UBCA: A UTILITY BASED CLUSTERING ARCHITECTURE FOR PEER TO PEER NETWORKS

by

BRENT JASON LAGESSE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2006

ACKNOWLEDGEMENTS

I would like to thank Mohan Kumar for all of his help in reviewing and working with me on my thesis and other research works. I would also like to thank Matthew Wright and Mike O'Dell for their assistance in my education and research along with their willingness to serve on my thesis committee. I would like to thank my parents for their encouragement in all that I've done. I also want to thank Jonathan Holm for first piquing my interest in computer science and Patrick Wagstrom for introducing me to pervasive computing. Finally, I'd like to thank all the members of PICO who have helped me with my work by asking me questions and giving me feedback.

April 14, 2006

ABSTRACT

UBCA: A UTILITY BASED CLUSTERING ARCHITECTURE

FOR PEER TO PEER NETWORKS

Publication No. _____

Brent Jason Lagesse, M.S.

The University of Texas at Arlington, 2006

Supervising Professor: Dr. Mohan Kumar

Use of the Peer-to-Peer (P2P) architecture has recently spread in popularity. File sharing and ad hoc networks have contributed to the architecture's usage. P2P generates new challenges in scalability, fairness, and quality of service. Current solutions tend to fall into two main areas: incentives and system design. Incentive-based approaches appeal to the self-interested nature of peers by requiring service to the system in order to access resources. System design includes distributed hash tables and graph-theoretical based designs which have seen some success, but also result in new problems. We introduce a Utility-Based Clustering Architecture, UBCA, designed to address scalability, fairness, quality of service, and load distribution through the use of implicit incentives. The UBCA runs on peers and clusters during the execution based on mutual utility gained as a

result of the grouping. Simulation of the UBCA shows improved bandwidth and latency per access and reduced overhead costs.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

CHAPTER 1

INTRODUCTION

Peer to Peer (P2P) computing is now commonly used in software development [16]. Since the architecture has direct implications in dynamic and decentralized environments, it has become part of the foundation of mobile and distributed computing. P2P computing provides advantages in resource replication, a decentralized nature, improved availability, and flexibility for dynamic systems [2]. Despite all of its advantages, the P2P architecture introduces several new problems that must be addressed.[1,2,6,22] High overhead costs are introduced to systems based on P2P since resource locations are not known, but rather must be discovered. As a result, excess queries and broadcasts are sent that do not result in the access of a resource. These extra communication messages can clutter a network and reduce performance. In addition to increased overhead, P2P-based systems scale poorly. P2P-based systems also can be dominated my non-contributors which only consume resources and provide nothing.

We examine current solutions to problems in P2P-based systems and introduce a utility-based solution which addresses problems of increased overhead, fairness, and load distribution. This solution utilizes techniques from distributed computing and game theory in order to reduce overhead costs and select the best resources to access. Each peer is abstracted into an agent that consumes and provides a set of resources with certain quality and quantity attributes, such as the pages per minute on a printer or the number of

jazz music files in a music archive. By performing this abstraction, peers can be clustered together to satisfy each other's consumptions best so that there is less need for a peer to request help from outside the group.

In order to provide a concrete example, we can examine the interaction of players in sports, particularly basketball. In basketball, each player possesses certain skills which contribute to the team; however each player is also dependent on the provisions of other players in order to operate at or near their maximum potential. For example, a *guard* is a player who is quick, a good ball-handler, a good passer, and a good shooter; however, the point guard cannot do much without the ball, so he or she needs somebody (including potentially himself or herself) to provide him or her with the ball. This can be done by players providing such actions as rebounds, steals, and passes. A *center* is typically a taller player who plays near the goal and makes short shots well and grabs rebounds due to height. For this case, the center provides close shots and rebounds, but needs to consume the shots of other players and passes from them. Evidently, a team will not do as well with 5 great players with a minimal set of skills as they will with 5 players whose skills complement each other. This analogy can be taken even further to realize that 5 players on a team is an artificial limit placed on leagues by rule makers. Due to the cost of having to satisfy the needs of all players, a team that is formed based on compositional need may not increase their performance by continually adding players. Each player that joins the team adds an additional overhead cost, so at some point, the addition of another player will not benefit of the team.

As a result of simulating the utility-based clustering architecture (UBCA), we have

seen positive results in terms of overhead messages, bandwidth and latency in accessing resources, and classification of peers. The primary use of the UBCA is expected in pervasive computing applications. In pervasive computing environments, many agents are brought together and communicate in order to perform tasks to make the lives of the users better.

In Figure 1, an example of peer clustering is shown. This example demonstrates three groups of peers which have mutually derived utility from each other existing in a system with peers that either cannot currently benefit groups or themselves. These peers act as though there were in a normal version of the P2P system and are not affected by the groups, other than the likelihood that they will have less requests sent to them. The peers handle the significant load of their interactions within the group, so they do not need to interact with other peers as often.

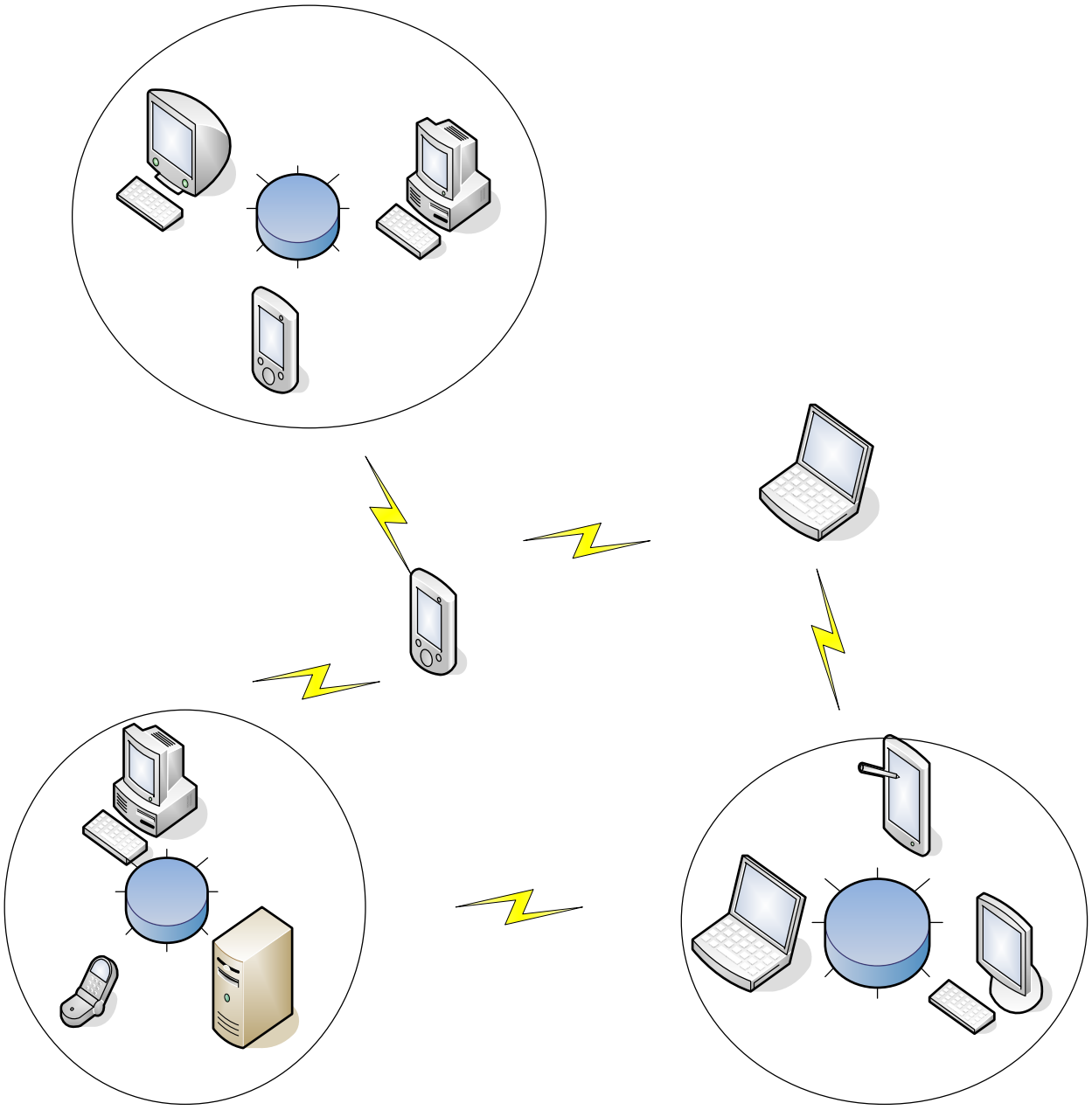

Figure 1: An example of grouped and non-grouped peers interacting in a P2P network

CHAPTER 2

BACKGROUND

2.1 Pervasive Computing

Computational hardware continues to be produced in smaller forms and become more abundant in every day life as technological and production techniques improve. In fact, we are beginning to reach a point where computing has become ubiquitous. As computers reach a point of ubiquity, they also become more intertwined in networking and information sharing. Ideally the conglomeration of computers should be completely transparent from the perspective of a human; however, in most cases we are far from achieving this goal.

In 1991, Mark Weiser noted that "*the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*" [32] This concept has become the core of Pervasive Computing. While it includes work done in distributed and mobile computing, pervasive computing goes further to integrate itself invisibly into all things. Satyanarayanan identifies four new research thrusts for pervasive computing. These are effective use of smart spaces, invisibility, localized scalability, and masking uneven conditioning.

Smart spaces bring together the physical infrastructure of an area with the computational infrastructure. They are an excellent environment in which to research pervasive computing. Many research efforts have begun in this area, including MavHome at University of Texas Arlington, Gaia at University of Illinois Urbana-Champaign, and Vigil at

University of Maryland Baltimore-County. These research efforts address and raise questions in service provision, trust, communication and format protocols, and resource discovery; however, all of these issues are manifestations of the other three research thrusts identified by Satyanarayanan [25].

MavHome [9] is a smart space with significant focus in pro-active automation. Work in MavHome focuses on utilizing artificial intelligence techniques in order to learn inhabitant patterns. The goal of this work is to create a home that acts as a rational agent and maximize comfort while minimizing operation and interaction costs through the ability to predict mobility patterns and device usage. The MavHome architecture is rooted in the house agent, which communicates with Rooms/Robots and then further through them to Appliances/Robots. The house agent is divided into four layers: Decision, Information, Communication, and Physical. The decision layer is in charge of selecting actions to take based on information it receives from other layers. The information layer works to collect, maintain, and generate information useful for decision making. The communication layer exists to route communications between the users and the house and the house and external resources. Finally, the physical layer is the actual hardware devices in the house. Together, these layers work to create a proactive smart environment.

The Gaia project [24] at University of Illinois approaches the smart space concept by bringing the functionality of an operating system to physical spaces. Through extending the operating system to include concepts of context and location awareness, mobile computing devices, and actuators, they are seeking to be able to build generic applications to function without the knowledge of the underlying hardware. Using middleware, the Gaia project

builds an application layer on top of their application framework and Quality of Service measures, which sit on top of the Gaia kernel. These measures work to provide fault tolerance and invisibility to the smart space environment.

Vigil [13] at University of Maryland Baltimore County is a third generation pervasive computing infrastructure designed to utilize role-based services. The work is agent based where software agents provide services which are registered with service managers and are accessible to client agents (either humans or computers) based on the needs of their role. These roles are wrapped in digitally signed role-certificates and are utilized within the Rei framework which is an engine using rights, prohibitions, obligations, and dispensations to create logic-inference rules. Furthermore, Vigil can be extended as has been done with UMBC's EasyMeeting [7] which provides context awareness and privacy protection. With this extension, Vigil has been able to pro-actively utilize context to disappear into the background while providing a basic meeting smart space

In developing pervasive computing applications and infrastructures such as these, one of the most prominent architectures used as been the Peer to Peer computing. Due to the nature of this architecture, in which agents both provide and consume resources rather than all looking towards a centralized source, systems can become robust, but it also introduces other potential problems such as trust, corruption, discovery, and traffic issues. Because of and despite these characteristics of peer to per computing, many projects and research thrusts in pervasive computing utilize the architecture to approach their goals in a pervasive computing system, including transparency, reliability, and content/resource distribution.

2.2 Peer-to-Peer Architecture

Peer-to-Peer systems break from the typical client/server architecture which specifies that one node must provide a service while others node must contact the server node in order to receive the service. In Peer-to-Peer systems, this distinction disappears. Each node (or peer) ideally has the some capabilities and services and acts as both a client and a server. Peer-to-Peer broadly defines a range of computing systems. From a purist standpoint, peer-to-peer systems are created where each peer has a completely equal level (but not the same set) of tasks and abilities, which result in a purely decentralized system. However, we open the label of peer-to-peer to many other systems that utilize super-nodes such as Kazaa, systems that have decentralized distribution of tasks, but a centralized searching mechanism such as Napster, and systems such as Gnutella, which is close to a pure peer-to-peer system, but does require bootstrapping. According to Androutsellis-Theotokis and Spinellis:

> *Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.* [2]

Androutsellis-Theotokis and Spinellis state that there are three types of structures and three types of centralization. These are Unstructured, Structured Infrastructures, and Structured Systems for structures and Hybrid, Partial, and None for centralization. Unstructured systems content and services are distributed with no particular method. This requires a

search to locate the desired content or service. This causes problems in with scalability and availability, but those can be somewhat mitigated through the use of advanced searching techniques and replication. The main advantage of these types of systems is that they are excellent at supporting very transient populations. Structured networks provide excellent scalability when a node needs to perform non-ambiguous searches. The main disadvantage of the structured Peer-to-Peer network is that the structure is extremely difficult to maintain due to the highly transient nature of Peer-to-Peer systems.

As a very pure and widely used peer to peer system, the Gnutella architecture [21] will be a main focus in future discussions. The basics of Gnutella employ a query flooding protocol which includes five main messages:

- Ping – Discovers hosts
- Pong – Replies to a ping
- Query – Searches for a File
- Query Hit -Replies to a Query
- Push – Request to Download for firewalled peer

Gnutella works by bootstrapping peers into the network. This method can differ depending on the implementation of the Gnutella protocol, but generally involves connecting to some initial peer or set of peers and requesting a list of available peers to connect to. Whenever a peer requires access to a file, it then sends a query to each connected peer (usually a small number on the order of a single digit). Each peer that receives a query then does two things. First, the peer checks to see if it can satisfy that query. If this is the case, the peer will then return that result. This is done in two forms. If the peer is not firewalled, it returns the result

directly to the original requesting peer; however, if it is firewalled, the peer will return the resulting query hit along the same route of peers that it received the query. The second action a peer takes is forward the request to its connected peers (if a peer has already forwarded a unique request, it will drop the request in most systems to avoid eternal looping which would bring the system to overload). Theoretically speaking, all members of the system will eventually receive the query. Unfortunately, Gnutella does not scale since the regeneration of each requests creates significant amounts of traffic. Peers can then utilize multiple copies of the file that has been queried in order to download pieces from several peers at once, thus reducing load on any particular peer and utilizing available bandwidth better.

Peer-to-Peer computing does an excellent job of adapting, providing fault tolerance, handling transient populations of peers and the instability caused by transient populations [26]; however, there is significant lack of success in defining the infrastructure to promote common protocols and interoperability. One of the key areas of focus in peer to peer computing is the need for improved performance. Generally speaking, peers can be modeled as selfish agents, seeking to improve their own utility while not caring about the system as a whole. While this is not always the case, it needs to be accounted for. As a result, many mechanisms have been designed to address freeloading peers and to promote and reward a more communal perspective of distributed computing in peer to peer systems.

Peer to Peer systems are utilized extensively in pervasive computing. Since the goals of pervasive computing include transparency and scalability, the advantages of the peer to peer architecture should become evident. Furthermore, pervasive environments are usually

considered a conglomeration of dynamic agents that act together for the benefit of the users (who can be considered agents in themselves). This makes modeling systems as groups of peers an obvious choice since the architecture so closely resembles the physical environment. Two examples of the use of P2P in pervasive computing are shown in the PICO (Pervasive Information Community Organization) project at the University of Texas at Arlington and Scarlet, a middleware for context-awareness, at Illinois Institute of Technology.

PICO [17] uses Camileuns (devices) and Delegent (software agents) as the basis of its architecture. A Camilieun is generically defined by its identifier, its set of characteristics (ie, Operating System, CPU, Memory, etc.) and its set of functionalities (ie, communications, data sensors, etc.). A Delegent is generically defined by its identifier and its functional description. By allowing Delegents to live on devices and provide functionality, PICO creates communities, which are the set of Delegents in the community, the set of goals of the community, and the set of community characteristics (ie, the community manager, resources need by the community, etc.). The PICO Delegents can be considered peers that provide and consume services. PICO currently relies on JXTA to provide communications between peers.

Scarlet [31] is a pervasive middleware project for context-aware computing with goals of cross-platform compatibility, scalability, modularity, and extensibility. The architecture for Scarlet utilizes the peer to peer architecture whenever possible in order to improve scalability. Scarlet uses pure peer-to-peer communications in all aspects of communication except in service discovery, in which handled in a very similar form to DNS. The main communications occur between Context Providers (ie, sensors) and Context

Consumers (ie, an agent) in the form of SOAP messages. The performance of the system is largely based on the performance of these communications.

Since Scarlet relies so heavily on peer-to-peer communications, the need for optimizing these communications is essential to providing a seamless context provision. Since peer-to-peer communications assists in offloading the work of routing and satisfying requests from a single point and distributes it to the peers, the system becomes scalable and usable. Thus the system becomes useful as an aspect of an effective smart space.

While there are many advantages to using the Peer to Peer architecture, there are also problems within it. To begin, agents must discover who has resources that they need. This process requires communications which can bog down a network's resources if enough agents are searching at once. Furthermore, its peer to peer architecture relies on the fact that each peer both shares and consumes. When too many peers fail to share and become consumers, the architecture begins to resemble a client server architecture. Peers that only consume are referred to as free loaders.

There are many conceivable ways to improve efficiency, and thus performance, of a peer to peer system. One method is to offer incentives to promote sharing. Another is to optimize communications such that less communications are wasted and yield no return. Another method, especially utilized in content distribution systems, is to devise mechanisms to promote distribution of resources in a more homogeneous manner, so that the load of satisfying requests would be distributed more evenly and that the hop distance required to find a resource would be decreased.

Buragohain, Agrawal, and Suri begin their paper, “A Game Theoretic Framework for

Incentives in P2P Systems" [5], by stating research that shows that 50% of users in Napster and Gnutella systems utilize "short sessions" which last less than an hour. Additionally, both systems have high levels of free loaders that purely consume and provide no contribution to the systems. It was stated that approximately 25% of Gnutella users fall into this category. What these two facts result in is a significantly negative turn in performance for the system. Short sessions result in large periods of time where resources become unavailable, thus grouping load rather than spreading it out. Furthermore, it is stated that as more users become free loaders, the Peer-to-Peer system loses its nature and begins to resemble more of client/server architecture. The paper then goes on to explain the differences between a monetary payment incentive system (a payment is made to consume services and the entity is paid when its services are consumed) and a differentiated services incentive system (where nodes that contribute more receive a higher level of quality of service). The authors introduce their idea of providing a formal model for incentives through differentiated service utilizing Nash equilibrium (the point at which no player can improve his utility my changing only his strategy). The authors begin by establishing that each node in the system is a "rational, strategic play, which wants to maximize his utility by participating in the P2P system." [5] The authors introduce the concept of Nash equilibrium, but then are quick to admit that its computation is non-trivial since there is no known polynomial time algorithm for a general N person game. The authors describe the ideal Peer-to-Peer system as one that is a cooperative game where all players work together for the benefit of both themselves and the system as a whole; however, the authors then point out that this is rarely the case, so instead they classify a general Peer-to-Peer system as a non-cooperative game. Next the

authors establish the basics of their system. In this, metrics are defined and they establish a dimensionless contribution, *d*, which is the user's contribution divided by a base level contribution defined by the system architect. The goal of their system is to have every user contribute at least a *d* value of 1. The authors then define a benefit matrix which details the amount that a node will benefit from the contribution of $d = 1$ from each node. Then given that value, each node that requests from the user will be accepted with a probability P(d) such that P is any reasonable probability function. The function does not affect the qualitative results of the system. They then define a function U(i) which describes the total utility node i will experience from joining the system. The authors then apply Nash equilibrium equations using an iterative learning model to this system based on the utility function for each node. Their results proved positive, showing an increase in average contribution and a decrease in free loading peers.

2.3 Group Management

Group management is a method by which a system attempts to improve efficiency by supplying as much of the full benefit of the entire system, but in a much smaller capacity. Groups are advantageous in that they approximate a full system at a significantly smaller overhead cost. Therefore, scalability, robustness, and efficiency can all be improved at a small trade-off cost in reliability and replication. Furthermore, these schemes often seek to penalize freeloaders and promote fairness within the system. Group formation can occur many different ways and based on many different factors. Grouping also creates a natural avenue through which a system can be modeled as services rather than just individual

components.

Koo, et al present work on Group Management in utilizing peer to peer systems for content distribution [15]. In this case, it is argued that the set of peers in a group should have the most disjoint set of content (provisions). They solve the problem by utilizing integer programming; however noting that there is a large overhead computational cost, so there is ongoing effort to produce a low cost approximation. Furthermore, Koo utilizes upload to download ratio and "price" in the form of pieces of content as incentive for peers to participate. This penalizes freeloaders, assists in selecting partners, promotes fairness, and helps the system to adapt to a potentially non-cooperative environment.

Ogston, et al have worked with decentralized clustering in large (2,500-160,000 agents) multi-agent systems. In their work, they seek to define high-quality (most well matched) clusters of agents based on their attributes. Thus, each agent is defined as a set of data items with a small number of links to other agents. Each agent also has a number of objectives based on their characteristic attributes. Each agent is provided two methods to improve their clusters and achieve their objectives. The first is to combine their sets of links with other agents to increase the magnitude of their neighborhood. The second is the ability to break weaker links in favor of stronger matches. Additionally, a limit on the size of groups is also enforced.

After the bootstrapping phase where the agents get their initial set of links, the agents go through four phases:

1. Connecting – Using some rule, clusters choose some subset of their matched links to become connected links.

2. Mixing – Using a random permutation, each cluster mixes its unmatched adjacent links
3. Matching – Using a turn-dependent matching probability, each agent tests its unmatched links and makes matched links.
4. Breaking - Using a breaking probability, clusters downgrade some matched links to unmatched

While they do not explore performance metrics, they do show relatively decent scalability in matching quality (99% for small systems and 95% for large systems) which significantly outperforms k-means clustering in each case.

2.4 Related Techniques

While we have addressed work primarily in the area of Peer to Peer Computing, it is not unreasonable to take into consideration work in other related fields which have potential application to peer to peer computing.

Since the architecture of peer to peer computing is highly related to parallel processing and can be utilized in a similar fashion, this makes sense as our first consideration. As expected, due to communication overhead, parallel processing reaches points of diminished returns. To overcome this, Weissman and Grimshaw [33] utilize a heuristic which is guided by a run-time cost estimate based on the result from their callback framework which deals with computational costs (number of PDU's, Computation Complexity, and architecture costs), and communication costs (topology, communication complexity, and overlap in the communication and computation phases). A total cost is then

computed with the heuristics which explore configurations of the best clusters first, but do not guarantee an optimal solution since exploring all configurations would cause exponential growth in cost of processors and clusters.

CHAPTER 3

UBCA DESIGN

3.1 Goals

At a high level, the goal of my work is to form dynamic communities of peers. In designing UBCA, three main goals stood out.

- Improved scalability
- Improved performance
- Incentive to share resources

The accomplishment of these goals will create a more efficient and useful architecture for designing applications that rely on P2P.

3.1.1 Scalability

Scalability is the ability of a system to grow without a drop in performance significant enough to inhibit the use of the system. Scalability exists in different forms, for instance, size, geography, and administration. The design will primarily address size and geographic scalability.

Size scalability, a metric in which Gnutella systems fail [22], is of first concern. As new peers enter the system, the performance of the system should not degrade. While this metric is not easily definable by itself, it is easy to observe as more peers are injected into the system by examining the decrease in performance as measured by latency and bandwidth.

While some performance degradation is tolerable, the goal is to prevent the system from degrading severely into an unusable state.

Since Gnutella relies on message forwarding to connected peers in order to discover resources, the number of requests can potentially increase exponentially [22]. Accounting for the simplest set of TCP and IP headers, Ritter shows a query packet will be 83 bytes. While 83 bytes may seem inconsequential, there is potential of producing gigabytes of traffic by a single request if there are enough peers that need to be reached. Likewise, and even more bandwidth-consuming, the response messages are exponential and even more detrimental to the network. Furthermore, the sheer number of requests that must be handled in the network will bog down routers and increase latency and affect response time. One solution to deter this type of system overload is to set a maximum hop count on query messages [16]. The downside of this approach is that it inhibits a peer's ability to find resources outside of the range of the maximum hop count.

When the UBCA is applied to ad-hoc peer to peer networks, it is highly applicable to geographic scalability. While the focus is in an Internet style network, it is evident that the UBCA can be applied to an ad-hoc network. In the ad-hoc network, the location of the peer becomes an issue in more ways than in an Internet style network (which typically only reflects latency and possibly bandwidth to a lesser extent in geographic scaling). In the ad-hoc network, another factor, connectivity, is introduced, as the paths to a resource are potentially mobile, present a higher chance of exhibiting a dynamic nature.

In order to demonstrate that the UBCA improves the scalability of a system, the simulations need to show an improvement in the overhead traffic generated in order for the

system implemented to access the resources needed. The three primary contributors to overhead traffic are queries, queries forwarded, and query hits. While the specific application of the UBCA to mobile ad-hoc networks is not simulated in this work, it is expected that UBCA implementations on mobile ad-hoc networks will further improve their scalability.

3.1.2 Performance

System performance is measured in two ways. The first is the performance of the system as a whole, and the second is application specific performance.

The first consideration to take into account is bandwidth. Obviously, this is more important in some applications than in others; however, it is a necessary consideration for the general case and can be "weighted out" if the case does not need to require bandwidth capabilities.

Latency is time required for a trivially sized message to propagate from one peer to another and back. This has the most effect on transactions that occur often and are not bandwidth intensive. This is not only essential for response time, but also for resource utilization.

Performance will be tested based on overall system performance. In order to show that performance is improved, the simulations will need to show improvements in bandwidth per access and latency per access.

3.1.3 Incentives

Incentives are the approach to getting self-interested peers to share resources [5,12]. Most systems with incentives utilize a currency type approach such as karma in Kazaa. The primary problem with incentives is that they introduce many other problems. These issues primarily involve securing transactions and preventing counterfeiting. In the UBCA, incentives are currency-less. These incentives are implicit in the clustering of the peers. Since the group is based on mutually increased utility, each peer must contribute sufficiently, and if an peer is not part of a group, their performance will drop back to the levels of non-UBCA systems. Futhermore, incentives will improve the quantity and quality of resources accessed since these two aspects are factored in to the computation of utility.

Quality refers to the quality of a resource. The function to derive the quality is defined by the peer itself and may differ significantly based on type of resource and the needs of the user. These quality factors are defined in the resource definition file, and vary widely based on the resource. For instance, a quality factor of a printer might be its maximum DPI, or the quality factor of a song might be its bit rate.

Quantity is quite simply the amount of a resource that a peer offers. For instance, if the type is memory, it could be defined as the number of megabytes. This value might be part of a range (such as in the memory example), or it may also be a 0 or 1 value based on whether or not the resource is available (ie. a particular HTML file).

In order to show that the goal of providing incentives is met, it must be shown that utility is increased by providing resources and not leaching off of the system. The utility

increase is shown by comparing utility with the number of resources shared.

3.2 Architecture

Requesting Peer	Responding Peer
ReceiveQueryHit()	GenerateQueryHit()
if (CalculateAdditionalUtility() > 0)	ReceiveRequest()
SendRequest()	if (CalculateAdditionalUtility() > 0)
ReceiveResponse()	SendResponse()
If (CalculateAdditionalUtility() > 0)	if (receiveAck())
AddToGroup()	AddToGroup()
SendAck()	

3.2.1 Utility

Utility is the defining metric for forming a group. Utility is defined as the sum of the benefits minus the sum of the costs.

Equation 1:Utility

$$\text{Utility} = \sum (\text{Benefit} \;\; \text{Cost})$$

When the utility is greater than 0 for a peer, it implies that the formation of a group is beneficial for those involved. The metrics constraining the value of utility and selection are based on bandwidth, latency, memory, and cpu cycles. [26]

3.2.2 Benefit

Benefit is the value derived from the sum of the weighted averages of all the consumed resources less the cost of consuming those resources. The increase in benefit from one quantity of a resource can model a diminished return, which is common in economic systems, or linear increase, which be seen in situations where replication is valuable.

The cost to consume a resource takes into account a sum of the weighted ratio of the resource's latency to the average latency for that resource and the weighted ratio of the average bandwidth of that resource to the specific resource's bandwidth. The consideration of cost to consume creates a situation in which only the resources that are effectively more accessible to the consumer are selected.

Equation 2: Benefit

$$Benefit = \sum (w_0 Q(R_a)/Q(R_c) + w_0 Qn(R_a)/Qn(R_c)) - Cost_{consume}$$

Equation 3: Cost to Consume

$$Cost_{consume} = ((w_1(L/L_m)) + (w_2(B_m/B)))$$

Where:

w is the weight of the parameter
Q is the quality function for the resource
Qn is the quantity of the resource
R_a is the value of the particular resource
R_c is the value needed/preferred
L is the latency of the instance resource being considered
L_m is the mean latency of all instances of the resource being considered
B_m is the mean bandwidth of all instances of the resource being considered
B is the bandwidth of the instance resource being considered

3.2.3 Cost

The cost of joining a group is determined by analyzing how taxing it is on the peer to provide that resource. This value is determined by taking the ratio of the bandwidth required by the resource to the bandwidth available, the ratio of the memory required by the resource to the memory available, and the ratio of the CPU cycles required by the resource to the CPU cycles available. The bandwidth available is the effective bandwidth which is determined as the average bandwidth determined experimentally by the running average of the bandwidth to resources consumed. These values are then weighted by the amount relative importance to the peer, as with the benefit, but then the sum itself is weighted again based on the relative amount this resource provides out of the entire group in comparison to the amount that is consumed by the group.

Equation 4: Cost to Provide

$$Cost_{provide} = (w_1(B_r/B_a)) + (w_2(M_r/M_a)) + (w_3(C_r/C_a))$$

Where:

w is the weight of the parameter
B_r is the expected bandwidth required by providing the resource
B_a is the expected bandwidth available from the perspective of the peer
M_r is the expected memory required by providing the resource
M_a is the expected memory available from the perspective of the peer
C_r is the expected CPU time required by providing the resource
C_a is the expected CPU time available from the perspective of the peer

3.2.4 Selection

Selection entails determining which resource is optimal to request. The value for selection is given by taking the quality of the resource and subtracting from that weighted

value of the ratio of the latency of that resource to the mean latency and the weighted value of the ratio of the average bandwidth to the bandwidth of that resource.

Equation 5: Selection Value

$$SelectionValue = w_1 Q(R) - ((w_2(L_i/L_m)) + (w_3(B_m/B_i)))$$

Where:

w is the weight of the parameter
Q(R) is the quality value for the particular resource
L_i is the latency of the instance resource being considered
L_m is the mean latency of all instances of the resource being considered
B_m is the mean bandwidth of all instances of the resource being considered
B_i is the bandwidth of the instance resource being considered

3.2.5 Data Structures

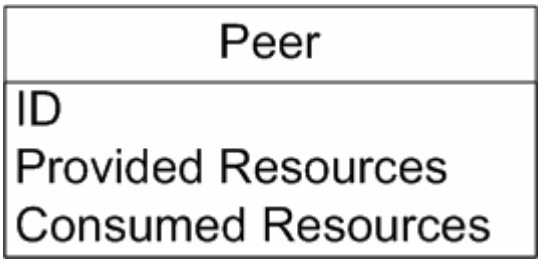

Figure 2: Peer Data Structure

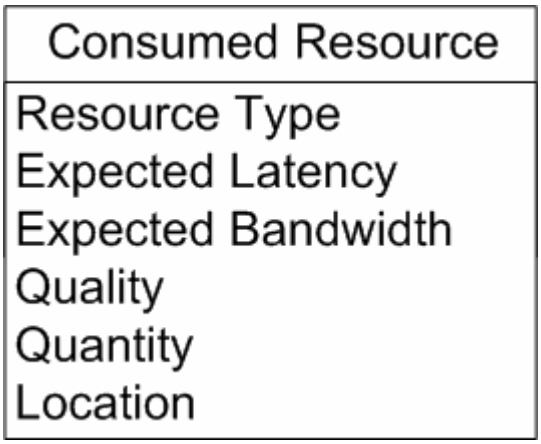

Figure 3: Consumed Resource Data Structure

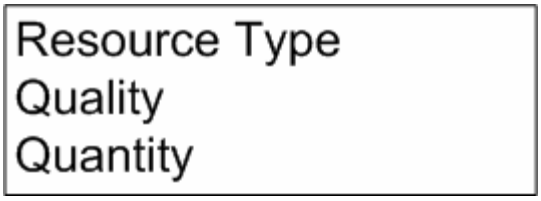

Figure 4: Provided Resource Data Structure

The UBCA data structures are used to represent the resource production (Figure 4) and consumption (Figure 3) of both the individual peer (Figure 2) and the collective resources of the group. A resource record contains the type of resource, the expected latency to access the resource, the expected bandwidth to the resource, the qualities of the resource, and the address of the resource.

The resource's expected latency is a running average of latency values as determined from communications with that peer. While many methods of calculating this average may be sufficient, we find that an approach similar to TCP timeout by weighting 75% of the value on previous measurements and 25% of the current measurement does well to adapt to changes in latency, but without overreacting to a single sample. The approach to bandwidth parallels that of latency. Transmission rates are taken empirically during communication and the value of the weighted average is stored. The type of resource is obviously the system's identifier for the resource (possibly a hash code or a specified resource ID); however, the quality of the resource is slightly less obvious. The resource quality is the list of attributes of a resource that could cause its quality to vary. For instance, a storage resource's quality could include the available space. A file, for instance, a song, could have resource qualities of bit rate and frequency. These quality properties can be defined in a definition file that establishes the resource's existence to the hosting peer or even sampled at load time when the

peer becomes active or resource is shared and then stored in memory.

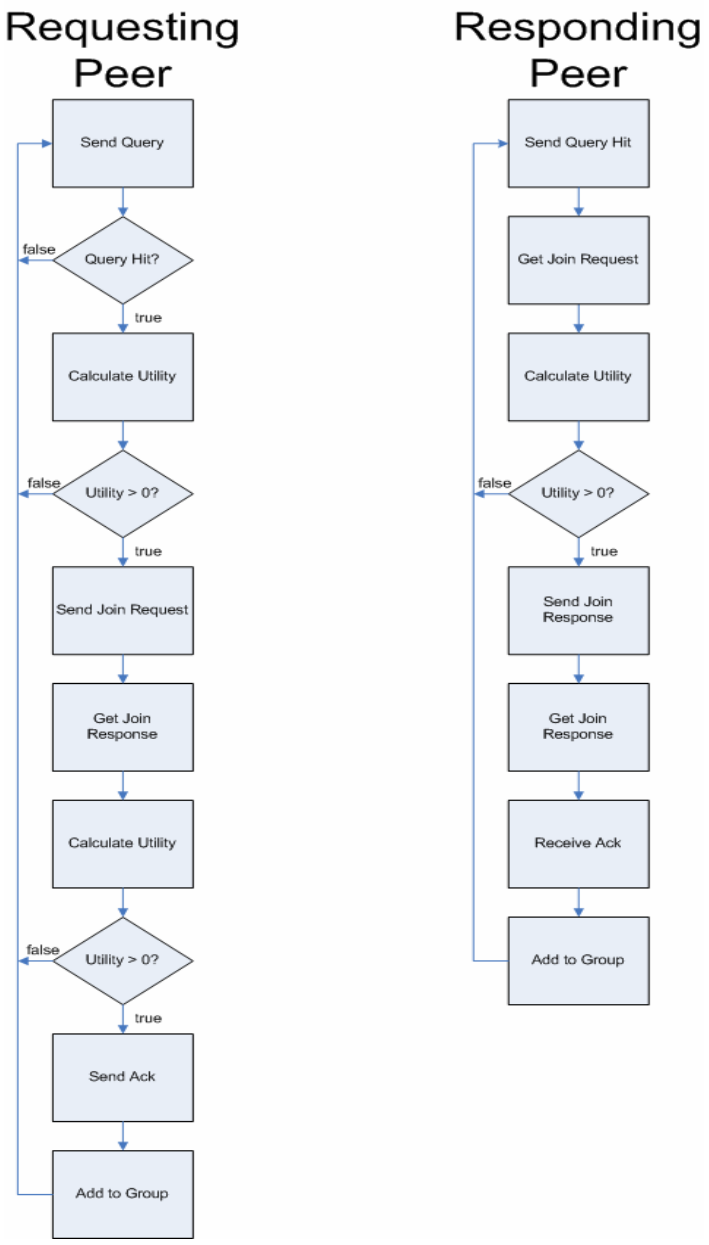

Figure 5: Peer Communication State Flow

3.2.6 Communications

Communications utilize and extend the Gnutella communication protocol [30]. The extension contains two major portions. The first portion is utilized to establish a group. The second is intra-group communication.

Upon receiving a successful query hit, a peer analyzes the peer that generated the query hit through a utility function and determines if the benefit based on that resource outweighs the cost of adding the peer to the group. If that is the case, then the peer initiates a group request by sending a group request message with the current group's set of resource provisions and consumptions. The peer then moves into a response-wait state. Upon receiving this invitation, the peer who supplied the query hit analyzes the provisions and consumptions with its utility function to determine if it accepts. If joining the group is beneficial to the peer, it then returns a response to the request containing its provisions and consumptions and enters a response-wait state. The original querying peer then performs a full utility analysis of the provisions and consumptions and decides if the peer is acceptable. If so, the peer is added to the group. Upon this acceptance, the peer will return a response to the waiting peer informing it of the decision and the peer will accept the group's provisions and consumptions. After each peer has added, they return to normal state.

Intra-group communication permits either lazy or active communication depending on the needs of the application. The communications that take place within the group are utilized to maintain the link state of the group. Each peer capable of permitting another peer into the group maintains a set of resources of each peer's consumption and production. In

order to update these lists, when a new peer is added, the peer that admits it broadcasts a message to the group announcing the new addition of resources provided and consumed. Furthermore, when a peer departs from a group, that peer broadcasts an exit message for the other peers to remove its record. Since peer-to-peer networks are often ad-hoc, mobile, and dynamic,[16] peers are likely to exit without warning. If a peer departs the group for any reason without sending the exit message, any peer that fails to access the peer broadcasts a message to warn other of the potential departure. When a peer receives a sufficient number of these messages (by default, one) the peer removes the departed peer's record from the data structure.

3.3 Theoretical of Implications

3.3.1 Group Sizes

One major concern that differs from the work of Ogston, et al. is the sizing of groups. Ogston presents evidence of successful convergence rates of groups with fixed sizes [20], but does not address situations where group sizes are undefined or dynamic.

Since the algorithm is based on additional benefit, the size of the group will only grow as it is beneficial for the group. The utility functions provide a point where the overhead of adding another peer to the group will exceed the benefit. Since the benefit derived from adding more of a resource diminishes as the resource is continually replicated, if it is given that there are a finite number of desired resources, it is then possible to show that every group will have a limit where the utility will decrease by adding another peer. If

needed, an artificial limit can be placed on the size groups; however, it will diminish the utility of the system.

3.3.2 Network Utilization

One of the effects of P2P computing is the increased network traffic [22] from searching for resources. In addition to the overhead caused by this traffic, the effective time of the network is wasted. The algorithm pits the overhead cost of establishing and maintaining groups against the cost of doing “dumb” searches for resources. A “dumb” search is one in which there is no strategy in the search and each peer forwards queries to all of its neighbors, as in Gnutella. Since the cost in bandwidth of maintaining groups is relatively close to that of “dumb” searches when compared to the overall bandwidth typically available to peers, we discard this factor and look to a more important issue. This issue is the time it takes to actually acquire and utilize a resource. Since a peer will be in a group only if the peer itself gains benefit from participating in the group, the majority of its important resource needs are met within the group. Because of this fact, a peer will require no additional network access time, which is significantly more costly in comparison to CPU or memory access time, the ratio of time spent utilizing the network bandwidth for accessing the resource as opposed to utilizing the network bandwidth for overhead costs greatly increases. Hence, the overhead traffic is compacted such that the peer's time is better utilized.

3.3.3 Flexibility

Since the algorithm is designed to be general case, it can be easily modified to meet constraints that might be enforced by a particular aspect of the system. For instance, if the need arises, it is simple to enforce a limit on the group size. Additionally, systems which need to utilize the benefits of loosely or tightly formed groups are free to do so within the parameters of the algorithm. The form of communication is also not prescribed, as the UBCA works with either lazy or strict propagation of information.

CHAPTER 4

IMPLEMENTATION DETAILS

4.1 GroupSim

GroupSim is a Java-based simulator designed for simulating P2P systems. It implements the Gnutella architecture and the UBCA present in this thesis built on top of Gnutella. This implementation decision was made since Gnutella is not highly structured and is decentralized. The simulator takes the following inputs:

- number of classes of resource in the system
- number of peers in the system
- number of peers to connect to initially
- latency distribution of a peer
- bandwidth distribution of a peer
- maximum number of resources consumed by a peer
- maximum number of resources produced by a peer
- probability of a peer accessing resources
- whether or not the UBCA is enabled
- number of partitions of peers

Peers can be extended from a base peer class which provides random access to resources. Extending the peer class can allow for particular classifications (ie, leechers,

servers) of peers to be simulated, or to simulate related resources by defining that certain resources are provided and consumed with a higher probability by one class of peer. The controlling values set in the system parameters are defined by these values. These are applied to all peers that extend the basic peer class and are utilized by the basic peer as follows:

- Classes of resources - the set of possible resources a peer can provide or consume.
- Number of peers in the system - the maximum number of active peers in the system. Since the simulations model dynamic situations, it is likely that not all peers are currently available at any given time.
- Number of peers to connect to - the number of peers originally connected to each peer in the bootstrapping process.
- Maximum latency and bandwidth - both mean values, standard deviations, and distribution type
- Maximum number of resources consumed and produced – defines the range from 0 to the maximum provided that a peer with consume or produce, respectively.
- Probability of the peer to access – likelihood that a peer is to choose to access a resource at the given time.
- Number of partitions – partitions the resources into linked sets of resources to each other

Simulations statistics are organized into two types. The first is the underlying metrics. These are the core statistics of the system, such as average latency, bandwidth, and

so on. These values are essential to determining the performance of how well the system can be tuned to particular needs. The second type of simulation is the high level statistics. These figures provide a big picture of the architecture. For instance, how well it can be used to fit specific needs, such as ad-hoc clustering. The former simulations show the potential of the UBCA and the latter show the application of the UBCA.

Measurements were taken from 500 trials of the simulation software. As a result, the simulations provide a narrow range for the 95% confidence interval. The 95% confidence interval is give by a normal distribution in the form of:

$$\bar{x} - 1.96(\frac{\sigma}{\sqrt{n}}) \leq \bar{x} \leq \bar{x} + 1.96(\frac{\sigma}{\sqrt{n}})$$

Where the empirical average of the simulation results is $\bar{x}$, the empirical standard deviation of the simulation results is σ, and the number of simulations performed is n.

4.1.1 Average Bandwidth Per Access

Average Bandwidth per Access takes the average bandwidth a peer transfers per each access. This metric is an essential consideration in large data transfers, for instance content distribution. This is taken by determining the bandwidth of every access that occurs in the system over the course of a simulation, then dividing it by the number of accesses.

4.1.2 Average Latency Per Access

Average Latency per Access parallels the average bandwidth per access. In this case, however, the metric represents the performance of high numbers of accesses at potentially lower bandwidth concerns. This metric is determined by taking the latency of every access over the course of the simulation, then dividing it by the number of accesses.

4.1.3 Queries Initiated

The queries initiated shows how many queries are issued during the course of the simulation. This does not include the queries forwarded by a peer, which are captured in the next statistic.

4.1.4 Queries Forwarded

The queries forwarded are the resulting queries that are forwarded to other peers when a peer receives a query. Only unique queries are forwarded. This statistic is a key factor in scalability.

4.1.5 Query Hits

The query hits are the number of query hit packets that make it back to the original querying peer successfully. This value is important in many factors, including measuring efficiency by the success rate of queries with respect to the available resources.

4.2 Implementation Options

4.2.1 Strong Versus Weak Groups

Groups can be either weak or strong. Strong groups are better defined and have a group leader who organizing and orchestrate the membership of the group. This is considered a strong group since the group is well defined and everybody is fully aware of everybody else and they each maintain the same group-state information. A weak group is one in which all the peers act independently and there is no leader. In this case, group membership is granted by any member of the group. This provides a more flexible and robust group; however, it also presents a problem of inconsistency, since it is feasible that two peers will admit other peers into the group simultaneously which, individually would

benefit the community, but together decrease the utility of the community.

4.2.2 Propagation of Information

Another issue to consider is how the group information propagates. While this is not a focus in the work, it should be considered in system design. For the architecture provided and implemented, lazy propagation is used in intra-group communications. This reduces the difficulties of having to assure synchronization; however, the trade-off is that it is now possible to make errant decisions based on incomplete information. It is expected that this will not have a significant effect on the system and will be tested in simulation.

CHAPTER 5

RESULTS

In this chapter, we present the results of the simulation in order to demonstrate key strengths in the architecture in addition to its ability to satisfy our goals established previously. Unless otherwise noted, all simulations were performed with the following parameters:

- 100 classes of resources
- 5 initial connections at bootstrap
- Normally distributed latency
- Normally distributed bandwidth
- Resources consumed to provided ratio of 1 (25 to 25)
- No Peer Classifications
- The simulations were run for 10, 20, 50, 100, 200, 500, and 1000 peers and graphed values were extrapolated from those results

The first consideration is the traffic cost due to Gnutella as opposed to our UBCA implementation on Gnutella. To demonstrate this, we show that our architecture reduces the queries initiated, queries forwarded, and query hits generated (which results in messages being sent) while still maintaining performance in other metrics.

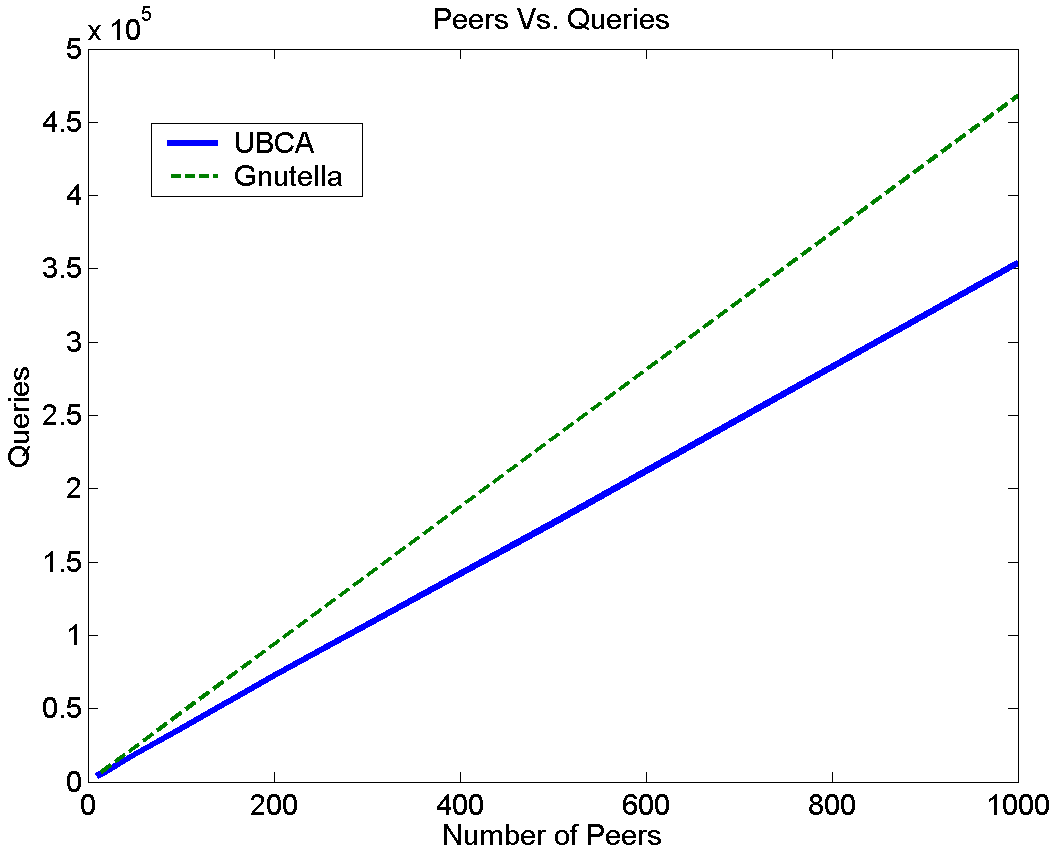

Figure 6: Number of Peers Vs. Queries

As Figure 6 shows, the number of queries initiated is drastically reduced in the UBCA implementation. The 95% confidence interval for the average number of queries of Gnutella is plus or minus 71.80 queries Sent. The 95% confidence interval for the average number of queries of UBCA is plus or minus 1093.81 queries Sent. The reason that the UBCA implementation varied so much more than the standard Gnutella implementation is that the effectiveness of the UBCA depends on peers being able to find resources in peers with high bandwidth and low latency connections quickly. Since the structure of the network and the allocation of resources are generated randomly, there will be cases when the UBCA

performs closer to the Gnutella implementation than others; however, even with the worst case of the 95% confidence interval, the UBCA still outperforms the Gnutella implementation.

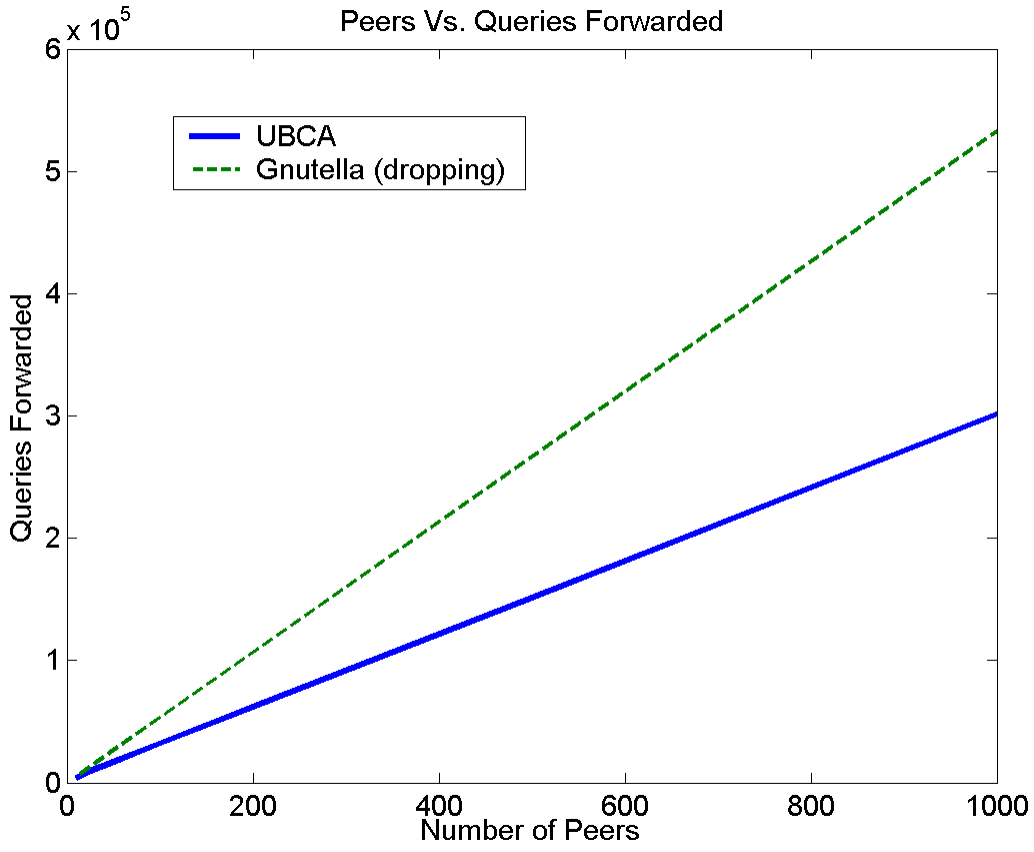

Figure 7: Number of Peers Vs. Queries Forwarded

As Figure 7 shows, the number of queries forwarded also is drastically reduce in the Grouped Architecture. The 95% confidence interval for the average number of queries forwarded of Gnutella is plus or minus 247.6 queries forwarded. The 95% confidence interval for the average number of queries forwarded of UBCA is plus or minus 422.35 queries Forwarded.

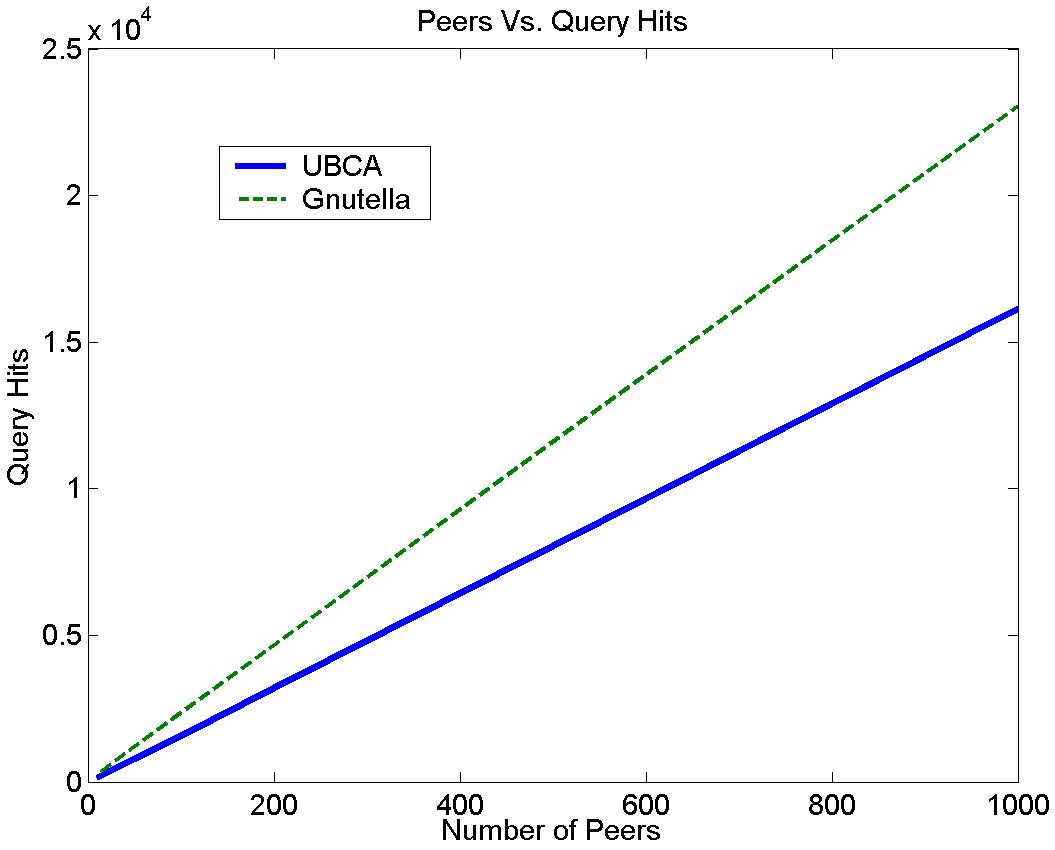

Figure 8: Number of Peers Vs. Query Hits

As Figure 8 shows, the number of query hits generated also is drastically reduced in the Grouped Architecture. The 95% confidence interval for the average number of query hits of Gnutella is plus or minus 30.77 query hits. The 95% confidence interval for the average number of query hits of UBCA is plus or minus 26.82 query hits.

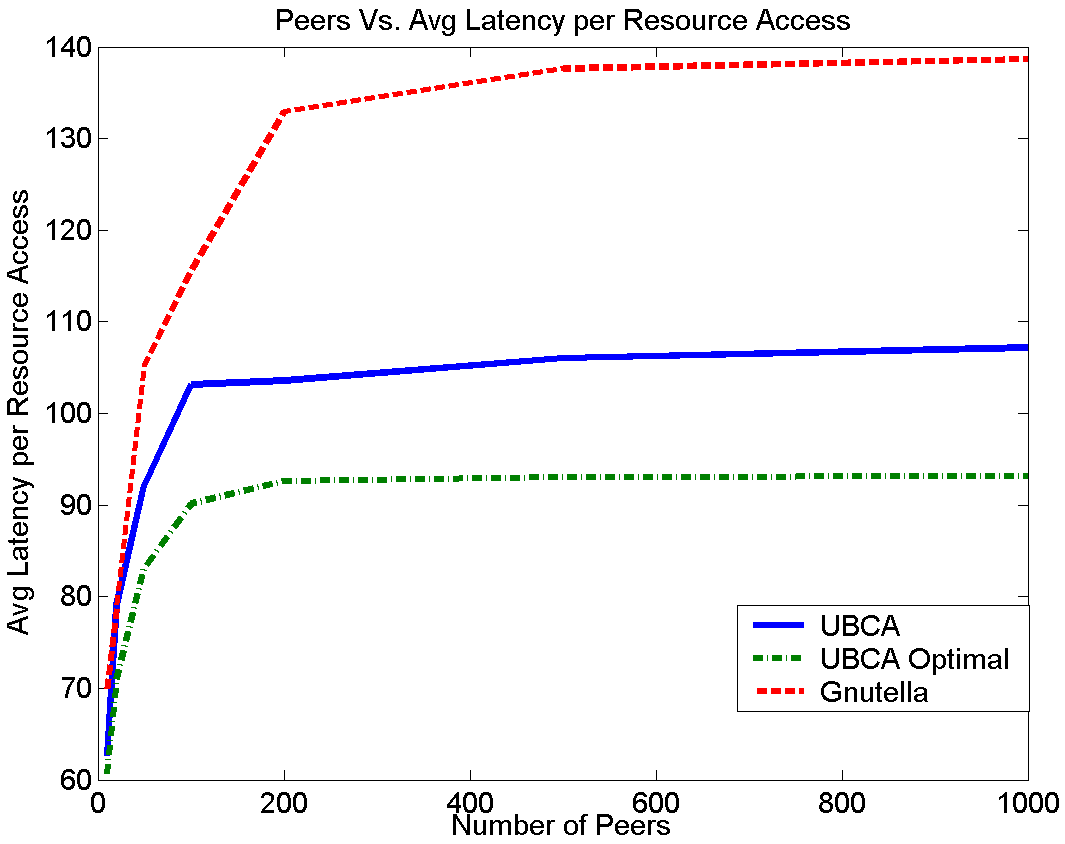

Figure 9: Number of Peers Vs. Average Latency per Resource Access

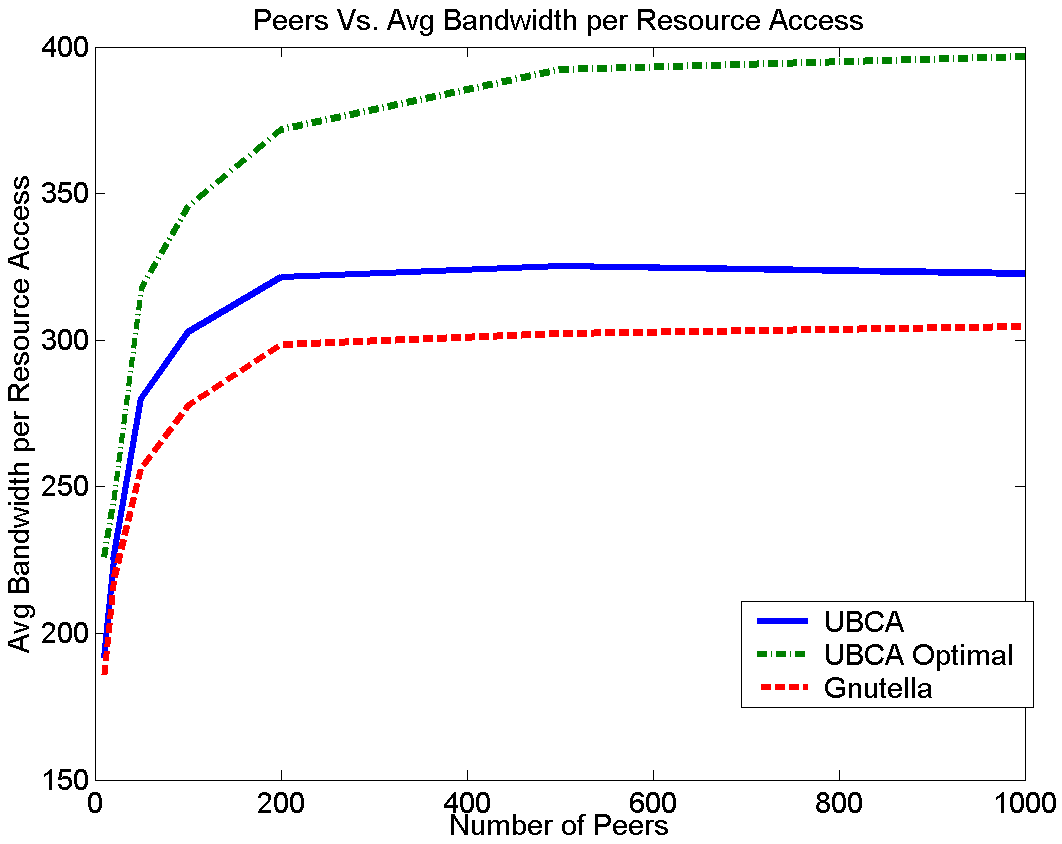

Figure 10: Number of Peers Vs. Average Bandwidth per Resource Access

Figure 9 and Figure 10 compare the average latency and average bandwidth values per resource access. The UBCA plot is the simulation using the UBCA with a uniform weight (all aspects are considered equal. The optimal plot is a UBCA implementation where latency, for Figure 4, and bandwidth, for Figure 5, were given the entire weight of the decision. The Gnutella plot is the result of the standard Gnutella implementation. The comparison shows performance of the UBCA implementation's with respect to the best possible choice that could have been made over all peers that a peer could have known of at the time of the resource access. The 95% confidence interval for the average bandwidth

values of Gnutella is plus or minus 0.76 KB/s. The 95% confidence interval for the average bandwidth values of UBCA is plus or minus 0.91 KB/s. The 95% confidence interval for the average bandwidth values of UBCA Optimal is plus or minus 0.88 KB/s. The 95% confidence interval for the average latency values of Gnutella is plus or minus 5.90 ms. The 95% confidence interval for the average latency values of UBCA is plus or minus 3.61 ms. The 95% confidence interval for the average latency values of UBCA Optimal is plus or minus 3.44 ms.

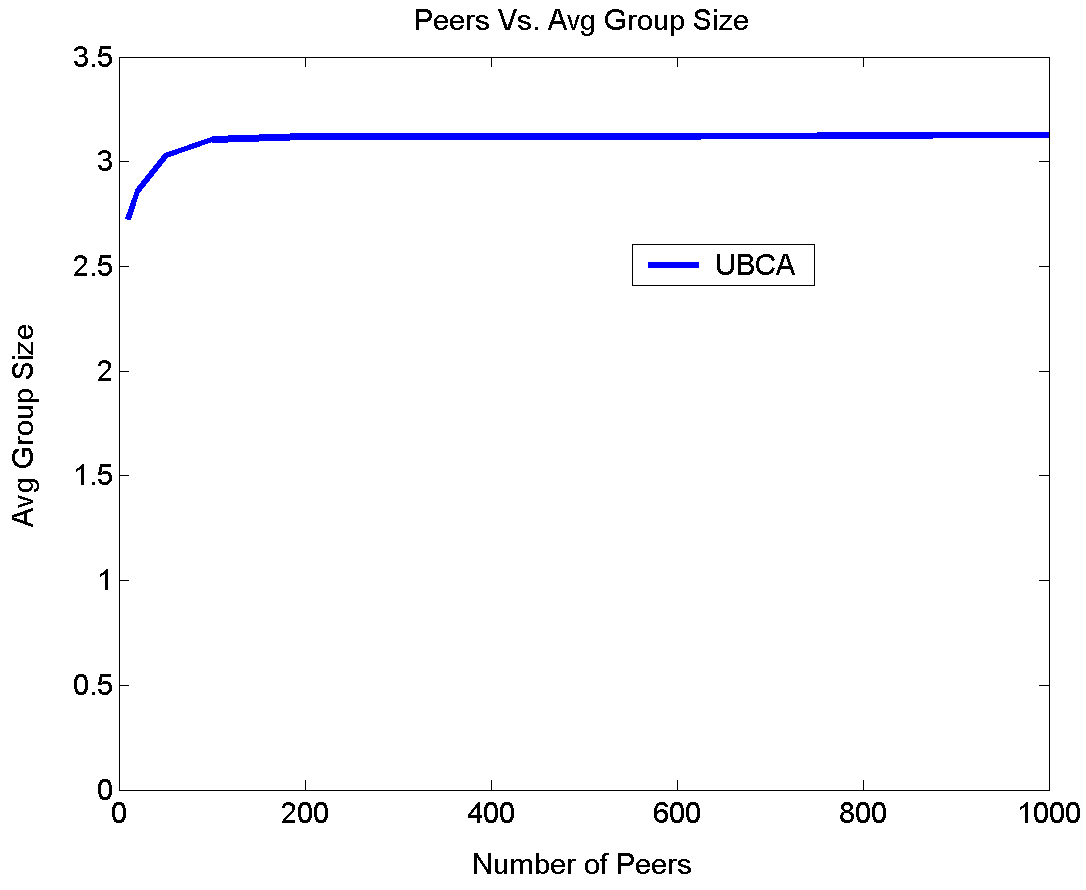

Figure 11: Number of Peers Vs. Average Group Size

The average group size shown in Figure 11 is not an essential metric to performance, but it does show approximately how many peers need to be grouped together in order to

provide the performance shown previously and that the size of the group is basically constant with respect to number of peers. It also demonstrates the low load on memory required for the algorithm to operate efficiently. The 95% confidence interval for the average group size of UBCA is plus or minus .0099 members.

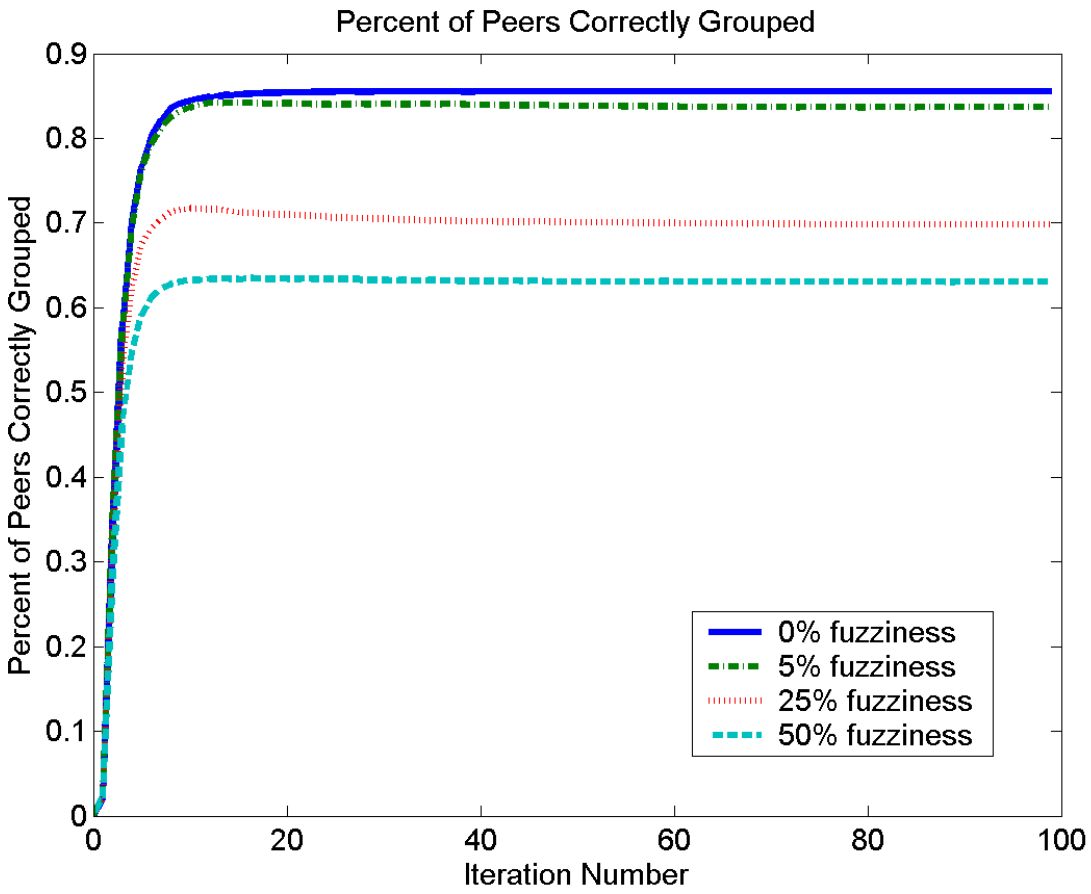

Figure 12: Percent of Peer Correctly Grouped at Each Iteration

In order to determine the UBCA's performance as a classification algorithm, we produced the results shown in Figure 12 by iteratively examining the current status of groups after each round (in this case, the algorithm executes the same; however, time is not

considered in the same sense as the network simulations since no resources are actually accessed. A round is the occurrence in which a peer has the opportunity to submit one query and handle any current incoming queries and group requests.).

The UBCA implementation's ability to group peers scales constantly with the number of peers. In order to determine the governing factors of algorithm's ability to group peers, other factors were varied until a significant change was discovered. In this algorithm, the governing factors for grouping efficiency are the ratio of resources shared/consumed to the number of resource classes and the fuzzy difference between peer types. Figure 12 shows a high classification rate of peers when the intersection of peers is null (in many cases peers cannot be grouped because of their poor latency and bandwidth values).

CHAPTER 6

APPLICATIONS

6.1 PICO

PICO is a framework which facilitates the formation of dynamic, mission-oriented communities of software agents [17]. Due to its agent-based dynamic nature, along with its promotion of the formation of communities, PICO is a prime candidate for utilizing the UBCA described in this paper. Since PICO's goal is to create communities of agents for the completion of a particular meeting, our clustering work can be utilized in order to form a community with improved utility, rather than one with just an unknown set of available agents. Furthermore, specific missions can utilize different weights within the UBCA in order to adapt it to cluster peers in order to more aptly satisfy the goals of the community. The application of the UBCA is a method to improve performance and reliability of the system.

6.2 RoboCup

Robocup is an international competition for artificial intelligence and robotics [23]. Robocup uses soccer as its standard for competition. It encompasses several leagues including simulation, small, middle, 4-legged, and humanoid. The simulation is done completely on computer while the others use various robots as described by the league type.

Within the context of this competition, I propose an extension of competition. In

current competition, all competition is based on techniques of artificial intelligence and robotics. I propose an additional competition on each league where a set of robots/agents are presented with definitions of their abilities (speed, passing accuracy, shooting accuracy, passing power, shooting power, etc.) and the teams compete by drafting these competitors. This new competition would use the familiar platform of soccer to introduce work in iterative, self-interested organization. In this case, selecting the correct combination of players to work together could be accomplished with the UBCA presented in this paper.

As this architecture applies to the problem presented, its goal would be to put together the best team given the available players at the time of the selection. This is a direct application of the architecture to a scenario directly related to the basketball one described at the beginning of this paper. Depending on the nature of the competition, the architecture could be applied as-is (in the case that each agent is free to join whichever team offers admission) or more likely, could be applied with only one side's utility in consideration (in the case that there is a draft process where each team incrementally receives the right to add one player to its team each round). Regardless of the case, the architecture can be applied to best match together a group of players to compete. Since “best” is subjective, the architecture would use the skills needed by each player to compete best to put together a well matched team where each player complements the other players on the team.

6.3 Content Distribution

Content Distribution Networks (CDNs) are systems which typically reside at many key points in the network infrastructure in order to transparently satisfy the requests of users

more efficiently [2,4]. P2P-based CDNs move content between peers in order to more quickly satisfy user requests and reduce the bandwidth associated with satisfying the request from a provider "further" away. Similar to the architecture presented in this paper, this is often done by intelligently directing requests to peers which will reduce the number of hops or which will be capable of responding more quickly to the request. Additionally, Content distribution can be accomplished by routing requests through a caching proxy. For instance, in a university environment, requests may be routed through a proxy server which will satisfy a request with a page found in its cache. This is done since University students tend to have similar interests and needs, so the content they consume will be similar.

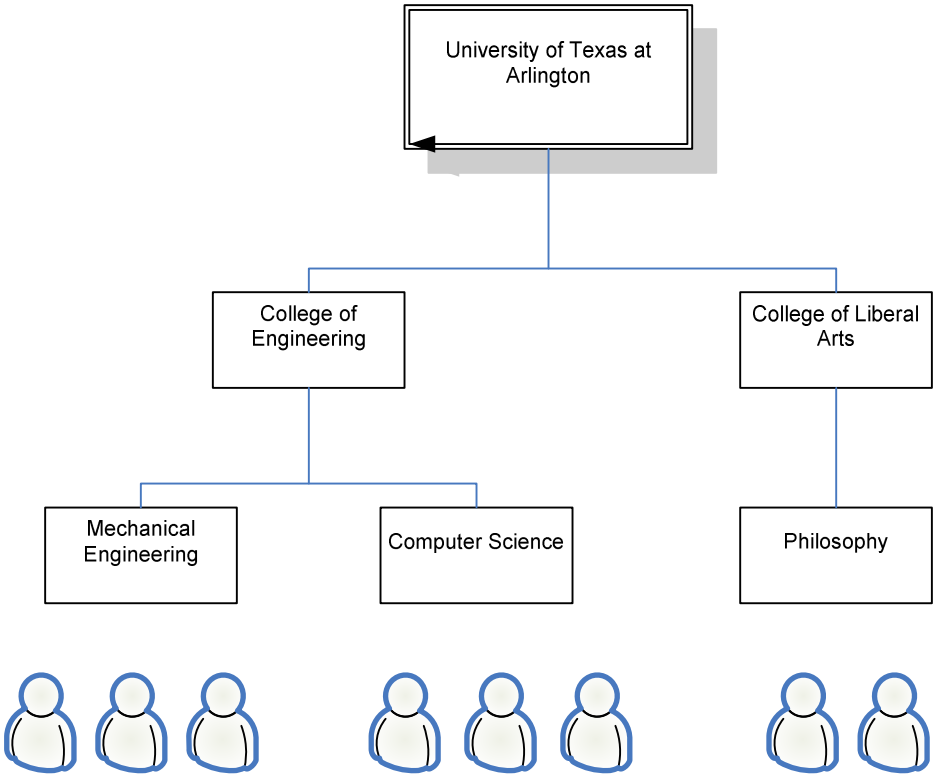

Figure 13: Logical Departmental Layout

One situation where this application would arise is a peer to peer network of college students all living in a particular dorm. This network consists of students who have accessed and cached academic websites from their courses, departments, colleges, and University. In this network, there are several classifications of students. For instance, we can have Computer Science students, Mechanical Engineering students, and Philosophy students. In such a scenario, each classification of students shares Provided (cached) and Consumed (request) resources (websites) most similar to each other; however, due to their membership in other similar groups, they also share provisions and consumptions with students outside of their classification.

The application of the UBCA described in this paper would group the most similar students together with consideration of the accessibility of those resources. Obviously, each particular major would tend to group together most strongly, but Computer Science & Engineering students might also group together with Mechanical Engineering students who are very accessible to them because of their similarities in the College of Engineering. Likewise, the engineering students have minimal affinity towards Philosophy students (but a non-zero amount due to their similarities in the overall University website). The probability of groupings would be based on this shared affinity and produce clusters of students which can easily access each other's cached websites to minimize the load on the University's central servers.

CHAPTER 7

CONCLUSION

UBCA provides an improvement to P2P systems, particularly Gnutella. UBCA takes a unique approach to improving P2P systems by clustering peers together based on mutual utility derived from the clustering. UBCA meets its design goals by improving scalability, increasing performance, and increasing resource availability/accessibility.

UBCA has been shown in simulations to increase bandwidth per access, reduce latency per access, and reduce the overhead costs of system operation over Gnutella. Furthermore, UBCA uses the increased performance as a currency-less incentive for peers to share more resources and not free-load. The implementation of UBCA will encourage the replication and access of files in distribution networks.

The next step in this line of research is to implement the UBCA in the applications mentioned previously and examine their empirical performance. Another area off work to consider is to examine application specific optimizations of UBCA such as defining the proper weights, or dynamic mechanisms for assigning weights for an application such as streaming multimedia in an ad-hoc network. There is also potential for future work in examining implementations of the architecture in mobile environments and testing the result of those implementations.

REFERENCES

1. Adar, E., Huberman, B. Free Riding on Gnutella. First Monday, volume 5, number 10, October 2000.

2. Androutsellis-Theotokis, S. and Spinellis, D. A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys, Vol. 36, No. 4, December 2004, pp. 335-371.

3. Banavar, G. Beck, J. Gluzberg, E., et al. Challenges: An Application Model for Pervasive Computing. Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking, 2000.

4. Biddle, P., England, P., Peinado, M., Willman, B. The Darknet and the Future of Content Distribution. ACM Workshop on Digital Rights Management, 2002.

5. Buragohain, C., Agrawal, D., Suri, . A Game Theoretic Framework for Incentives in P2P Systems. Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P'03). 2003.

6. Chawathe, Y., Ratnasamy, S., Breslau, L., et al. Making Gnutella-like P2P Systems Scalable. SIGCOMM, 2003.

7. Chen, H. Finin, T. Joshi, A., et al. Intelligent Agents Meet the Semantic Web in Smart Spaces. IEEE Internet Computing. Vol. 8, No. 6, November 2004.

8. Cohen, E., Shenker, S. Replication Strategies in Unstructured Peer-to-Peer Networks. SIGCOMM, 2002.

9. Cook, D., Youngblood, M., Heierman, E., et al. MavHome: An Agent-Based Smart Home. First IEEE International Conference on Pervasive Computing and Communications, 2003.

10. Douceur, J. The Sybil Attack. Microsoft Research.

11. Galil, Z. Efficient Algorithms for Finding Maximum Matching in Graphs. Computing Surveys, Vol. 18, No. 1, March 1986.

12. Golle, P., Leyton-Brown, K., Mironov, I., Lillibridge, M. Incentives for Sharing in

13. Peer-to-Peer Networks. In Proc. of the Third ACM Conference on Electronic Commerce, Tampa, Florida, USA, October 2001.

14. Kagal, L., Undercoffer J., Joshi A., Finin, T. Vigil : Providing trust for enhanced security in pervasive systems, October 2001

15. Khuller, S. and Raghavachari, B. Graph and Network Algorithms. ACM Computing Surveys, Vol. 28, No. 1, March 1996.

16. Koo, S. G. M., Rosenberg C., Xu, D. Analysis of Parallel Downloading for Large File Distribution. FTDCS 2003.

17. Krishnan, N. The JXTA Solution to P2P. http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html. October 19, 2001.

18. Kumar, M., Shirazi, B., Das, S. K., et al. Pervasive Information Communities Organization PICO: A Middleware Framework for Pervasive Computing, IEEE Pervasive Computing, July-September 2003, pp. 72-79.

19. Loguinov, D., Kumar, A., Rai, V., Ganesh, S. Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience. SIGCOMM'03. 2003.

20. Nuno C., Romero L., Santiago J., et al. Gaming, Fine Art, and Familiar Strangers. IEEE Pervasive Computing, vol. 03, no. 1, pp. 35-37, Jan-Mar, 2004.

21. Ogston, E., Overeinder, B., Van Steen, M., and Brazier, F.: A Method for Decentralized Clustering in Large Multi-Agent Systems. Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2003.

22. Ripeanu, M. Peer-to-Peer Architecture Case Study: Gnutella Network. Technical Report, University of Chicago, 2001.

23. Ritter, J. Why Gnutella Can't Scale. No, Really. Unpublished. 2001.

24. Robocup. http://www.robocup.org

25. Roman, M, Campbell R. GAIA: Enabling Active Spaces. 9th ACM SIGOPS European Workshop, September 17-20, 2000.

26. Satyanarayanan, "Pervasive Computing: Vision and Challenges", IEEE PCM August, pp. 10-17, 2001.

27. Schoder, D., Fischbach, K., Schmitt, C. Peer to Peer Computing: The Evolution of a Disruptive Technology. Chapter 1: Core Concepts in Peer-to-Peer Networking. Idea Group Publishing, 2005.

28. Seika, B., Kshemkalyani, A., Singhal, M. On the Security of Polling Protocols in Peer-to-Peer Systems. Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04). 2004

29. Sih, G. and Lee, E. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 2, February 1993.

30. Takemoto, M., Sunaga, H., Tanaka, K., Matsumura, H., Shinohara, E. The Ubiquitous Service-Oriented Network – An Approach for a Ubiquitous World based on P2P Technology. Proceedings of the Second International Conference on Peer-to-Peer Computing (P2P'02). 2002

31. The Gnutella Protocol Specification v0.4.

32. Wagstrom, P. Scarlet. Master's Thesis. 2003.

33. Weiser, M. "The Computer for the Twenty-First Century," Scientific American, pp. 94-10, September 1991

34. Weissman, J., Grimshaw A., West E., Lyot E. Metasystems: An approach combining parallel processing and heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 21(3):257--270, 1994.

BIOGRAPHICAL INFORMATION

Brent Jason Lagesse received his B.S. in Computer Engineering in May of 2004. He then entered the University of Texas at Arlington in August where he graduated with his M.S. in Computer Science in May of 2006. During this time he spent a year working at Lockheed Martin Missiles and Fire Control in addition to pursuing his academic interests. He will spend the summer of 2006 working with Lawrence Livermore National Laboratory, and then begin pursuing a PhD in the fall of 2006. His primary research interests include pervasive computing, multi-agent systems, computational economics, and trust-based systems.