak consistency constraints apply not only to data but also to metadata, but the consequences could be quite different.

Although collaborative applications (classified according to the Resource Usage dimension) may require more elaborate global state control than isolated ones, the two dimensions are independent. For example, one can envision very tight global state control even though the peer action is limited to simple query-response.

QoS Constraints: Timing, and more generally quality of service (QoS) requirements during the resource usage phase (i.e., excluding the initial setup and final windup phases) determine to a large extent whether a P2P solution is feasible in a given environment and how to provide it. In some cases (e.g., solution of large scientific problems), there are really no QoS requirements to speak of and a best effort service works just fine. These define the low end of the "QoS requirements" dimension. The next large class along the QoS dimension are the online query/response type of applications that have "moderate" QoS requirements, e.g., a good response time coupled with a rather low failure/loss rate. The high end of the QoS dimension (or "tight requirements") is occupied by applications that involve transfer of a continuous stream of bits. Real-time audio and video are typical applications that require that the stream starts flowing within a few seconds of the request and have very low delay, delay jitter, packet loss and failure probability.

B. Environmental Attributes

In addition to the basic nature of P2P applications, the environment in which the application operates has a

[3] "Registration" doesn't necessarily imply a centralized control.

tremendous influence on the design and usefulness of the application. For example, implementing a collaborative application in a laboratory LAN environment may be relatively straightforward, but making it usable among a set of home PC's that intermittently connect to public Internet may be plain infeasible. A characterization of the operating environment involves many distinct attributes, some of which may be more important than others depending on the application requirements. The major attributes in this regard are listed below:

1. Network latency: Ranges from uniformly low (e.g., for a high-speed LAN) to highly variable (e.g., for general WAN).
2. Security concerns: Ranges from low (e.g., corporate intranet) to high (e.g., public WAN).
3. Scope of failures: Ranges from occasional isolated failures (e.g., a laboratory network of workstations) to frequent failures, possibly including massive failures that result in network partitions.
4. Connectivity: Ranges from always-on (e.g., nodes in a business LAN) to occasional-on (e.g., laptops and other mobile devices).
5. Heterogeneity: Ranges from complete homogeneity to complete heterogeneity (in hardware, O/S, protocol stack, services and application interfaces).
6. Addressability: Ranges from easy (e.g., all nodes directly accessible and have DNS entries) to very difficult (e.g., nodes behind a different NATs/fire-walls and no DNS entries).

The distinction between application dimensions and envionmental attributes may appear a bit arbitrary but was motivated by inherent application characteristics vs. techniques to make the application robust. For example, coping with intermittent connectivity requires certain techniques, which could be applied to any application domain. However, the security dimension does appear to straddle application/environment boundary. One could attempt to classify applications according to their security needs, which would make it a dimension. On the other hand, one could also focus on the level of difficulty in providing security or on the kinds of security threats that must be dealt with, which would make security an environmental attribute. We take the latter view since a single application (e.g., e-commerce) may require a range of security measures depending on the sensitivity of the information being manipulated. Also, the provision of various levels of security involves the same basic issues including authentication, integrity, nonrepudiation and access control.

For the purposes of application classification, it is useful to identify the two extremes of these attributes as "friendly" and "hostile". A friendly environment is typically synonymous with a LAN, but may extend to WAN as well depending on the application requirements. For example, a wide-area corporate intranet that uses VPN (virtual private networks) to provide security and SLAs (service level agreements) to provide assured bandwidth for inter-site communications may be almost as friendly as a LAN environment. On the other hand, harnessing the idle computing power of home-PC's necessarily requires dealing with a very hostile environment. It is possible to consider the friendliness of the environment as yet another dimension in our taxonomy; however, because of the multifaceted nature of the "friendliness", such a classification may not be very revealing.

C. Classification of Emerging Services

Let the ordered 5-tuple $(x_i, i = 1..5)$ represent a point in the taxonomy space with $x_i =$"-" indicating a don't care. In defining the dimensions above, we only listed 2 or 3 values in each case mainly to convey some idea about the range of each dimension. It is understood that real applications could fall anywhere within the indicated ranges along each dimension. Nevertheless, it is useful to place known applications in this rather limited, discretized space for illustrative purposes. Figure 3 shows the first 3 dimensions pictorially and indicates some applications and application environments in each category. For ease in depiction, we have represented both first and second dimensions along the x-axis by using the combinations (org, org), (scat, org) (org, scat), and (scat, scat) where "org" stands for organized and "scat" for scattered. The boxes corresponding to the combination (org, scat) does not have any entries as we do not know of any such applications. However, this represents an interesting class, where the data is hosted on a set of globally known nodes, but access to the data scattered among the agents. The reason to do this may be to provide tight control over the data access. The hosted data may contain parts owned by individual agents, or different data views may be accessible to different agents.

We acknowledge here that some applications (and particularly the application environments like Legion, Globe, Magi, JXTA, etc.) may cover several boxes in Fig. 3, but are shown in only one place. Thus, *the placement shown here should be considered purely illustrative rather than judgmental about the relative merits of the products mentioned.*

The current web-browsing service can be represented by (organized, organized, isolated, loose, moderate) and the e-commerce service is represented by (organized, organized, collaborative, tight, moderate).[4] In the latter case, the resources of interest are located in the web-server, LDAP servers, search server, database server, personalization server, etc. and these might interact in complex ways (although normally most interactions are handled either by the web-server or by the middleware).

The Napster computing model can be classified as (scattered, organized, isolated, —, loose), and Gnutella/Freenet as (scattered, scattered, isolated, —, loose). The distributed hardware resource sharing models (e.g., sharing of spare CPU cycles and disk capacity) can also be described as (scattered, organized, isolated, —, loose) since the "client" has complete knowledge of all the agents that currently host the resource and either there is no inter-

[4]E-commerce typically requires more stringent QoS constraints than web-browsing, but as stated above, we are characterizing applications here only in terms of stated extreme values.

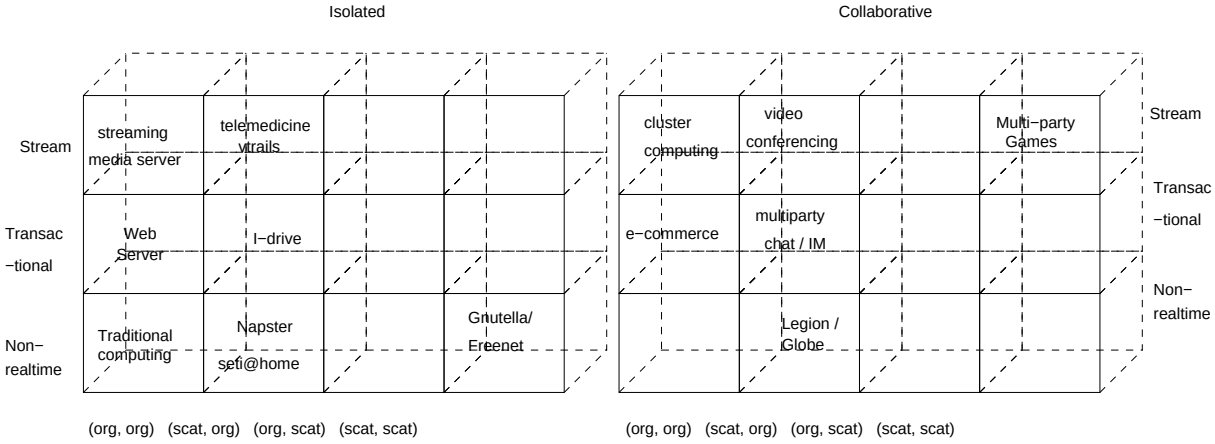


Fig. 3. A pictorial representation of the proposed taxonomy (Along the x-axis, first dimension is data, and second is control)

action between the resource usage at various nodes (as in SETI@HOME model) or only one instance is used at a time (as in a i-drive like model where a client can store some or all of its files at multiple places).

Past work on network of workstations has attempted to provide interactive resource usage in a friendly environment using both organized and scattered control. The recent work on WAN operating systems has addressed provision of interactive services in a hostile environment. These can be described as (scattered, organized, collaborative, tight, moderate). Here we indicate the global state model as "tight" since generally all instances of an object are closely tracked. Legion provides an object based service model such that objects can be replicated and located arbitrarily transparently. It achieves this by a three level naming scheme which maps human readable names to Legion Object Identifiers (Globally unique in time and space) which map at run-time to address and port of an active instance of the object. Globe also provides a similar separation between object name and address. These WAN operating systems stress flexible resource usage in a distributed environment and thus not geared towards lightweight computation or tight real-time constraints.

In the current Internet, providing services that require tight real-time constraints has been quite a challenge. For certain services such as delivery of streaming audio/video, the standard solution is to download the "files" in advance to "edge servers" that are closer to the points of consumption. Such a scheme provides better timing properties during the play by introducing a startup delay at the beginning. Recently, a P2P approach for handling this has been proposed (See `www.vtrails.com/product.htm`). Here, users with broadband connections that hit the original web-site cache the webcast locally so that others could be served using this copy. This can be classified as (scattered, organized, isolated, loose, tight) operating in a hostile environment. Note that in this application, there is only one way delivery of information and there is no interaction between various recipients.

Another P2P application indicated in Fig. 3 is so called Virtual Private Web (VPW) which allows a private group of people (e.g., family) to share content (pictures, videos, etc.) without outside exposure or explicit searches of what is located where. This application can be classified as (scattered, scattered, isolated, loose, loose) and must be designed for a hostile environment.

More general multiparty interaction is an important emerging category of applications. For example, multiparty chat and instant messaging (IM) are already well entrenched, except that they are currently implemented using a client-server paradigm. P2P computing is ideally suited for these applications. Assuming a primarily textual interaction, these applications can be classified as (scattered, organized, collaborative, tight, moderate). If a significant use of audio is allowed, the last attribute would be "stream". Multiparty games over the Internet are also a rapidly expanding application and is ideally suited for P2P paradigm [11], [7]. Assuming a substantial use of live video and audio, these would be classified as (scattered, scattered, collaborative, tight, tight).

In video conferencing implementations, the control may be either centralized (i.e., performed at a manager node) [6] or distributed (performed cooperatively by all participants) [2]. However, the participants do not discover one-another via any sort of search, instead, the information about the ongoing video conference is either known a priori, or can be found through some central place. Thus, video conferencing can be classified as (scattered, organized, collaborative, tight, tight) in our taxonomy.

Telemedicine is a vast emerging area that facilitates access to patient records from all sources, remote diagnosis based on such records, cooperative diagnosis by involving a number of remotely located physicians, remote checkup and diagnosis of live patients, and even remote surgery using virtual reality (VR) displays perhaps involving multiple surgeons. References [4], [3] discuss some applications of patient record pooling and cooperative diagnosis which could be recast in P2P paradigm. In case of live checkups and diagnosis, the simplest case would involve a point to point channel between a physician and a patient capa-

ble of transmitting streaming audio (e.g., heart-beat) and video data. This would be considered as an "isolated" class of applications in our taxonomy. In more sophisticated cases, multiple physicians may be involved simultaneously, thereby resulting in a "collaborative" type of application. In either case, it is expected that the location of the physician(s) and patient(s) would be based on some a-priori information, and thus this application can also be considered as (scattered, organized). We have also considered this as one with tight QoS constraints by assuming that that the interaction between physicians involves real-time transmission of audio and video. In such cases, the timing, reliability, and accuracy requirements of this application are so stringent that this application cannot be reasonably supported by the current Internet and is considered as one of the drivers for the Internet2 project (`www.Internet2.edu`).

V. Summary and Research Issues

In this article, we proposed a taxonomy to classify the evolving peer-to-peer (P2P) computing paradigm and its associated applications. In the taxonomy, we classified P2P applications based on five dimensions: resource (data) location, control (metadata) location, resource usage, global state control, and QoS constraints. We also discussed environmental attributes (including friendliness, security and connectivity) to address implementation issues for P2P applications. The examination of P2P landscape according to the proposed taxonomy points to regions that are not well explored (e.g., empty boxes in Fig. 3) and this could lead to new P2P applications and application platform capabilities. The taxonomy also points to a number of interesting research issues that need to be explored in order to achieve the potential of P2P computing.

Briefly, some of the important research issues include (a) devising intelligent and efficient mechanisms for propagating queries and response through the network, (b) data propagation through the network to enhance search efficiency without attendant problems of instabilities, oscillations, or unnecessary overhead, (c) devising new consistency models, synchronization mechanisms and other support needed for scalable management of global peer state, (d) coping with network address translation and fire-walls in providing interaction between peers, (e) coping with intermittent connectivity and presence, (f) lightweight and nimble migration protocols for quickly relinquishing resources from a P2P participant when the machine owner requires those resources exclusively, (g) ensuring robustness, security, authentication and access control in a very hostile environment, (h) user-level (instead of device-level) security and authentication, and (i) performance characterization of P2P computing environment to enable comparative evaluation of many design choices.

A more detailed discussion on these issues can be found in the detailed version of the paper available at `kkant.ccwebhost.com/download.html`. Reference [14] presents a random graph model and shows some preliminary results to address issue (i). In addition to these, many of the traditional distributed systems issues (e.g., naming, binding, synchronization, fault tolerance, state management, discovery, security, authentication, auto-configuration, task-migration, etc.) need to be revisited since we now want solutions that must scale to millions of nodes spread over networks with unknown characteristics.

References

[1] T.E. Anderson, et. al., "A case for NOW (Network of workstations)", IEEE Micro, vol 15, no 1, Feb 1995, pp54-64.

[2] I. Beier and H. Koenig, "GCSVA – A multiparty video conferencing system with distributed group and QoS management", Proc. of 7th Intl. conference on computer communications and networks, 1998, pp594-598.

[3] R.D. Bella, et. al., "An inter/intranet multimedia service for telemedicine", Proc. of 23rd Euromicro conf, 1997, pp379-386.

[4] T. Bui and S. Sankaran, "Group Decision and Negotiation in Telemedicine: An application of Intelligent mobile agents as nonhuman teleworkers", Proc. of 30th Hawaii Intl. conference on system sciences, Vol 4, 1997, pp120-129.

[5] I. Clarke, "A Distributed Decentralized Information Storage and Retrieval system." M.S. Thesis, Division of Informatics, Univ of Edinburgh, UK, 1999.

[6] S.T. Chanson, A. Hui and E. Siu, "OCTOPUS – A scalable global multiparty video conferencing system", Proc. of 8th Intel. conference on computer communications and networks, 1999, pp97-102.

[7] R. Corchuelo, D. Ruiz, M. Toro, and A. Ruiz, "Implementing multiparty interactions on a network computer", Proc. of 25th Euromicro conference, Vol 2, 1999, pp458-465.

[8] M. Dahlin, et al., "Cooperative file caching: Using remote client memory to improve file system performance", Proc. of first conference on O/S/ design and implementation, Nov 1994.

[9] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[10] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", HotOS VIII, Schoss Elmau, Germany, May 2001.

[11] L. Gautier, C. Diot, and J. Kurose, "End to end transmission control mechanisms for multiparty interactive applications on the Internet", Proc. of IEEE INFOCOM, 1999, Vol 3, pp1470-1479.

[12] A. Grimshaw, et al., "Wide-Area Computing: Resource sharing on a large scale", IEEE Computer, May 1999, pp1-9.

[13] R.K. Joshi and D.J. Ram, "Anonymous Remote Computing: A paradigm for parallel programming on interconnected workstations", IEEE Trans on software engineering, Vol 25, No 1, Jan 1999, pp75-90.

[14] K. Kant and R. Iyer, "A Performance Model for Peer-to-Peer File Sharing Services", `kkant.ccwebhost.com/download.html`.

[15] J. Kubiatowicz, D. Bindel et al., "OceanStore: An architecture for global-scale persistent storage," Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000) (Boston, MA, November 2000), pp. 190201.

[16] B.E. Martin, C.H. Pedersen, and J.B. Roberts, "An object based taxonomy for distributed computing systems", IEEE Computer, Aug 1991, pp17-27.

[17] S.J. Mullender, G. Rossum, et. al., "Amoeba: A distributed operating system for the 1990s", IEEE Computer, vol 23, no 5, pp44-53, May 1990.

[18] S. Ratnasamy, P. Francis et al., "A Scalable Content-Addressable Network," ACM SIGCOMM'2001, San Diego, Aug 2001.

[19] I. Stoica, R. Morris, et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," ACM SIGCOMM'2001, San Diego, Aug 2001.

[20] F. Tandiary, et. al., "Batrun: Utilizing idle workstations for large-scale computing", IEEE parallel and distributed technology, Summer 1996, pp41-49.

[21] Roy D Williams et. al., "Parallel Computing Works" Morgan Kaufmann Publishers.

[22] Ian Foster, Carl Kesselman, Jeffrey M Nick and Steven Tuecke, http://www.globus.org/research/papers/ogsa.pdf