# BIOLOGICALLY INSPIRED PEER-TO-PEER DISTRIBUTED FILE SYSTEM

By

Sergio Guido Camorlinga

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Faculty of Graduate Studies
University of Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Biologically Inspired Peer-to-Peer Distributed File System

BY

Sergio Guido Camorlinga

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

Doctor of Philosophy

# TABLE OF CONTENTS

# List of Figures

*Number* *Page*

University of Manitoba

Abstract

# BIOLOGICALLY INSPIRED PEER-TO-PEER DISTRIBUTED FILE SYSTEM

by Sergio Guido Camorlinga Díaz

Chairperson of the Supervisory Committee: Professor Ken Barker
Department of Computer Science

A fundamental problem in the current second generation of Peer-to-Peer algorithms and applications is that they are based on schemes and models that have predetermined, predefined centralized and distributed techniques. The majority of second-generation Peer-to-Peer systems assume stable environments. Workarounds are commonly used to fix them. However, when subject to dynamic contexts, the algorithms and applications are brittle, un-scalable, and limited to continue providing services.

A Peer-to-Peer Distributed File System as an application developed on top of a Peer-to-Peer system, inherits second-generation Peer-to-Peer drawbacks, unless special workarounds are provided. The Peer-to-Peer Distributed File System has limitations to manage its distributed file services effectively in decentralized, self-organized, *ad-hoc* environments. This thesis tackles these problems by first presenting a new computing paradigm called the Emergent Thinker and then applying the Emergent Thinker to design and develop a Biologically Inspired Peer-to-Peer Distributed File System. As a collateral contribution, the thesis provides a paradigm and a methodology that can be used to provide Systems Research a new approach to solve distributed systems problems.

The thesis introduces the *Emergent Thinker* paradigm, an area-wide logical computing entity, named after a philosopher that continuously analyses information and has emergent computed solutions for new and/or existing requests. The Complex Adaptive System (CAS) emergent computation model and the CAS propagation model are proposed as mechanisms to achieve the

Emergent Thinker. The Thinker is intended as an alternative approach for new and current design and implementation challenges in systems research.

The Biologically Inspired Peer-to-Peer Distributed File System (BPD) implements the Emergent Thinker paradigm. The BPD design and implementation is the major thesis contribution. The biological inspiration comes from the natural and biological models used in the implementation of the CAS that form the basis of its services. The BPD provides file system services in a complex information system environment that is dynamic, self-organized, *ad hoc* and decentralized. Some of the characteristics that make BPD distinctive include an architecture design and implementation that merges CAS models with Peer-to-Peer computing; CAS models for distributed services (storage, retrieval, search, and replication); and the use of commodity peers for storage and retrieval in a complex dynamic environment.

Systems research is an area where complex adaptive systems can provide innovative schemes and models. These schemes and models can provide emergent solutions (i.e. swarm properties) to a variety of design and implementation challenges in large distributed systems; some already exist but others have not been conceived before.

The Emergent Thinker paradigm offers an alternative computational model for complex information system environments with the use of Complex Adaptive Systems models and the swarm properties that emerge from it. The thesis builds and describes the design and implementation of the Biologically Inspired Peer-to-Peer Distributed File System that corroborates the hypothesis.

# Acknowledgments

There are many people in my life that have helped to achieve personal and professional goals that I will always be in debt to them. Particularly important is my advisor Dr. Ken Barker. I will always be grateful to your mentorship and guidance. Your prompt help and assistance to research problems I faced during this PhD journey made things easier to approach and resolve. Even though you are in Calgary now, the distance was not obstacle to be continuously communicated. Nowadays different mediums exists that remote collaboration and research can be done effectively. The proof is that most of the thesis research experiments were done on large servers residing in the University of Calgary. Your teaching at the distributed databases course was enjoyable and learned seminal ideas for distributed system design and implementation. This seminal knowledge seeded in me the idea to carry on my thesis research in distributed systems. I appreciate the fact that you had to fly to Winnipeg for many weekends, giving up your personal time to be here and teach. Dr. Barker, I thank you for all your assistance and support to this journey.

I want to thank my other members of the PhD committee Dr. Peter Graham, Dr. John Anderson, Dr. Jeff Diamond and Dr. William Perrizo. Dr. Graham and Dr. Anderson taught me fundamental graduate courses that gave me basic ideas for my thesis research topic. Dr. Graham brought to my attention the topic of Distributed File Systems and Peer-to-Peer computing as potential research areas to explore. I knew I wanted to do research in distributed systems but I was not sure how I could contribute new knowledge and expand current systems research. That it was when Dr. Anderson introduced me in his graduate course teaching the topic of Complex Adaptive Systems, and my

internal bell clicked with the thought of mixing all four subjects (i.e. distributed systems, distributed file systems, peer-to-peer computing and complex adaptive systems) into one research thesis topic and the Biologically Inspired Peer-to-Peer Distributed File System idea was born in early 2002. Since then I am so excited about Complex Adaptive Systems that everything I do and research, I see it first from a Complex Adaptive Systems perspective. Dr. Diamond has been similarly helpful by executing collaborative research in related projects with Telecommunications Research Labs. Dr. Diamond's comments have been helpful to improve my research work. I thank you all for your teaching and guidance that help me a lot in my thesis work. Also I am thankful to Dr. William Perrizo for accepting serve as thesis external examiner and provide valuable feedback.

Also, I am grateful to Dr. Blake McClarty for his support to carry out the PhD program. Dr. McClarty and I had initial talks regarding pursuing a PhD and he was immediately supporting the idea back in 2001. I thank Dr. McClarty for all his help to pursue the PhD.

I show gratitude to all professors, graduate students and friends that I have met at the University of Manitoba, University of Calgary and St. Boniface General Hospital Research Center during these years. Thank you for your friendship and assistance.

Similarly I express thanks to St. Boniface General Hospital Research Centre Medical Informatics group, the University of Manitoba departments of Radiology and Computer Science, and the University of Calgary department of Computer Science that without their support and availability of resources, making the PhD would have been very difficult.

My family is the most important part of my life. I have always been a family person and always enjoy being with them. My mom Dolores and dad Roberto guided me in my initial years and showed me that being a family and keeping your values help you to deal with a difficult world. My daughters Paola and Gaby that have grown up quickly and continue growing remind me that life is short and that we have to enjoy it together while we can. My wife Guadalupe has been lovely and helpful supporting me in this journey. Guadalupe has kept me on the ground and help me to not to divert from my goals. I hoped I did not bore her much with my squirrels' talks. I dedicate this thesis to my beloved family.

And thanks God for giving me the opportunity to live during these exciting times and be a researcher.

Sergio Camorlinga, 2001-2005.

# Glossary

**API**. Application Programming Interface

**BDM**. Back-end Data Manager

**BPD**. Biologically inspired Peer-to-peer Distributed file system

**CAS**. Complex Adaptive System

**CEC**. CAS Emergent Computation

**CPM**. CAS Propagation Model

**CPU**. Central Processing Unit

**DFS**. Distributed File System

**DS**. Distributed System

**DSF**. Data Sharing Facility

**FDM**. Front-end Data Manager

**FS**. File System

**IP**. Internet Protocol

**LAN**. Local Area Network

**LDM**. Local Data Manager

**LDS**. Large Distributed System

**MAS**. Multi Agent System

**NFS**. Network File System

**OS**. Operating System

**P2P**. Peer to Peer

**PC**. Personal Computer

**POSIX**. Portable Operating System Interface

**TCP**. Transport Control Protocol

**TNC**. Transport, Network and Communication

**TTL**. Time to Live

**WAN**. Wide Area Network

**JXTA**. Juxtapose

# *Chapter 1*

# 1. Introduction

*"It is possible to make things of great complexity out of things that are very simple. There is no conservation of simplicity"*

*- Physicist and Computer Scientist Stephen Wolfram*

## 1.1 Basics

A Peer-to-Peer (P2P) system is defined as a network of a large number of nodes[1] that operate in a decentralized manner to provide services. Services provided are distributed applications that make use of the P2P system. Recently many distributed applications have been researched and developed [Rat02, Kel02, Dov03, Yia01]. Examples of this research include:

- *Replication-based Data Archive Systems* [Row01, Dru01, Dab01] whose goal is to provide distributed data storage services with high availability, reliability, performance, and scalability;

[1] The terms 'location', 'peer' and 'nodes' mean the same across the thesis.

- *File-sharing Systems* [Cla00, Nap01, Gnu01] whose goal is to provide an infrastructure where many nodes contribute storage and network resources to produce a pool of sharable files;

- *Application-layer Multicast Systems* [Ban03] whose goal is to provide wide area application content distribution services to enable one-to-many and many-to-many communications;

- *Event notification Service Systems* [Cab01] whose goal is to deliver information sent by event publishers to clients who have subscribed to those information events; and

- *Distributed File Systems* (DFS) [Fu00, Bol00], which relate to replication-based data archives, but are more focused to provide a seamless file system in a distributed system of peer nodes.

According to Tanenbaum *et al.* [Tan02] a distributed system is defined as *a collection of independent computers that appears to its users as a single coherent system.* By making use of this definition, a Peer-to-Peer Distributed File System (P2P DFS) is defined as a distributed system formed from a collection of independent nodes that provides seamless file system services to the user applications. A file system gives storage, retrieval and management services to the user files in a computing system.

## 1.2 Motivation

Historically, first generation *Peer-to-Peer* (P2P) systems, like Napster [Nap01], Gnutella [Gnu01] and Freenet [Cla00], among others, demonstrate the potential capabilities of P2P computing in terms of scalability, resource sharing (e.g. storage and network resources), and load balancing. First-generation P2P systems could grow to thousands of peers; share their resources (e.g. files), and balance the load across all peer members. These capabilities are long-term challenges for large distributed systems.

First-generation P2P systems provide workable schemes that have some limitations in their designs. Napster, for instance, relies on a central database to match peers looking for and providing a specific file. The selected peers later communicate directly in a "peer-to-peer fashion". The central database scheme is a limiting factor in terms of reliability, performance, and scalability. Gnutella and Freenet eliminate the central database, and instead rely on network flooding of messages to look for a specific file. The message is given a time-to-live (TTL) value that avoids a complete network flood. This scheme, although distributed, does not guarantee search success and makes poor use of network resources like bandwidth and packet messages, and peer's processing computations.

Recently, different independent research groups have developed second-generation P2P systems. Second generation P2P systems primarily focus on P2P search and look-up services [Bal02]. A commonality of these systems is that most of them use a hash function. The hash function receives as input a file identifier (e.g. file name) and produces a key. The key with high

probability uniquely and globally identifies the file within the system. Each node in the system is then responsible for a range of the hash function keys. The research group contributions differ primarily by two facts: the way the P2P algorithms map and assign the file keys into the set of peers and the way the P2P algorithms search for these keys. The main examples of these second-generation P2P systems are Chord [Sto01], Content Addressable Networks [Rat01], Tapestry [Zha01], and Pastry [Dru01].

The second-generation P2P hash-based search and look-up mechanisms provide a virtualization of the namespace [Kel02, Dov03]. The namespace virtualization is achieved when hashed file keys are mapped to the closest peers by comparing keys and hashed peer names (e.g. hashed IP address). The closest peer is responsible for storing and managing the file, instead of the original file source [Rat02]. In this way, files can be assigned anywhere within the P2P system. Hash functions evenly distribute output values that cause the files to be evenly allocated across peers. The hash-based search and look-up mechanisms usually have bounds on the number of hops utilized when searching and looking data in the overlay network of peers [Mil02]. An overlay network is a logical organized network on top of a physical network using end or intermediate peers [Dov03]. A hop occurs each time a network packet is forwarded to the next router. The more hops, the longer it takes to search and look-up data. Despite these advantages (i.e. namespace virtualization, hop bounds), several drawbacks have been identified for second-generation P2P mechanisms though. Some of the drawbacks are described next.

In terms of file attributes, the second-generation P2P mechanisms lose file locality and application information [Kel02, Har02]. Application information

is the relationships existing among files given by the application context. File keys can hash into any value of the allowed range so that the file destination can be any peer within the system. The even hash-value distribution causes related files to be dispersed throughout the system. This dispersion loses the files' spatial locality. Spatial locality refers to the physical proximity that related files usually have [Kel02]. The lost locality can bring high latency and increased network resource consumption when related files are searched and fetched. Latency is the time it takes when a file request is made until the file is available to the requester [Sar02a]. Also, file application information like data relationships that are usually expressed by hierarchies is lost. The relationship lost is because application related objects can be mapped anywhere. These drawbacks limit the capability of second-generation P2P systems to optimize pre-fetching, network routing, and efficient searching [Kel02]. Pre-fetching is reading ahead files that can be accessed in the future. Network routing is finding the best physical network path to move the file across [Sar02b]. Efficient searching is locating files for fast retrieval with a minimum number of resources [Har02].

Another drawback is related to peer's homogeneity. Second-generation P2P mechanisms assume peer's homogeneity for computing, storage and networking capabilities [Rat02]. The assumptions are widely used when the algorithms estimate complexity, number of hops and latency in a P2P network. However, there is a wide variation of computing, storage and networking capabilities per peer. It is common to find peers with several orders of magnitude difference in capabilities (e.g. networking bandwidth, processing power, storage capacity, etc.). There are a variety of peers participating in a real P2P system [Sar04]. A real P2P system is quite different from the P2P system usually used in a controlled environment such as a research or

academic lab [Lv02]. The peer differences could easily result in a bottleneck and source of inefficiency for second-generation P2P systems [Rat02]. For example, hot spots either in routing or file-provider peers can be created. A hot spot is created because the demand for service (e.g. reading a file) is greater than the capacity to provide the service. The hot spots limit the P2P system capability to fulfill file requests. Workarounds are usually developed to alleviate the hot spot scenario. For instance, caching and replication mechanisms workarounds along the file pathway have been developed [Sto01, Kub00]. These caching and replication schemes try to minimize hot spots in data-provider peers; however they ignore routing hot spots [Rat02]. Routing hot spots occur when overlay network peers are routing requests (e.g. read and write requests) from a source peer to a destination peer and the overlay network peers are overloaded with routing requests.

Previous hot spot drawback example illustrates a prevailing problem in systems research (e.g. distributed systems, operating systems, *etc.*) that is not particularly exclusive of P2P computing. Systems research has produced algorithms, models and solutions to many of its design and implementation challenges [Mil99]. Unfortunately, these algorithms and models are restricted when subject to computing environments that are dynamic, self-organized, *ad hoc* (e.g. in terms of connectivity, operatively, *etc.*) and decentralized [Kav99]. Some examples of these dynamic environments are P2P systems [Lib02], pervasive computing environments [Vah02], some grid systems (e.g. grids with no common domain across the Internet) [Cim02], and sensor networks [Zam04]. The limitations of the algorithms and models can be exposed in different ways like scalability, performance, and reliability issues, among others when subject to large distributed system environments [Sar04]. Workarounds are sometimes implemented to alleviate their limitations.

However, the fundamental problem is that these schemes are usually based on algorithms and models that have predefined centralized and distributed techniques. A different approach is required for the dynamic, self-organized, decentralized and *ad hoc* operation of large distributed systems like those usually found in a P2P environment [Mil02].

As briefly mentioned in Section 1.1, first and second generation P2P systems are recently being used as building blocks in a variety of new research efforts [Rat02, Kel02, Dov03]. The research efforts explore large distributed system applications running on top of a P2P system. *Distributed File Systems* (DFS) [Fu00, Bol00] are one active area within these research efforts (i.e. P2P DFS research). A DFS could be defined as a distributed system that provides an abstraction layer responsible for offering transparent data access and storage to user applications running in a local or wide area network of computers and storage devices regardless of location.

DFSs have been developed since the late 1970's. Initial DFSs like Locus [Wal83], Andrew [Sat85], Sprite [Ous88], Coda [Sat90], and many others [Bor89, Bur00] developed seminal DFS concepts. Some seminal concepts still apply and are used in modern DFS designs and implementations. For instance, the basic idea of file replication to increase availability and reliability is essential to any DFS, as is the use of caching as a mechanism to improve availability and minimize latency. Current DFS research tries to minimize or eliminate the use of servers, having a more distributed P2P-based structure [Fu00, Bol00]. The underlying P2P environment on which they are based directly affects the DFS' characteristics. A P2P DFS inherits drawbacks existing at the P2P layer unless workarounds are provided or a new computing paradigm for distributed computations is put in place.

*Multi-Agent Systems* (MAS) provide a computing model formed by a community of entities (e.g. agents, individuals, *etc.*). The entities are independent of each other, working with no specific leadership, but with a local goal that translates into collective result(s). This changes the way we see things because domain community goals are not results of special entities but are the comprehensive, emerging global results of the sum of each of the entities' actions that are frequently executed independently from each other.

Researchers have used natural and economic models to design MAS algorithms to solve different kinds of resource management and optimization problems. For example, a common natural model used is that of ant behaviours [Bon00a, War98]. Although each ant undertakes very simple activities, the global outcome of the collective accomplishes goals that are very difficult without the ants behaving indirectly as a team. Ant behaviours are an example of what is known as *Complex Adaptive Systems* (CAS) [Mon02a].

CAS have a large number of members with simple functions and limited communication among them. They exploit their synergy to solve complex problems and accomplish global behaviours. The *Swarm Intelligence* [Bon00b] that emerges from the global behaviours of simple members boasts autonomy and self-sufficiency, which allows them to adapt quickly to changing environmental conditions. It is called "swarm" because of the large number of members that participate. Several other insects and animal societies have been studied [And00] where emergent team or group behaviours exist. The interesting aspect of CAS is that they present team or group models that solve complex distributed problems, which otherwise would be difficult with deterministic, centralized or distributed techniques.

CAS are characterized by the decentralization, self-organization, *ad hoc* operation and fault resilience of their members. These characteristics make CAS a suitable model for research to develop a new computing paradigm for large distributed systems that could potentially overcome the existing limitations of current distributed techniques. Therefore, this research thesis introduces a new computing paradigm called the *Emergent Thinker*. The Emergent Thinker is an area-wide logical computing entity, named after a philosopher that continuously analyses information and has emergent computed solutions for new and/or living requests. Living requests are those computations that the Emergent Thinker continuously works on. Computations are emergent because they arise as result of the CAS member activities.

The CAS-based Emergent Thinker computing paradigm is applied to the design and implementation of a P2P DFS thereby providing more efficient data access and storage. It is more efficient because it supports the decentralization, self-organization, *ad hoc* operation and fault resilience of the peer members. The Emergent Thinker is used on the P2P DFS research and development and helps to overcome some of the existing limitations of first-generation and second- generation P2P systems such as those described previously. The Emergent Thinker computing paradigm applied on a P2P DFS results in the *Biologically Inspired Peer-to-Peer Distributed File System* (BPD). The BPD is biologically inspired because the CAS based algorithms utilized are based on natural biological behaviours of squirrels.

## 1.3 Shortcomings of current P2P DFS

Many *Large Distributed System* (LDS) environments are characterized by the dynamic, self-organizing, and *ad-hoc* connectivity of its members (i.e. peers, nodes, locations, *etc.*). Examples of these environments are unstructured P2P systems [Mil02, Lv02], pervasive computing environments [Vah02], sensor networks [Zam04], and some grid systems [Cim02] (e.g. grids that have no common domain across the Internet), and so forth.

The only constant is the variability of the members' connectivity to the LDS. One moment a member can be present and the next unavailable either because the member is turned off, the member fails, or the member is not connected [Ghe03]. Dynamic LDSs are more common in the Internet community and large organizations as compared to other static, structured LDS schemes commonly documented in the literature (e.g. [Rat01, Sto01, Zha01]).

P2P systems can be classified as either structured or unstructured systems. Structured P2P systems usually have stable peer configurations with few or no changes in the number of peers. Conversely, unstructured P2P systems are dynamic, heterogeneous and *ad-hoc* in the operation and connectivity of their peers and variable in peer characteristics.

Many algorithms developed in the second P2P generation [Sto01, Rat01, Zha01, Dru01] and the applications that make use of them, assume the existence of a structural P2P system in their designs. The assumptions regarding the peer stability (in terms of operability, connectivity, etc) and peer homogeneity (in terms of equal configuration and capacity) have created doubts about the feasibility of the algorithms and applications when subject to

dynamic, *ad-hoc* environments that are characteristic of unstructured P2P systems and are commonly used.

It is unclear how and what second generation P2P algorithms and applications are suitable or appropriate for unstructured P2P systems [Sar04, Lv02, Ghe03]. Unstructured P2P systems are commonly found on the Internet and in large organizations. The fundamental problem is that second generation P2P algorithms and applications are based on schemes and models that have predefined centralized and distributed techniques. Second generation P2P systems assume stable environments with either fixed or controlled member growth. Workarounds are commonly used to fix them. For example cache mechanisms are heavily used to minimize latency when data is read. However, when subject to dynamic contexts, the algorithms are brittle, un-scalable, and constrain to provide services with limitations.

A P2P DFS, as an application developed on top of a P2P system, will inherit second-generation P2P drawbacks unless special workarounds are provided. Such a P2P DFS will have limitations in managing its distributed file services effectively in decentralized, self-organized, and *ad-hoc* environments. Such P2P DFSs are limited to providing appropriate fault resilience when members fail. This thesis makes two related contributions described below.

First, the thesis provides a novel paradigm to obtain distributed computation. The new paradigm is named the *Emergent Thinker* paradigm. Chapter 3 describes the Emergent Thinker. The Emergent Thinker paradigm uses CAS models to provide emergent computations. The CAS models utilized by the Emergent Thinker paradigm can be based on metaphors coming from sciences like biology, cognitive science, economics and related areas. Due to its reliance on CAS models, the Emergent Thinker paradigm handles aspects of

decentralization, self-organization, and *ad-hoc* operation to provide scalable and reliable distributed computations in large distributed systems. These are precisely the characteristics addressed in a P2P DFS implemented in unstructured P2P systems described in this thesis.

Secondly, the thesis investigates and develops a *Biologically Inspired Peer-to-Peer Distributed File System* (BPD). The BPD is the main topic of the thesis. The BPD instantiates the Emergent Thinker paradigm and consequently tackles the problems of DFSs, which have been implemented on top of second generation P2P systems for use in LDSs. In the BPD, novel CAS algorithms based on squirrel behaviors are developed. The biologically-based CAS schemes are highly adaptable and scalable across a variety of P2P system characteristics. The CAS adaptability and scalability are suitable for the dynamics of unstructured P2P systems. The BPD represents a novel P2P DFS design and implementation that merges CAS models with P2P computing. The BPD uses CAS models for distributed services including storage, retrieval, search and replication. Further, the BPD utilizes commodity peers for storage and retrieval in a complex dynamic environment.

The BPD contributions go farther and demonstrate how systems research as a discipline could benefit from the Emergent Thinker paradigm. Systems research, as a branch of Computer Science, aims to design and implement novel architectures and information systems. The Emergent Thinker paradigm offers a large experimental area to expand current system research. The BPD touches only a small piece where the paradigm can be applied within operating system file services. The operating system provides schemes and mechanisms to properly use computer system resources. The generic term *operating system* may refer to either a centralized or distributed operating system. An operating

system has many service areas where the Emergent Thinker paradigm could be instantiated. For example program execution, security, error detection and response, system accounting, *etc.* to mention only a few. These areas could use a methodology similar to the one employed in the BPD to achieve emergent computations that facilitate building functional blocks for their services in a LDS. Eventually a complete computing device based on CAS emergent computations might be developed. The more the research community can experiment with CAS models, the more the research community will understand their complexity and applicability in systems research. The BPD provides one example of how this can be accomplished.

## 1.4 A Biologically Inspired Peer-to-Peer Distributed File System

This thesis introduces the *Biologically Inspired Peer-to-Peer Distributed File System* (BPD). The BPD is based on MAS-based CAS algorithms and methods to implement essential P2P DFS services. P2P DFS services are system calls that user applications use for their file management and access (e.g. file read, file write, *etc.*). The BPD uses the Emergent Thinker computing paradigm as its architecture model. The Emergent Thinker paradigm presents a novel architecture to provide distributed computations in large distributed systems.

The BPD also introduces original CAS algorithms whose swarm behaviours provide emergent computations to the BPD components. This approach offers a sustainable alternative for providing distributed computation. It is sustainable

because the BPD copes with the decentralization, self-organization, *ad hoc* and dynamic operation of large distributed systems usually found in a P2P DFS. The BPD capabilities, which are dealing with the dynamic characteristics of large distributed systems, are achievable with the utilization of the Emergent Thinker paradigm that otherwise could be difficult to obtain with traditional approaches or techniques. Furthermore, the BPD enhances CAS algorithms with the use of specially designed distributed data model schemes to implement DFS data structures.

BPD presents a new design approach for a DFS on top of a P2P system. BPD experimental results show that reliability and scalability for DFS services in large P2P systems are maintained. The results are promising because they are achieved across many P2P characteristics such peer storage capacities, number of peers, peer storage demands, peers' heterogeneity, *etc.*

P2P computing could potentially offer scalability, load balancing and resource sharing that can be exploited by a DFS. However suitable models must be in place to make them achievable. CAS based algorithms developed in this research have provided models with emergent solutions that alleviate existing P2P issues for scalability, decentralization, and *ad hoc* operation while providing DFS services with reliability. Experimental results indicate that service reliability is obtained with good resource management and load balancing.

The major contributions of this thesis can be summarized as follows:

1. A new computing paradigm called the Emergent Thinker based on complex adaptive systems emergent computations is reported.

2. A novel DFS on top of a P2P system called the Biological Inspired Peer-to-Peer Distributed File System (BPD) is designed and implemented. This design and implementation make use of the Emergent Thinker paradigm and CAS based algorithms modeled after the natural biological behaviors of squirrels.

3. Demonstrates the design of the P2P DFS is correct and functional.

4. Novel CAS algorithms and distributed data model schemes have been proposed and their performance evaluated within a P2P DFS design implementation.

5. Experimentally proves that novel CAS algorithms guarantee an emergent solution to DFS functions and services.

6. A next generation P2P DFS system design that considers well-balanced resource management.

7. Demonstrates the application of a DFS while providing file management services in a dynamic, *ad hoc*, decentralized and scalable P2P system.

## 1.5 Outline of Thesis

An essential proposition of the thesis is that biologically inspired mechanisms provide adaptive complex adaptive system models and schemes. These models

and schemes can help alleviate the rigidity of preconceived algorithms and schemes commonly found in distributed systems and their applications. The distributed systems' rigidity creates limitations on services' scalability, brittleness and reliability among other things.

The case is made for the design and implementation of a P2P DFS application called the Biologically Inspired P2P DFS (a.k.a. BPD). Research is done to design and develop the BPD. CAS models and schemes are investigated to achieve biologically inspired squirrel algorithms for distributed data allocation and search. Experimental research by means of software simulations and a BPD prototype are developed to sustain the thesis. Results corroborate our hypothesis and demonstrate the CAS models and schemes feasibility in distributed system computations. The following chapters give details of the BPD design and implementation carried out by the thesis work.

CAS emergent computations are a consequence of many factors related to the CAS members and their environment. This research focuses on the CAS members' behaviors, mechanisms and interactions to explore and sustain my hypotheses. This work has utilized a MAS approach to first understand and establish CAS knowledge and its application to distributed systems, and it assumes that the connectivity among CAS members to be in place at the application layer.

Future extensions of this research might consider physical connectivity, communication paths and other physical communication constraints that could have an impact on CAS emergent computation performance. However this is out of the scope of this thesis and is left in the realm of future research. Section 8.1 discusses some insights about how the 'physical networking' factor among CAS members might be included in prospective work.

The rest of this thesis is organized as follows:

- Chapter 2 extends this introduction by providing fundamental definitions and related work relevant to this thesis. Interdisciplinary related work taken from Operating Systems (i.e. DFS), Distributed Systems (i.e. P2P) and Complex Adaptive Systems is summarized.

- Chapter 3 introduces the Emergent Thinker paradigm to provide emergent computations in large distributed systems. The CAS Emergent Computation and the CAS Propagation Models are reported as building blocks for the novel Emergent Thinker paradigm. CAS complexity and the Emergent Thinker utilization to provide relative computing on hierarchical and interrelated CASs are also discussed.

- Chapter 4 presents the *Biologically Inspired Peer-to-Peer Distributed File System* (BPD) design, describes details about how all components fit together and how it makes use of the Emergent Thinker model. The chapter reports the BPD design architecture including an architecture stack overview and constituent layers, the BPD peer network, the API, and the control tables. Data consistency and mobility are also discussed.

- Chapter 5 describesthe CAS algorithms used within BPD and describes how they achieve satisfying emergent solutions to assigned DFS services and functions. The chapter details the CAS allocation and discovery algorithms for the implementation of the BPD emergent computations.

- Chapter 6 describes the implementation of the BPD design and related issues. Two software simulators are designed and implemented. The simulators are test beds for the BPD CAS allocation and discovery algorithms respectively. The BPD prototype implementation results are reported. The BPD protocols and stack layers implementations are also described.

- Chapter 7 covers the performance of CAS algorithms in terms of scalability, reliability and availability of data for the BPD. Experimental results are shown for large CAS simulations and then results in the BPD implementation. The results are shown for a variety of P2P characteristics. The two simulators and the BPD prototype described in Chapter 6 are exercised to carry out the experiments.

- Chapter 8 makes some concluding remarks, gives a summary of contributions and suggests some directions for future research.

# *Chapter 2*

# 2. Background

*"It is the best possible time to be alive, when almost everything you thought you knew is wrong..."*

*- Playwright Tom Stoppard*

This thesis deals with interdisciplinary work drawing from the fields of Operating Systems (specifically DFSs), Distributed Systems (specifically P2P), and Complex Adaptive Systems. Therefore, the background section is divided into discussions of some important definitions together with reviews of related work on each of these topics.

## 2.1 Distributed File Systems

For more than two decades the interconnection of computers (servers, clients) and storage devices by means of a network has been a fruitful research area. Operating System (OS) researchers have created a variety of solutions to the challenges and problems posed by many issues related to computer and storage device interconnection [Yia01, Dog98]. Required information to

conduct business resides anywhere in the enterprise or in dispersed geographical locations [Pap93, Man99, Has00]. Users need file access anywhere in the network for files with information that may be located anywhere in the system [Gri99, Van99]. Such ubiquitous file access across the network poses the challenge of offering transparent local and remote file access to users. It is transparent file access because the user is not aware of the file location [Whi01]. A Distributed File System (DFS) provides the mechanisms and services for the user applications that are accessing files, which reside somewhere within the system [Wal83, Sat85, Bir93] (Figure 2.1).

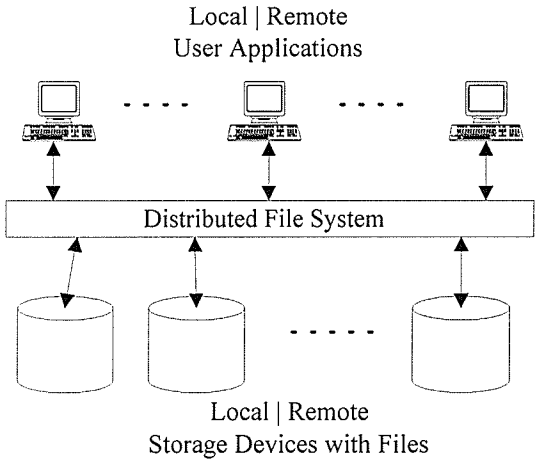


Figure 2.1: A Distributed File System Abstraction Layer

User applications can be anywhere in the network and the files can, therefore, be stored either locally or remotely relative to their users [Wal83]. A *local area network* (*LAN*) or a *wide area network* (*WAN*) interconnects the storage devices and computers. If the user applications are aware of the file location then a *network file system* (*NFS*) is in place. If the user applications do not

know the file location then a *distributed file system* (*DFS*) is providing services. In this thesis, the application awareness of file location is what differentiates an NFS from a DFS. A DFS provides an abstraction layer (Figure 2.1) that hides all the complexity of where and how to access and store a file in a distributed system of computers and storage devices [Lev90]. The applications only make requests for file services, and the DFS will provide those services on behalf of the storage and computing devices [Tan02].

### *2.1.1 DFS Characteristics and Concepts*

Some DFS characteristics and concepts are now described. These characteristics and concepts make designing and building a DFS an exciting research field when it is implemented on top of a dynamic, large distributed system like a P2P system.

#### *2.1.1.1 Naming Scheme*

A naming scheme determines how file names are structured. A mapping scheme then translates logical names, possibly indirectly, to physical locations [Sat85, Bak02]. In a local file system, this mapping is relatively straightforward because all physical devices usually reside within the boundaries of the same computer. However, in a DFS, files may reside anywhere in the network and the naming scheme must be robust enough to provide location transparency and independence [Wal83, Sat85, Lev90, Bir93, Whi01]. Location transparency means that the logical name does not give any hint as to where the file resides. Location independence means that the file can be moved anywhere within the system and the logical name should not have to change [Gri99, Whi01]. Mapping is complex because physical data locations may be updated continuously as files are stored and accessed from remote and

local applications. Consequently naming scheme scalability is important because good performance is required to execute fast mappings when the applications access data for storage and/or retrieval. Naming scheme scalability is defined as the DFS capability to support large number of mappings.

*2.1.1.2. Application Programming Interface (API)*

The API provided by a DFS should be analogous to that provided by a regular File System (FS) [San85, Lev90]. The API similarity helps to simplify porting applications from a regular FS to a DFS and vice versa. The DFS API should provide at least the same functionality provided by a regular FS but in the context of a distributed system of computers and storage devices [Wal83]. It is challenging to provide equivalent DFS services to local FS services when files can reside anywhere on many interconnected devices with multiple applications simultaneously storing and accessing these files.

*2.1.1.3 Integrity*

Data integrity schemes help ensure that data files are not corrupted when are accessed by user applications. A file may reside remotely and be accessed concurrently by multiple applications. Furthermore, the file most likely is divided in blocks, which could reside in different storage devices physically dispersed throughout the distributed system [Mor02]. The DFS must ensure that the file's integrity is not jeopardized [Sat85]. For instance, when several applications update or access a file, the data should reflect the last change within the distributed system. Otherwise, there is a risk that different applications could change the same data simultaneously when accessing the file, thereby corrupting the DFS [Lev90]. The file image seen by all applications should be the same at any given instant in time.

#### *2.1.1.4 Concurrency Control*

Several applications could access files or a single file simultaneously. A DFS must ensure that when several applications update the same file, they avoid file system conflicts and corruption [Lev90, Bak02]. The DFS concurrency control mechanisms guarantee that file data integrity is maintained. The challenge is to maintain these controls when various applications are reading and writing the same file [Bir93, Van99]. The DFS must reflect file updates to all interested applications with minimum delay thereby avoiding file corruption and inconsistency. Interested applications are those applications concurrently accessing the same file(s).

#### *2.1.1.5 Security*

Security schemes must be robust to ensure that only applications with correct permissions are accessing files [Sat85]. The DFS security complexity, unlike a regular FS security, arises because security is not only required at one centralized point of access, but at multiple access points within the system, which most likely are far away from each other [Bir93, Whi01]. Research to develop a security scheme for large distributed systems that is secure with acceptable performance but that is also flexible and scalable is an ongoing research problem [Sat85, Bir93, Whi01]. Having acceptable performance means to detect, correct and protect against computer attacks (e.g. intrusions, viruses, *etc.*) with efficient use of resources (e.g. computing time, network bandwidth, *etc.*). Flexibility means to be able to adjust the security schemes to new security threats as they appear [Sat85, Gri99], whereas scalability requires coping with distributed systems growth while providing the same or better security. Security becomes even more intricate when the file is partitioned, possibly replicated, and scattered throughout the distributed system [Van99].

*2.1.1.6 File Location*

Within a regular FS, file location is simpler than DFS file location as files reside within the same computer. In a DFS, the files can be anywhere and can be accessed from any point within the system [Sat93]. A DFS could place a file wherever it will enhance the performance and scalability of the system. Thus, to provide file location flexibility, the DFS must address some challenges. For example, when a file is located far away in the network from its access point(s), it is more expensive than having it nearby because more resources (e.g. network, computing processing, storage, *etc.*) are used to move it to the place it is needed [Sch84, Van99]. A DFS design must consider resource efficiencies [Liu00]. If all file data is located remotely, file accesses from applications could clog the network causing high latencies [Bak02]. There is a trade off between how much data should be located locally and how much should be located remotely because it has an impact on other DFS characteristics as well (e.g. security, availability, concurrency control, *etc.*) [Sat85]. The simple question of "where to put the data?" is a challenge for DFS researchers, because this design decision will have an impact on system performance and scalability. Resource utilization is another challenge; a file location scheme can impact how to better utilize existing resources in addition to affecting how efficient backup and restore schemes are for file recovery. These decisions translate into a more or less effective system in terms of support and maintenance.

*2.1.1.7 Availability*

Availability means having file access whenever the user applications require it [Sat85]. Availability is limited when the network is partitioned and related file data is dispersed across network partitions or there is a single copy, unless techniques like caching and replication are used [Sat85, Van99, Kuz02]. Data

caching and replication techniques help to ensure 24x7 data availability within a large distributed system [Wak83, Liu00]. Disk caching techniques try to load into memory the file data that most likely will be accessed by applications [Bak02], whereas replication techniques try to copy the file data to other locations so that if the original location fails, then there is another location where the file can be accessed [Kuz02]. Both caching and replication schemes must consider integrity, concurrency control, and security characteristics [Van99]. These can be seen as contradictory features so the challenge is how to trade off among them to provide a highly available DFS. For instance, if the number of replicas is large, then ensuring that file integrity is never compromised is expensive in terms of resources (e.g. network messages, computations to validate data, *etc.*) needed to provide file integrity [Bak02].

### *2.1.2 DFS Evolution*

This section briefly overviews a number of previous DFSs and introduces a DFS taxonomy spanning three generations according to their contributions:

- First Generation (1980's → early 1990's). Seminal concepts and algorithms are introduced.

- Second Generation (early 1990's → mid 1990's). Seminal concepts and algorithms are improved with some degree of innovation.

- Third Generation (late 1990's → today). Previous concepts and algorithms are enhanced and new ideas and larger innovations are proposed, primarily due to the development of new technologies in other related fields (e.g. networking, storage and computing processing).

*2.1.2.1 First Generation DFSs (1980's -> early 1990's)*

This generation developed seminal DFS concepts. DFSs were limited in the scope and number of services offered (e.g. Locus [Wal83], Andrew [Sat85]); nonetheless later DFS implementations followed first-generation designs for the next two decades. Client-server architectures typify these DFSs. Servers controlled important DFS management aspects (e.g. storage, access synchronization, version control, *etc.*), and clients made requests to these servers. Locus [Wal83] also deployed a basic peer-to-peer architecture in its design though. Client-server DFS architectures imposed limitations on scalability but the DFS architectures were implemented over several networked computers, which alleviated the limitation to some degree. These implementations were characterized by client and server OS homogeneity, typically based on UNIX. Manual procedures were common when performing several DFS management activities. For instance, in the Andrew DFS [Sat85], manual procedures are required when a user moves to another workgroup of stations. Security schemes are rudimentary or non-existent as they assume a trusted network environment, although the Andrew DFS [Sat85] implemented basic security. Seminal ideas for file replication, caching, and global hierarchical naming schemes first appeared during this generation.

*2.1.2.2 Second Generation DFSs (early 1990's -> mid 1990's)*

Second generation DFSs continued utilizing the initial client-server architecture (e.g. Coda [Sat90], Frangipani [The97]). This generation improved some of the deficiencies or limitations of seminal DFS. Resilience to failures was improved by means of stronger file replication and caching mechanisms. This facilitated network fragmentation support and mobile users (e.g. Coda [Sat90]). Scalability was addressed by defining better algorithms that allowed growth as required by the environment. Fault tolerance was also

improved by replication support [Sat90]. Second-generation DFSs still assumed a more or less homogenous-networked computer environment. Automation of system administration was incorporated to ease administrative activities (e.g. adding, removing or updating servers). Load balancing was also introduced to improve performance beyond that achieved by file replication and caching schemes alone.

*2.1.2.3 Third Generation DFSs (mid 1990's -> today)*

Previous DFS concepts evolved in a number of ways reflected in systems such as JetFile [Gro99], xFS [Bol00], and Pangaea [Sai02]. One of the most important is the base architecture moved from a traditional client-server to a peer-to-peer model. The server is almost eliminated or its utility minimized in some third generation DFSs. Thus, any computer with resources within the DFS can provide FS services to others and access FS services from others. Computer system heterogeneity is also supported (e.g. Windows, Linux, Unix, *etc.*) so a broader range of systems is capable of connecting to the DFS.

Scalability was also expanded; there are DFS computers with resources working anywhere, not only at a well-controlled environment (i.e. a closed setting where all computers participating in the system are under same domain and/or behave neatly without major problems) as in previous generations. This generation does not assume a trusted network environment anymore. This required that existing security schemes be improved and made efficient. Security techniques were either improved or created using such techniques as digital signatures, cryptography, and encryption of stored files.

File replication and caching were also improved to provide better availability and reliability as required by the new heterogeneous environment. Optimistic storage and concurrency control strategies are now commonly used to improve

DFS efficiency. DFS administration is automated as much as possible to eliminate or minimize the need for manual intervention. The implementation of a serverless DFS demands new paradigms in DFS design [Wan98] to provide distributed services. Some paradigm examples include formal verifications, threading mechanisms, fast messaging layers, and new kernel interfaces among others. The new paradigms make DFS implementation quite challenging because the algorithms and schemes are distributed and concurrent.

## 2.2 Complex Adaptive Systems

*Complex Adaptive Systems* (CAS) are characterized by having a large number of members with simple functions and with limited communication among them [Res94]. They exploit their synergy to solve complex problems and accomplish global behaviors. The global behavior of such systems is an emergent result or outcome of the simple member activities (Figure 2.2). *Emergence* is a fuzzy CAS concept [Dec01]. Using definitions from related literature [Dec01, Sim96, Hey03, Ger04] and a dictionary [Hou82], *Emergence* is defined as the appearance (unpredicted or predicted) of new collective characteristics or phenomena in the course of biological or social interactions.

The *swarm intelligence* [Bon00b, Tar02] that emerges from the global behaviors of simple members boasts autonomy and self-sufficiency, which allows the members to adapt quickly to changing environmental conditions. Swarm intelligence is called 'swarm' because of the large number of members

that participate. The independent activity of the CAS members provides autonomy and self-sufficiency since no leader or central control is required. Members can appear or disappear; their own CAS dynamics adjusts the number of members. Each member executes locally and its action is propagated to others and/or to the environment (either directly or indirectly).

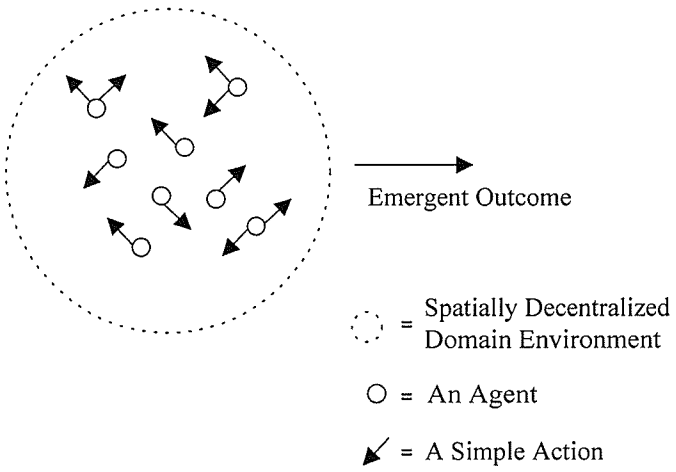


Figure 2.2: A Generic CAS

Figure 2.2 shows a depiction of a generic CAS. The agents (a.k.a. members) execute simple actions independently. There is limited or no communication among agents [Whi02]. A common form of communication among agents is defined by *stigmergy* [Mon01]. Stigmergy is a term used to describe a form of indirect communication mediated by changes in the environment [Dic97]. It is indirect because no direct interaction exists between the agents. Stigmergy is divided into *sematectonic* stigmergy and *sign-based* stigmergy. Sematectonic stigmergy occurs when physical changes in the environment are used to communicate information with no direct interactions between the CAS agents. Sign-based stigmergy occurs when something is placed in the environment

and then affects the behavior of future activities of the CAS agents. Other forms of communication could also exist (e.g. direct exchange of small amounts of information). The BPD (Biological Inspired Peer-to-Peer Distributed File System) biological models use stigmergy and are further described in Chapter 5.

All agents exist within a spatially decentralized environment as in Figure 2.2. The domain is artificially set-up to define a territorial scope for the agents and so they are easier to conceptualize. The emergent outcome from the domain environment is said to be self-organized [Tar02]. It is self-organized because it stabilizes to a relatively steady point in the system's dynamics. It is like taking a snapshot in time. The continuity of the member activities could take the CAS into other self-organized state. As time progresses in the life of a CAS, each of the member activities can stay the same, increase, decrease, or stop. The CAS as a whole is continuously self-adapting [Bon01]. The CAS emergent outcomes are the result of this self-adaptation and can then be utilized to provide emergent computations for large distributed systems. This idea is further expanded in Chapter 3 when the Emergent Thinker model is described explaining how emergence provide computing solutions in large distributed systems.

Emergent computations can be used to solve complex distributed problems, which would be difficult to solve using deterministic, centralized or distributed techniques [Bon00b, Bon01, Tar02]. Some CAS models have been applied successfully to different problem domains. Ant behavior based CAS models have been applied in the telecommunications routing domain. White [Whi97] describes a search process that solves telecommunications network routing problems. His work is based on ant models and uses genetic algorithms to

generate adaptive behaviors. Bonabeau *et al.* [Bon00b] describe how ant behaviors can be used to solve the classical traveling salesman problem effectively, but not necessarily with the best solution. These examples illustrate an important aspect that has been observed in CASs. Due to their inherent randomness and unpredictable emergent behaviors, CASs do not guarantee the best outcome possible. This can be seen as a disadvantage or as a strength from the perspective that a sufficient, though non-optimal, solution can emerge cost effectively.

Dorigo *et al.* [Dor96] also applied ant behaviors to the traveling salesman problem and other combinatorial optimization problems. Dorigo *et al.*'s work is characterized by the use of positive feedback that optimizes the search domain. Solnon [Sol02] blended ant CAS algorithms with local search techniques and pre-processing steps to solve generic constraint satisfaction problems. Deadman *et al* [Dea00] based their work on CAS to simulate common pool resource management problems and to explain the consequences of individual actions on these common pools. The Anthill project [Bab02, Mon01, Mon02b] applies ant foraging behaviors to design, implement, and evaluate peer-to-peer applications. Anthill's metaphor of ant foraging has also been applied to CPU load balancing [Mon02b].

Bonabeau *et al.* [Bon01] used a number of fundamental CAS characteristics (e.g. flexibility, robustness and self-organization) and explored potential applications on business processes and how to obtain a self-organizing enterprise that can adapt quickly to fast evolving markets. Bourjot *et al.* [Bou03] introduced a new swarm model based on *social spiders*. Their contribution is the inclusion of non-local information being brought into local processing, which is different from other ant colony optimization approaches.

White *et al.* [Whi02] showed how social insect metaphors can be used in the management of mobile agent systems. They addressed two aspects of swarm population management: maintenance of population density and upgrading agents over time.

An alternative metaphor to CAS in resource allocation and optimization problems is the use of economic models instead of biological models. Gibney *et al.* [Gib98] proposed a framework of self-interested agents that have limited communication and context information, but use a market economy model to exchange information and allow cooperation between them to solve the allocation problem of telecommunication paths using sellers and buyers of communication resources. Similar work by Kuwabara *et al.* [Kuw96] enhances the market economy model by proposing using learning in the agent's behaviour to improve the decision process between buyers and sellers. The consequence is a reduction in oscillations during resource allocation. Chavez *et al.* [Cha97] propose a similar model but apply it to the allocation of processor resources, with learning abilities that adapt more appropriately on load fluctuations and message delays. Kurose *et al.* [Kur89] analyze some microeconomic decentralized algorithms to allocate file resources. They achieve an optimal allocation through a series of iterations that consume resources (communication, storage) and take time to converge. However, the number of constraints they include in their simulation models limits its application in real contexts. Gelfand *et al.* [Gel02] use a MAS approach to trade space in a DFS by applying some learning processes in the agents. This approach provides allocation fairness among peers to avoid some peers consuming too many system resources.

The variety of metaphors in CAS models exemplifies the ubiquity of complex systems in our lives [Bar97]. Although only some examples have been reviewed here (related to biological and economic metaphors) many other examples exist in other domains like world ecosystems (e.g. desert, rain forest, ocean, *etc.*), the brain, a corporation, a society, governments, the human body (e.g. physiological and psychosocial perspectives), the weather, *etc.* The solutions that emerge from the simple activities of a swarm's members and the similarity with other natural complex systems (e.g. ecologies, brains, social systems, immune systems) have created a growing interest from the research community to explore CAS models as mechanisms to achieve solutions that otherwise would be difficult to obtain for complex problems [Bon00b, Tar02]. Interest has risen from areas as diverse as artificial intelligence, cognitive sciences, computational economics, mathematics, optimization, biology, psychology, neuroscience, and engineering [Vos03].

This thesis uses CAS models as an alternative approach to existing distributed system methods and expands current CAS work by using biological models that are based on squirrels' behaviors. A squirrel-hoarding behavior inspire a biologically CAS model that provides interesting global emergent outcomes.. Many groups have studied the hundreds of squirrel species found worldwide [Shu01, Con00]. Squirrels hoard various nuts, acorns, and other small foodstuffs by caching acorns in small hoards over a large geographic area. "*Acorns*" will be the term used in this thesis to refer to all foodstuffs collectively. This is acceptable because it is the consequence of the hoarding activities, not the objects being hoarded that is of interest. When food becomes scarce the squirrels return to their caches. Their failure to recover a great percentage of the horded acorns also facilitates the growth of new trees. Squirrels, in conjunction with Jays [Shu01], are responsible for the vast oak

extensions through North America after the glacial age 10,000 years ago. This is an example of individuals' simple behaviour leading to the reforestation of vast geographical extensions. From another perspective, squirrels are allocating resources (land space) to storage demands (acorns) in such a way that resources are balanced. The global consequence is a populated forest of oak trees across a wide geographic area.

The most interesting squirrel behavior is hoarding, rather than foraging, as hoarding leads to a global outcome of which the individual squirrels cannot possibly be aware. The interest is in the way squirrels spread out acorns, nuts and other small food pieces in an area to obtain a resource allocation and dissemination. This thesis explores how to exploit these techniques to design and implement resource allocation and data dissemination suitable for distributed systems algorithms. The algorithms are used for DFS P2P data storage and search services. The following chapters detail the algorithms and their use in the BPD.

Squirrel hoarding behaviors can be summarized as:

- random gathering and burying of small acorns in an area geographically close to its nest;
- investigation of various random locations ("sniffing" several places) before deciding where to put the acorn;
- possibly deciding not to hoard its food if there are other squirrels around; and

- possibly working in small teams with others with whom they are familiar, while avoiding places inhabited by unknown or 'strange' squirrels. Strange squirrel is defined as a squirrel not belonging to the squirrel team either from the same location or another location.

## 2.3 Peer-to-Peer Computing

*Peer-to-Peer* (P2P) computing offers unique characteristics and advantages that provide alternatives to classical client-server architectures for data retrieval, storage, and management [Par01, Mil02, Tal03]. For example, Aberer *et al.* [Abe02] describe an efficient P2P binary search algorithm, but its assumption of uniform data allocation on peers limits its applicability to a real environment. JXTA [Wat02, Sun03] offers a network-programming platform with a set of protocols specifically designed as a foundation for P2P systems. Useful algorithms for distributed search and services enhancement for P2P systems have been analyzed [Wat02, Lie02]. Replication services to increase availability in P2P have also been explored [Sai02, Ran02, Gee02]. The adaptive model presented by Ranganathan *et al.* [Ran02] offers interesting insights about how a completely distributed scheme can offer solutions to the availability problem in P2P networks by dispersing data. Geels [Gee02] provides replication next to the client access point with a dissemination tree structure to support replication.

Casassa *et al.* presents a hybrid P2P architecture that makes use of a centralized trusted control component [Cas01]. This system is adaptive to peer behaviour since it is responsive to the assessment of their trustworthiness and

reliability by means of this central component. Its scalability is questionable though because of the central component. Talia *et al.* [Tal03] overview the commonality and synergy between grid computing and P2P for security, connectivity, access services, resource discovery and fault tolerance. P2P offers an alternative to grid computing by providing distributed schemes that can reduce the centralized schemes commonly used in grid implementations (e.g. the Globus toolkit [Cim02]). Ciminiera *et al* [Cim02] compares P2P with Grid implementations in terms of discovery, management and description of resources, in addition to providing a comprehensive survey of these technologies and some of their implementations.

P2P distributed computing based on mobile agents has recently been explored [Kle02, Pen02, Bon02]. This work has in common the design and implementation of different P2P services and functions by means of agents, which are distributed across the P2P environment. The Anthill project [Mon02a, Bab02, Mon01, Mon02b] uses CAS and P2P computing to provide a framework that could be used to develop P2P applications. Anthill uses ant behaviours to develop its framework and is applied to CPU load balancing [Mon02b]. Anthill primarily focuses on P2P services like security, communication, neighbor management, and scheduling.

This thesis leverages CAS and P2P computing to provide a novel paradigm for distributed computations called the Emergent Thinker (Chapter 3). The Emergent Thinker [Cam04c, Cam05b] provides a model that offers emergent solutions to complex distributed environments. It makes no assumption about the underlying distributed system environment (e.g. P2P environment). The BPD [Cam04b, Cam05b] instantiates the paradigm and uses squirrel hoarding behaviors as its CAS metaphor. The squirrel-based hoarding approach reflects

a better way to allocate and disseminate resources than existing foraging approaches such as the Antill project [Mon02a, Bab02, Mon01, Mon02b] because it is a more natural reflection of this activity [Cam03a, Cam03b, Cam05a]. The allocation and dissemination of resources are used to store and search for data in a DFS [Cam04a, Cam04d]. The Emergent Thinker paradigm is highly flexible since any CAS metaphor can be utilized. Furthermore, the paradigm can be applied to other research problems in distributed systems like aggregation, scheduling, assignment, *etc.*

Recently several P2P DFSs have been investigated and/or implemented [Mut02, Ric02, Ste02, Bro02]. This suggests that P2P together with other application areas such as DFS are interesting research fields. These research efforts attempt to apply DFSs in a P2P context offering new and/or enhanced capabilities. The schemes implement the DFS services described in Section 2.1.1. Contrary to this work, this thesis presents a novel scheme based on CAS models to provide emergent distributed computations, and then utilizes emergent computations on top of a distributed system, which could be either a P2P system or any other type. We now overview some related P2P DFS research. The intention is to show representative work and their main contributions.

The Ivy P2P DFS developed at MIT [Mut02] provides a novel DFS implementation scheme based on distributed logging to track DFS activities. Although it permits maintaining distributed metadata consistency without locking, it pays a penalty for supporting this scheme because its performance suffers. Based on their results, it is two to three times slower than Sun NFS [San85]. Due to its reliance on logging mechanisms, Ivy operates best on fully connected networks. Furthermore, Ivy periodically constructs file system

snapshots to avoid traversing FS logs but this adds extra processing overhead to its operation.

Hewlett-Packard Labs Clique project's [Ric02] goal is to develop a P2P server-less DFS based on optimistic replication algorithms. Clique uses epidemic-style replication to achieve a consistent FS global view. They rely on an IP multicast transport layer to replicate data across all currently active peers in a multicast group. This 'bulk' replication can be a source of wasted resources, beyond the reconciliation processing required for the weak consistency update policy used, since there are no limits on the number of file modifications. Basic security is provided based on out-of-band password keys[2] used to encrypt data communications. Clique's benefits reside in its transparent operation primarily for disconnected users, system convergence at global level, and a no lost update policy.

The Eliot DFS [Ste02] uses a separate file system metadata service from the actual data block service. This allows Eliot to use an immutable P2P layer. The P2P layer is characterized by a dynamic network of constantly failing and arriving nodes, which provides the storage layer. The P2P layer is immutable because data blocks do not change; new blocks are created when updates occur. Eliot centralizes the metadata service on a trusted, replicated database. Issues of scalability, consistency, coherence and performance are clear limitations of their approach because of its centralized metadata management.

The Mammoth P2P FS [Bro02] is another effort to implement a read/write P2P DFS. Mammoth handles all coordination at the granularity of files instead of peers as was used in previous efforts. Per object (file) policies are used to

[2] Keys are out-of-band because they are given separately.

specify how an object is stored and replicated. Important issues of scalability, performance, and reliability are only touched upon.

Another research area related to P2P DFSs is global distributed archival systems. The goal of archival systems is to provide distributed data storage services with high availability, reliability and scalability. The archival's goal differs from a P2P DFS goal because the latter is more focused to provide a seamless file system in a distributed system of peer nodes. Some archival system examples are Silverback [Wea01], Pasis [Wyl00], DSF [Dub02] and zFS [Rod03].

Silverback [Wea01] uses "erasure codes" to fragment a file and provide durability and persistence. Erasure codes are mechanisms to encode-decode data blocks to recover the blocks in case of failure. Silverback makes use of secure hashing as a mechanism to implement globally unique ids and Tapestry [Zha01] as a distributed infrastructure to route and locate data. Silverback lacks security mechanisms and has scalability and fault tolerance issues on data writes. Pasis [Wyl00], which is similar to Silverback, uses several threshold schemes as a foundation to support data availability and confidentiality. The threshold schemes (e.g. secret-sharing schemes or information dispersal protocols) encode, replicate, and divide information into multiple pieces that are stored in different peers.

DSF (Data Sharing Facility) from IBM Research Laboratories proposes the distribution of all aspects of storage management (including metadata and file data) across cooperating machines that are interconnected by a network. DSF applies some previous concepts of logical volumes similar to Frangipani [The97] and serverless xFS [Bol00]. zFS extends DSF and achieves scalability

by separating storage management from file management and by dynamically distributing file management across the network.

# *Chapter 3*

# 3. The Emergent Thinker Paradigm

*"Everything should be made as simple as possible, but not simpler"*

*- Physicist Albert Einstein*

Our existence is formed and surrounded by *Complex Adaptive Systems (CAS)* [Bar97]. Global properties emerge from these systems that enable them to provide functions and/or services. CAS examples permeate our lives. We find them in biological, economic, social, neural, immunological, ecological, and cosmological systems to mention a few.

The following examples illustrate the ubiquity of CAS in our life:

- Blood cells working together to provide a set of functions like the defense of the organism by means of phagocyte activity of white cells or leukocytes [Pur83],
- Neurons forming centers and circuits where brain activities emerge for different processes like vision, thinking, hearing, sensing, and many others [Nic01],

- People trading goods and services to come out with a fair market price level for those goods and services,

- A society or group of people that exchange and communicate ideas and behaviors to eventually appear with characteristics that identify them as a whole in terms of culture, conducts, language, and so forth,

- Groups of genes with mechanisms and processes (e.g. transcription, splicing, translation, and others) rising with amino acids and proteins structures [Hun93].

- Proteins and amino acids using 3-D folding mechanisms and interacting with other proteins and environment to emerge with biological functions [Coh04].

In all these 'everyday' systems, a CAS is found where the interrelation of many elements somehow produces an emergent outcome. Formally, a CAS is defined as a system with many elements with simple activities and interrelation among them, whose synergy is transformed into a global emergent outcome. The emergent outcome provides a function or service for the CAS operation, survival and/or defense. The interrelation of the elements can occur between the elements themselves and/or between the elements and their environment. The interrelation is a communication and exchanging mechanism among elements.

The concept of a *swarm property* represents a CAS emergent outcome. Swarm properties arise from the activities of the mass of the CAS elements within their environment. Emergence is defined as the dynamic appearance of new properties that provide functions and/or services. Emergence varies when

slight variations of the element activities and environment exist. For instance, there are no human beings alike although everybody has a similar universe of CASs providing biological functions. Because the elements are continuously exercising their activities in different ways and the environment can be varying, the emergent swarm property dynamically changes, sometimes within 'stable' boundaries and sometimes out of 'stable' boundaries. Thus, a swarm property is formally defined as the emergence of dynamic outcomes from elements and environment activities.

To exemplify the concept of 'stable' boundaries, the human body temperature is given as an example of a dynamic swarm property. Oral body temperature usually stays within a range of 98.6 °F ±1. Blood vessels open and close to keep internal heat and consequently the oral body temperature within range. This simplified CAS consisting of blood vessels and its emerging swarm property (i.e. body temperature) are dynamically changing. However, the swarm property could be taken out of stable boundaries for a period of time when external forces appear (e.g. a fever due to infection). Thus, a stable boundary is defined as a useful operational range of the swarm property in the environment where the CAS exists.

CAS emergent outcome has been previously called *swarm intelligence* [Bon00b, Tar02], however this definition limits the comprehensive scope that emergence has. The swarm intelligence concept restricts CAS emergence as something related to intelligence. CAS emergent outcome is wider and as such it is called swarm property. Swarm properties expand the emergent outcome meaning to any function, service or characteristic coming out from elements and environment activities, whether intelligent or not. Furthermore, the

'swarm' concept is being used to represent a group of elements, not necessarily in large quantities.

After this introduction to 'everyday' CAS ubiquity and its associated swarm properties, a technical reader can be asking herself[3] what 'everyday' CAS and emergence have to do with computer science. As we will see in the following sections, a lot of know-how from 'everyday' CAS can be drawn to computer science. This knowledge transfer can be used to research and develop novel algorithms and systems within computer science domains like systems research. The Emergent Thinker is proposed as a paradigm to model and simulate CAS know-how. The Emergent Thinker paradigm is described later.

However, knowledge transfer is not only one way: computer science can similarly provide structural representation, simulation and formal definitions for CAS. All these can enhance our CAS understanding and provide a better know-how for CAS domains. For example, simulation and modeling techniques make tools available to obtain a better understanding of emerging population behaviors that help to identify consumer trends and eventually provide better marketing strategies.

In computer science, particularly in distributed systems and systems research, it is well known that systems design is currently facing a continuous increase in complexity for the large number, interdependencies and heterogeneity of systems components [Her05]. Furthermore, systems pervasiveness (i.e. ubiquity) across the environment and a non-stop evolution of component technology add extra difficulty to systems design.

[3] The female pronoun herself is used throughout, but the pronoun himself similarly applies.

A fundamental aspect to tackle is how to research and develop systems under the dynamic, *ad-hoc* and decentralized characteristics of the components and environment. Traditional centralized or distributed methods with known constraints do not work appropriately in these circumstances. We need new ways and models to achieve scalable and adaptable computing for highly dynamic contexts. Otherwise our implemented systems will be brittle, limited, and costly to maintain and support.

CAS models and their associated swarm properties are proposed as building blocks for an alternative computing paradigm that provides systems mechanisms and schemes in highly dynamic, *ad-hoc*, and pervasive environments. Emergent computing (i.e. CAS-based) is introduced as an alternative paradigm for systems research and development. Thus, two hypotheses are established. The first hypothesis claims that swarm properties modeled after 'every day' CAS systems will provide a new form of computing functions and services. The second hypothesis states that the mechanisms of the CAS elements and environment activities will provide the computing processing means to achieve the swarm properties.

But before CAS principles and ideas can be applied, models and schemes are needed to guide their use on the solutions of distributed systems problems. These models and schemes are frameworks that define the design architecture to follow when using CAS in distributed systems. For example, in cognitive sciences it is well known that the mass activity of neurons somehow emerge with abstract thinking that provide human intelligence and behavior. A Model architecture could guide research into these neuron processes and help to find an answer to some of the questions about how human thinking is achieved.

Thus this chapter introduces a new computing paradigm called the *Emergent Thinker* [Cam04c]. The Emergent Thinker provides a framework to achieve the design architecture for the Biologically Inspired Peer-to-Peer Distributed File System (BPD). The importance of the Emergent Thinker is derived from the need to have a blueprint for the investigation of CAS models that provide solutions to some distributed systems design challenges like those commonly found in unstructured peer-to-peer systems. The Emergent Thinker guides, structurally, the design and implementation of the BPD.

Formally, the Emergent Thinker is an area-wide logical computing entity, named after a philosopher that continuously analyses information and has emergent computed solutions for new and/or existing requests. The Emergent Thinker uses CAS domains and their associated swarm properties to provide emergent computation for large, highly dynamic distributed systems. The instantiation of the Emergent Thinker on the BPD design and implementation will corroborate the hypothesis that CAS based computation is a valid alternative paradigm to provide scalable computation for large distributed systems. Thus, the CAS emergent computation model and the CAS propagation model are the mechanisms to achieve the Emergent Thinker. These models are described in Section 3.2 and 3.3 respectively.

## 3.1 Emergent Self-* Properties

A self-* property is defined as a global property that characterizes CAS and give system-wide tools to tackle the variable challenges of the system environment [Can04, Lem04]. Currently there is a flurry of self-* properties

being identifiable for computer systems as necessary properties to have in order to face the dynamic, *ad-hoc* and scalable computer environments. The '*' indicates meaning multiplicity and can take a specific value such as configuration, balance, aggregation, decision, *etc.* to represent self-configuration, self-balance, self-aggregation, self-decision, *etc.* respectively. For instance, some self-* properties discussed in the literature include self-made [Van04], self-management [Jel04], self-organization [Deu04], self-configuration [Can04], self-adaptive [Lad04], and many others.

Self-* properties can be seen as emergent outcomes arising from the CAS that exist within a given system environment. Emergent self-* properties are associated with the currently active CAS. For example, when somebody has a skin cut, blood cells immediately come to the scene of the injury and work together to somehow protect and help the skin to recover (e.g. to avoid infection) [Pur83]. We have an emergent outcome that gives the person a self-protection property. Our immune system activates the corresponding CAS to get the body's self-protection based on the body's medical condition. This perspective of self-* properties as CAS outcome suggests that self-* is an instantiation of the swarm property concept introduced previously. Thus, the swarm property concept expands and generalizes the self-* concept.

Emergent self-* properties are either continuously present or show up when they are required (as in our previous example on the immune system). System designs need to identify what self-* properties are required to tackle the problems and scenarios the system will face in its life. For example self-* properties like self-learning, self-growth and self-configuration are some desirable property examples that could help a system face unknown and unpredictable scenarios.

Self-* properties for the unknown are important to cope with unforeseen scenarios that can cause the system to fail. For instance, the immune system fails to self-protect us when it finds stronger and/or strange viruses. Therefore the immune system cannot defend itself, causing us to get sick. This gives some evidence that biological CAS has limits in the emergent outcome functions. However, it is flexible enough to learn and receive external assistance (e.g. an immunization vaccine) to adjust its mechanisms that help the immune system to improve coping with intruders. This gives a better self-protection property and indications of the biological CAS flexibility to learn and grow.

The CAS Emergent Computation (CEC) model to be introduced in the next section not only generalizes and extends the self-* property concept but also provides a simple representation for it. The CEC model constitutes a computing element that provides swarm properties; either system functions, services or characteristics. The CEC model is the building block for the Emergent Thinker paradigm.

## 3.2 The CAS Emergent Computation (CEC) Model

A profound computing paradigm shift takes place with the use and modeling of CAS. People are used to linear computing dependencies when designing systems. For instance, when program code is developed, it is relatively easy to see how the program achieves its results. Frequently a "divide and conquer" approach is followed to understand the program pieces and forecast its behavior. In CAS this is not the case and many components are working

simultaneously and communicating among themselves in such a way that non-linear or group synergies occur continuously. These synergies are emerging persistently providing functions, services and/or characteristics (i.e. swarm properties). This perspective changes the way people see computing. A different view is in place because emergence somehow arises with outcomes currently not clear as the previously 'divide and conquer' approach. The Emergent Thinker paradigm tries to provide means to achieve the understanding of the new computing perspective.

A common feature of previous CAS related work is the existence of simple agents with local functions (i.e. action, activity, property) and a simple communication mechanism that is either direct or indirect. The communication mechanism exists within the domain environment. This thesis proposes a basic computing model that generalizes previous work called the *CAS Emergent Computation (CEC)* model (see Figure 3.1).

In the CEC model, agents follow simple rules to affect their states and/or environment to generate an emergent pattern formation that produces a system-wide result (Figure 3.1). The term *emergent pattern formation* means pattern formations in agents, the environment, or both. The system wide result is interpreted as an emergent computation (i.e. swarm property) that solves a particular distributed system problem (e.g. aggregation, resource allocation, classification, assignment, path selection, decision, *etc.*). A model hypothesis is that all computation to be provided can be obtained by emergent computations of simple activities.

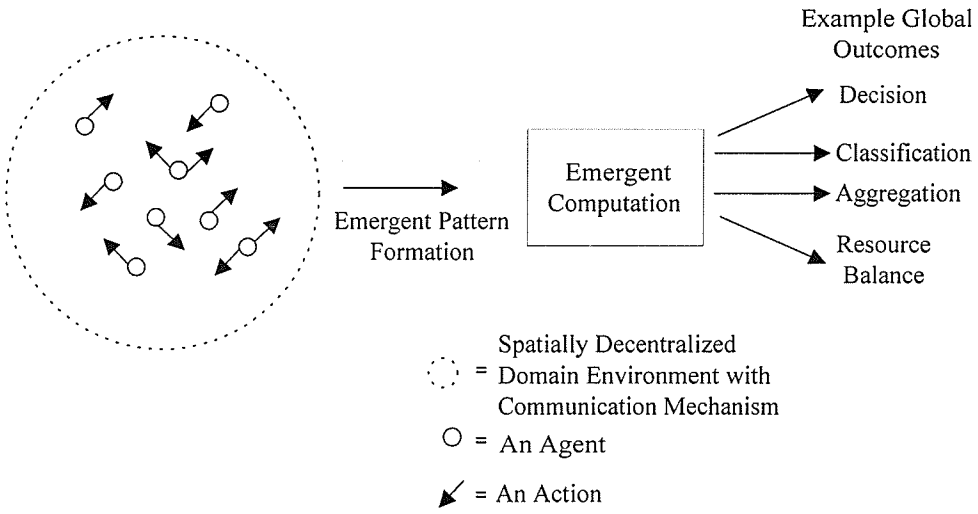


Figure 3.1: CAS Emergent Computation Model

Agent rules are simple activities that usually consist of elementary operations. Examples include arithmetic/logical operations on local or neighbor data, moving and/or storing data locally or to a nearby neighbor, exchanging data with local neighbors, *etc*. Usually agent activities are within local surroundings unless there are direct links to remote or far-reaching neighbors.

It is important to stress the difference between an *emergent* computation and a *regular* computation. A regular computation is a CPU computation such as arithmetic and logical operations. An emergent computation is at a higher abstraction level. An emergent computation is a domain-wide (e.g. system-wide, problem-wide, *etc.*) computation relative to the perception of a domain viewer. A domain viewer is an entity (e.g. person, computer, device, *etc.*) that has knowledge and perceives global characteristics about the domain.

Emergent computations produce values and/or results that are interpreted by the domain viewer as swarm propertiy assessments. The swarm properties are used to solve or provide solutions to complex systems. In the Emergent Thinker paradigm, the emergent computation outcome is a swarm property that resolves a particular distributed computing problem. Emergent computations can make use of regular computations to achieve their outcomes, and could also use other emergent computations (e.g. low-level hierarchical emergent computations) as part of their computations (See Section 3.5).

An important basic research question is identifying the relationship that exists between the emergent computation and the local agents' properties or actions. Other research questions include problems like messages exchanges, speed, property sets, feedbacks, edge of chaos, *etc.* By focusing on the cause-effect relationships, which are dynamic and non-linear, it could lead us to identify necessary and sufficient conditions to obtain emergent computations (i.e. swarm properties). Further, it helps us to understand how a specific property produces an emergent outcome. All these will allow us to control and manage the agents' properties to obtain desired global outcomes. This leads to a key question: How can we identify these relationships? This thesis proposes the *CAS Propagation Model.*

## 3.3 The CAS Propagation Model (CPM)

The *CAS Propagation* Model (CPM) is based on the simple idea of amplification by propagation. The way the different agents connect and exchange information (directly and/or indirectly) will have a direct impact on

the other agents and the system as a whole. Different approaches can be used to interconnect CAS agents [Wat98], either physically or logically (e.g. fully connected, small world pattern, random, small set of neighbors, *etc.*). These connectivity approaches together with exchanging mechanisms and behaviours generate different global outcomes from the same local properties. This occurs because of the way the local properties are propagated and exchanged across the domain and are consequently amplified for different global, emergent swarm properties. The main point that the CPM makes is about the propagation and its affects on the amplification of the agents' actions to eventually give an emergent result, called a global swarm property (Figure 3.2).

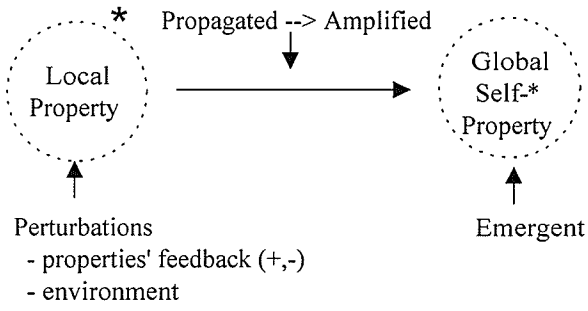


Figure 3.2: CAS Propagation Model

Furthermore, the way actions are propagated and exchanged will have an impact on the perturbations (i.e. properties' feedback, environment feedback), which subsequently affect the local agents' properties, which then adapt their agents' behaviors accordingly. If these perturbations are too high, the system could become chaotic. The CPM is a fertile area of experimental research that can lead to an understanding of the fundamental linkages between local properties and emergent, global swarm properties.

CPM is studied, like the majority of swarm systems developed, through computer simulation. Though it has been very difficult to mathematically model systems with many degrees of freedom, the availability of powerful computers has made it possible to construct and explore model systems of various degrees of complexity [Hey03]. A degree of freedom is a local property that can take on different values (e.g. real numbers) varying over a finite or infinite interval. Computer modeling is the best means to quantitatively and qualitatively research CAS [Dec01, Hey03]. This thesis follows this approach and does an extensive investigation of heuristics and algorithms that later are modeled in CAS computer simulations.

Based on the CEC model (Figure 3.1) and using the CPM experimental model (Figure 3.2), the CEC model is extended to develop the *Emergent Thinker* paradigm. The Emergent Thinker paradigm provides computing services by means of CEC model instantiations. The *Emergent Thinker* is named after a philosopher that continuously analyses information and has emergent computed solutions for new and/or "living" requests. Living requests are those computations that the Emergent Thinker continuously works on.

## 3.4 The Emergent Thinker Paradigm

The Emergent Thinker is an area-wide logical computing entity (Figure 3.3) based solely on CAS models and algorithms to provide function services (i.e. swarm properties), which are the mechanisms used by an application program to request services from the Emergent Thinker.

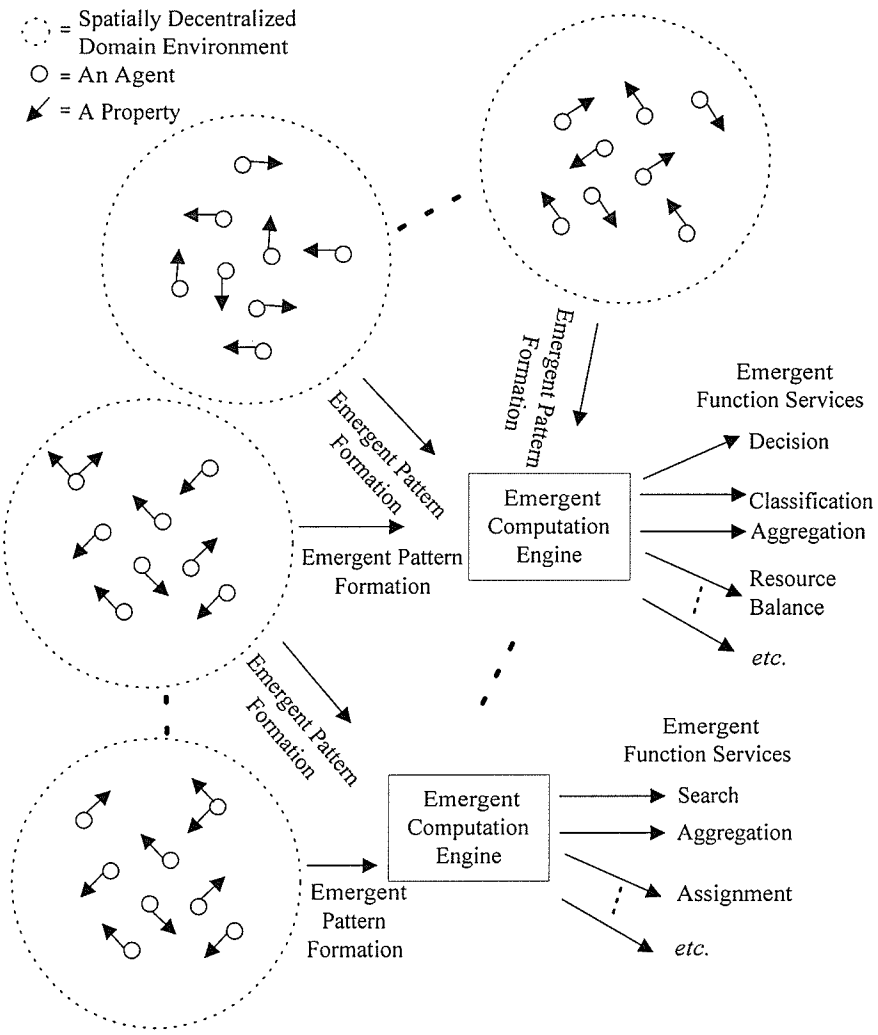


Figure 3.3: The Emergent Thinker Paradigm

The Emergent Thinker uses the CAS Emergent Computation model (Figure3.1) as its building block. It has CAS domain environments that are implemented in decentralized contexts (e.g. P2P environments, pervasive environments, multi-processor computers, grids, *etc.*) providing emergent function services. The emergent function services are analogous to system calls provided by a regular operating system, but with completely different

functionalities and purposes (at higher abstraction levels). The "Emergent Computation Engines" (Figure 3.3) are located wherever emergent services are required (i.e. they are pervasive engines). This is possible because the services, functions and/or characteristics (i.e. swarm properties) are continuously emerging from the system itself. These engines are access points to observable and/or interpreted global properties.

An emergent computation engine, besides providing access to emergent computations, also helps to insert information back into the Emergent Thinker system. This inserted information can be:

- new requests for emergent computations,
- new requests for information,
- perturbations and/or data from external entities to the emergent thinker, and
- other related data that must be supplied.

The initial Emergent Thinker paradigm assumes logically isolated CAS domains. The CAS domains are logically isolated from the perspective that they compute independently from one other, but they might be running on the same physical hardware and network of computing devices. Each CAS domain is usually implemented in software (e.g. software agents' domains). However, this does not imply that CAS domains could not be implemented in hardware as well (e.g. mini-robots' domains).

The Emergent Thinker provides a rich paradigm to research and develop computing services based on CAS and their associated swarm properties. The

Thinker generalizes and extends the self-* concept by making use of the CEC model besides providing a simpler representation. Although the initial Thinker has a flat hierarchy of CEC instantiations, it is enough to support simultaneous swarm properties coexisting within the same environment. For instance, the BPD design and implementation (Chapters 4-8) instantiates the Emergent Thinker with two CEC domains providing essential distributed file system services.

The richness of the Emergent Thinker paradigm is shown when the BPD is designed and implemented. Most of the work in the past regarding CAS considers "emergent behaviour" to be some kind of ant activity whereby each ant has a common goal – namely to work for the good of the colony. The ants all have common goals, they work independently, and what emerges is behaviour for the good of the colony itself. However, the BPD instantiation of the Emergent Thinker changes that premise of the CAS philosophy by using squirrels as a metaphor. The squirrels are in fact not working for the good of a colony where you would expect to get an emergent behaviour rather, the squirrels are working for themselves. They are 'thinking' for themselves and as such are motivated by their own success – not that of the colony. 'Thinking' for themselves meaning the fact that the squirrels are acting and behaving for their own sake.

However, the squirrels' behaviour results in the 'thoughtful production of the forest' as explained in Section 2.2. 'Thoughtful production' is used to stress the fact that the forest is created similar to a careful reasoned architect creation, to something that is well planned and designed. In other words, what emerges is not behaviour (that is what the squirrels are doing) but 'thought' – that is, forestation (i.e. a swarm property). 'Thought' is the emergent

computation out of the CAS domain solving a particular distributed system problem. Hence the Emergent Thinker has CAS domains with individual agents (i.e. elements) being self-motivated that leads to a consequence that is the same as a well thought out strategy, of which, none of the agents can possibly be aware since it is not the motivation for their individualist behaviour. Thus 'thought' emerges from the Emergent Thinker paradigm instantiation.

The emerged 'thought' is a fundamental difference achieved by the Emergent Thinker respect other CAS based schemes (e.g. ant models). The concept of purposed independent behaviours (i.e. ants) versus un-purposed independent behaviours (i.e. squirrels) shows an essential difference about how emergence is achieved among these CAS models and subsequently their global CAS outcome.

Swarm properties coming out of the Emergent Thinker provide functions, services and characteristics useful to carry out computations. This is exemplified when the Emergent Thinker is used as a paradigm to implement BPD services. The Emergent Thinker provides the structural layout to establish CAS-based services for distributed computing systems.

## 3.5 The Complexity of CAS

An initial step is taken in the Emergent Thinker paradigm when isolated CAS domains are assumed to provide emergent function services (Figure 3.3). However, nature shows that most likely this is not the case and CAS

domains are highly interrelated, hierarchical and recursive. For example, functional magnetic resonance medical imaging of the brain indicates that many neuronal circuits are active in different ways, orders and patterns when brain activity is happening [Fra04, Nic01]. This example gives evidence of the complexity of the many CASs working in different ways to achieve a given human behavior. A neuronal circuit constitutes a CAS providing a swarm property coming out from the activity of all neuronal circuits.

Interrelation of CAS elements can be multidimensional. A given element can communicate and exchange information in different ways to other elements according to the emergent function service provided. For instance, at certain point in time, elements can be part of a CAS and next time they can be part of a different hierarchical CAS arrangement as shown in the brain imaging example. This multidimensional interrelation can also occur simultaneously due to the multifaceted activity that some CAS elements have.

CAS element activities are considered simple from the perspective of the global outcome they achieve. However the CAS element itself could be another CAS that achieves its function by the emergence of other inner CASs, subsequently the elements of these CAS provide their function by other inner CASs and so forth. This brings us a hierarchical and likely interrelated view of CASs.

For instance [Hun93], a skin tissue CAS has elements that provide their functions from the cell CASs working together that emerge with color, humidity, texture and other skin properties. Then the cell CAS elements provide their functions by the emergence of inner CASs of the cell parts (e.g. cytoplasm, membrane, organelles, *etc.*) that work together to provide cell properties. Similarly a cell part like organelle is another CAS whose elements

provide their functions from ribosome and protein CASs. Subsequently the ribosome CAS can be seen as another CAS that has elements (e.g. proteins and ribonucleic acids) processing genetic instructions to emerge with amino acids and other proteins; and so forth (Figure 3.4). This example exemplifies a hierarchical structure of CASs and shows an interrelated CAS environment for the cell and organelle CAS. CAS interrelation means that CAS are not only vertically related but horizontally related as well to provide emergent outcomes at higher levels in the CAS hierarchy. For simplicity, we are skipping the horizontal inter-CASs communication and exchange, which it is left in the realm of future research. This example is meant to show the complexity of CAS.

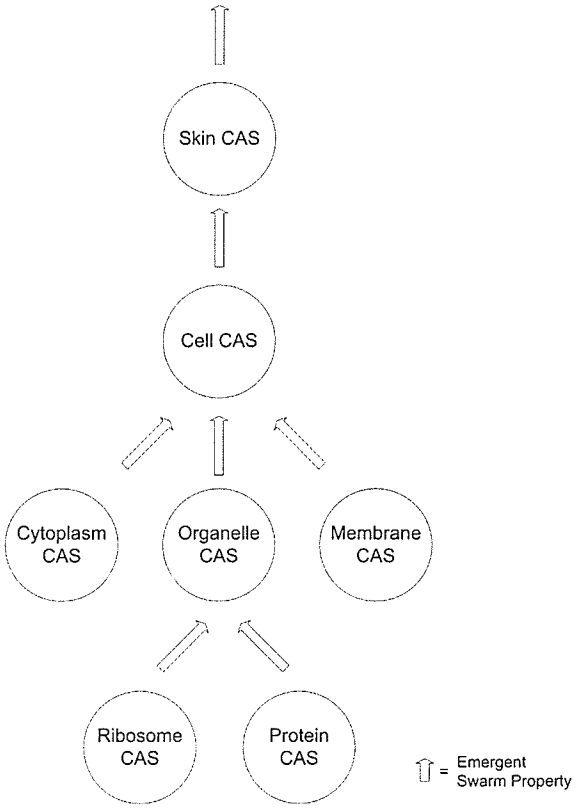


Figure 3.4: CAS Path

The hierarchical and interrelated view of a CAS is defined as the CAS path. A CAS path can grow upward and downward. A CAS path is relative to the emerged outcome that it is analyzed. For example, there will be different CAS paths for skin color, texture and humidity. Also, there can be similar CASs in paths that have different emergent outcome because the same CAS in the path could provide different emergent outcomes. An example is when the same ribosome organelles emerge with different amino acids and proteins based on ribosome activity [Coh04].

CAS complexity provides a rich environment for new ways to provide computing. Ideally, a computing machine can be developed that dynamically changes CAS domains and their associated interrelations, orders and patterns corresponding to the CAS emergent outcome provided. The Emergent Thinker is an initial attempt in this direction that establishes a computational environment of isolated CAS domains with a flat hierarchy providing swarm properties.

## 3.6 A Paradigm for the BPD and Relative Computing

The Emergent Thinker guides structurally the design and implementation of the BPD. Thus, the BPD is experimentally validating the hypothesis that CAS-based computing (and their associated swarm properties) is a viable paradigm for systems research.

This concept is proposed to be extended to potentially develop a complete CAS-based computing machine that is called the Emergent Thinker Machine (ETM). The ETM can provide new computing abstractions, whose characteristics and scope are in the realm of future research, though nature offer principles and targets to set them up.

There are two perspectives to be briefly described. One perspective views the ETM at the user level and the other views the ETM at the system level.

The user level ETM is a machine with a large amount of processing units providing computing by emergent mechanisms. CAS domains are created with CAS elements living within these processing units. The CAS domains provide swarm properties for the user level ETM.

The system level ETM creates CAS domains with the use of many user level ETM machines. The system level ETM provides computing at larger scope with emergent mechanisms compared to the user level ETM.

The ETM could be designed to support services similar to those implemented by a regular operating system with the use of swarm properties. This is equivalent to what has been done with the BPD design and implementation. However, a new abstraction for computing services can be defined. Hence, the concept of *Relative Computing* is established. Relative Computing is the abstract provisioning of swarm properties at different levels. It is abstract because it models 'everyday' CAS emergent outcomes. It exists at different levels because the swarm properties vary according to the level where the swarm properties exist within the system. Following the example in Figure 3.4, an ETM can be built to provide Relative Computing services (i.e. swarm properties) for the skin CAS at various abstraction levels.

Another example for Relative Computing providing computing services at different abstraction level is a brain related example. In the brain, synapses[4] provide services by the CAS that are formed by membranes, molecules and ions. Microcircuits[5] provide services by the CASs formed by synapses. Neurons provide services by the CASs form by the microcircuits and so forth [13]. Computing is at different functional levels according to the abstract level that is used. In this case, Relative Computing provides its services according to the perspective level that is desired by the problem being solved and/or function being offered.

The benefits for Relative Computing are large. For instance, in the brain example, Relative Computing can develop advanced tools and approaches essential for the understanding of the structure and function of the brain. The ETM built for these swarm properties could develop models to simulate neuronal circuits that will allow understanding brain mechanisms and processes. This know-how can then be applied for the treatment of brain diseases, understanding brain behavior, development of biomedical technology, analyzing the effects of treatments and drugs, and many other medical applications.

Relative Computing uses the ETM, which leverages the Emergent Thinker paradigm as its foundation architecture to provide swarm properties. Relative Computing redefines the way computation is comprehended because with Relative Computing there are a variety of functions, services and characteristics that currently only nature knows and controls. The Emergent

---

[4] Synapse is the site where neurons make functional contact.

[5] Microcircuits are patterns of synapse connections that are stereotyped and distinctive, which mediate specific types of operations carried out by those patterns.

Thinker with its BPD instantiation is a first endeavor in this direction. The BPD design is described next.

# *Chapter 4*

# 4. BPD Design

> *"There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."*
>
> *- Computer Scientist C. A. R. Hoare*

The *Biologically Inspired Peer-to-Peer Distributed File System* (BPD) is an instantiation of the Emergent Thinker paradigm. The BPD merges Complex Adaptive Systems (CAS) models with Peer-to-Peer (P2P) computing to implement a Distributed File System (DFS). The BPD design and implementation presents a novel alternative to existing deterministic, centralized or distributed techniques proposed in the majority of current P2P and DFS research. BPD models natural behaviors in its foundation services to solve distributed systems' design challenges. First the BPD is overviewed and how the Emergent Thinker paradigm is applied is described. Then the BPD architecture design is detailed.

## 4.1 The BPD and the Emergent Thinker

### *4.1.1 The BPD Overview*

The BPD assumes an environment that has computing devices with *ad hoc* behavior (i.e. joining and leaving the network) and no central server or controller. The BPD is intended to scale from a few peers to thousands of peers allocated across a distributed system within an organization. The BPD targets an environment where users can seamlessly and dynamically share their storage resources to provide a peer-to-peer (P2P) distributed file system (DFS).

Medium and large organizations have hundreds or thousands of employees with personal computers (PCs) located within several branches that probably are physically apart from each other (e.g. large financial and management companies). The BPD targets this environment where users can seamlessly and dynamically share their storage resources to provide a P2P DFS. The P2P DFS is used for the employees' application and document management needs. For example, if the average user, in a 1000-employee organization, has available 50-gigabyte of spare space on their PC, then the BPD potentially can create a 50-terabyte P2P DFS. The BPD is built from a set of commodity peers available across the enterprise where participants share their resources. Since the enterprise peers can fail at any moment, the BPD's adaptability and fault-tolerance by means of its use of CAS models alleviates the situation with peers failing and improves the reliability and data availability.

Application semantics are supported by the BPD. This helps the BPD to permeate application domain context into the DFS data management. The

supported semantics relate to the way data is created and later retrieved, specifically data hierarchies (i.e. application dependencies) and data relationships (i.e. data proximity within the hierarchy). There are many examples in our everyday working environments where data semantics exist and are used when applications access data. For instance, in a large insurance company, the majority of services and products center on the customer data and the customer's insurance portfolio (e.g. customers and their insurance for personal belongings like house, car, business, *etc.*). There is likely to exist a clear relationship and hierarchy among related data. Another example is a regional hospital facility where services support patients and their medical records (e.g. patient health records with lab results, radiology images, prescriptions, diagnosis and treatments, bills, *etc.*). Data hierarchies and relationships usually exist in every organization.

Data hierarchies and relationships are created and retrieved by user applications that might be located in many different places. This situation creates a physical fragmentation of related data in different, potentially distant, locations. Although data is fragmented in several storage "islands", data semantic hierarchies and relationships continue to exist and should be supported to provide convenient and efficient data retrieval and storage. For this, the BPD introduces the concept of a bag, where a bag is a hierarchy of related data. Furthermore, the BPD extends standard File System (FS) Application Programming Interfaces (APIs) to support bag semantics and provide better access to related data and their relationships.

Contrary to conventional hierarchical FSs with a common global root as is usually found in other existing DFS designs (e.g. [Ghe03]), the BPD has a floating multi root hierarchical FS. Roots are defined according to application

semantics and are dynamically maintained as required by the client application. There could be several directory hierarchies (i.e. bags) per client in a given time. This supports data independence and relationships that exist in application data. For this, the concept of a bag holder is also introduced that dynamically holds several bags.

Files have a life cycle of creation, use (i.e. read, write) and deletion. The life cycle is commonly characterized by continuous updates for a relatively short period of time after creation to be either deleted or stored in a device [Ous85]. Once the file is stored, the file has few, if any changes for some time. During this period of time, the file usually has several read retrievals until eventually it is permanently stored or deleted. The BPD supports this life cycle of files. The BPD provides data management that follows usage patterns when accessing files during the file life cycle.

### *4.1.2 How BPD Compares to Other P2P DFSs*

The BPD expands current research on P2P DFS with CAS computing models. CAS computing provides a P2P DFS with implicit adaptability, fault tolerance and scalability. This differentiates the BPD from other previous reported research work in P2P DFSs.

The BPD, similar to GFS [Ghe03], consists of a P2P community of commodity devices whose quality and quantity guarantees some of them are likely to fail at any given time. However, the BPD automatically adapts to failure using CAS models instead of the constant monitoring pre-designed into GFS. The BPD peers participate in the DFS according to their capabilities and

logically are members of a community; unlike the GFS which has a master server with chunk servers as storage slaves for GFS clients.

Previous work typically uses a lower layer lookup service to store data across a set of servers (e.g. CFS [Dab01] uses Chord [Sto01], PAST [Row01] uses Pastry [Dru01], Oceanstore [Rhe01] uses Tapestry [Zha01], *etc*). These lookup services guarantee a lookup time in some number of hops but they lose data locality (Chapter 1). The lost data locality can bring latency in data update and retrieval [Kel02], unless special workarounds are designed. The BPD uses a different approach by keeping the metadata in each peer and using CAS computing layer services to store, retrieve, and search for data with implicitly high data locality.

DFSs like Frangipani [The97], xFS [And95] and others use caching mechanisms to offer a global file cache. The BPD does not have any global cache since it relies on commodity local FS caches to provide some level of local caching. Further, the BPD has a policy to keep data local to minimize access delays. After the client closes the file, the BPD then allocates the file in the distributed system. This policy helps the BPD to follow the life cycle of many file changes made by the client before permanently saving the file.

Data management is done dynamically at the clients with a clear separation from storage using file management similar to zFS [Rod03] and Elliot [Ste02]. However the BPD is simpler and faster, as each client does its data management locally and independently.

Semantic file systems [Gif91] and data archive management systems [Tho03, Han96, Jef98] have associated metadata via attribute-value pairs similar to the BPD. However, these systems have servers to maintain semantic access that is

contrary to the BPD, where this functionality is spread out across the P2P system. Semantic access means data retrieval by attribute value.

*4.1.3 Applying the Emergent Thinker to the BPD Design*

Figure 4.1 shows a high level view of the BPD. A P2P system with hundreds or thousands of computing devices forms a complex environment where peers (Figure 4.1, small circles) continuously connect and disconnect. Each peer has agents that execute basic actions (in Figure 4.1 these actions are depicted with arrows) independent from each other, with minimum or no communication among them. The emergent computations achieved by the agents' actions provide computing services required by the DFS, which is spread out across the P2P environment.

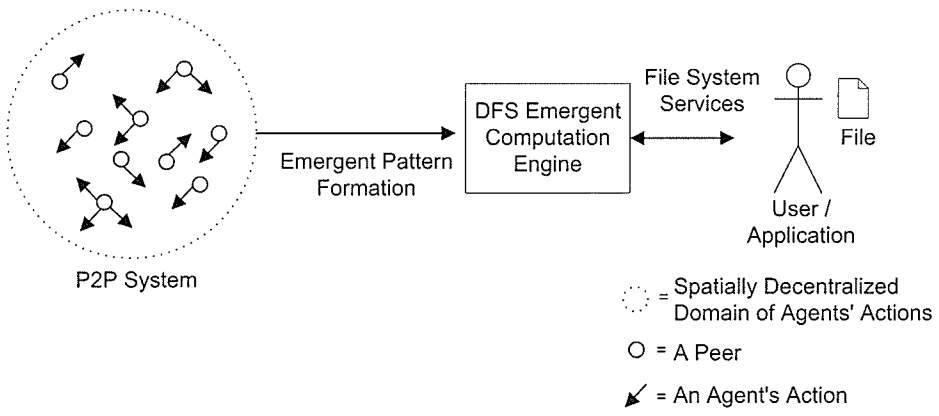


Figure 4.1: BPD High Level View of the Emergent Thinker Paradigm

A user (or application[6]) accesses File System (FS) services for its file management needs through calls to the DFS emergent computation engine. A

[6] The terms 'user' and 'application' mean the same across the thesis.

DFS emergent computation engine resides in each peer that provides access to the environment. When an application requires a FS service, the engine inserts a request into the complex environment and the response emerges from it to be delivered to the application that initially made the request. For each BPD DFS service provided, there is an independent spatially decentralized domain of agents' actions that execute on the same physical P2P system.

Before the CAS Propagation Model can be experimentally used, essential DFS building block functions must be identified. These DFS building block functions must be suitable to be provided by emergent computations and also meet the characteristics desired in a DFS. The DFS building block functions constitute the major components of the FS services that are offered to the applications. I use a top-down approach to analyze application FS services and identify the common building block functions. For example, Bach [Bac86] describes the Unix operating system FS services provided to applications. My analysis yields the following primary DFS building block functions:

1. Allocation services to implement storing data blocks and/or complete files across the P2P.

2. Retrieval services to implement reading data blocks and/or complete files from the P2P system.

3. Replication services to implement a data replication scheme to increase storage reliability and availability.

4. Discovery[7] services to implement a P2P system-wide data or file system search.

These four primary DFS block functions are the key operations that need to be implemented by emergent computations. However, further study shows that replication services (using variants) can be implemented with allocation services. Further, directory and file tables, which are resident at the local peer, can be used to guide retrieval services of known data, while discovery services can be used to guide retrieval services of unknown data. Thus, retrieval services can be implemented using local data management techniques (known data) and using discovery services (unknown data). Thus, there are two essential DFS building block functions: Allocation and discovery function services.

Allocation and discovery services are essential distributed building block services. Both are implemented by emergent computations. *Emergent function services* are defined as those basic DFS services that come out of CAS emergent computations.

Figure 4.2 shows a more detailed BPD instantiation of the Emergent Thinker paradigm. The current BPD design and implementation uses one CAS (i.e. one spatially decentralized domain of agents' actions) for each emergent function service. However, these CAS domains run on the same physical P2P complex environment such as each CAS domain is logically independent but physically execute on one physical P2P complex environment. Chapter 5 details the CAS algorithms that implement the BPD emergent function services of allocation

[7] The terms 'discovery' and 'search' mean the same across the thesis.

and search, while Chapters 6 and 7 describe their implementation and experimental results.

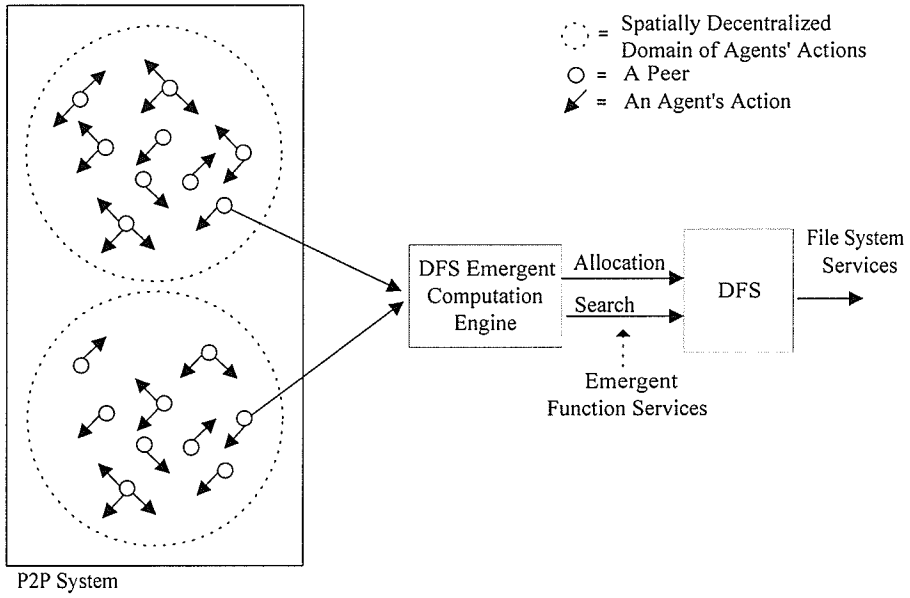


Figure 4.2: BPD Instantiation of the Emergent Thinker Paradigm

## 4.2 The Design of the BPD

### *4.2.1 Design Overview*

The BPD architecture stack (Figure 4.3) runs on every peer. It is divided into three layers that together provide file system services to applications running on the peer. Section 4.2.2 gives details on these layers. Briefly the layers are:

- The Data Glue layer that processes file system calls and provides metadata services to keep control of files and directories. It consists of the Front-end Data Manager (FDM), Local-end Data Manager (LDM), and Back-end Data Manager (BDM). These managers are responsible for the metadata services. The Data Glue layer also provides the Application Programming Interface (API), the Command Line Interface, and the Graphical User Interface to client applications.

- The CAS layer that provides the DFS emergent computation engine responsible for the complex adaptive systems that provide data allocation[8], retrieval, replication and search emergent function services.

- The Transport, Network, and Communication (TNC) layer that interconnects peers across the distributed system with networking and communication services. It interfaces to the physical peer communication network.

Recall that BPD introduces the concept of a *bag* to represent a container where related data is kept. A bag is defined as a complete directory hierarchy of related files. The implicit locality of the CAS models used by the BPD localizes files that are in the same bag. In this way, data locality is implemented using bags. *Bag items* (or just *items*) are used to manage bags. An *item* is defined as a file, a folder within a bag or the bag itself. All FS application-programming interface (API) are defined around the bag item concept.

---

[8] Allocation and search emergent services implement the retrieval and replication emergent services as discussed in Section 5.1.3.

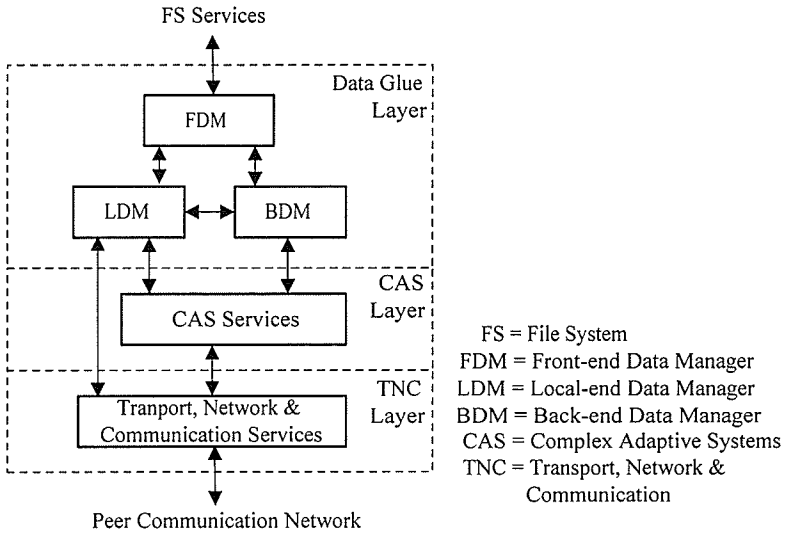


Figure 4.3: BPD Architecture Stack

A *bags' holder* (i.e. *holder*) is a virtual place where multiple bags can dynamically exist. There is one *holder* per client computer. A *holder* grows as more bags are loaded into it and shrinks as bags are unloaded from it. Bags are loaded automatically by changing to a different bag not already loaded when the application accesses them.

Each BPD peer is independent of the others. A peer collaborates with other peers as shown in Figure 4.4. The TNC layer provides a communication path between the peers. The peers with their software entities (i.e. agents) residing within the CAS layer work together when executing CAS models. A global DFS outcome will emerge that balances resources, searches for data, maintains locality and scales according to the peer-to-peer system characteristics. Figure

4.4 also shows how the BPD architecture layers are implemented in each peer. Chapter 6 describes the implementation.

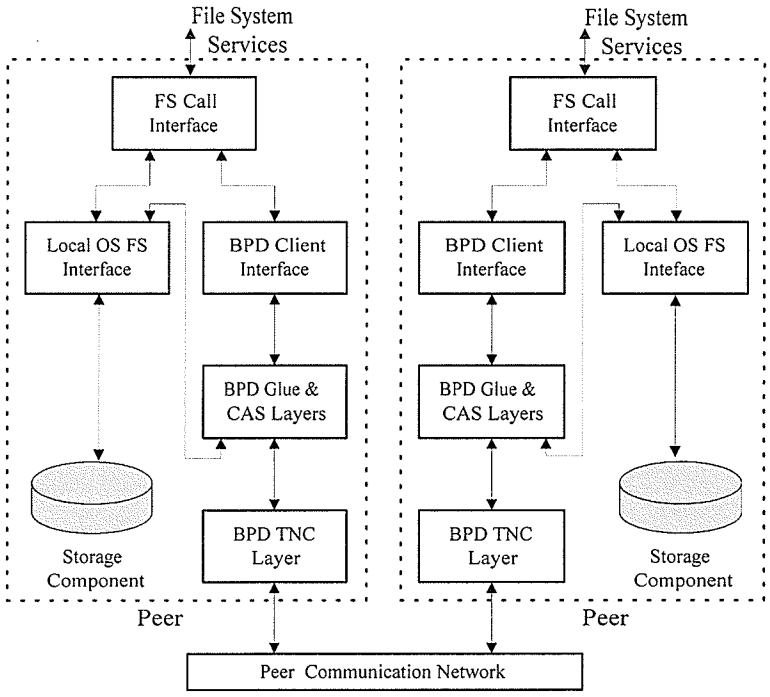


Figure 4.4: The BPD Peer System

It is important to stress that software entities (i.e. agents[9]) live in each peer at the CAS layer. The software entities execute natural behavior analogies that are described in Chapter 5, which produce the DFS emergent computations required within the P2P complex environment.

[9] Also known as squirrels in the CAS analogy used in the thesis.

### *4.2.2 Design Architecture of the BPD*

#### *4.2.2.1 The API Interface*

The BPD Application Programming Interface (API) is similar to regular APIs provided by other file systems. The BPD API is not POSIX (Portable Operating System Interface) [Ope05] compliant though. However, the BPD supports analogous standard FS APIs like create, delete, open, close, read, write, *etc*. Standard FS APIs system calls are renamed and/or extended to reflect a more appropriate functionality for BPD services and concepts. Chapter 6 details the API system calls that are implemented. An API listing is as follows:

- createItem( ): Creates a bag item within a bag.
- openItem( ): Opens an already created bag item.
- closeItem( ): Closes an opened bag item.
- removeItem( ): Deletes a bag item from a bag.
- grabItem( ): Reads a bag item's contents.
- putItem( ): Writes data to a bag item.
- putItemFlag( ): Writes the item flag to a bag item.
- grabItemFlag( ): Reads the item flag from a bag item.
- addItemAtt( ): Adds an item attribute to a bag item.
- grabItemAtt( ): Reads an item attribute from a bag item.
- putItemAtt( ): Writes an item attribute to a bag item.
- listItemAtt( ): Lists all item attributes from a bag item.

- removeItemAtt( ): Deletes an item attribute from a bag item.
- searchItem( ): Search for items that fulfill a search criterion.

#### *4.2.2.2 The Data Glue Layer*

The Data Glue layer provides transparent access to data. It consists of by three components (Figure 4.3):

- The Front-end Data Manager (FDM) does the processing and handshaking for FS API requests between the application and the BPD.
- The Local-end Data Manager (LDM) manages a bag's metadata structures.
- The Back-end Data Manager (BDM) builds a new bag's metadata structures and together with the CAS layer provides the emergent allocation service.

The Glue and CAS layers implement two essential protocols to support BPD APIs (see Chapter 6). The protocols include algorithms for data storage and discovery.

There is one basic structure and four auxiliary structures to support metadata associated with *bag items*. A *bag item* (or *item* for simplicity) can be a file, a folder within a bag, or the bag itself. Figure 4.5 shows these structures and the information the structures manage.

The item metadata structure (Figure 4.5) keeps general metadata for a *bag item*. It has fields for name id, name, owner, size, date (creation, modification, and access) and pointers to auxiliary structures.

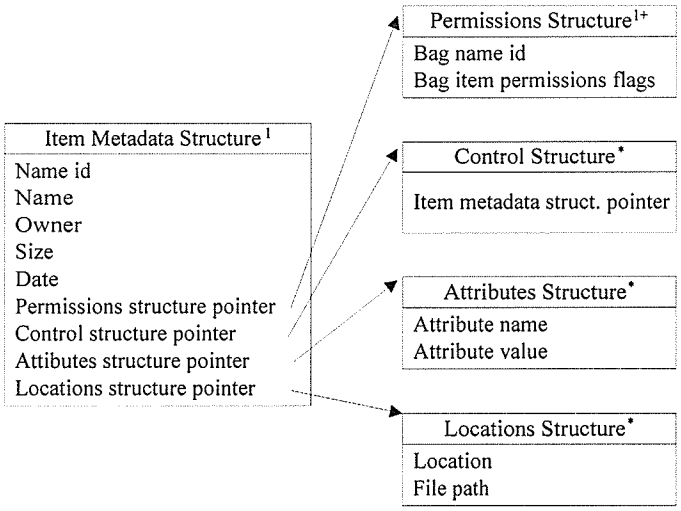


Figure 4.5: BPD Metadata Structures[10]

The permissions structure specifies the read/write/execute permission rights for an item within a given bag and the bag name id where the item belongs. If the item is a file, then the item can belong to one or more bags according to application dependencies and relationships. There can be different item permissions for each bag where this item belongs. If the item is a bag, the bag name id in the permissions structure will be the same as the name id in the item metadata structure.

Pointers to keep the hierarchical structure of a bag are maintained by the control structure. The control structure links an item's metadata structure

[10] Cardinality per metadata structure: '1' = one, '1+' = at least one or more, '*' = zero or more.

representing a parent item to another item's metadata structure representing a child item and so on for each relationship between items (Figure 4.6).

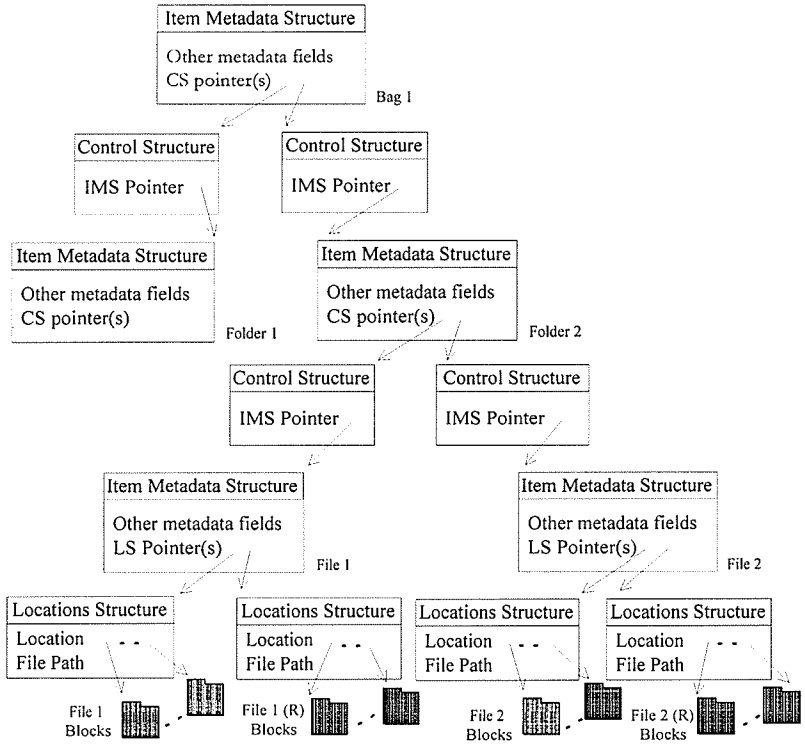


Figure 4.6: A BPD Bag Hierarchy Example[11]

Figure 4.6 shows an example of a hierarchy linkage of a bag item (i.e. 'Bag1') with two bag items (i.e. 'Folder 1' and 'Folder 2'). 'Folder 1' is empty and 'Folder 2' has two bag items associated with it (i.e. 'File 1' and 'File 2'). The files are allocated in blocks within the P2P systems and have a replication factor of one (e.g. 'File 1 (R)').

Clients can assign attribute-value pairs to each bag item. These attributes provide semantics specific to the item. They are used to create, search for, and access semantically related bags. Once a bag is found, a client can access all related bag items in a hierarchical manner. Since a bag is kept locally near to its original source, its retrieval is fast and efficient. Consequently, both existing data dependencies and existing data relationships are supported.

An actual physical file location with its associated path is kept in a location structure. If a file is replicated, then several location structures exist (Figure 4.6). The current BPD implementation supports block level granularity. Granularity refers to the way the FS allocates files into the storage layer. For example, if a file is divided in blocks and the blocks are the units being stored, we have a block level granularity. The BPD may physically stores a block on any peer; however the CAS layer takes care to put bag items that are related locally near. It is also possible to extend the location structure to support block grouping (e.g. block clusters) or complete files so a different granularity can easily be supported (e.g. cluster level granularity).

When file blocks are loaded to a local peer from a remote location, a temporal location structure is created to hold the file block copy pointers. This minimizes network traffic and access latency. An updated file is written back to the original location(s) when the file is closed and the temporal location structure is deleted. If the file is not updated, the BPD only deletes the temporal location structure.

If an original file is updated by an application 'B' while an application 'A' makes changes locally, the changes made by the application 'B' are lost when

---

[11] CS means Control Structure, LS means Location Structure, and IMS means Item Metadata Structure.

application 'A' closes and saves changes to the original file. This situation is easily avoided with the use of permissions or file locks. For example, if a file cannot be changed, then permissions can be set up to avoid other applications making changes to it.

Although auxiliary structures are shown as different entities (Figure 4.5), they are implemented as dynamic vectors inside the item metadata structure object (e.g. by using Java hash vectors). This implementation gives flexibility in the number of data pointers and a fast retrieval for the pointers inside the item metadata structure.

The Glue layer is also responsible for maintaining the bags' hierarchies within a bag created in the local peer. These bags are stored in the peer persistent storage as objects. The bags are dynamically loaded into memory when the BPD is executed at the client peer. When a bag item (e.g. a file) is retrieved or stored, the Glue layer communicates directly to the peer responsible for the item via the CAS and TNC layers. The CAS layer implements allocation and retrieval algorithms to efficiently balance resources. The TNC layer provides the physical communication between the peers.

The Glue layer managers either the assembly or disassembly of file blocks for the client application for a write or read operation, respectively. When a request comes from a client, the Glue layer managers immediately know if they can fulfill the request by checking with locally managed data structures, otherwise they ask other layers for assistance to start building the bag that satisfies the request.

When new bag items are stored, the Glue layer uses the CAS layer to allocate storage within the DFS. The Glue layer relies on the CAS layer to accurately

balance resource usage and achieve data locality. Once the bag items are allocated, the Glue layer updates the item's metadata control tables. Furthermore, the Glue layer together with the CAS layer cooperate with other peers' Glue and CAS layers to search for bag items with specific attributes and/or to form new bags.

#### *4.2.2.3 The CAS Layer*

The *CAS Layer* provides services to allocate, retrieve, and search for data resources (e.g. bag items) across the P2P system. These CAS services can be based on simple biological behaviors commonly observed in animal, insect, and other societies. The CAS layer uses the services of the TNC layer to establish linkages to other peers within the BPD system. Chapter 5 describes the CAS algorithms used by the BPD.

Section 4.1.3 previously identified that allocation and discovery services are essential distributed building blocks for the BPD. The CAS layer supports these services.

*1) Allocation services* are called when the Glue layer executes data storage and/or replication protocols (as described in Chapter 6). These services find a suitable peer where the file can be stored. Suitability is achieved by both balancing global distributed system resources and by selecting a peer locally near to its bag contents. If the allocation is successful, the CAS layer returns information to the Glue layer about the peer where the Glue layer can put the item (e.g. file block). If allocation fails then an error status is returned to the Glue layer that communicates the failure status to the application.

*2) Discovery services* find peers that can provide items with associated attribute-values, which are passed as parameters to the respective API call. Discovery services are called when looking for bag items and/or constructing bags according to the semantic meanings defined by the attribute-value parameters. If the discovery service is successful, the CAS layer returns a list with all peers that can provide the required item to the Glue layer and then, subsequently, to the application. The application will then request the Glue layer to communicate directly with the selected peer that provides the item according to permissions and access flags defined in the permissions structure (Figure 4.5). If discovery fails, then an error status is returned. The error status could be either because the item is not found or an actual error occurred (e.g. a communication error).

*4.2.2.4 The TNC Layer*

At the bottom of the BPD architecture stack (Figure 4.3), is the transport, network, and communication (TNC) layer that provides services by physically interconnecting nodes in the P2P distributed system. The TNC layer relies on existing networking and communication protocols and technologies to connect to other peers.

There are different ways the TNC layer can be implemented. The implementation depends on the way the peers are physically laid out within the organization where the BPD is deployed. If all peers belong to a well-segmented subnet, then perhaps only a simple communication protocol among peers is required. A network protocol like TCP/IP can be used to connect to other peers in the subnet. The well-segmented subnet manages and keeps references to the peers that participate in the P2P DFS. In this case, all

communication can be socket based between the peers that provide FS services to the client applications.

If a more complex peer network exists in the organization compared to a well-segmented subnet then a more robust TNC layer is required. This scenario will most likely occur when not all the peers in a subnet participate in the BPD or there are different locations with firewalls and peers belonging to different subnets. A P2P platform like JXTA [Sun03] or .NET [Mic03] can be used to implement the required services. The platform services of membership, transport, security, and lookup, among others, are required for these more complex physical peer networks. All communication is P2P platform-based between the peers (e.g. JXTA-based).

Figure 4.4 shows how the TNC layer uses the peer communication network to establish direct linkages among the peers to support the communication needs of the other layers.

Both Glue and CAS layers make use of TNC layer services to communicate to other peers. The CAS layer executes CAS algorithms that communicate to other peers to work collaboratively and eventually provide emergent global solutions, whereas the Glue layer uses a direct communication to other peers to store and retrieve bag items.

### *4.2.3 User Control Tables*

There are two user control tables used to store pointers to bag items: the *user bag holder* and the *user items* tables. Bag items are bag hierarchies and items. The bag items are loaded, opened, or both (Figure 4.7). The *user bag holder*

table represents the bags' holders and resides at each BPD peer. It has an entry for each bag hierarchy existing in the local peer. A peer loads remote bags before making them accessible to applications. The user bag holder is then updated with an entry for the loaded bag(s).

The *user items* table resides at the client session instance, which is a running BPD client instance that uses DFS services provided by the peer. Each client session instance communicates to a BPD peer instance to obtain services. The user items table has an entry for each bag item identifier that is opened and directly accessible. Figure 4.7 shows an example of the relationship between the user control tables.

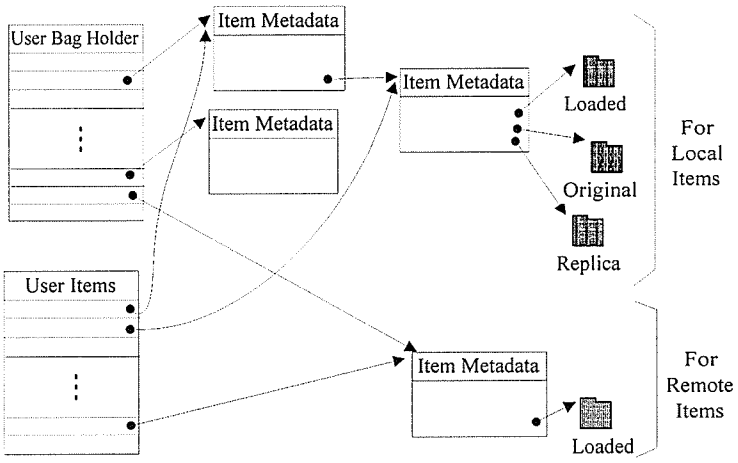


Figure 4.7: User Control Tables

In Figure 4.7, there exist three bags loaded at the peer. The first bag has a bag item (i.e. file item metadata with three pointers to the original, replicated and loaded file blocks). The second bag is empty, whereas the third bag is a remote bag with loaded file block copies. The bag holder table keeps pointers to all bags for peer control whereas the user items table keeps direct pointers to

opened bag items for application access (e.g. a pointer to a bag and a pointer to a file).

### *4.2.4 Data Mobility*

Bag item data moves through various states during its life cycle. The BPD supports the data life cycle shown in Figure 4.8.

These states represent the various states file data exists in within the P2P DFS.

- Birth state – data has been created and management resources must be allocated. The Glue layer allocates temporary item metadata structures.

- Active state – Applications access and store data. Resources are allocated locally for online access (e.g. local cache, main memory, local disk). Data grows and shrinks based on application activity. When a bag item is closed, the data it contains transitions to the rest state, which causes the bag item to be allocated in the P2P system. The CAS layer allocates space for the data using CAS algorithms and the Glue layer makes and/or updates permanent item metadata structures as per application activity carried out against the data.

- Rest state – data resides on P2P storage resources for subsequent access in the future. When the data that is in the rest state has been accessed (e.g. reading a bag), it transitions from the rest state to the active state for faster access. When the data that is in the rest state has not been accessed for a certain period of time (e.g. lease expiration); it may be backed up, compressed or moved to the dead state for deletion.

- Dead state – data has been trashed and previously allocated resources associated with it can be recycled. Its metadata structures are also deleted.

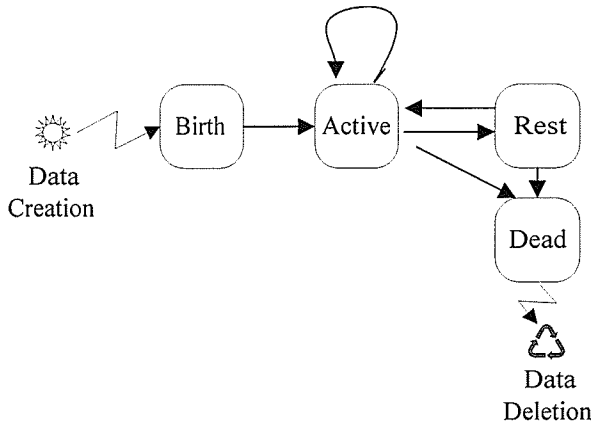


Figure 4.8: Bag Item Data Life Cycle

*4.2.5 Data Consistency*

File sharing semantics are relaxed because concurrency control is simplified as described next. The BPD implements a combination of session semantics with immutable files. Session semantics are supported from the perspective that modifications to an open file are only visible locally to the client that makes the updates to the file. After the file is closed and stored in the P2P system, then those modifications will be available to the other clients. Clients that have cached previous file copies have old data and must re-read the file to update its contents.

Immutable files are files that do not change. The virtual storage layer that is formed by all BPD peers only supports create and read operations on files. When a file is locally updated, the new copy overwrites an existing one when

it is allocated within the distributed system with the same name. Although files cannot be updated in the distributed system, the metadata that controls them can be. Consequently, from this perspective the BPD virtual layer supports immutable files that can be updated locally and overwritten remotely. If a peer application reads a file while another peer application is replacing it, BPD can detect that the file has changed and notify those applications reading from the file.

Next chapter expands the CAS algorithms used within BPD. It describes how these CAS algorithms achieve satisfying emergent solutions to assigned DFS services and functions. The chapter details the CAS allocation and discovery algorithms for the implementation of the BPD emergent computations formed at the CAS layer.

# *Chapter 5*

# 5. Complex Adaptive Algorithms

*"It is one of the universal miracles of nature that huge assemblages of particles, subject only to the blind forces of nature, are nonetheless capable of organizing themselves into patterns of cooperative activity"*

*- Quantum Physicist Paul Davis*

Research has been undertaken in this thesis work on using CAS Emergent Computation and the CAS Propagation models (Chapter 3) to experiment and implement BPD emergent computations. The emergent computations were based on biological squirrel behaviors. This research provided the development of CAS algorithms and provided a novel metaphor for this kind of CAS [Cam03b, Cam04a, Cam04d].

Chapter 4 identified allocation and discovery services as essential BPD distributed building block services and hypothesized that they can be implemented by emergent computations. This chapter describes the CAS allocation and discovery algorithms for the implementation of the BPD emergent function services. Section 2.2 described the squirrel hoarding

behaviors. These behaviors are extended and applied to the CAS allocation and discovery problems.

The allocation and discovery algorithms to be described next provide distinctive CAS characteristics for distributed resource allocation and discovery (analogous as described in [Var02]) including:

- Flexibility – rapid adaptability to changes in the distributed environment such as *ad-hoc* peer existence.
- Robustness – in case of individual failures, the system adapts and continues.
- Self-organization – no global administration is required as all individuals are self-guided.

## 5.1 CAS Algorithms for DFS Allocation Services

### *5.1.1 Squirrel-based Emergent Allocation Algorithms Overview*

For allocation services (e.g. storing data), the most interesting squirrel behavior is hoarding, rather than foraging, as hoarding leads to a global outcome of which the individual squirrels cannot possibly be aware. We are interested in the way squirrels spread out acorns, nuts and other small food pieces in an area to obtain a resource allocation balance. We exploit these techniques to design and implement resource allocation services suitable for distributed systems.

The squirrels-based CAS exists in a P2P environment. Each peer is a location that has one or more caches where squirrels can hoard acorns. An acorn symbolizes a piece of something (e.g. food) that needs to be stored. Squirrels live in these peers in small groups. When they have acorns, they go through the peers, "sniffing" to find a cache suitable for storing the acorn following one of the behaviours described in section 2.2. Each peer is both a provider and user of caches (i.e. storage resources). Figure 5.1 illustrates such an environment.

Coordination among the squirrels is achieved by sign-based stigmergy. Sign-based stigmergy occurs when something is placed in the environment and then affects the behavior of future activities by the CAS members. When a squirrel allocates an acorn to a peer, it indirectly affects the allocation decision of subsequent squirrels. We consider allocating an acorn as something that it is deposited in the environment and not as a physical change to the environment. Squirrels thus follow a sign-based stigmergy interaction.

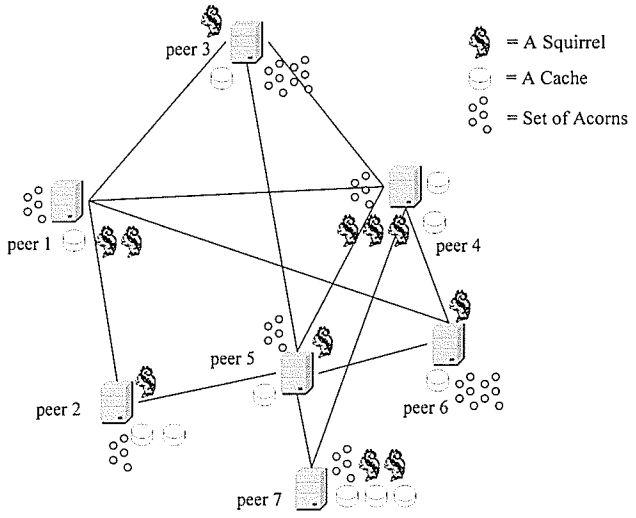


Figure 5.1: Squirrels-based CAS Environment

A simple example of the *Squirrels* System sign-based stigmergy is shown in Figure 5.2. This example assumes that the *Sniffing and Burying* Algorithm described in Section 5.1.2 is used to allocate acorns. . In the example a newly created acorn in peer 5 (e.g. a new file data block) must be allocated somewhere in the P2P system. A squirrel in peer 5 wakes up and "sniffs" (i.e. explores nearby) peers 2, 1 and 4 before deciding where to allocate the new acorn. Previous acorns deposited by other squirrels in peers 2 and 4 influences the squirrel to decide to use the cache in peer 1. Consequently, the squirrel deposits the acorn in peer 1, which subsequently influences other squirrels in their behaviors when allocating new acorns. The cache in peer 1 is selected because it has fewer acorns than the other sniffed caches.

Each peer (i.e. location) has different capabilities such as the number of caches and available acorns capacities for hoarding in the system. If there are no more acorns to store in a peer, the squirrels living there sleep until there are more to hoard. Each cache is assumed to store the same amount of acorns and each peer may have different storage resource availability because each has a different number of caches. This is important because variation in the availability of storage resources may create strong inefficiencies in a distributed system so the *Squirrels*-based system homogenizes each peer.

In terms of usable resources, each peer has a variable number of caches. The *Squirrels*-based system balances the caches thereby balancing each location according to its storage resource availability. In terms of demand, each location requests a different amount of storage resources because they have a different number of acorns to allocate (e.g. file data blocks).

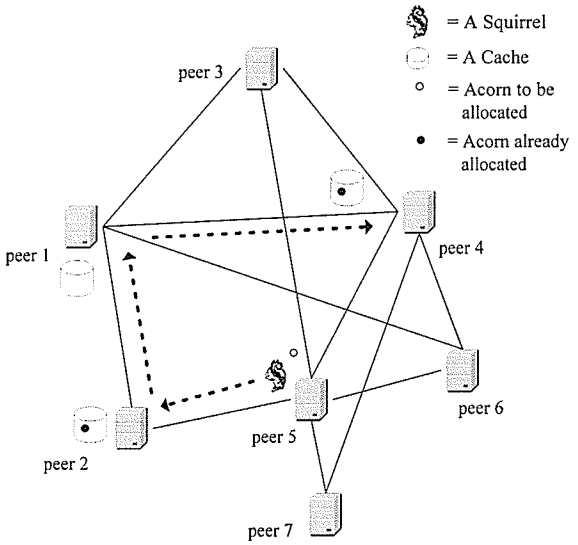


Figure 5.2: Squirrels Sign-based Stigmergy

In other words, each peer is a producer and consumer of storage resources. The goal is efficient allocation of storage resources by balancing resource allocation. This thesis claims that the allocation algorithm which evenly distributes the resources over the distributed resource system is better than one that does not. The empirical reasoning is that by having a common resource unit across peers gives each peer certain capacity to participate in the distributed system (e.g. a storage capacity unit). Each peer can have different multiples of the common resource unit according to the peer's capability. If the common resource unit is evenly used across the whole system then each peer is balanced according to the peer's capability. Furthermore the system as a whole obtains a better performance using the available resources. Clearly, other metrics must be considered but this discussion is in the realm of future research.

Thus, the balanced resource allocation is measured using the variance in storage allocation. Variance is a measure of the spread of a distribution and is the averaged squared deviation of each distribution element from its mean:

$$\sigma^2 = \sum_{i=1}^{N} (x_i - \mu)^2 / N$$

where $\mu$ = mean, $N$ = number of elements, and the mean is the sum of all element values divided by the number of elements:

$$\mu = \sum_{i=1}^{N} x_i / N$$

If we let $x_i$ be the number of acorns allocated in a peer's cache $i$, the variance will indicate how uniformly the system is allocating acorns among peer caches. That is, if the deviation $\sigma \rightarrow 0$, then we have a better storage resource allocation because each peer has a more equal assignment of acorns allocated per cache. Thus, we can have peers with a different numbers of caches provided each is the common unit of storage resources. Further, if we balance caches, then a well-balanced system is achieved across peers according to their storage capabilities (e.g. processor capabilities can be considered in the metrics instead).

*5.1.2 CAS Allocation Algorithms*

This section describes three algorithms that are based on squirrel hoarding behaviors. They implement squirrel CAS behaviors that provide global emergent solutions to the resource allocation problem in a P2P distributed system. The *Burying* algorithm simulates a squirrel behavior of random gathering and burying of small acorns in a geographic area. The *Sniffing and Burying* algorithm simulates a squirrel behavior that investigates various locations ('sniffing' places) before deciding where to bury the acorns, and the *Sniffing and Burying with No Strangers* algorithm expands previous algorithms to simulate a squirrel behavior of not hoarding acorns if there are other squirrels around. A 'sniffing cycle' is defined as the repeated investigation of various locations by the squirrel. The behavior of working in small teams with familiar squirrels will be implemented later when there is more than one squirrel per location. When a squirrel selects the best cache from a sniffed set, the squirrel chooses the cache with most space available to allocate the storage space for the acorn.

For all algorithms, the following apply:

acorn = data block or chunk to allocate.

$n$ = number of peers in a distributed system.

$loc_j$ = a peer within the distributed system. Each peer has '$x_j$' acorns to allocate, '$p_j$' caches to store acorns in, '$m_j$' squirrels living there, and $j = 1,\ldots, n$.

$s_{kj}$ = squirrel $k$ at location $j$, $k = 1,\ldots, m$ and $m >= 0$.

$c_{ij}$ = cache $i$ at location $j$, $i = 1, .., p$ and $p >= 0$, each cache stores '$y$' acorns, $y>0$.

*5.1.2.1 The Burying Algorithm*

The burying algorithm has the squirrel gathering acorns from its home peer and randomly searching for another peer (that could be its own) at which to cache the acorn. If the selected peer is full (i.e. no more space in any of the selected peer's caches), then the squirrel will pause. After pausing, the squirrel will jump from its current peer to another peer to test for available space. This cycle repeats until a suitable place is found or the squirrel dies looking for one.

Formally the algorithm is:

```
current = j
For each squirrel s k, j
  If home loc j has acorns to bury
    Select a path randomly to go to another loc current+1
    Move squirrel s k, j to loc current+1
    If any space free in cache c i, current+1 ∀ Ci in loc current+1
      Put acorn in c i, current+1
      Return squirrel s k, j to home loc j
      current = j
    Else
      current = current + 1
    End if
  End if
End for
```

*5.1.2.2 The Sniffing and Burying Algorithm*

This algorithm is similar to the "*burying*" algorithm above in that the squirrel will gather acorns from its home peer and randomly look for another peer (that could be its own) to cache the acorn. However, the squirrel will sniff several peers before deciding where to place the acorn. The selected peer will be the "best" option for the squirrel based on a specific (set of) metrics. If all sniffed peers are full, (i.e. no more space is left in any of the sniffed peers' caches), then the squirrel pauses and jumps to another peer. This cycle repeats until a suitable cache is found or the squirrel dies.

Formally this algorithm is:

```
current = j
For each squirrel s k, j
  If home loc j has acorns to bury
    For each sniffing cycle
      Select a path randomly to go to other loc current+1
      Move squirrel s k, j to loc current+1
      Sniff and record loc current+1 in location set
      current = current + 1
    End for
    Select best cache c i, j+1 ∀ Ci in sniffed location set
    If ∃ cache c i, j+1 found
      Put acorn in cache c i, j+1
      Return squirrel s k, j to home loc j
      current = j
    End if
  End if
End for
```

*5.1.2.3 The Sniffing and Burying with No Strangers Algorithm*

This algorithm is similar to the previous one in that the squirrel will gather acorns from its home peer and randomly investigate several peers to find one (which could be its own) in which to cache the acorn. However, now the squirrel will sniff only peers not being visited by other squirrels at the present moment of life to determine the best peer to place the acorns. From an algorithmic perspective, the present moment of life is defined as the current external 'For' cycle of the following algorithm that it is executed by a squirrel to bury acorns. Thus, the selected peer is the best available one that has not been visited by a stranger. I.e. The selected location has not been visited by other squirrels at the present moment of life. A stranger is another squirrel from the one allocating the acorn. If all sniffed peers are full or visited by strangers, then the squirrel will pause and move to another peer. This cycle repeats until a suitable peer is found or the squirrel dies.

Formally this algorithm is:

```
current = j
For each squirrel s k, j
  If home loc j has acorns to bury
    For each sniffing cycle
      Select a path randomly to go to other loc current+1
      Move squirrel s k, j to loc current+1
      Sniff loc current+1
      If loc current+1 not previously visited by another
      squirrel
        Record loc current+1 in location set
      Else
        current = current + 1
      End if
    End for
    Select best cache c i, j+1 ∀ Ci in sniffed and not visited
    location set
    If ∃ cache c i, j+1 found
      Put acorn in cache c i, j+1
      Return squirrel s k, j to home loc j
      current = j
    End if
  End if
End for
```

*5.1.3 Allocation Algorithms Discussion*

In the algorithms, *sniffing and burying* and *sniffing and burying with no strangers*, the best cache selection process from the sniffed set can be based on different parameters. This thesis uses existing free space to decide which cache is the best to select from the sniffed set (i.e. the cache that has most space available among those sniffed by the squirrel will be selected to allocate storage space for the current acorn being saved). Although this is an empirical approach, a hypothesis is that by following this approach an emergent CAS global result of storage allocation balancing will be obtained. Empirical approach means that it is corroborated by experiments and created with no mathematical formalization. This will be corroborated by the experiments described in Chapter 7. A motivation for this selection process is that when

there are a set of similar sources with different amounts, an approach to make them even (in terms of amount) is to take from those that have most. This is seen in economy, when the wealthiest pay more taxes to create a more even middle class society. E.g. Canadian economy with a stronger middle class society compared to US economy. Further, the algorithms use a standard, fixed cache acorn size (e.g. a file data block) as a common unit of allocation analogous to a simple paging scheme in an operating system. This eliminates external fragmentation within a cache. Other possible metrics to select the best cache from the pool of sniffed caches could include first-cache-found, last-cache-found, random-cache-found, and so forth. Although these alternative selection metrics provide fast execution because they are simple; they do not seem to have an optimization justification in terms of resource balancing. They are heuristics for fast execution. Future research might exercise these metrics and analyze their impact.

In all the algorithms, a squirrel will be constrained to a maximum number of steps to find a suitable cache to store the current acorn being allocated (i.e. to converge to a solution). If this maximum number is reached, then the squirrel will give up and die. This maximum number is set to avoid a squirrel searching forever without finding an appropriate cache to store the acorn in. This situation will most likely occur when there is no more storage space available (i.e. resources) but we still have acorns to allocate from the home peer. In effect, this scenario indicates that there is no feasible solution because the resource requirements exceed the capacity. This information will then be sent to the original requester.

## 5.2 CAS Algorithms for DFS Discovery Services

### *5.2.1 Squirrel-based Emergent Discovery Algorithm Overview*

A slight variation of squirrel behaviors in support of discovery services (e.g. data search) is also applied. The basic idea is that squirrels with simple hoarding activities disseminate acorns that contain identifiers. These acorn identifiers are used to dig out data acorns stored in the peer storage caches where they were allocated [Cam04a] and help to determine their locations.

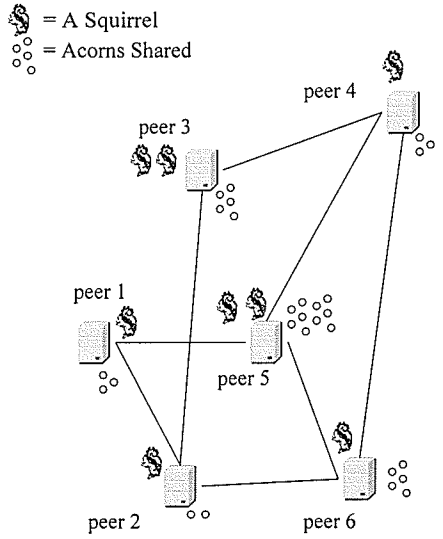


Figure 5.3: Distributed System with Sharable Data Acorns

Initially each peer has its own squirrels (up to a maximum) according to its capabilities (Figure 5.3). Each peer shares resources (e.g. files in sharable folders). Squirrels from different peers are independent of each other and the squirrels are unaware that their hoarding activities are used to search for data. The search emerges as a global outcome of the activities of the independent

members (i.e. squirrels) that perform simple activities but generate the desired system wide result. When a new member joins the P2P environment, there is no administrative activity to do besides joining the P2P environment.

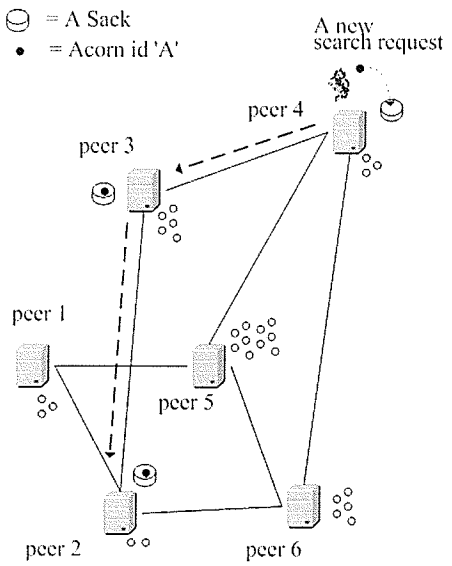


Figure 5.4: A New Search Executed by a Squirrel

When a new search "request" arrives at a peer (Figure 5.4), the peer's squirrel puts the acorn id in a sack together with other already existing acorn ids that have been put there by other squirrels (if any) and hoards them in nearby peers. 'Sack dissemination' is defined as the hoarding (copying) of the sack across several neighbor peers by the squirrel. 'Sack dissemination fan-out' is defined as the number of peers where the sack is disseminated. A squirrel leaves a sack (i.e. copies the sack's acorn ids) at each peer where the dissemination occurs. For example, in Figure 5.4 the new acorn id to be search

is put into a sack at peer 4 and then disseminated at peers 3 and 2. The sack dissemination fan-out is 2 in this case.

If a sack with acorn ids is placed in a peer, the acorn ids are searched for within the peer. Any acorn id that is found is notified of the original search peer, otherwise either the acorn id remains in the sack or the acorn identifier expires, terminating the data acorn search. New searches from this peer are added to the sack and are disseminated by the local squirrel to other peers (Figure 5.5).

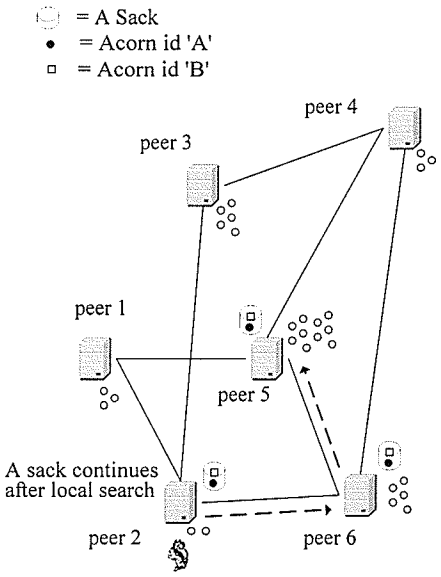


Figure 5.5: An Existing Search Executed by a Squirrel

The whole P2P environment is a living organism that has members joining and leaving continuously with no administrative burden. Once a peer joins, its sharable resources are available and the peer actively participates in any emergent search activities according to its capabilities (i.e. limited by the number of sharable resources and by the number of squirrels available). Each

squirrel works locally so that if the peer disappears (e.g. fails), the emergent search algorithm self-adapts to its new context without interruption. Neither loss of administrative statistics nor waste of resources occurs. Furthermore, by packing several acorn ids in sacks, the squirrels reduce the average number of messages they carry by a factor of 'n', where 'n' is the average number of acorn ids per sack over time. The more activity the P2P environment has, the greater the message reduction factor up to a limit (e.g. limited by message size).

When a peer leaves the P2P environment the squirrel moving the sack will choose another active peer, ignoring the absent peer. If the absent peer has no data from other peers then the peer can disappear smoothly without further processing from the remaining peers. If the absent peer has stored data from other peers, some form of replication scheme must have previously been used for the data that the leaving peer stores before disappearing. Data replication saves the same acorn across several peers in such a way that the probability of having all peers down (i.e. those storing a specific data) is near to zero. Data replication is thus used to increase the data availability when peers frequently leave the environment. The issue of deciding on an appropriate level of replication is not addressed in the thesis but is left for future research.

In both DFS CAS-based services, experimental research with the CAS propagation model will help to identify agent actions (i.e. properties) suitable for the BPD. Essential properties in the squirrel behaviors will be identified to provide DFS emergent computations (i.e. search and allocation services). Later the BPD implementation will use these properties and behaviors to become a biologically inspired DFS with emergent computations.

*5.2.2 CAS Discovery Algorithm*

There exist three acorn deposits per peer called the *living*, *search,* and *dissemination* caches. The living cache keeps all acorn searches originated from the peer that are still alive. The search cache holds acorns currently being searched for at the peer within the shareable resources of the P2P system. The dissemination cache holds acorns ready to be placed into search caches at other peers. A new acorn to be searched for is assigned to the living cache. This new acorn is also either placed in the search cache (i.e. to be searched for locally first) or placed directly into the dissemination cache (i.e. if the algorithm implementation does not search locally first). Dissemination is defined as the spreading of acorn id copies to nearby peers following a given squirrel behaviour.

The squirrel behaviour determines how the dissemination is achieved and the number of acorn id copies disseminated. For instance, when an acorn id has been disseminated in 3 peers, 3 additional acorn id copies exist in 3 different peers. Then from those 3 locations, the acorn id copies can be further disseminated simultaneously to other peers by other squirrels residing in those peers. This system wide dissemination process together with new acorns to be search and the existing searches executed locally provide a massive parallelism of activities carry out by all CAS members, which eventually emerge with a system-wide discovery service.

The CAS discovery algorithm requires the following definitions:

> $A_i$ is a new acorn with id *i* created by a query (e.g. a file identifier).
>
> $A_k$ is an existing acorn id *k* being searched at the peer
>
> $Loc_j$ is a peer *j* within the P2P system that participates independently in the emergent search and can create search queries.

$LC_j$ is the living cache at peer *j*.

$SC_j$ is the search cache at peer *j*.

$DC_j$ is the dissemination cache at peer *j*.

When a new search query is created at a peer, the following code sequence is asynchronously executed with respect to the CAS discovery algorithm to insert the query into the CAS search mechanism:

```
LCj <-- Ai
If local search performed first Then
  SCj <-- Ai
Else  DCj <-- Ai
Endif
```

Formally the CAS discovery algorithm is:

```
For each peer Locj
  Repeat forever
    For all acorns Ak in SCj
      Locally search for acorn Ak.
      If acorn Ak found Then
        Call back original search peer with status.
        Delete acorn Ak from SCj.
      Else
        DCj <- Ak
      Endif
    Endfor
    Peer's squirrels put all acorns Ak from DCj into a sack.
    Squirrel(s) disseminates sack(s) into other peers' SCjs
    following a given squirrel behavior.
  Endrepeat
Endfor
```

Most external 'For' loop is an independent activity continually executed by the CAS members living at each location $Loc_j$. The independent activity carries on a local search for acorn ids in the search cache $SC_j$. If the acorn id is locally found, then there is a callback to the original peer that created the search,

otherwise the acorn id is deposited on the dissemination cache $DC_j$. This is done for all acorns in the search cache $SC_j$. Once the acorns are in the $DC_j$, they are disseminated into other peers following the squirrel behaviour for further search in other locations.

When the original search peer receives a call back with a notification that an acorn has been found, the original search peer uses this information to retrieve the data acorn from the source peer. The call back is possible because the acorn carries with it the original search peer identification. The original search peer also deletes the searched-for $A_i$ from the source $LC_j$

*5.2.3 Discovery Algorithm Discussion*

There are some algorithm aspects that need to be clarified. The algorithm can be made much more complex by introducing more *'intelligence'* into it. However, a simple solution that obtains the search result as an emergent global property is the goal. One of the fundamental CAS premises is that simple mechanisms are amplified and a global synergy is achieved. Further, there are many '*tuning*' parameters in the algorithm. Some of them are described and their effects are analyzed in Chapter 7. Such 'tuning' parameters could answer questions like how many squirrels to have per peer? What behaviors squirrels should follow to disseminate bags? What sack size to use? What acorn life expectancy and termination policy should be followed? What acorn priorities to have when searching for data? These 'tuning' parameters will have an impact on the global outcome. One area of active research in CAS is to understand how local properties with different characteristics translate into global outcomes. This constitutes an important aspect of the analysis and experimentation undertaken by this research.

# *Chapter 6*

# 6. BPD Implementation

*"I have not failed. I've just found 10,000 ways that won't work."*

*- Inventor and Scientist Thomas Alva Edison*

The BPD implementation is composed of three phases. The first phase is the software implementation of CAS-based allocation service algorithms. The second phase is the software implementation for CAS-based discovery service algorithms and the third phase is the BPD prototype implementation. Both groups of CAS algorithms were described in Chapter 5, whereas the BPD prototype implementation will make use of knowledge and heuristics determined while carrying on phases 1 and 2. Consequently, Chapter 6 is naturally divided into three main sections covering each of the three phases.

The strength of CAS arises as a consequence of the emergent system wide functionality that is achieved by the simple 'programmable' activity of its members. At the same time a weakness of the current CAS state of the art is the lack of formal tools to predict or prove bounds on the performance of a CAS system under specific problem configurations. A simulation-based approach is commonly used to demonstrate the performance ranges of CAS applications.

Two simulators have been developed using Java under Linux to analyze and demonstrate the *Squirrel-based* CAS system performance for the implementation of phases 1 and 2. The simulators implement the *Squirrels* system class architectures for both DFS services. The class architectures are described in the next sections. The class architectures implement the CAS propagation and CAS emergent computation models (described in Chapter 3), while providing an experimental test bed for the algorithms.

In addition, sets of experiments will be performed to analyze the effectiveness of various squirrel behaviors on storage resource allocation and resource discovery in a P2P DFS. The various experiments are reported and discussed in Chapter 7, which also describes experiments for the BPD as a whole and how the results corroborate CAS results obtained in the CAS-based simulations. These experiments show how the thesis research addresses the shortcomings discussed in Chapter 1.

The BPD is implemented following the architecture described in Chapter 4. The BPD implementation uses a couple of protocols, described in Section 6.3that are used for the implementation of allocation and discovery DFS services. The BPD implementation makes use of CAS modeling results and experimentally instantiates the Emergent Thinker paradigm (described in Chapter 3).

## 6.1 CAS for DFS Allocation Services Implementation

The CAS squirrel-based class architecture for DFS allocation services is illustrated in Figure 6.1.

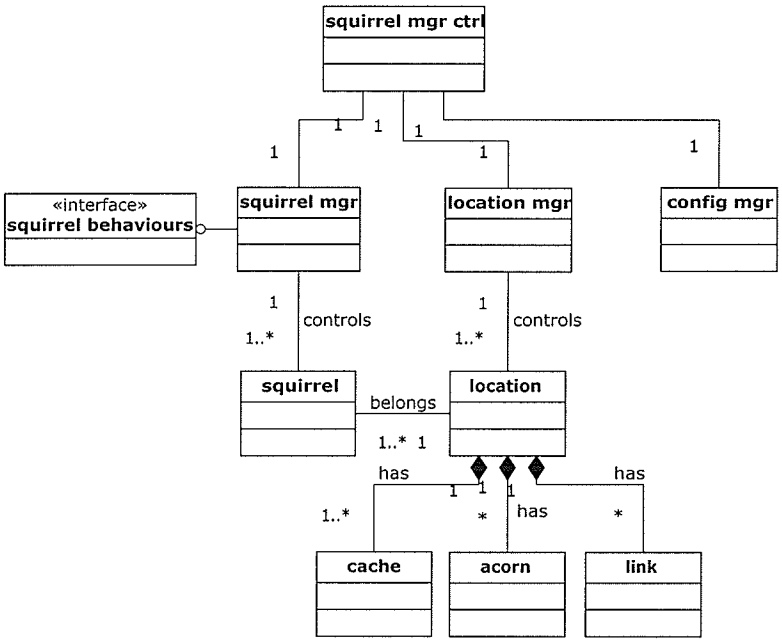


Figure 6.1: Class Architecture for DFS Allocation Services

This class architecture models the software implementation of the Squirrel-based CAS. The class architecture gives the overall picture of a CAS environment and is an instantiation of the CAS propagation and CAS emergent computation models (described in Chapter 3). The architecture supports the common CAS framework characteristics [Var02] including:

- An environment for the CAS members to live in and the system as a whole to be represented,
- a formal representation for each of the members existing on the system,
- evaluation criteria to make decisions at the member level and to evaluate system performance, and
- strategies to adapt member behaviors that consequently will have an impact on the global emerging solutions.

In the software class architecture there are three main groups of classes:

- Environment classes that describe the environment, represented by the location and location manager classes.
- Member classes that describe CAS members with their strategies, evaluation criteria, and behaviors, represented by the squirrel and squirrel manager classes.
- Utility classes to initialize and output results for system evaluation and performance, represented by the squirrel manager controller and configuration manager classes.

The location class, which contains cache, acorn and link classes, represents a geographic area (i.e. a peer) where a squirrel can bury acorns from its home location in places called caches. A cache can store several

acorns. There is also a link class that records paths between the different locations. One or several squirrels belong to a location (i.e. it is their nest). The squirrels pick up acorns from their home location (e.g. peer) so they can be buried in another location's cache. When there are several squirrels in a single (or home) location, they finish faster burying the home location's acorns faster.

The location manager class controls all locations in the distributed system by tracking their locations and managing them. These locations represent the environment where the CAS members live and execute.

The squirrel manager manages the squirrels, which are the CAS members responsible for storing and allocating storage resources to acorns that must be buried. The squirrel manager implements the interface squirrel behaviors, which defines the behaviors supported by this CAS based system. Three squirrel behaviors are implemented and described in Chapter 5. The squirrel manager implements the CAS members' strategies, evaluation criteria, and behaviors.

The squirrel manager controller controls both managers and provides the GUI to the user making use of the Squirrels-based CAS simulator. It provides all necessary interfaces to set up the CAS based system and outputs results to the working CAS-based system for its evaluation and system performance. This is accomplished with the squirrel manager controller using the configuration manager. Figure 6.1 also indicates the associations among the different classes defined in the squirrel-based CAS class architecture.

The Java-based simulator implements the squirrel-based CAS class architecture. The simulator allows dynamic setting of the following parameters:

- Number of squirrels per peer.
- Number of caches per peer.
- Cache size per peer.
- Number of peers.
- Network topology.
- Acorns allocated per peer.
- Number of times to sniff in sniffing behaviors.
- Simulation duration by defining the number of steps to execute.
- Logging of messages and activity on screen and to text files.

These parameters provide substantial flexibility in experimental design as the experimenter can define resource demands (acorn demands), resource availability (caches, cache size, locations), agent density (squirrels per peer), resource location (peer network topology), behaviors (sniffing times, behavior selection) and simulation environment (simulation steps, logging) so experiments can be set up in many interesting ways. Chapter 7 describes experiments that vary these parameters to analyze squirrel emergent behaviors for DFS allocation services for various interesting scenarios.

## 6.2 CAS for DFS Discovery Services Implementation

The CAS squirrel-based class architecture for DFS discovery services is illustrated in Figure 6.2. This class architecture extends the DFS allocation service class architecture and models the Squirrel-based CAS system for discovery services.

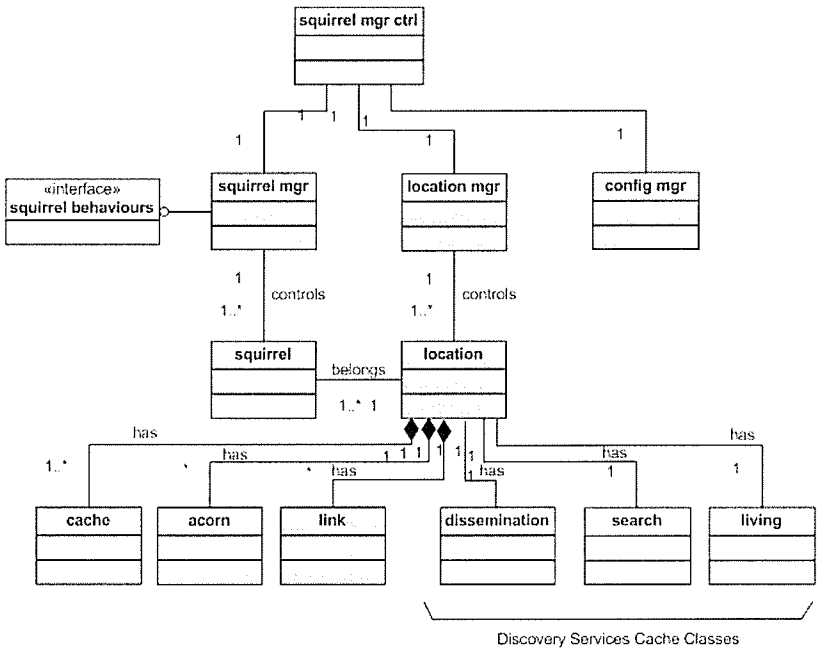


Figure 6.2: Class Architecture for DFS Discovery Services

The location class, which has cache, acorn, and link classes, is extended to include three additional classes: the dissemination, search, and living cache classes. The dissemination cache has acorns from peers (including the local peer) that are ready to be disseminated to other peers by the squirrels.

The search cache has acorns from peers (including the local peer) that are being searched for in the local peer. The living cache has the original acorns being searched for (i.e. the active ones) from the local peer (here, the local peer refers to the peer that owns the three cache instances within the distributed system).

Although not shown in Figure 6.2, the squirrel, squirrel manager, and squirrel behaviors classes are also extended to implement the behaviors and algorithms that are described in Chapter 5 for discovery services. Other classes maintain their meanings as in the class architecture for DFS allocation services.

The DFS discovery services simulator expands the DFS allocation services simulator to support the additional classes and behaviors in the DFS discovery service class architecture. Chapter 7 describes the experiments done using the simulator and analyzes squirrel emergent behaviors for DFS discovery services.

## 6.3 The BPD Prototype Implementation

A BPD software prototype has been built. The BPD prototype utilizes the CAS algorithms I developed and then modeled with the simulators. The CAS algorithms provide the BPD emergent function services and unique characteristics that differentiate the BPD from other P2P DFSs.

Several goals can be identified for implementing the BPD. The first goal is to prove the BPD concepts and instantiation of the Emergent Thinker paradigm.

The second goal is to corroborate the simulation results for emergent services in a P2P DFS implementation that validates the BPD adaptability and scalability across a variety of P2P characteristics. The third goal is to show that systems research has a viable paradigm for research with the BPD instantiation. These goals relate to the current P2P DFS shortcomings discussed in Chapter 1.

The BPD prototype uses the Java language. The BPD runs in each peer as illustrated in Figure 4.4. Most of the BPD design has been implemented as described in Chapter 4. Minor design issues that do not bring value to the thesis have been excluded and are mentioned in the respective layer implementation descriptions (Section 6.3.2 through Section 6.3.4)

Briefly, the data glue layer implements the bag item concept. The glue layer provides services to store, retrieve, manage, and search for bag items (i.e. files) across the P2P system environment. These glue layer services constitute the DFS API interface to client applications. The CAS layer implements both CAS algorithms for DFS allocation and discovery services. The TNC layer communication is TCP/IP socket-based among the peers. The Sections 6.3.2 through 6.3.4 overview each BPD layer implementation. Section 6.3.1 starts with a description of the two basic protocols for DFS emergent function services (i.e. allocation and discovery services).

### *6.3.1 BPD Emergent Function Service Protocols*

Two essential DFS building block functions are allocation and discovery function services. These function services provide support for the

implementation of the P2P DFS API system calls (Section 4.2.2.1). These are described next.

*6.3.1.1 Allocation Function Service Protocol*

From a high level perspective, the protocol for allocation function services receives item blocks (e.g. file blocks) from API system calls. The protocol then allocates the item blocks within the P2P system following one of the Squirrel-based CAS system that was modeled (Section 5.1). Finally, local management tables are updated to keep control of the item blocks. The pseudo-code for the protocol is:

Allocation Function Service Protocol

1. An API system call thread at the peer receives an item (e.g. file) to allocate from a peer client. For example when the item is closed (since the BPD supports session semantics - Section 4.2.5), the close API system call has an item ready to store.
2. The API system call spawns a child thread called Allocation Manager Thread to manage request messages to allocate item blocks (since the BPD implements block granularity – Section 4.2.2.2), then the initial API system call thread returns to the original API caller at the client peer to avoid blocking it.
3. For each item block the Allocation Manager Thread randomly sends an *allocation request* message to other peers. Then the Manager Thread terminates.
4. A new thread called Allocation Worker Thread wakes up when an *allocation request* message is received at a peerand does the following:
    a. Sniffs local peer, executes peer calculations (e.g. cache size available) and then updates the *allocation request* message.
    b. If this is the last sniff (e.g. in CAS simulations I found 3 sniffing steps are usually enough) then:

i. The Worker Thread selects the best location and sends a *call back request* message to the original peer.

ii. The Worker Thread terminates.

c. If this is not the last sniff then:

i. The Worker Thread forwards the updated *allocation request* message to another randomly chosen peer.

ii. The Worker Thread terminates.

iii. The Protocol applies step 4 again for the peer receiving the *allocation request* message.

5. A new thread called Allocation Call-back Thread wakes up when a *call back request* message is received at the original peer. The Call-back Thread does the following:

a. Performs calculations to allocate the item block.

b. Pushes the item block to the selected peer from the sniffed set of peers with an *allocate block request* message.

c. Receives management information from the Allocate Block Thread at the selected peer and updates the local management tables for the item.

d. The Call-back Thread then terminates.

6. A new thread called Allocate Block Thread wakes up when an *allocate block request* message is received at the selected peer. The Block Thread does the following:

a. Allocates a physical block of space and saves the item block in it.

b. Returns management information to the Allocation Call-back Thread.

c. The Block Thread then terminates.

#### *6.3.1.2 Discovery Function Service Protocol*

From a high level perspective, the protocol for discovery function services receives item search requests (e.g. file searches) from API system calls. The

protocol then disseminates the search requests through peer caches until the item is found or the search request expires. The protocol follows the squirrel-based CAS system that was modeled when searching for items (Section 5.2). The search service emerges from the activities of the peers' threads and messages (i.e. squirrels). Emerging search results are returned to the original client application that makes the original search request to the peer. The pseudo-code for the protocol is:

Discovery Function Service Protocol

1. An API system call thread at a peer receives an *item search request* message (e.g. file search) from a peer client. The API system call thread does the following:
    a. Saves the searched for item with attributes into the peer's *living* cache.
    b. Sends a *search locally item request* message with searched item to another peer (might be itself).
    c. Sleeps '*x*' seconds ('*x*' is an integer parameter to the call).
    d. Retrieves searched item statuses (e.g. "item found with peer locations" or "item not found") from the peer's *living* cache.
    e. Replies to the original peer client with the searched item statuses.
2. A new thread called Search and Disseminate Thread wakes up when a *search locally item request* message is received at a peer. The Search and Disseminate Thread does the following:
    a. Saves the searched for item to the local *search* cache.
    b. If the *search* cache has no items to search for then the Search and Disseminate Thread terminates.
    c. For each searched for item in the *search* cache
        i. If the searched item is found in the peer then the thread sends a *found item request* message to the original peer that created the search request.

ii. If a searched for item expires then the thread sends a *not found item request* message to original peer that created the search request.

iii. The thread moves the searched for item to the *dissemination* cache.

d. If the *dissemination* cache has no items to disseminate then the Search and Disseminate Thread terminates.

e. Packs all found searched for items from the *dissemination* cache in a sack and adds the sack to the *search locally item request* message.

f. Forwards the *search locally item request* message to another randomly selected peer.

g. The protocol resumes at step 2 for each chosen peer receiving a *search locally item request* message.

3. A new thread called Call-back Search Thread wakes up when a *found item request* or a *not found item request* message is received at the original peer that inserted the search request into the P2P system. The Call-back Search Thread does the following:

   a. If the searched for item that is referred to in the *found item request* or the *not found item request* message still exists in the *living cache* then the Call-back Search Thread updates the item statuses in the *living* cache.

   b. If the searched for item that is referred to in the *found item request* or the *not found item request* message does not exist in the *living* cache anymore then the Call-back Search Thread ignores the message.

   c. The Call-back Search Thread then terminates.

*6.3.1.3 Protocols Discussion*

There are few things worth noting in the allocation function service protocol and discovery function service protocol implementations. The approach is a best effort one. This means that the current BPD implementation does not

check if the threads are receiving messages sent to them before terminating. This is acceptable considering that the BPD is intended for a research environment to test and validate ideas and is not a commercial product. If the BPD becomes a commercial product then it will require validation and security in message transfers. These capabilities are not important at this stage, as the primary goals stated at the beginning of Section 6.3 are the focus.

Secondly, the CAS squirrel-based environment analogy is implemented with threads and messages. The squirrels are active when threads are spawned at a peer executing different roles as described in the protocols. The squirrels are moving when messages sent by the threads are flowing between the peers. The best CAS member properties, actions and behaviors are used when implementing the protocols. For example, the number of sniffing steps that the squirrels do, best squirrel behaviors for acorn allocation, and peer selection criteria are utilized.

Finally, synchronized access to *living*, *search* and *dissemination* caches in the discovery function service protocol is used. The synchronization is required to avoid squirrels (i.e. threads) racing for shared caches when making decisions and updating caches. This is provided by the native synchronization available in the Java language.

### *6.3.2 Data Glue Layer Implementation*

The data glue layer implements the API system calls made available to peer clients. The glue layer is multithreaded to spawn a child thread each time an API call is made by a peer client. API calls to create, open, close, remove, grab, put and search for items are implemented. API calls are implemented to

support threads for allocation and discovery protocols and for statistics reporting. The API calls implement essential bag item services. These services support the creation, storage, retrieval of, and searching for, bag items (Section 4.2).

Two bitmaps are implemented to manage the block allocation process. One bitmap, called the *Item Block Bitmap,* reflects what peer blocks are free and which are assigned. The other bitmap, called the *Item Metadata Bitmap,* reflects what item metadata structures are free and assigned.. Each group of blocks simulates a cache where squirrels can allocate, retrieve and search acorns among peers. This implementation uses one cache per peer. One cache per peer is sufficient to test for resource balancing among peers because all peers have same cache size.

A global structure called the *global block* contains general peer information like the total number of resources available (e.g. blocks, item metadata) and peer identification information (e.g. IP address, IP port).

A flat user bag holder structure is implemented and runs in each peer. A flat directory hierarchy is enough to demonstrate concepts for bag items with block level granularity (described in Chapter 4). A tree-like directory hierarchy only increases the programming effort but does not contribute to the thesis goals. Because a tree-like hierarchy can be implemented with local tables which the peer client manages.

The user item table structure runs in each BPD peer client and controls item identifiers assigned to each item per BPD peer client session. When a peer client opens or creates an item, an item identifier is assigned. The item

identifier is subsequently used to access the item (e.g. grab, put, remove, or close API calls).

An item metadata structure that has the fields listed in Figure 4.5 is used for bag item control. Item block table structures contain management information and the location of the peers responsible for an item block. The item block table structures allow the BPD to support block level granularity for temporary, persistent, and replica item blocks. The current implementation does not implement the attribute and permission structures. All searches are done for item names. The one attribute per item search is sufficient for testing to establish the main research problem of the emergent search services topic described in previous chapters. Having multiple attributes per item extends the programming effort but does not contribute to the research goals described in Chapter 1. The same applies to the permissions structure.

### *6.3.3 CAS Layer Implementation*

The CAS layer implements the allocation and discovery protocols (see Section 6.3.1). The CAS layer works closely with the glue layer threads to provide the emergent services. Squirrel behaviors modeled are used to implement the protocols. Squirrels are implemented with threads and messages that are dynamically created and removed within the P2P system. The squirrel dynamism comes as a consequence of the P2P activity. The more activity there is in the P2P system, the more squirrels exist; and conversely when there is less P2P activity, fewer squirrels exist.

The P2P DFS is self-organized as new resources are added automatically when new peers join the P2P system. The CAS-based protocols automatically

account for new peer resources and adapt to the current P2P environment. The same is true for the opposite when resources are removed. The BPD self-organizes to accommodate existing available resources. The self-organization is reflected when peer clients access, store, retrieve and search items across the P2P system.

### *6.3.4 TNC Layer Implementation*

Communication among peers is implemented with the TNC layer. The TNC layer implementation uses TCP/IP sockets to permit communication between the peers. A more robust communication scheme to interconnect peers could be easily used though (e.g. JXTA [Sun03]).

A simple name service is used to keep a listing of peers active within the system. This domain name service provides random peers. Squirrels use this service when squirrels are choosing a peer to sniff and/or when searching. Java random functions are used to generate variation in peer selection.

Proxy inter peer clients are implemented to support socket communication. The proxy inter peer client opens the communication channel with the other peers. The proxy supports two basic forms of messaging: either sends a message and returns, or sends a message and waits for reply results. Most of the CAS-based protocol messages use the first mode of communication that consists of sending a message and continuing the thread's execution.

# *Chapter 7*

# 7. BPD Experimental Results

*"Research is what I'm doing when I don't know what I'm doing."*

*- Rocket Scientist Wernher Von Braun*

Experimental results follow the three-phase BPD implementation. Experiments are performed for each phase and the results corroborate the hypotheses set-up across the thesis. The simulators and the BPD software prototype, which were described in Chapter 6, are used to perform the experiments. In each phase, experiments are followed with a discussion section to identify the main results achieved by the experiments. The experimental results support the initial claims from previous chapters.

## 7.1 CAS for DFS Allocation Services - Experimental Results

All Section 7.1 experiments analyze the three algorithms described in Section 5.1 for allocation services. Both sniffing algorithms were run with one to

several sniffing steps each, while the burying algorithm only requires one sniffing step at a time. The system wide resource variance in cache allocation is used as an indicator of how well the P2P DFS resources are balanced. Lower variance indicates better balance, as explained in Section 5.1.

The following metrics and notation are used to present the Section 7.1 results:

- beh 1 = Burying algorithm behavior
- beh 2 – x = Sniffing and burying algorithm behavior with 'x' sniffing steps, x = 1.. r, r>0
- beh 3 – x = Sniffing and burying with no strangers algorithm behavior with 'x' sniffing steps, x = 1.. s, s>0
- deviation = square root of variance in resources allocated (acorns) to locations' caches. This is the final global deviation obtained for all locations' caches. This value is determined after all demand has been allocated across the distributed system.
- % acorns/total cache capacity = percentage of resources to be allocated in relation to the total capacity of all locations' caches.

There are two major sets of experiments. The first experiment set's goal is to verify the Section 5.1 hypothesis, that the *Squirrel-based* CAS algorithms provide the basis for a well-balanced resource allocation across a distributed system. The second experiment set's goal extends the basic hypothesis to verify that balance is achieved and maintained when the scalability and reliability of the distributed system is taken into account. *Scalability* is defined as the sustainability of the system performance obtained in terms of resource allocation deviation when different parameters are increased, i.e. low deviation values remain unchanged despite some parameters being increased. Parameters are increased to evaluate the scalability including number of locations, more focused demand from fewer locations and number of squirrels

per location. *Reliability* is defined as the consistency or repeatability of the deviation value measurements across different scenarios.

For each experiment, the allocation demand starts at time zero. The allocation demand is given by the "% acorns / total cache capacity" ratio. Once this demand is known for a given experiment, the demand is uniformly distributed to a certain percentage of locations randomly chosen that will start generating their assigned demand. The whole demand is then allocated by the location's squirrels within the number of steps taken by the experiment for the complete distributed system. This simulates DFS environments where random proportions of locations generate the allocation loads but the virtual storage layer is supplied by the peer population.

For example, there can be a demand of 30% of storage capacity which comes (is generated) from 10% of the peer population, the remaining 90% may contribute storage resources, but this 90% peer population has no demand for allocation. Once a demand value has been assigned to a peer, the peer squirrels start working to allocate assigned acorns throughout the system.

The experimental plots in Sections 7.1 and 7.2 show deviation and steps results. These results show the system variability when the system changes as a whole under different scenarios. For example, Figure 7.3 shows the variability in the number of steps required to complete allocation for different behaviours across the distributed system as a whole when the demand for allocation ratio changes. Each plot's dot for a given behaviour is a final result after all demand has been allocated. The final result represents an independent experiment under the conditions stated for each experimental group.

*7.1.1 Experiment Set One: Achieving An Efficient Allocation*

In this experiment set, all experiments use a fixed topology of ten nodes and a fixed capacity of twenty acorns per cache. There is only one cache per location. The first set of experiments is divided into three different groups. The global resource allocation balance and convergence time with different resource demands was the main focus of these experiments to prove previously stated hypothesis.

Producers define resource demands that specify requested resources for allocation. A balanced resource allocation occurs when the calculated deviation of acorns allocated across the set of location' caches available in the distributed system is minimal. The smaller the deviation, the better the resource allocation balance obtained. The number of steps the squirrels take to allocate the acorns to be stored determines the convergence time. The fewer steps taken, the faster the whole system is at obtaining the balanced storage allocation. A 'step' is used as a unit of measure because it reflects a better abstraction than other measures units to calculate execution performance. A step is independent of hardware and software utilized in the experiments. For example, a measure unit like time delay to transfer acorns is sensitive and dependent on the network, hardware and software, whereas a step is independent and consequently a step permits a better results comparison between different experimental platforms.

Hence, steps are measured for the sniffing behaviour, for the squirrel and for the system as a whole. A 'sniffing step' is the movement of the squirrel from one location to another location to sniff some value (e.g. current cache capacity). A 'squirrel step' is defined as a series of sniffing steps to find a suitable cache among the sniffed set. For instance, a 'sniffing and burying'

behaviour with 2 sniffing steps means that a 'squirrel step' will take 2 'sniffing steps'. The term 'System step' is defined as the movement of all squirrels simultaneously and independently from their locations to other locations (i.e. executing a 'squirrel step' each) if and only if it is necessary for a given squirrel to move. That is, there could be 'system steps' where only those squirrels with acorns move, while other squirrels will sleep waiting for more acorns to be available to allocate.

'Steps' reported in this experiment set and the other sets refer to 'system steps' unless otherwise noted. For example, Figure 7.3 reports 3 steps for behavior 3-1 when allocating 10% of acorns / total cache capacity. This means that the system as a whole took 3 steps to execute the experiment and in each step the living squirrels that had acorns to allocate moved 1 'sniffing step' each.

Experiment Group 1:

Figure 7.1 presents the results from experiment group 1. These experiments explore resource allocation for all allocation algorithms under simple conditions. Other experiment groups explore additional parameters to analyze other algorithm characteristics. In these initial experiments, there is one squirrel per location seeking a cache to place its acorns in. The demand for resource allocation is evenly distributed across locations: each location wants to store an equal amount of data. The sniffing algorithms are run with 1 to 3 sniffing steps. Beh 2 – 3 and beh 2 – 2 offer the best allocation of resources while beh 3 - x (with x = 1,2 or 3) is between the others. Beh 2 – 2 and beh 2 – 3 obtain a deviation of less than 1 data unit across the whole system for all percentages of demanded acorns.

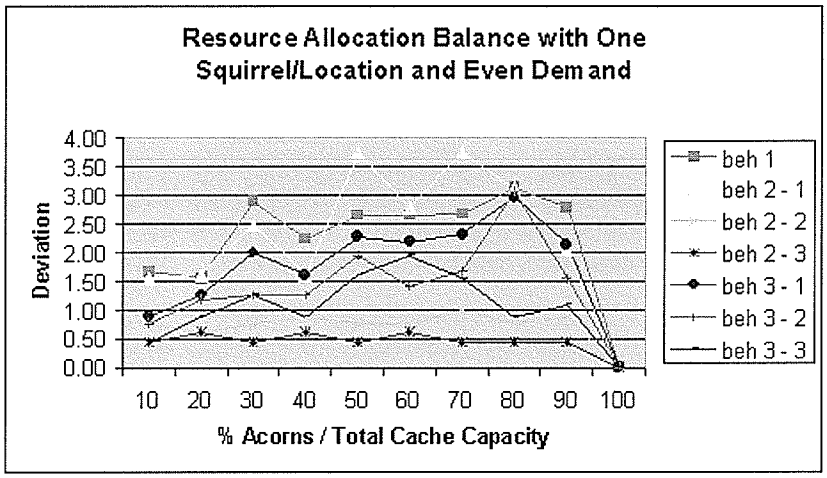


Figure 7.1: Experiment Group 1 Deviation Results

Experiment Group 2:

In these experiments, all conditions are fixed similar to experiment group 1 but the number of squirrels available to allocate resources per location is varied. Figure 7.2 and Figure 7.3 show the results when there are 4 squirrels per location. Better resource allocation was obtained when there were 4 squirrels for all behaviors for the experimental conditions used, suggesting an emergent global result of better resource allocation when more entities are working. Further, there is improvement (i.e. a decrease) in the number of steps required to converge when more squirrels hoard acorns.

Beh 2 – 3 followed by beh 3 – x continue to be the best performers in resource allocations, but beh 3 – x takes longer to converge to a final allocation, primarily when the proportion of acorns to total cache capacity approaches 100% (Figure 7.3). This is expected because when more acorns are to be allocated, the squirrels in beh 3 – x will need more steps to find a suitable

cache for placement because they must avoid another squirrel at the same cache. Thus the beh 2 – x convergence time is faster. For example, when beh 2 - 3 allocates 90% of total cache capacity (Figure 7.3), the whole distributed system takes less than 5 steps to achieve a deviation value of less than 0.5 units for the global resource allocation (Figure 7.2).

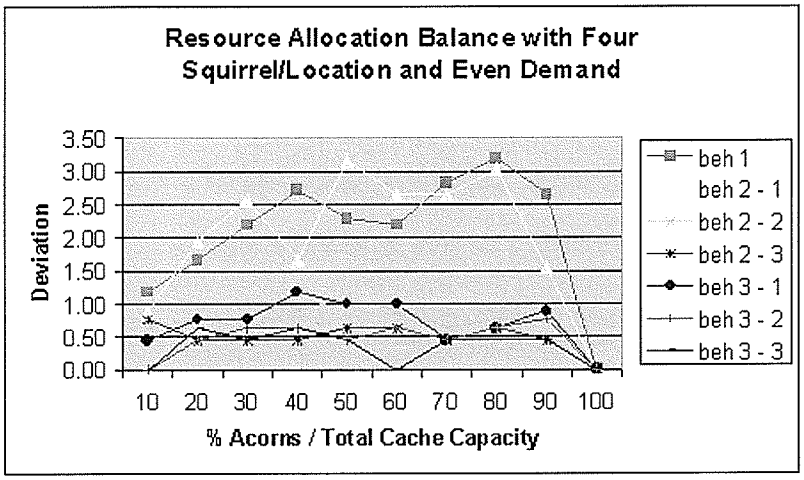


Figure 7.2: Experiment Group 2 Deviation Results

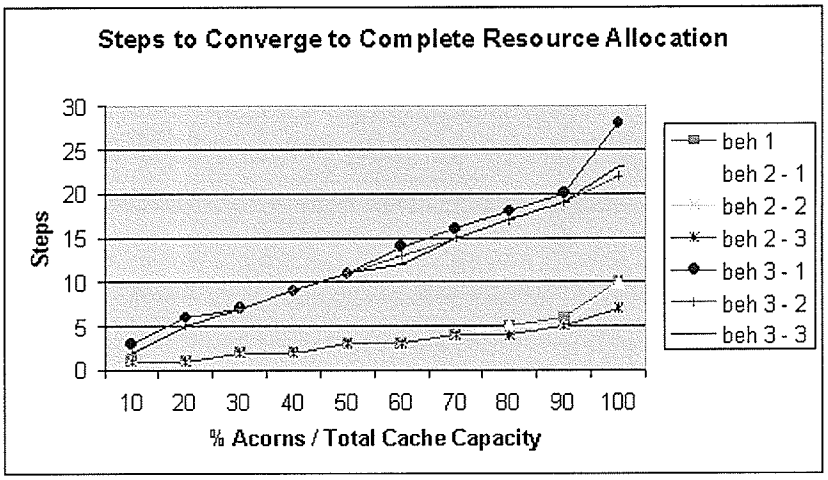


Figure 7.3: Experiment Group 2 Steps to Converge Results

Experiment Group 3:

In this experiment the demand is distributed unevenly across locations. The demand to allocate storage space comes from a few locations rather than all locations. We report two scenarios: a scenario with 50% of locations demanding storage resources (Figures 7.4 and 7.5) and another scenario with 20% of locations demanding storage resources (Figures 7.6 and 7.7).

The percentage of "locations demanding storage resources" means the proportion of locations that require storing data with respect to the total number of locations on the whole system. For example, it is possible that only 20% of the locations generate the entire demand to store acorns on the P2P DFS but all location caches have storage repositories available.

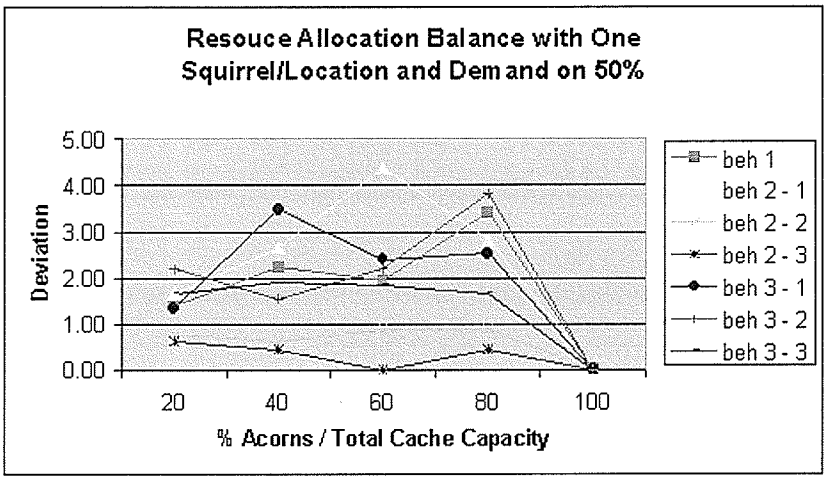


Figure 7.4: Experiment Group 3 Deviation Results on 50% Demand and 1 Squirrel / Location

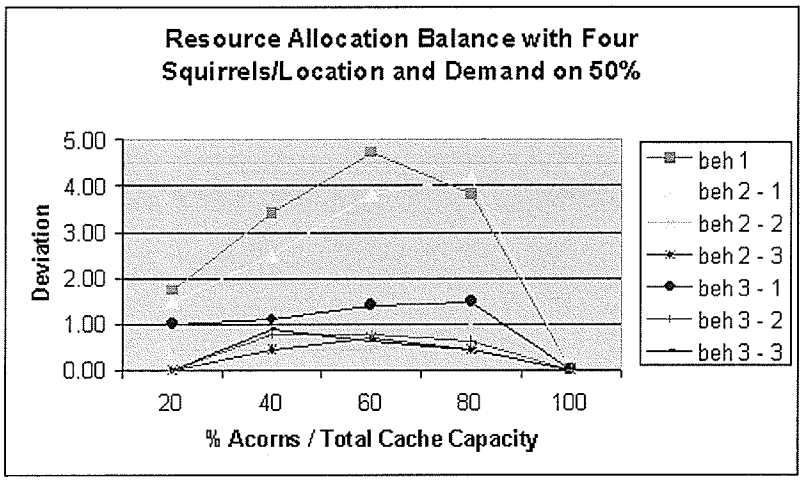


Figure 7.5: Experiment Group 3 Deviation Results on 50% Demand and 4 Squirrels / Location

It is important to note that beh 2 with 1 step (i.e. beh 2 – 1) is essentially the same as beh 1 since only one location is sniffed and selected. Consequently they both produce similar results as expected (Figures 7.4 to 7.7).

In these experiment scenarios, both beh 2 – 1 and beh 1 are greatly affected by demand coming from fewer locations with no improvement in deviation values even if the number of squirrels per location is increased. Beh 3 – x is also affected by a more focused demand on fewer locations, producing large deviation values compared to beh 2 - x (Figures 7.4 and 7.6) unless more squirrels per location are used (Figures 7.5 and 7.7). However, even with more squirrels per location Beh 3 – x deviation values are worse than beh 2 – x (x > 1) deviation values. Beh 2 – x with 2 or 3 sniffing steps produce good system wide resource allocations, generally, with a deviation of less than one data unit. This demonstrates that even with a very unbalanced demand of resources to allocate in the P2P DFS, the "*sniffing and burying*" algorithm with 2 or more sniffing steps is a good CAS algorithm to solve the problem of resource

allocation for the conditions stated in the experiments. The following section corroborates these results when distributed system scalability is taken into account.

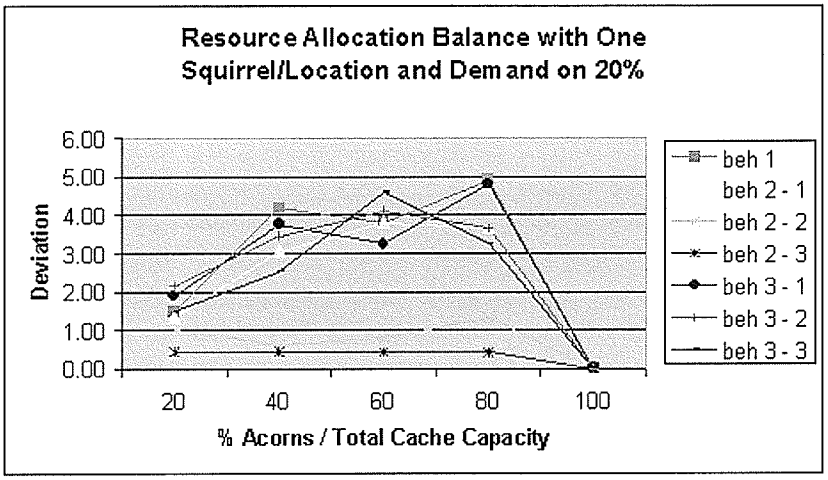


Figure 7.6: Experiment Group 3 Deviation Results on 20% Demand and 1 Squirrel / Location

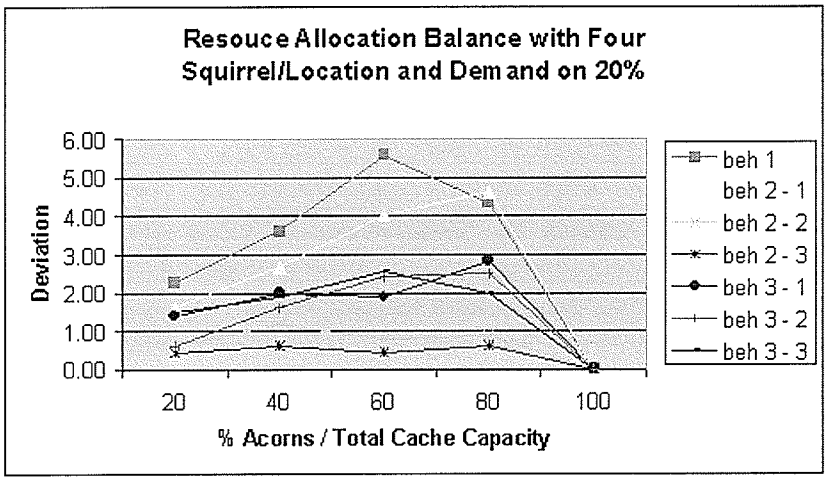


Figure 7.7: Experiment Group 3 Deviation Results on 20% Demand and 4 Squirrels / Location

### *7.1.2 Experiment Set Two: Allocation Scalability and Reliability*

These experiments analyze the scalability and reliability of the *Squirrels* system. The second set of experiments is divided into five different groups (experiment groups 4 to 8). These experiment groups analyze the characteristics of the *Squirrels* system as a foundation for resource allocation in distributed systems. The characteristics of the distributed system are varied in different dimensions including: size, resource demands, resource availability, and location characteristics. Changing these parameters helps to answer the question of scalability and reliability of the *Squirrel-based* CAS for distributed systems resource allocation. These experiments verify the extended hypothesis, which states that *Squirrels-based* CAS is scalable and reliable for providing a well-balanced resource allocation across a distributed system.

Experiment Group 4:

Group 4 experiments analyze the impact of having several squirrels per location to improve the resource allocation balance in addition to demonstrating the scalability and reliability of the algorithms across a span of P2P size configurations. A team or group of squirrels is assigned per location. There is only one team per location. In the experiments, each location has the same number of squirrels per team. If the location has acorns then the location's squirrels work and allocate the acorns across the P2P system, otherwise the squirrels rest until more acorns are generated at the location. When there is no demand at time zero, it is expected that squirrels will wait until more acorns are generated. Each location's squirrel team is independent

from each other (i.e. only the location's squirrels are responsible to allocate the location's acorns). The questions being answered are:

- Do these algorithms scale across a wide spectrum of P2P systems' size?

- What is a reasonable squirrel team size to obtain a good resource allocation balance?

Figures 7.8 and 7.9 show representative experimental results. These results are consistent across P2P systems ranging from those with few peers (10-50) up to several thousand peers (25,600). The peer team size ranges from 1 to 32 squirrels.

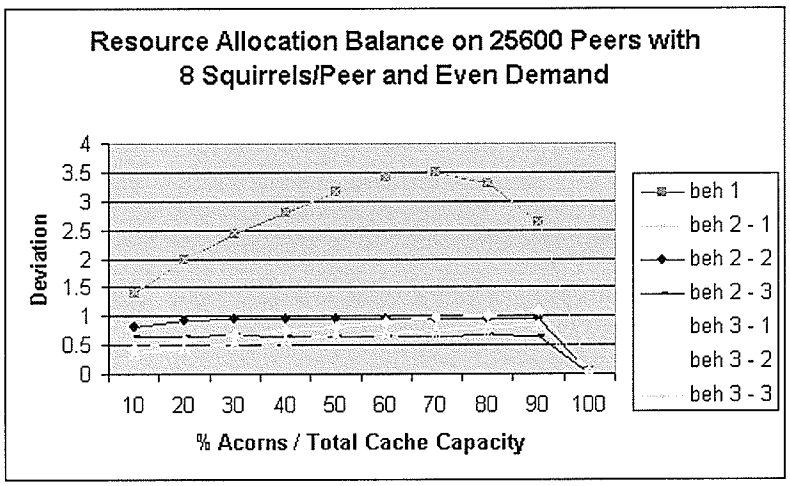


Figure 7.8: Experiment Group 4 Deviation Results with 8 Squirrels / Peer

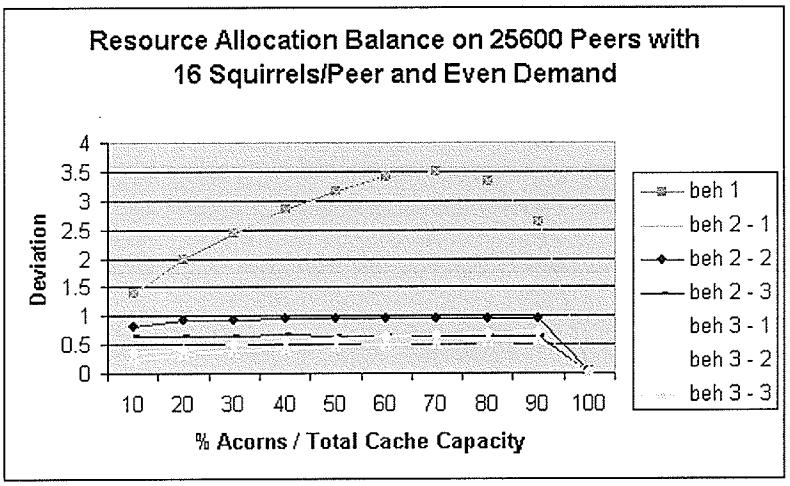


Figure 7.9: Experiment Group 4 Deviation Results with 16 Squirrels / Peer

We observe that team size improves resource allocation balance up to a point where the gains are marginal bringing less resource allocation improvement. This seems to occur experimentally at 4-8 squirrels per team (Figure 7.8), suggesting that larger teams (> 8 squirrels/peer) do not improve the resource allocation balance obtained measurably (Figure 7.9) for the experimental conditions stated. These results are maintained across all experiments carried on from 10 to 25,600 peers with one cache per peer and 20 acorns per cache capacity. The results' reliability also suggests that squirrel CAS based algorithms should scale well to larger P2P systems (> 25,600 peers), presenting similar results as those reported here.

Experiment Group 5:

These experiments analyze the impact of having several squirrels per team per location to improve convergence time. The questions being answered are:

- What is a reasonable team size to obtain good balance in a minimum number of steps?

- What behavior benefits from larger team size?

In terms of number of steps to convergence, Figures 7.10 and 7.11 show representative experimental results that are consistent across different P2P systems ranging from a few peers (10-50) up to several thousand peers (25,600). The number of steps to converge indicates the number of steps the squirrels take to allocate the acorns in the P2P system. Figures 7.10 and 7.11 show the number of steps to converge under different ratios of acorns to total cache capacity. We observe that team size decreases the number of steps required to converge up to a point where the gains are marginal.

Similar to the resource allocation balance experiments, this seems to occur experimentally at 4-8 squirrels per peer (Figure 7.10), suggesting that larger teams (> 8 squirrels/peer) do not significantly improve the number of steps required to converge (Figure 7.11) for the peer cache values used (1 cache per peer with 20 acorns per cache).

Experiment group 8 demonstrates that the cache size and number of caches per peer will have an impact on the number of steps required to do the job, but not on the allocation balance obtained. These results are maintained across all experiments carried out between 10 to 25,600 peers with different numbers of squirrels per peer (1 to 32). This also suggests that Squirrels based CAS algorithms are reliable and may take a reasonable number of steps to find a solution to larger P2P systems (> 25,600 peers).

The "*Sniffing and burying with no strangers*" algorithm (beh 3 – x) improves allocation balance more as the team size increases (Figure 7.9). However its gains are relatively marginal compared to the "*sniffing and burying*" algorithm (e.g. beh 2 – 3). Further, beh 3 – x takes many more steps to converge to a solution compared to the other two algorithms (Figures 7.10 and 7.11).

These results are consistent across all experiments from 10 to 25,600 peers with 1 to 32 squirrels per peer. Figure 7.12 shows similar results when the demand is fixed at 50% "Acorns / Total Cache Capacity" and the number of squirrels per location is varied. Based on these results, I conclude that beh 2 – x is the Squirrels based CAS algorithm that provides both good resource allocation balance and good convergence time for the experimental conditions stated.

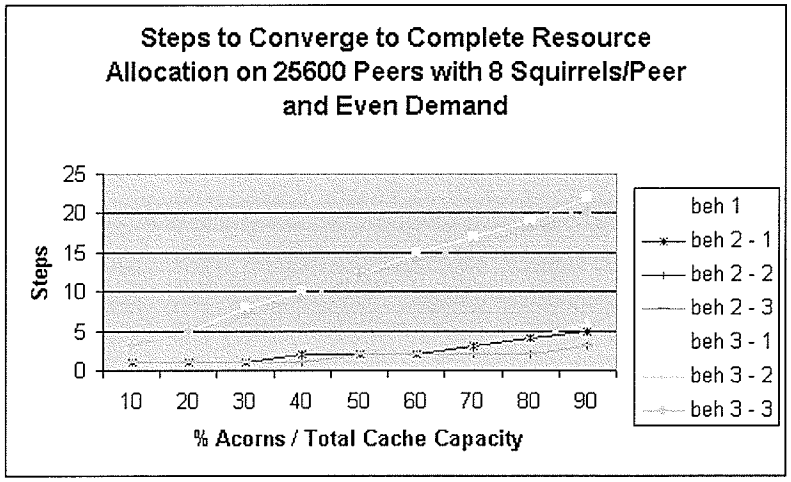


Figure 7.10: Experiment Group 5 Steps to Converge Results

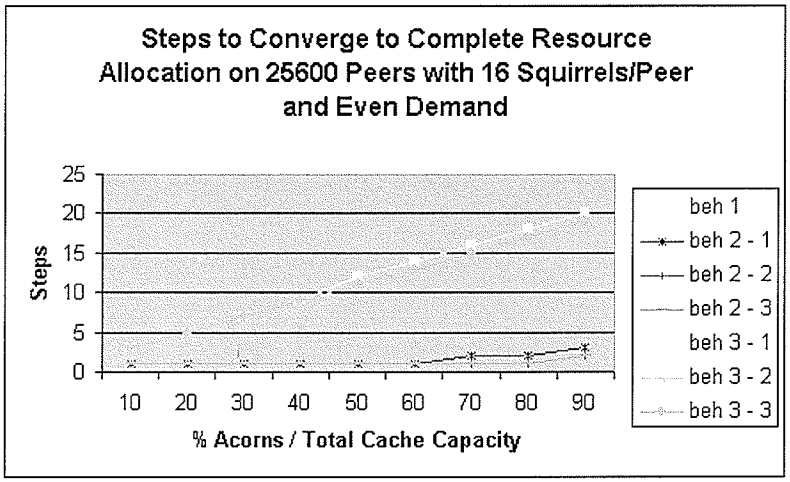


Figure 7.11: Experiment Group 5 Steps to Converge Results

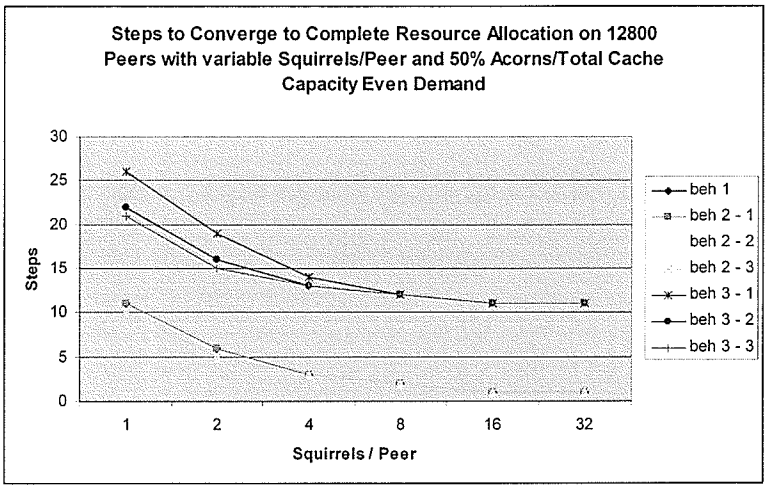


Figure 7.12: Experiment Group 5 Steps to Converge Results

An interesting aspect to note is that my results corroborate natural squirrel behaviors where small teams accomplish their real life hoarding goals (as described in Chapter 2). More effectively my results seem to suggest that the

squirrel CAS models provide reasonable balancing solutions with small teams (4 – 8 squirrels/location).

Experiment Group 6:

These experiments analyze the impact of varying the number of sniffing steps in both *Sniffing Burying* algorithms (beh 2 - x and beh 3 – x). With this group of experiments we answer the question of whether increasing the number of sniffing steps in both beh 2 – x and beh 3 – x algorithms bring improvement in resource allocation balance.

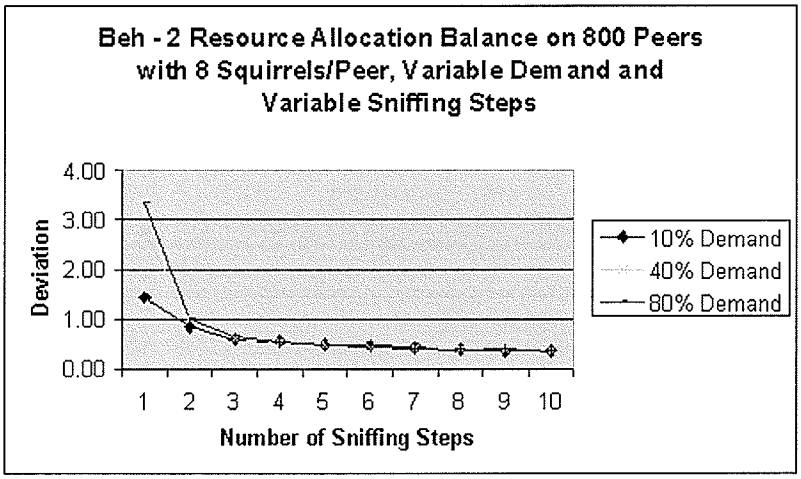


Figure 7.13: Experiment Group 6 Results with Variable Sniffing Steps

Figure 7.13 shows my results for the beh 2 – x algorithm and Figure 7.14 shows my results for the beh 3 – x algorithm. "% Demand" is the percentage of resources that are allocated in relation to the total capacity of all locations' caches for each experiment, when the number of sniffing steps is varied. These results indicate that the beh 2 – x algorithm obtains much of its resource allocation balance with 3 sniffing steps but smaller improvements accrue as the number of steps is increased. A similar outcome is observed for the beh 3 – x algorithm although a relatively greater improvement is obtained than beh 2 - x as the number of sniffing steps increases. Generally, beh 3 - x also obtains a reasonable solution with three sniffing steps. Comparative results were obtained across all experiments carried on from 10 to 25,600 peers with different number of squirrels per peer (1 to 32). These results are also consistent with natural real life observations that squirrels sniff few places before placing their acorns in their caches (as described in Chapter 2).

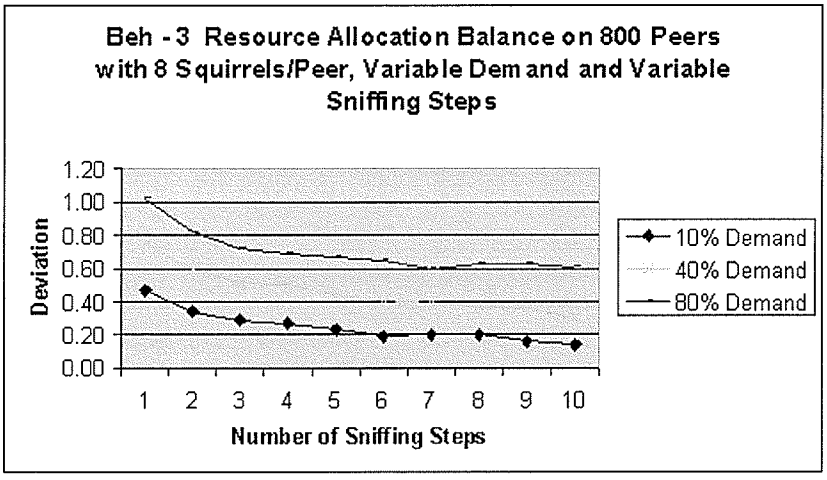


Figure 7.14: Experiment Group 6 Results with Variable Sniffing Steps

Experiment Group 7:

These experiments analyze the impact of changing the percentage of demand sources on the P2P system. A demand of 'x' percentage means that 'x' percentage of the peers generate all requests (demands) for storage. For example in Figure 7.16 a demand of 10%, means that 10% of the 3,200 peers (i.e. 320 peers) are requesting the different ratios of storage demands indicated in the graph's x-axis. The basic question answered with this experiment group is to validate the reliability and scalability of the squirrel based CAS algorithms across different demand percentage values. These values range from 50% to 1% as shown representatively in Figures 7.15, 7.16 and 7.17. Results indicate that beh 2 - x maintains a good global resource allocation balance of less than 1 deviation across different demand percentages. However, beh 3 – x starts to obtain higher deviations on resource allocation balance, as the demand is more focused in fewer peers. This corroborates my previous finding that beh 3 – x will need a larger team size to obtain a reasonable resource allocation balance with a deviation of less than 1.

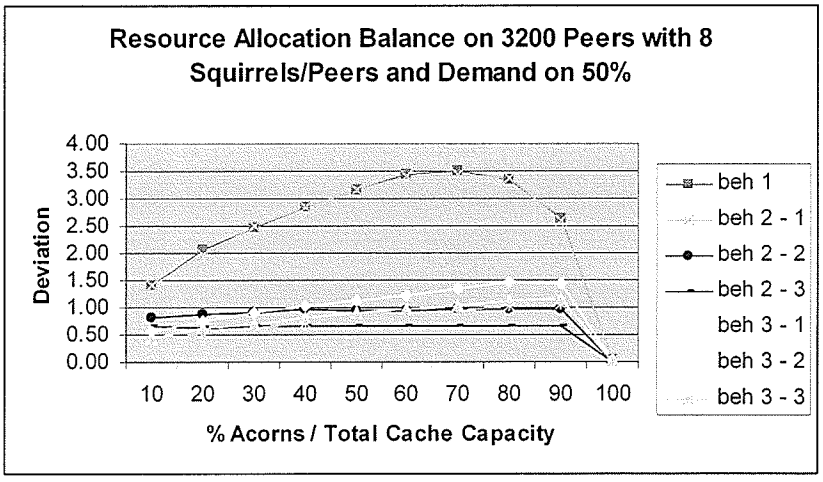


Figure 7.15: Experiment Group 7 Deviation Results on 50% Demand

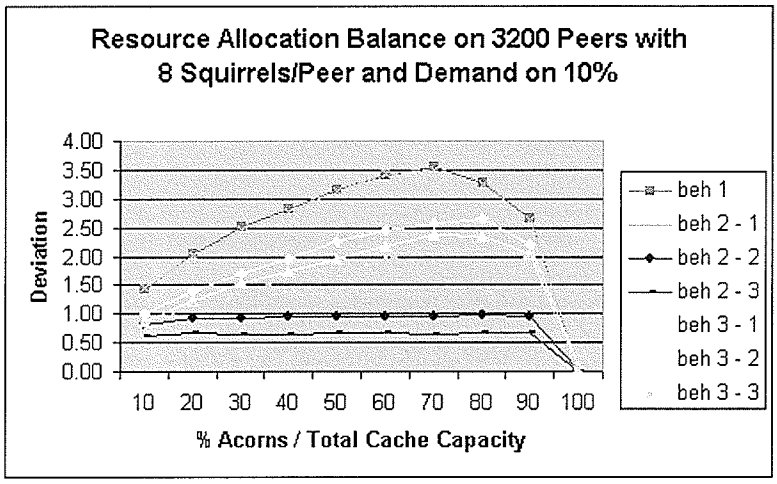


Figure 7.16: Experiment Group 7 Deviation Results on 10% Demand

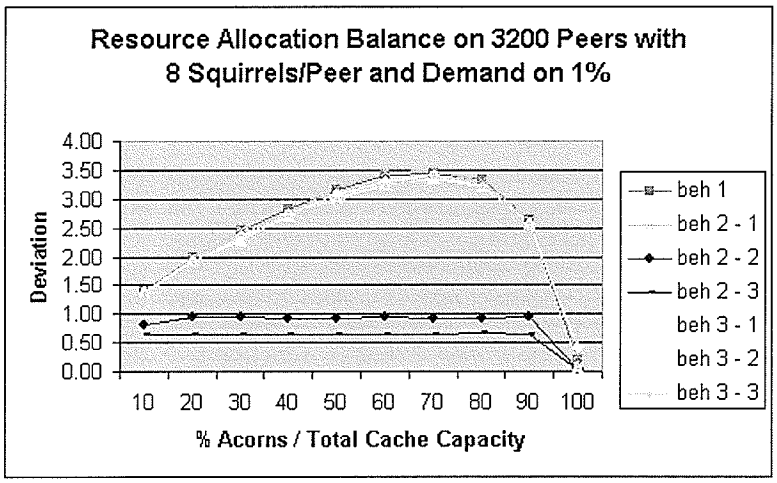


Figure 7.17: Experiment Group 7 Deviation Results on 1% Demand

One important observation in this group of experiments is that as the demand is more focused in fewer peers; all algorithms will require more steps to

converge as shown in Figure 7.18. Increasing the number of agents (i.e. squirrels) will minimize this and the CAS will reach the solution in fewer steps as indicated earlier. The promising result is that the algorithms' scalability and reliability are maintained even with demand coming from fewer peers; particularly the beh 2 - x algorithm.

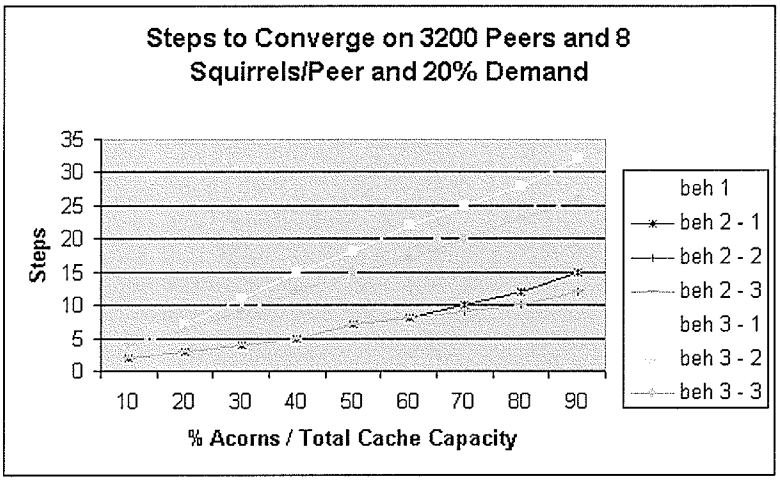


Figure 7.18: Experiment Group 7 Steps to Converge on 20% Demand

Experiment Group 8:

These experiments analyze the impact of increasing the number of caches available per peer as well as increases to the cache size. The amount of data resources to be allocated also increases because it proportionally varies on these two values. The scalability and reliability of the algorithms is tested across a span of P2P DFS size configurations ranging from 10 peers up to more than 25,600 peers.

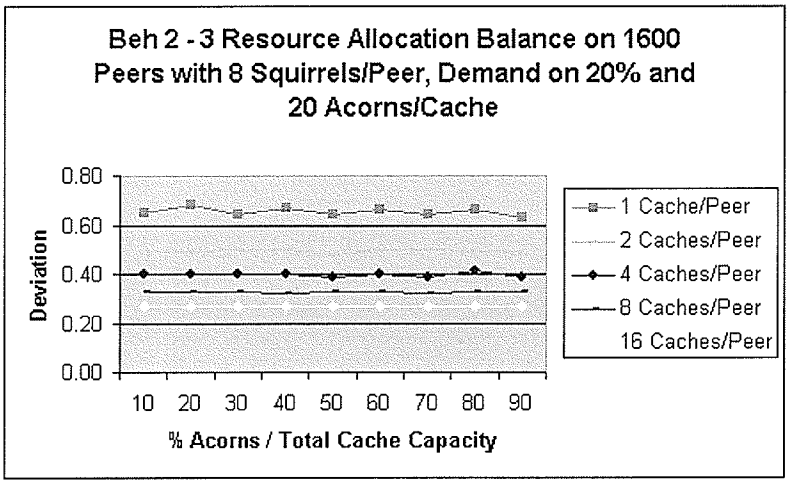


Figure 7.19: Experiment Group 8 Deviation Results for Various Caches per Peer Values

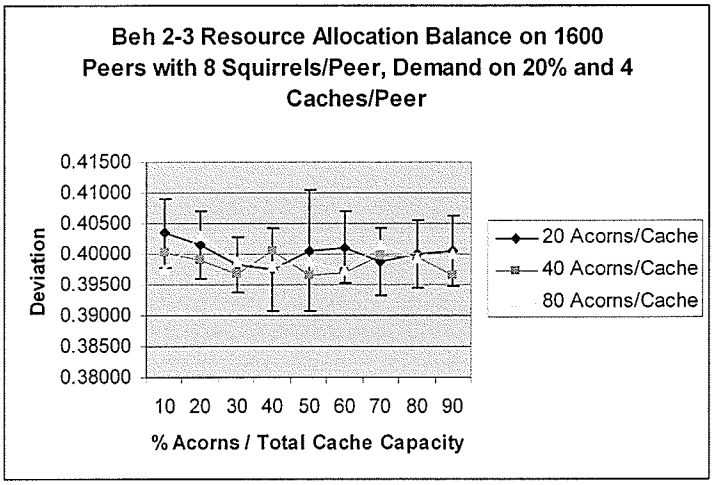


Figure 7.20: Experiment Group 8 Deviation Results with Error Bars for 20 Acorns / Cache

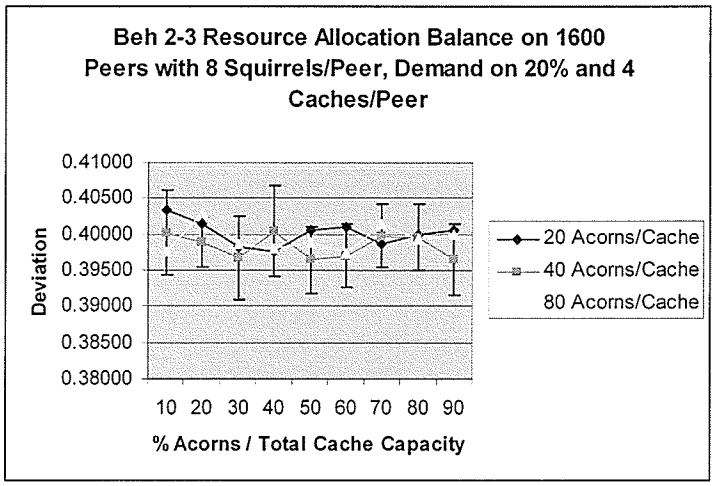


Figure 7.21: Experiment Group 8 Deviation Results with Error Bars for 40 Acorns / Cache

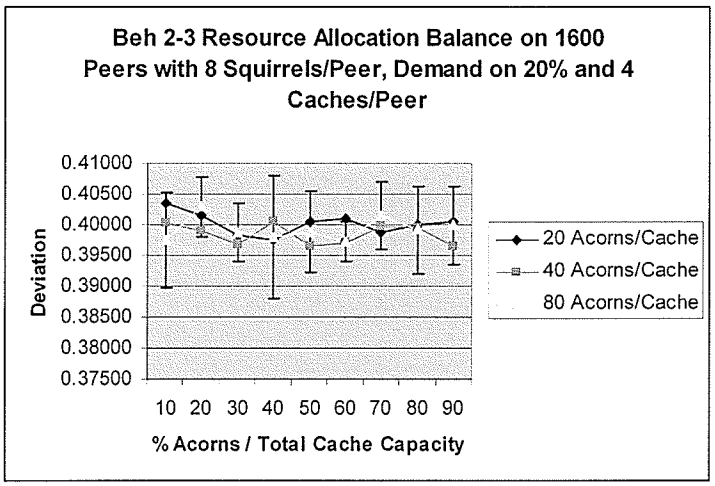


Figure 7.22: Experiment Group 8 Deviation Results with Error Bars for 80 Acorns / Cache

Figures 7.19, 7.20, 7.21 and 7.22 show results for the beh 2 – x algorithm under different scenarios with 1,600 peers. Figure 7.19 shows results when the number of caches per peer is changed. Figures 7.20, 7.21 and 7.22 show results when the number of acorns per cache is varied. Error bars in Figures

7.20, 7.21 and 7.22 indicate the standard deviation of the final deviation value found for 10 repeated experiments at the given "% Acorns / Total Cache Capacity" ratio. In both scenarios, the beh 2 – x algorithm maintained a good balance of resource allocation across P2P configurations tested. These results support the scalability and result reliability of the beh 2 – x algorithm given variability of the peer capacities.

### *7.1.3 Allocation Services Experimental Results Discussion*

Experimental results show that small teams are required to effectively balance resource allocation across a collection of P2P systems ranging from a few peers up to tens of thousands of them. Results reliability suggests that low deviation values in resource allocation are maintained irrespective of P2P system size. The emergent behavior resulting from the activities of each squirrel allocating storage resources obtains a well balanced P2P system in a reasonable number of steps. The larger the team per peer the faster a solution can be obtained, however more load on the system could occur. Consequently, a tradeoff between the number of agents (squirrels) working and acceptable speed to obtain a solution is required. The trade off for the conditions set up in the experiments indicates that team sizes of 4 – 8 squirrels per peer will obtain solutions in only a few steps across different P2P sizes, offering scalability in its application no matter the P2P system size. The consistency of results indicates that a solution is obtained reliability.

Both sniffing algorithms require few sniffing steps to make a good decision. A value of three sniffing steps seems to give both sniffing algorithms an emergent behavior of system wide storage balance that is relatively good in terms of low deviation values for resource allocation and in the number of

steps to converge to a global solution. This result is important, because without it, the whole system will take longer to converge to a solution if more sniffing steps are added in each allocation cycle. We want an algorithm that can achieve a balanced resource allocation with as few sniffing steps as possible, which is also scalable and reliable.

Requests to store or allocate resources can be evenly distributed across all system members or biased to a few members that consume the majority of resources available. Experimental results show that proposed algorithms, particularly the *Sniffing and Burying* algorithm, obtains a well balanced resource allocation across different demands ranging from evenly distributed down to 1% of peers being responsible for requesting storage resources. These results are obtained even with small teams of squirrels, though larger numbers of steps to converge to a solution are required as the proportion is reduced. Choosing the team size will depend on how fast a solution is required and on peer capabilities to support larger team sizes.

In terms of increasing the peer capacities by varying the number of caches available and the cache size, the *Sniffing and Burying* algorithm continues to produce, in the allocation experiments, a well-balanced resource allocation across different P2P systems with various resource demand proportions.

The *Sniffing and Burying* algorithm's scalability and result reliability is maintained across different P2P system configuration metrics (P2P size, resource demands, resource availability and peer characteristics). All experimental results obtained indicate the robustness of the algorithm's global outcome that emerges from the individual activities of each of the agents (i.e. squirrels) working with no direct interaction among them.

Squirrels use *stigmergy* to coordinate their behaviors and achieve the system goal of resource allocation balance. The *Sniffing and Burying* algorithm is a squirrel based CAS algorithm that solves the complex problem of resource allocation balance on a P2P distributed system. Further, due to the similarity with resource allocation on distributed systems, *Sniffing and Burying* algorithm might be a good resource allocation algorithm for any kind of distributed system (e.g. P2P, grid, or any other distributed system).

This experimental research work has focused on storage resource allocation, while retrieval has been left aside. This is because the implicit assumption is that once an acorn is allocated, local information manages its location, thereby making it trivial for the peer to retrieve the data previously allocated (Section 4.1.3). Most interesting is the issue of distributed data discovery. Discovery services experimental results are described next.

## 7.2 CAS for DFS Discovery Services - Experimental Results

The experiments in Section 7.2 analyze the CAS emergent search algorithm described in Section 5.2 for discovery services. The Java software simulator described in Section 6.2 was used to run the experiments. The simulator runs test cases with thousands of locations using different parameters. This section compares the obtained results to blind search algorithm performance results. Blind search algorithms do not have any, or use only minimum information, about other peers or object's locations when searching for data in a distributed system.

Experimental results were compared to results from the random walkers' blind algorithm that uses Breadth-First Search (BFS) [Lv02, Yan02]. The random walkers' algorithm selects a set of neighbors' nodes to set up search paths using "walkers". A walker is a software agent that it is created at the source search location and which represents the query message. The basic idea of random walkers is to forward a query message to a randomly chosen neighbor at each step until the object is found. To reduce delays, the number of 'walkers' can be increased. For further details on the random walkers' blind algorithm refer to Lv *et al.* [Lv02].

In this thesis the random walkers' blind algorithm is called the *Neighbor Set BFS*. The Neighbor Set BFS algorithm presents relatively good search results with a relatively good use of resources compared to other blind algorithms [Tso03]. Another reason for comparing experimental results to Neighbor Set BFS results is because the CAS Emergent Discovery algorithm is most similar to blind search algorithms for its self-adaptability to peer failure.

For each experiment, data was generated across the P2P system using the balanced resource allocation algorithms described in Chapter 5 [Cam03a, Cam03b, Cam04a]. The allocation algorithm used obtains a balance of resources under many P2P characteristics. Its resource allocation deviation is usually less than one, which means that each location participates according to its storage capabilities and a system wide resource balance is achieved.

For the Section 7.2 experiments the storage ratio is 40%, which means that the system-wide total sharable storage capacity utilization is 40%. Each storage cache location has a 20-acorn capacity. The location search ratio is the proportion of the total number of locations that are searching for data. The location search ratio is set at 40%, unless noted otherwise. For example, if there are 1000 peers in the system, then 400 peers will be searching for data.

Data search requests can be divided in two groups. Searching for known data and searching for unknown data. Known data is data created at a given location and searched for by the same location. Unknown data is data created by a location but searched for by a different, random, location not necessarily related to original location (e.g. searching for the existence of a given file). This research explores the emergence of discovery services within a distributed system. Hence searching for known data is controlled and managed by the BPD data glue layer tables (Chapter 4) and poses no challenge from the emergent perspective. The challenge is the emergence of discovery services for unknown data when many locations are working together so the search service can emerge. Thus, the experiments reported search only for unknown data to evaluate the emergence of discovery services.

Unknown data to be searched for is randomly generated and assigned to the number of locations that are searching for data (i.e. the location search ratio).

The assigned locations are randomly chosen. For example if an experiment searches for 5% of allocated data with 30% location search ratio. The 5% data is randomly selected from the total data allocated and this data is randomly assigned to 30% of the existing locations, which are also randomly chosen. The 30% of locations will generate a search (if there is enough data for each since the number of locations can be greater than the number of searches to carry out) but all locations could participate in the search. Hence the discovery service emerges from the activities of all locations as shown in the following experimental results.

Three experiment sets are presented in this section as representative of the experiments undertaken. The first set focuses on comparing the CAS Emergent Discovery method with the Neighbor Set BFS method for different P2P system sizes. Set two analyzes the impact of the number of dissemination steps in the emergent results (i.e. fan-out impact) while the third set analyzes the performance when the data search percentage increases with respect to the total amount of stored data.

### *7.2.1 Experiment Set One: The Achievement of Efficient Discovery*

These experiments compare the performance, in terms of number of steps, when the CAS Emergent Search and the Neighbor Set BFS find 100% of the requested search data. In the Emergent Search, a step is a complete dissemination of the squirrel (i.e. dissemination fan-out). In the Neighbor Set BFS, a step is a complete search in each walker's neighbor set by all the random walkers.

For Emergent Search, a sack dissemination fan-out of 5 locations is used with one squirrel per location to disseminate sacks into other locations. Refer to

Section 5.2 "CAS Algorithms for DFS Discovery Services" for further details on the Emergent Search algorithm.

For Neighbor Set BFS we use a 5-neightbor set size per random walker and 8 random walkers per location. Consequently, Emergent Search uses 8-times fewer messages than Neighbor Set BFS per step when all random walkers are disseminating data.

The results are shown in Figure 7.23. As the number of locations grows, Neighbor Set BFS takes longer to find all search results whereas Emergent Search not only finds them in fewer steps but also shows scalability by maintaining a smaller step count across different P2P sizes.

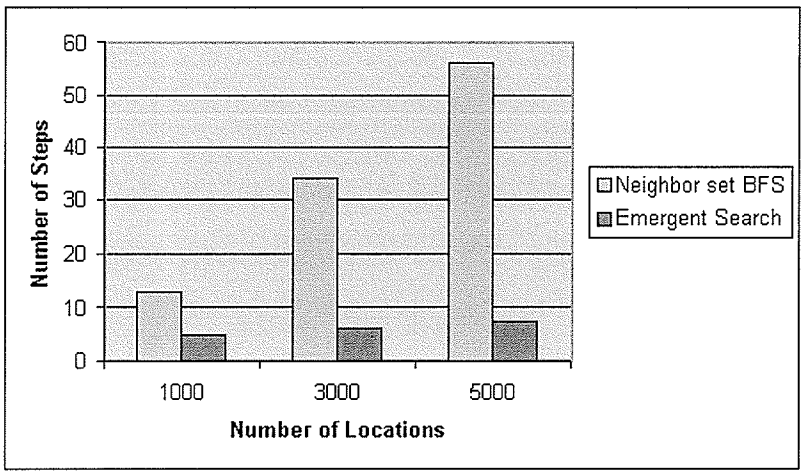


Figure 7.23: Number of Steps per Number of Locations Performance

*7.2.2 Experiment Set Two: Discovery Services Fine Tuning*

These experiments evaluate the impact on search when we have different dissemination fan-out for both Emergent Search and Neighbor Set BFS. We use a P2P system of 4,000 locations and 5% search proportion with respect to

the total stored data. This means that from all sharable data, 5% of it is being search for. Experiments to search for 100% of this 5% search proportion were undertaken. The results shown in Figure 7.24 indicate that the Emergent Search requires only small dissemination fan-out to successfully find 100% of the search data with fewer steps than the Neighbor Set BFS algorithm. Emergent Search is reliable and maintains approximately the same number of steps for a broad range of dissemination fan-outs. This means that with 3 or 4 fan-out disseminations, Emergent Search provides a P2P global search using few steps. Neighbor Set BFS requires many neighbors per set for each location's random walker. This means that Neighbor Set BFS uses more messages and is slower to provide a system wide search service as compared to the Emergent Search.

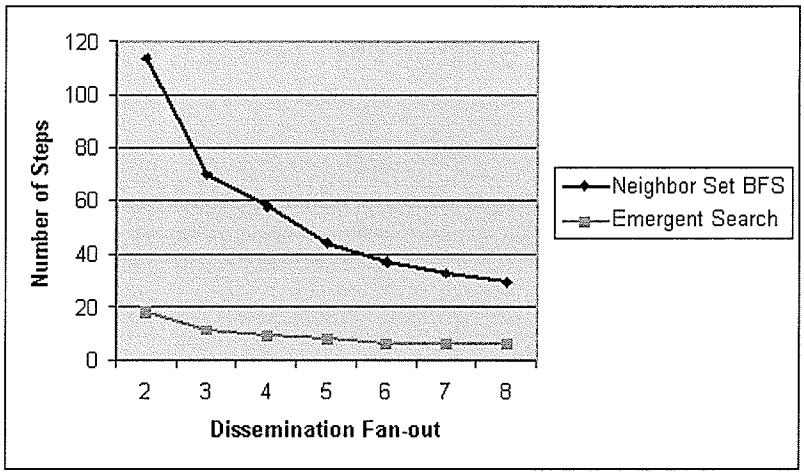


Figure 7.24: Number of Steps per Dissemination Performance

*7.2.3 Experiment Set Three: Discovery Scalability and Reliability*

In these experiments the impact of an increase in search percentage relative to the total stored data on the performance of both Neighbor Set BFS and Emergent Search methods is explored. A P2P system of 1,000 locations is used and search ratios ranging from 10% to 90% of the total stored data. In all results the 100% of the search ratio is used (e.g. in the case of 10% ratio, we search for 100% of this 10%). The results (Figure 7.25) indicate that Emergent Search scales and maintains a reliable good performance across the different ratios whereas Neighbor Set BFS explodes thereby requiring many disseminations steps to find all requested data.

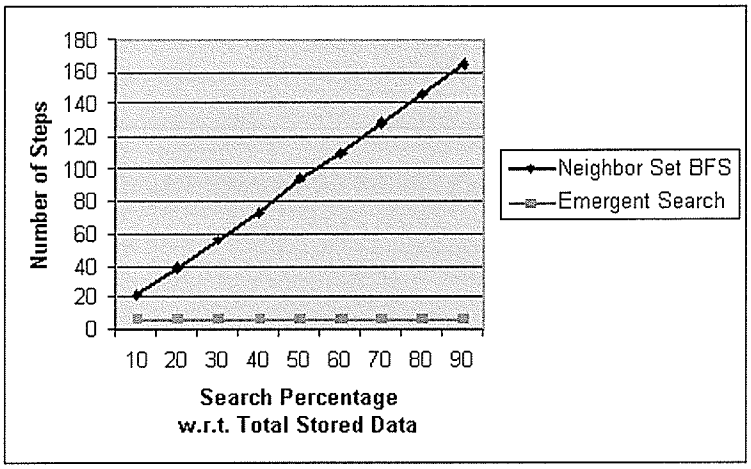


Figure 7.25: Number of Steps per Search Percentage Increase Performance

*7.2.4 Discovery Services - Experimental Results Discussion*

There are different areas where the CAS Emergent Discovery algorithm can be modified to improve the emergent search outcome. For example, each time an acorn is disseminated to a location we can check to see if the acorn has been

deposited there before to avoid search duplication. Acorn life expectancy and termination conditions can also be used to abort its dissemination. Correct dissemination aborts could potentially improve message size, bandwidth, and computation overhead but at an administrative cost. A commercial implementation or future research work could include these optimizations and assess their impact on the algorithm's simplicity and performance. The experimental implementation checks with the original search location to abort acorn dissemination if the acorn has already been found.

Current squirrel behavior to disseminate acorns has been kept simple to facilitate randomly jumping to the nearest location available. However, it has been shown in CAS allocation that sniffing locations before dissemination [Cam04a] helps to understand current location usages that improve global outcomes. A future research approach could attempt to explore different behaviors and evaluate their global impact. Changes to acorn sack sizes and the number of squirrels used to disseminate sacks may accelerate the global outcome. The current implementation uses only one squirrel that takes the location's sack and disseminates it to other locations. However, future research might explore having the same sack disseminated by different squirrels (i.e. a more aggressive approach) or set up a limit in the sack size and wake-up squirrels as sacks are filled up to a limit according to the location capacity to afford squirrels.

## 7.3 BPD Implementation - Experimental Results

The goal of the BPD implementation experiments is twofold. The first is to demonstrate that the Emergent Thinker paradigm, which is based on CAS emergent computation, can be instantiated and used to provide DFS services. The BPD implementation demonstrates the Emergent Thinker instantiation. Executing applications that use DFS services from the BPD in a P2P network demonstrates that the BPD effectively delivers a truly P2P DFS. The first experiment set provides some application examples that use the BPD for their file management services in a P2P system. These examples show how applications can make effective use of the BPD system calls when requesting file services.

The second goal of the BPD implementation experiments is to demonstrate that previous results obtained in the simulators are actually provided by the BPD. This is important because it shows that CAS-based computation is highly adaptable and scalable across a variety of P2P characteristics on real implementations and not only in simulation. The BPD adaptability and scalability makes it suitable for the dynamics of unstructured P2P systems.

Experiment sets two and three target the corroboration of results for the BPD in terms of allocation and discovery services, which are the two essential building blocks to provide DFS services (as described in Section 4.1.3).

For all experiments in this section, the peers were run with a different assigned IP port number in a large Linux computer. This approach is acceptable because it allows experimental control and validates research goals. The disadvantage is that there is a limit in the maximum capacity of peers that the

large Linux computer can handle. The limitation is established by the Linux computer memory and processing capacity. However, with the resources available at the University of Calgary Computer Science Department I ran fairly large P2P networks that help sustain the claims and the disadvantage was minimized. Another disadvantage is related to the peers sharing the CPU processing. Sharing processing might seem a constraint but in reality is a worse case scenario for the CAS-based computations, because even though each peer thread is independent in terms of computations, they have to wait for execution on the scheduling assignment within the Linux OS thereby limiting the thread execution concurrency. This means that CAS-based computation is slower, but the emergent outcomes are still present.

### *7.3.1 Experiment Set One: The Achievement of a Workable BPD*

Java was used to implement the BPD. For convenience, client applications that make use of the BPD are written in Java too. The BPD exports an API interface class that client applications import to make use of P2P DFS services. Small client applications were developed to test the BPD. Client applications to carry out the following sample functionality were implemented:

- create items
- create 'n' items
- create, put and close items
- create, put, close and remove items
- create and put items
- open items
- open and grab items
- open and remove items

- search for an item
- search for 'n' items

The following Java program listing shows a stripped example of the basic structure of a client application using the BPD. The structure shown is for the "create, put and close items" demo application.

```
import bpd.client.*;

public class createPutCloseDemo {
  private bpdClientAPI myBPDClient;

  public createPutCloseDemo() {
  }

  public void initBPDClientAPI(String ipadd, int portno) {
    myBPDClient = new bpdClientAPI(peerip, portno);
  }

  public int creatingItem() {
    int opStatus = 0;
                                        //item is created
    opStatus = myBPDClient.createItem("item01", 0x00);
    return opStatus;

  }

  public void puttingItem(int itemid) {
    int opStatus = 0;
    byte[] buffer = new byte[500];

    for (int i = 0; i < buffer.length; )
        buffer[i++] = 'A';          //some data is written
    opStatus = myBPDClient.putItem(itemid, buffer,
                                   buffer.length);
  }

  public void closingItem(int itemid) {

    int opStatus = 0;              //item is closed
    opStatus = myBPDClient.closeItem(itemid);

  }
```

```
  public static void main(String[] args) {

    createPutCloseDemo clientApp;   //client app instance
    int itemId;                     //item id
    String peerIP;                  //peer ip address
    int portNo;                     //peer port no

    peerIP = bpdClientConstants.defaultServer;
    portNo = bpdClientConstants.defaultServerPort;

    clientAppDemo = new createPutCloseDemo();
    clientAppDemo.initBPDClientAPI(peerIP, portNo);
    itemId = myDemo03.creatingItem();     //create item
    clientAppDemo.puttingItem(itemId);    //put data in item
    clientAppDemo.closingItem(itemId);    //close item

  }
}
```

Test client applications used in the experiments usually first import the BPD API client interface class and any other classes the application needs. Then client methods are implemented for each of the services the application does besides the initial constructor. Methods to initialize the BPD proxy client, create item, put data in the item and close the item are shown in the example code. The BPD proxy client implements TCP/IP socket communication to the peer. The TCP/IP socket communication sends and receives client requests for DFS services. The BPD proxy client is used to send API system calls to the peer (e.g. create item) and receive status and results back to the client. Subsequently the BPD peer will receive the client requests and process the requests within the P2P system. Finally, at the end of the sample application listing, the main method executes the client methods to access file management services in the P2P DFS. The main method constitutes the execution path for the client application.

Furthermore, Linux test scripts were developed to run many client application instances requesting services to the peers available within the distributed system. Other scripts were executed as well to collect statistics for allocation and discovery services when hundred of peers were using the BPD. These experimental results are shown next.

### *7.3.2 Experiment Set Two: The BPD Allocation Services*

A P2P system was set up for experiment set two. Peers had a maximum capacity of 2048 items and 10 blocks per item. The peer storage capacity was set to 8192 blocks. Each block was 16 kilobytes. There was only one cache per peer with a capacity of 128 megabytes (i.e. 8192 blocks times 16 kilobytes). At maximum usage, each peer could allocate the 2048 items with 4 blocks per item in average. Different capacities could be set up in the peers but these values gave more than enough capacity to carry on the experiments to validate previous simulation results.

Each time an experiment is run, a clean and new P2P system environment was built with the capacities specified in the last paragraph and a variable number of peers. Figure 7.26 and 7.27 show representative results for two P2P systems with 500 and 700 peers, respectively. An experiment consisted of creating a clean P2P system environment and then creating client applications that requested DFS services. The client applications were independent threads that stored items in the P2P system. Each client application stored 10 items and each item had 2 data blocks. The x-axis in Figures 7.26 and 7.27 indicate the number of client applications that were concurrently requesting DFS allocation services.

The BPD implements the CAS allocation algorithms by using the squirrel behavior properties that emerge with a self-adaptable system-wide resource allocation balance. The BPD implementation uses the sniffing and burying algorithm with three sniffed steps. The sniffing and burying algorithm is the best squirrel behavior that provides resource allocation in the CAS simulations. The allocation function service protocol implements the sniffing and burying squirrel algorithm (Section 6.3.1.1).

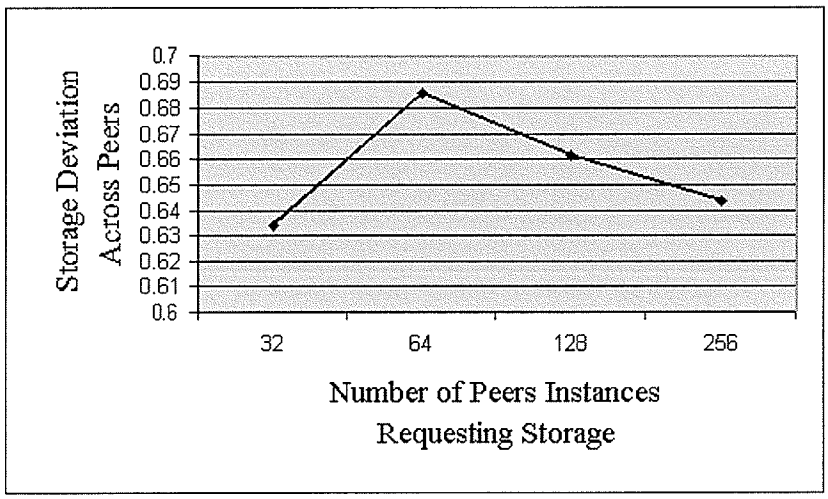


Figure 7.26: Storage Allocation in a 500-peer P2P System

The CAS-based allocation shown in Figures 7.26 and 7.27 obtained a balance of storage resources under a variety of P2P characteristics (including P2P size, number of peers storing data, P2P system-wide storage capacity utilization, and peer storage capacity). The storage allocation deviation was usually less than one, which means that each peer participated according to its storage capabilities and a system wide resource balance was achieved. The results in

Figure 7.26 and 7.27 are similar to those obtained in the CAS algorithm simulations carried out in P2P systems with more than 25,000 peers (Section 7.1) [Cam03b, Cam04a].

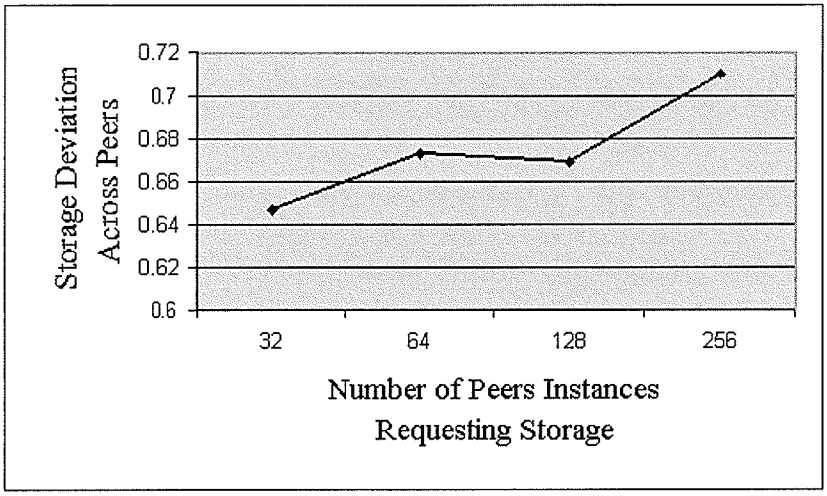


Figure 7.27: Storage Allocation in a 700-peer P2P System

*7.3.3 Experiment Set Three: The BPD Discovery Services*

Similarly to experiment set two, each time an experiment was run in experiment set three, a clean P2P system environment was built. The P2P system environment had the same capacity characteristics as those described in Section 7.3.2 and a variable, large number of peers. After a clean P2P system environment was built for an experiment, items were allocated across the P2P system. These items were searched for to test the discovery services.

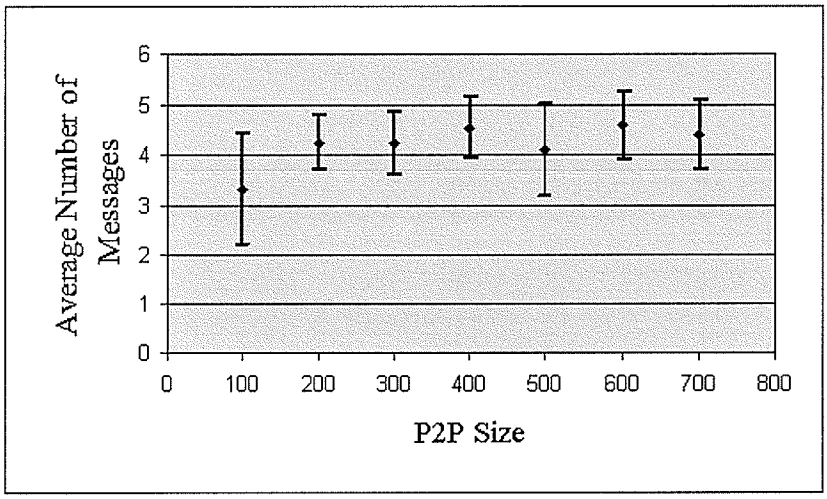


Figure 7.28: DFS Search Scalability

Representative discovery service results are shown in Figures 7.28 and 7.29. For the first group of results (Figure 7.28), the number of peers varies in the P2P environment and a fixed number of concurrent peers execute searches for data. The average number of messages and the standard deviation were collected. The average number of messages is only computed when the BPD successfully searches for all requested items. A message is an acorn sack transfered from one peer to another (Section 5.2).

Figure 7.28's x-axis shows the P2P size used in each experiment. The number of peers searching for items was set to 10 in each P2P size experiment. Each peer searched for 2 items. In terms of sharable storage (i.e. items available for searching), there were 100 peers with 10 sharable items per peer. All stored and searched data were randomly generated in each experiment. The peers that participated in the experiments (either allocating or searching for items) were randomly selected as well.

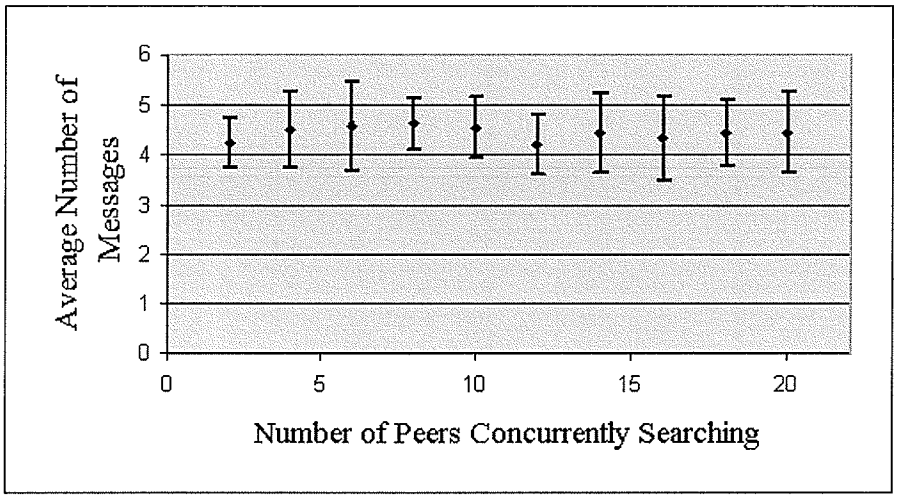


Figure 7.29: DFS Concurrent Search

For the second group of results (Figure 7.29), the number of peers is fixed at 400 and the number of concurrent peers searching for data varies. The average number of messages and the standard deviation were collected. There were also 128 peers with 10 sharable items per peer. Peers and data were randomly selected and generated in each experiment too.

It is important to stress that in both groups there was no data replication; there was only one copy per item; and no indexes or auxiliary item location tables existed. The search emerges as a global outcome of the behavior of the CAS members that disseminates acorn sacks across the system. Results are comparable to with emergent search simulations with thousands of peers previously reported in Section 7.2 [Cam04d]. Both Figures 7.28 and 7.29 results suggest that success in emergent discovery services is maintained across a variety of P2P characteristics.

### *7.3.4 BPD Experimental Results Discussion*

The BPD implementation results successfully demonstrate that the BPD delivers a platform that can provide P2P DFS services. The BPD uniqueness comes as a consequence of instantiating the Emergent Thinker paradigm and using CAS models for essential DFS services. Hundreds of peers have been tested concurrently using the BPD to provide DFS services. Results that were obtained in the experiments correlate with the simulation results for thousands of peers generated previously. This suggests that the BPD is adaptable to P2P system changes and scales to a large number of peers, while maintaining success in the services provided.

# *Chapter 8*

# 8. Conclusion

*"The hardest thing to understand is why we can understand anything at all"*

*- Physicist Albert Einstein*

Systems research is an area where complex adaptive systems can provide innovative schemes and models. These schemes and models can provide emergent solutions (i.e. swarm properties) for a variety of design and implementation challenges in large distributed systems. Some already exist but others have not yet been conceived before.

In this thesis, a generic CAS emergent computing model was proposed. This model was later extended to become the *Emergent Thinker* paradigm that provides a different approach to distributed systems computing with the use of swarm properties. The CAS propagation model was proposed as an experimental means to understand and layout the fundamentals of the CAS emergent computing model.

The Emergent Thinker paradigm offers an alternative computational model for complex information system environments with the use of Complex Adaptive Systems models and the swarm properties that emerge out of it.

The Biologically Inspired Peer-to-Peer Distributed File System (BPD) implemented the Emergent Thinker paradigm. The BPD provides file system services in a complex information system environment that is dynamic, self-organized, *ad-hoc* and decentralized. Some of the characteristics that make the BPD distinctive include an architecture design and implementation that merges CAS models with P2P computing; CAS models for distributed services (storage, retrieval, search, and replication); and the use of commodity peers (by using simulated devices) for storage and retrieval in a complex dynamic environment.

The BPD design identified allocation and discovery services as essential DFS building blocks and stated the hypothesis that they can be provided by emergent computations. Consequently, a large part of the thesis research focused on CAS as a model to solve the problem of resource allocation and data discovery in a P2P distributed system. Typical work on resource allocation and discovery for distributed systems has followed traditional centralized or distributed, pre-designed techniques, which are usually not flexible or resilient to failures. CAS is inherently flexible and resilient, so the approach proves to be high adaptable to changes and resilient to agent or system failures. Consequently, CAS models should be suitable for distributed environments where there exists a continuously changing environment with no distributed deterministic algorithm capable of coping with the inherent dynamism.

The thesis uses CAS ideas to produce emergent global results from the activities of simple agents (i.e. squirrels). The squirrel's hoarding nature has been studied, exploited, and simulated on a P2P DFS. The work is based on a new CAS metaphor, that of squirrels rather than previously reported (e.g. ant metaphor) and it produces a truly novel result. Experimental results obtained corroborate the hypothesis that a well-balanced resource allocation and reliable resource search in a distributed system with variant characteristics can be achieved.

The proposed *Squirrel-based* CAS software architecture is sufficiently flexible that new behaviors can readily be defined and implemented in the model. These can then be incorporated in the simulation tool and tested with only minor changes to analyze their results. If successful, then they may give new insights to understand CAS systems.

CAS has proven to be a valuable model to be applied to the solution of resource allocation and search problems in distributed systems. Its application in the P2P computing model has produced interesting CAS algorithms results. These CAS algorithms are leveraged for use in the BPD.

The BPD CAS algorithms, although simple, provide emergent global outcomes achieved by the synergy of the quasi-independent activity of its members. Because each distributed system member is independent of each other, self-adaptation and self-organization is achieved across the system. These are desirable characteristics to have in *ad-hoc* computing environments to provide continuous global DFS services.

## 8.1 Future Research

The Emergent Thinker paradigm is applicable to application domains where emergent computations are used to resolve specific, domain-oriented problems. Some examples of application domains include artificial intelligence, cognitive sciences, computational economics, optimization, biology, psychology, neurosciences, and engineering.

Current BPD research assumes logically isolated CAS domains. The CAS domains are logically isolated from the perspective that they compute independently of each other; eventhough they might be running on the same physical hardware. Future research could analyze and obtain the fundamental bases when the CAS domains are interrelated at different levels and forms (e.g. hierarchical, intersected, dependant, *etc.*). Then use this know-how to produce new alternatives and solutions where existing schemes are not good in application domains as those listed in the last paragraph.

My BPD experiments focused on the validation of the Emergent Thinker paradigm. Future work could research new BPD services and performance metrics. For instance replication services that make use of emergent allocation services. Another idea to explore could be the implementation of bag hierarchies and associated semantic search that could make use of the emergent discovery services. This could provide new approaches for semantic searches and data management.

Furthermore, there are several paths where related research to the BPD can be explored. One extensive area is the research in alternative biological behaviors

where emergence exists. Experimental simulation of these behaviors can potentially lead to useful new ways to compute. These new computing approaches could provide solutions to systems research problems that are difficult to tackle by other means due to its complexity. For example, alternative natural mechanisms can be investigated for the BPD discovery and allocation services that could bring additional benefits in terms of scalability, reliability and performance (e.g. finding an acorn faster in the environment, obtaining rapid result convergence, enhancing resource allocation, *etc.*).

Another area of research is the application of the Emergent Thinker model to other distributed operating system functions and/or services. This thesis applies the Emergent Thinker to the design and implementation of a distributed file system. Many other applications and services could also be investigated. Areas to look at might include process execution, location services (e.g. name services), replication services, resource allocation (e.g. memory, caches), and distributed security services among others.

My research assumed that the connectivity among CAS members took place at the application layer. The research ignored the physical networking among systems (e.g. peers). The physical interconnections among peers provide the networking environment and are used by the CAS members (e.g. squirrels) to execute their behaviours. The reason to ignore the physical networking in this research has to do with the BPD architecture stack (Figure 4.3). The decision to find an appropriate system (i.e. peer) is currently left to the TNC layer that will optimize the selection of the next system. The TNC layer controls and has information about the physical connectivity among systems (e.g. peers). Thus, the CAS layer can rely on the TNC layer and assume that the TNC layer will provide the best system (e.g. peer) that it is most favorable in bandwidth, traffic load, distance and other physical networking factors. Future work might

improve the TNC layer that optimizes system selection for the CAS members when they need to move among them to execute CAS member behaviours. Then, performance metrics can be taken to analyze what impact that TNC layer has on the application layers of the BPD architecture.

Another approach to extend research that considers networking factors among systems is to make CAS member behaviours more intelligent. The extra intelligence could give CAS members additional information to select the appropriate system (i.e. peer) in terms of traffic load, bandwidth, processing capacity, *etc*. However, this approach must be used carefully because the CAS premise of simplicity can be broken and return to old approaches of large and bulky system designs. The extra intelligence added to CAS member behaviours could have performance improvements though. This future work might alter CAS member behaviours to consider networking metrics (e.g. bandwidth, distance, *etc.*) besides behavioral metrics when deciding to move to another system. Then, performance metrics might be taken. These performance metrics could analyze the impact that changed behaviours with networking metrics have on the emergent outcomes of CAS.

Resource allocation services are not unique to P2P DFSs. Distributed resource allocation occurs in multiple different domains (communications, manufacturing, job scheduling, computer resources, *etc.*) so that ideas and results presented in this thesis for allocation services are equally applicable. This is because the principles existing in the distributed allocation of resources, whatever they are, are in essence quite similar.

Finally, for the theorist, developing CAS mathematical modeling can be an exciting area, although uncertain due to the lack of formal tools. The effort to carry out CAS mathematical modeling research could be extensive in time and

resources though. But developing CAS theory could bring benefits that have implications that are applicable to many different sciences (e.g. biology, cognitive science, economics, *etc.*) by helping to explain the many processes that occur in lots of natural and artificial distributed systems mechanisms.

# Bibliography

[Abe02] K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidth, "Improving Data Access in P2P Systems", *IEEE Internet Computing*, January – February 2002, pp 58-67.

[And00] C. Anderson, and N. Franks, "Teams in Animal Societies", *Behavioral Ecology*, Volume 12, Number 5, September 2000, pp 534-540.

[And95] T. Anderson, M. Dahlin, J. Neefe, *et al*, "Serverless Network File Systems", *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain, CO, USA, December 1995, pp 109-126.

[Bab02] O. Babaoglu, H. Meling, and A. Montresor, "Anthill: A Framework for the Development of Agent Based Peer-To-Peer Systems", *University of Bologna Technical Report UBLCS-2001-09*, Bologna, Italy, September 2002.

[Bac86] M.J. Bach, "The Design of the Unix Operating System", *Prentice Hall Editors*, Englewood Cliffs, New Jersey, USA, 1986.

[Bak02] A. Bakker, I. Kuz, M. vanSteen, A. Tanenbaum, and P. Verkaik, "Global Distribution of Free Software (and other things)", *Proceedings of the 3rd International SANE Conference - 2002 MECC*, Maastricht, The Netherlands, May 2002.

[Bal02] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking Up Data in P2P Systems", *Communications of the ACM*, Volume 46, Number 2, February 2003, pp 43-48.

[Ban03] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications", *IEEE Infocom 2003*, San Francisco, CA, USA, April 2003.

[Bar97] Y. Bar-Yam, "Dynamics of Complex Systems", *Addison-Wesley Press*, Reading Massachusetts USA, 1997, pp 1-15.

[Bir93] A. Birrell, A. Hisgen, C. Jerian, T. Mann, and G. Swart, "The Echo Distributed File System", *Technical Report #111, Digital Equipment Corporation, Systems Research Center*, Palo Alto CA USA, September 1993, pp 1-22.

[Bol00] W. Bolosky, J. Douceur, D. Ely, and M. Theimer, "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs", *Proceedings of the ACM SIGMETRICS 2000 Technical Conference*, Santa Clara, CA, USA, June 2000, pp 34-43.

[Bon00a] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for Optimization from Social Insect Behaviour", *Nature 406*, Macmillan Publishers Limited 2000, pp 39-42.

[Bon00b] E. Bonabeau, and G. Theraulaz, "Swarm Smarts", *Scientific American*, Volume 282, Number 3, March 2000, pp 54-61.

[Bon01] E. Bonabeau, and C. Meyer, "Swarm Intelligence, a New Way to Think About Business", *Harvard Business Review*, May 2001, pp 106-114.

[Bon02] E. Bonsma, and C. Hoile, "A Distributed Implementation of the SWAN Peer-To-Peer Look-Up System Using Mobile Agents", *Proceedings of the International Workshop on Agents and P2P Computing (AP2PC 2002)*, Bologna, Italy, July 2002, pp 100-111.

[Bor89] U. Borghoff, and K. Nast-Kolb, "Distributed Systems: A Comprehensive Survey", *Technical Report No. TUM-I8909*, Technical University Munchen, Munich, Germany, November 1989, pp 1-85.

[Bou03] C. Bourjot, V. Chevrier, and V. Thomas, "A New Swarm Mechanism Based on Social Spiders Colonies: From Web Weaving to Region Detection", *Web Intelligence and Agent Systems: An International Journal*, Volume 1, Number 1, 2003, pp 47-64.

[Bro02] D. Brodsky, A. Brodsky, J. Pomkoski, S. Gong, M. Feeley, and N. Hutchinson, "Using File-Grain Connectivity to Implement a Peer-to-Peer File System", *Proceedings of the IEEE International Workshop on Reliable Peer-to-Peer Distributed Systems*, Osaka, Japan, October 2002, pp 318-323.

[Bur00] R. Burns, "Data Management in a Distributed File System For Storage Area Networks", *University of California at Santa Cruz*, PhD Thesis, Chapter 1, March 2000, pp 1-16.

[Cab01] L. Cabrera, M. Jones, and M. Theimer, "Herald: Achieving a Global Event Notification Service", *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau Germany, May 2001, pp 87-94.

[Cam03a] S. Camorlinga, and K. Barker, "Multiagent Systems Storage Resource Allocation in a Peer-to-Peer Distributed File System", *University of*

*Manitoba Computer Science Department TR 03/01, University of Calgary Computer Science Department TR 2003-716-19*, Winnipeg - Calgary, Canada, January 2003, pp 1-22.

[Cam03b] S. Camorlinga, and K. Barker, "Scalability and Reliability of Complex Adaptive System Based Algorithms in a Peer-to-Peer System", *Proceedings of the 5th International Workshop on Computer Science and Information Technologies ( CSIT'2003)*, Ufa Russia, September 2003.

[Cam04a] S. Camorlinga, K. Barker, and J. Anderson, "Multiagent Systems for Resource Allocation in Peer-to-Peer Systems", *ACM International Conference Proceedings of the Winter International Symposium on Information and Communication Technologies WISICT 2004*, Cancun Mexico, January 2004, pp 42-47.

[Cam04b] S. Camorlinga, and K. Barker, "Design of a Biologically Inspired Peer-to-Peer Distributed File System", *University of Calgary Technical Report 2004-737-02*, Department of Computer Science, Calgary, AB, Canada, February 2004.

[Cam04c] S. Camorlinga, and K. Barker, "The Emergent Thinker", *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems*, Bertinoro, Italy, May 31-June 2, 2004, pp 69-72.

[Cam04d] S. Camorlinga, and K. Barker, "Emergent Search in Large Distributed Systems", *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, Workshop on Games and Emergent Behaviors in Distributed Computing Environments*, Birmingham, UK, September 2004.

[Cam05a] S. Camorlinga, and K. Barker, "A Complex Adaptive System Based on Squirrels Behaviors for Distributed Resource Allocation", *Accepted for publication as original research paper in the Journal of Web Intelligence and Agent Systems*, IOS Press Publishers, 2005 (to appear).

[Cam05b] S. Camorlinga, and K. Barker, "A Biologically Inspired Distributed File System: An Emergent Thinker Instantiation", *Self-Star Properties in Complex Information Systems Hot Topics Volume of the Springer Lecture Notes in Computer Science Series*, O. Babaoglu, C. Fetzer, M. Jelasity *et al.* Editors, Springer-Verlag, Volume 3460, June 2005.

[Can04] G. Canright, "Self-* Information Systems: Why Not?", *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems*, Bertinoro, Italy, May 31-June 2, 2004, pp 101-104.

[Cas01] M. Casassa, L. Tomasi, and R. Montanari, "An Adaptive System Responsive to Trust Assessment based on Peer-to-Peer Evidence Replication and Storage", *Technical Report TR HPL-2001-133*, Trusted E-Services Laboratory, Hewlett-Packard Bristol, Bristol, UK, June 2001, pp 1-21.

[Cha97] A. Chavez, A. Moukas, and P. Maes, "Challenger: A Multi-Agent System for Distributed Resource Allocation", *Proceedings of the 1st International Conference on Autonomous Agents*, Marina del Rey, CA, USA, February 1997, pp 323-331.

[Cim02] L. Ciminiera, A. Sanna, and C. Zunino, "Survey on Grid and Peer-to-Peer Network Technologies", *Technical Report TR#EGSO-DE01_01-D02-021001*, European Grid of Solar Observations, October 2002, pp 1-58.

[Cla00] I. Clarke, O. Sandberg, B. Willey, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System", *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability,* Berkeley, CA, USA, July 2000, pp 311-320.

[Coh04] J. Cohen, "Bioinformatics – An Introduction for Computer Scientists", *ACM Computing Surveys*, Volume 36, Number 2, June 2004, pp 122-158.

[Con00] Connecticut Bureau of Natural Resources, Wildlife Division, "Wildlife in Connecticut: Gray Squirrel", *Connecticut Department of Environmental Protection*, 2000. See docs at http://dep.state.ct.us/burnatr/wildlife/factshts/gsqrl.htm

[Dab01] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS", *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, Banff, Canada, October 2001, pp 202-215.

[Dea00] P. Deadman, E. Schlager, and R. Gimblett, "Simulating Common Pool Resource Management Experiments with Adaptive Agents Employing Alternate Communication Routines", *Journal of Artificial Societies and Social Simulations*, Volume 3, Number 2, March 2000.

[Dec01] E.H. Decker, "Self-Organizing Systems: A Tutorial in Complexity", *Vivek – A Quarterly in Artificial Intelligence Journal*, Volume 13, Number 1, 2001, pp 1-13.

[Deu04] A. Deutsch, "Self-Organization in Interacting Cell Networks: From Microscopic Rules to Emergent Behavior", *Proceedings of the International*

*Workshop on Self-* Properties in Complex Information Systems*, Bertinoro, Italy, June 2004, pp 65-68.

[Dic97] G. DiCaro, and M. Dorigo, "Antnet: A Mobile Agents Approach to Adaptive Routing", Technical Report IRIDIA/97-12, Universite Libre de Bruxelles, Bruxelles, Belgium, 1997.

[Dog98] A. Dogac, C. Dengi, and T. Oszu, "Distributed Object Computing Platforms", *Communications of the ACM*, Volume 41, Number 9, September 1998, pp 95-103.

[Dor96] M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: Optimization by a Colony of Cooperating Agents", *IEEE Transactions on Systems, Man and Cybernetics*, Volume 26, Number 1, 1996, pp 29-41.

[Dov03] D. Doval, and D. O'Mahony, "Overlay Networks: A Scalable Alternative for P2P", *IEEE Internet Computing*, Volume 7, Number 4, July – August 2003, pp 79-82.

[Dru01] P. Druschel, and A. Rowstron, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems", *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001, pp 329-350.

[Dub02] Z. Dubitzky, I. Gold, E. Henis, J. Satran, and D. Sheinwald, "DSF – Data Sharing Facility", *IBM Research Laboratories Technical Report*, Haifa, Israel, 2002, pp 1-20.

[Fra04] S. Frangou, X. Chitins, and S. Williams, "Mapping IQ and Gray Matter Density in Healthy Young People", *NeuroImage*, Volume 23, 2004, pp 800-805.

[Fu00] K. Fu, F. Kaashoek, and D. Mazieres, "Fast and Secure Distributed Read-Only File System", *Proceedings of the 4th USENIX Symposium on Operating Systems design and Implementation (OSDI 2000)*, San Diego, CA, USA, October 2000, pp 181-196.

[Gee02] D. Geels, "Data Replication in OceanStore", *Technical Report UCB/CSD-02-1217*, Computer Science Division EECS, University of California, Berkeley California, November 2002, pp 1-19.

[Gel02] B. Gelfand, A. Esfahanian, and M. Mutka, "An Agent-Based Approach to Enforcing Fairness in Peer-to-Peer Distributed File Systems", *Proceedings of the 9th International Conference on Parallel and Distributed Systems ICPADS'02*, Taiwan, ROC, December 2002, pp 157-162.

[Ger04] C. Gershenson, and F. Heylighen, "How Can We Think the Complex", *Managing the Complex Volume One: Philosophy, Theory and Application*, Richardson, Kurt (ed.), 2004.

[Ghe03] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System", *Proceedings of the 19th ACM Symposium on Operating System Principles*, Bolton Landing, NY, USA, October 2003.

[Gib98] M.A. Gibney, N.R. Jennings, "Dynamic Resource Allocation by Market-Based Routing in Telecommunications Networks", *Proceedings of the 2nd International Workshop on Multiagent Systems and Telecommunications (IATA-98)*, Paris, France, July 1998, pp 102-117.

[Gif91] D. Gifford, P. Jouvelot, M. Sheldon, and J. O'Toole, "Semantic File Systems", *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, Pacific Grove, CA, USA, October 1991, pp 16-25.

[Gnu01] Gnutella. http://www.gnutella.com. June 2001.

[Gri99] A. Grimshaw, A. Ferrari, F. Knabe, and M.Humphrey, "Wide-Area Computing: Resource Sharing on a Large Scale", *IEEE Computer*, Volume 32, Number 5, May 1999, pp 29-37.

[Gro99] B. Gronvall, A. Westerlund, and S. Pink, "The Design of a Multicast-based Distributed File System", *Third Symposium on Operating Systems Design and Implementation*, New Orleans, Louisiana, USA, February 1999, pp 251-264.

[Han96] D. Hansen, and D. Adams, "A Database Approach to Data Archive Management", *Proceedings of the 1st IEEE Metadata Conference*, Silver Spring, Maryland USA, April 1996.

[Har02] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica, "Complex Queries in DHT-based Peer-to-Peer Networks", *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, MIT Faculty Club, Cambridge MA USA, March 2002.

[Has00] W. Hasselbring, "Information System Integration", *Communications of the ACM*, Volume 43, Number 6, June 2000, pp 33-38.

[Her05] K. Herrmann, G. Muhl, and K. Geihs, "Self-Management: The Solution to Complexity or Just another Problem?" *IEEE Distributed Systems Online*, Vol. 6, No. 1, 2005.

[Hey03] F. Heylighen, "The Science of Self-Organization and Adaptivity", *The Encyclopedia of Life Support Systems EOLSS*, Eolss Publishers, Oxford, UK, 2003.

[Hou82] Houghton – Mifflin Company, "The Houghton Mifflin Canadian Dictionary of the English Language", *Houghton Mifflin Canada Limited*, Markham, Ontario, Canada, 1982 Edition, 1982, pp 427.

[Hun93] L. Hunter, "Molecular Biology for Computer Scientists", *Artificial Intelligence and Molecular Biology Chapter 1*, L. Hunter Editor, AAAI Library, 1993, pp 1-45.

[Jef98] K. Jeffrey, "Metadata: An Overview and Some Issues", *Proceedings of the 11th ERCIM Database Research Group Workshop Metadata for Web Databases*, Sankt Augustin, Germany, May 1998.

[Jel04a] M. Jelasity, W. Kowalczyk, and M. van Steen, "An Approach to Massively Distributed Aggregate Computing on Peer-to-Peer Networks", *Proceedings 12th Euromicro Conference on Parallel, Distributed and Network based Processing*, La Coruña, Spain, February 2004.

[Jel04b] M. Jelasity, A. Montresor, and O. Babaoglu, "Grassroots Self-Management: A Modular Approach", *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems*, Bertinoro, Italy, June 2004, pp 85-88.

[Kav99] K. Kavi, J. Browne, and A. Tripathi, "Computer Systems Research: The Pressure is On", *IEEE Computer*, January 1999, pp 30-39.

[Kel02] P. Keleher, B. Bhattacharjee, and B. Silaghi, "Are Virtualized Overlay Networks Too Much of a Good Thing?", *1st International Workshop on P2P Systems (IPTPS 2002)*, MIT Labs, Cambridge, MA, USA, March 2002, pp 225-231.

[Kle02] S. Kleijkers, F. Wiesman, and N. Roos, "Mobile Multi-Agent System for Distributed Computing", *Proceedings of the First International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002)*, Bologna, Italy, July 2002, pp 145-156.

[Kub00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czervwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage", *Proceedings of the ACM ASPLOS'2000*, Cambridge, MA, USA, November 2000, pp 190-201.

[Kur89] J.F. Kurose, and R. Simha, "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems", *IEEE Transactions on Computers*, Volume 38, Number 5, May 1989, pp 705-717.

[Kuw96] K. Kuwabara, T. Ishida, Y. Nishibe, and T. Suda, "An Equilibratory Market Based Approach for Distributed Resource Allocation and Its Applications to Communication Network Control", *in Scott H. Clearwater Ed., Market Based Control: A Paradigm for Distributed Resource Allocation*, World Scientific, Singapore, 1996, pp 53-73.

[Kuz02] I. Kuz, M. vanSteen, and H. Sips, "The Globe Infrastructure Directory Service", *Computer Communications*, Volume 25, Number 1, January 2002, pp 835-845.

[Lad04] R. Laddaga, and P. Robertson, "Self Adaptive Software", *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems*, Bertinoro, Italy, June 2004, pp 137-140.

[Lem04] R. de Lemos, "Self-* and Predictability: Are these conflicting system capabilities?", *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems*, Bertinoro, Italy, May 31-June 2, 2004, pp 105-108.

[Lev90] E. Levy, and A. Silberschatz, "Distributed File Systems: Concepts and Examples", *ACM Computing Surveys*, Volume 22, Number 4, December 1990, pp 321-374.

[Lib02] D Liben-Nowell, H. Balakrishnan, and D. Karger, "Observations on the Dynamic Evolution of Peer-to-Peer Networks", *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, MIT Faculty Club, Cambridge MA USA, March 2002.

[Lie02] R. Lienhart, M. Holliman, Y. Chen, *et al*, "Improving Media Services on P2P Networks", *IEEE Internet Computing*, January – February 2002, pp 73-77.

[Liu00] W. Liu, M. Wu, X. Ou *et al*, "Design of an I/O Balancing File System on Web Server Clusters", *Proceedings of the 2000 International Workshop on Parallel Processing ICPP'00*, Toronto, On, Canada, August 2000, pp 119-125.

[Lv02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", Proceedings of the 16th International Conference on Supercomputing, New York, NY USA, June 2002, pp 84-95.

[Man99] F. Manola, "Technologies for a Web Object Model", *IEEE Internet Computing*, January – February 1999, pp 38-47.

[Mic03] Microsoft Corporation, "Microsoft .Net", *Microsoft .Net Home*, see docs at http://www.microsoft.com/net, 2003.

[Mil99] D. Milojicic, D. Black, B. Bolosky, F. Kaashoek, J. Liedtke *et al*, "Operating Systems – Now and in the Future", *IEEE Concurrency*, January-March 1999, pp 12-21.

[Mil02] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne *et al*, "Peer-to-Peer Computing", *Hewlett-Packard Technical Report HPL-2002-57*, HP Laboratories Palo Alto, Palo Alto CA, March 2002, pp 1-51.

[Mon01] A. Montresor, "Anthill: A Framework for the Design and Analysis of Peer-To-Peer Systems", *Proceedings of the 4th European Research Seminar on Advances in Distributed Systems (ERSADS 2001)*, Bertinoro, Italy, May 2001.

[Mon02a] A. Montresor, H. Meling, and O. Babaoglu, "Toward Self-Organizing, Self-Repairing and Resilient Large-Scale Distributed Systems", *University of Bologna Technical Report UBLCS-2002-10*, Bologna, Italy, September 2002, pp 1-6.

[Mon02b] A. Montresor, H. Meling, and O. Babaoglu, "Messor: Load-Balancing through a Swarm of Autonomous Agent", *Proceedings of the 1st International Workshop on Agents and P2P Computing*, Bologna, Italy, July 2002, pp 125-137.

[Mor02] T. Moreton, I. Pratt, and T. Harris, "Storage, Mutability and Naming in Pasta", *Proceedings of the International Workshop on Peer-to-Peer Computing at Networking 2002*, Pisa, Italy, May 2002, pp 215-219.

[Mut02] A. Muthitacharoen, R. Morris, T.M. Gil, and B. Chen, "Ivy: A Read/Write Peer-to-Peer File System", *Proceedings of the 5th Symposium on*

*Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, USA, December 2002, pp 31-44.

[Nap01] Napster. http://www.napster.com. June 2001.

[Nic01] J. Nicholls, A. Martin, B. Wallace, and P. Fuchs, "From Neuron to Brain, Fourth Edition", *Sinauer Associates Inc Publishers*, Sunderland, MA USA, 2001.

[Ope05] The Open Group, "The Single UNIX Specification, Version 3", *Available at http://www.unix.org/version3*, August 2005.

[Ous85] J. Ousterhout, H. Da Costa, D. Harrison, *et al*, "A Trace-Driven Analysis of the Unix 4.2 BSD File System", *Proceedings of the 10th Symposium on Operating System Principles (SOSP'85)*, Orcas Island, WA, USA, December 1985, pp 15-24.

[Ous88] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch, "The Sprite Network Operating System", *IEEE Computer*, Volume 21, Number 2, February 1988, pp 23-36.

[Pap93] M. Papazoglou, "On the Duality of Distributed Database and Distributed AI Systems", *Proceedings of the 2nd International Conference on Information and Knowledge Management*, Washington DC USA, November 1993, pp 1-10.

[Par01] M. Parameswaran, A. Susarla, and A. Whinston, "P2P Networking: An Information-Sharing Alternative", *IEEE Computer*, July 2001, pp 31-38.

[Pen02] L. Penserini, L. Liu, J. Mylopoulos, *et al.*, "Modeling and Evaluating Cooperation Strategies in P2P Systems", *Proceedings of the International*

*Workshop on Agents and P2P Computing (AP2PC 2002)*, Bologna, Italy, July 2002, pp 87-99.

[Pur83] W. Purves, and G. Orians, "Life, the Science of Biology", *Sinauer Associates Inc Publishers*, Sunderland Massachusetts USA, 1983.

[Ran02] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities", *Proceedings of the Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop*, Berlin, Germany, May 2002, pp 376-381.

[Rat01] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, "A Scalable Content Addressable Network", *Proceedings of the ACM SIGCOMM 2001 Technical Conference,* San Diego, CA, USA, August 2001, pp 161-172.

[Rat02] S. Ratnasamy, S. Shenker, and I. Stoica, "Routing Algorithms for DHTs: Some Open Questions", *1st International Workshop on P2P Systems (IPTPS 2002)*, MIT Labs, Cambridge, MA, USA, March 2002, pp 45-52.

[Res94] M. Resnick, "Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds", *MIT Press*, Cambridge Massachusetts USA, 1994.

[Rhe01] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon and J. Kubiatowicz, "Maintenance-Free Global Data Storage", *IEEE Internet Computing*, Volume 5, Number 5, September-October 2001, pp 40-49.

[Ric02] B. Richard, D. Mac Nioclais, and D. Chalon, "Clique: A Transparent, Peer-to-Peer Collaborative File Sharing System", *Technical Report HPL-*

*2002-307*, Hewlett-Packard Laboratories Grenoble, France, October 28th, 2002, pp 1-10.

[Rod03] O. Rodeh, and A. Teperman, "zFS – A Scalable Distributed File System using Object Disks", *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, CA, USA, April 2003, pp 207-218.

[Row01] A. Rowstron, and P. Druschel, "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility", *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01),* Banff, Canada, October 2001, pp 188-201.

[Sai02] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam, "Taming Aggressive Replication in the Pangaea Wide-Area File System", *Proceedings of the 5th Symposium on Operating Systems Design and Implementation OSDI'02*, Boston, MA, USA, December 2002.

[San85] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "The Sun Network Filesystem: Design, Implementation and Experience", *Proceeding of the 1985 Summer USENIX Conference*, Portland, OR, USA, June 1985, pp 119-130.

[Sar02a] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems", *Proceedings of the Multimedia Computing and Networking Conference (MMCN'02)*, San Jose CA USA, January 2002.

[Sar02b] S. Saroiu, P. Gummadi, and S. Gribble, "Exploring the Design Space of Distributed and Peer-to-Peer Systems: Comparing the Web, TRIAD, and

Chord/CFS", *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, MIT Faculty Club, Cambridge MA USA, March 2002.

[Sar04] N. Sarshar, P. Boykin, and V. Roychowdhury, "Percolation Search in Power Law Networks: Making Unstructured Peer-to-Peer Networks Scalable", *Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing*, Zurich, Switzerland, August 2004.

[Sat85] M. Satyanarayana, J.H. Howard, D. Nichols, R. Sidebotham, A. Spector, and M. West, "The ITC Distributed File System: Principles and Design", *Proceedings of the 10th ACM Symposium on Operating System Principles*, Orcas Islands, WA, USA, December 1985, pp 35-50.

[Sat90] M. Satyanarayana, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment", *IEEE Transactions on Computers,* Volume 39, Number 4, April 1990, pp 447-459.

[Sat93] M. Satyanarayana, "Distributed File Systems", *Distributed Systems 2nd Edition*, Sape Mullender Editor, ACM Press Addison-Wesley Publishing Company, New York NY USA, 1993.

[Sch84] M. Schroeder, A. Birrel, and R. Needham, "Experience with Grapevine: The Growth of a Distributed System", *ACM Transactions on Computer Systems*, Volume 2, Number 1, February 1984, pp 3-23.

[Shu01] A. Shulman, and E. Nelson, "Foraging Behavior of Squirrels as a Function of Season Progression From Early Fall Towards Winter", *Sewanee Journal of Biological Research*, Volume 2, University of the South, Sewanee, Tennessee, USA, 2001.

[Sim96] H.A. Simon, "The Sciences of the Artificial, Third Edition", *The MIT Press*, Cambridge, Massachusetts, USA, 1996.

[Sol02] C. Solnon, "Ants Can Solve Constraint Satisfaction Problems", *IEEE Transactions on Evolutionary Computation*, Volume 6, Number 4, August 2002, pp 347-357.

[Ste02] C. Stein, M. Tucker, and M. Seltzer, "Building a Reliable Mutable File System on Peer-to-Peer Storage", *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, Osaka University, Suita, Japan, October 2002, pp 324-329.

[Sto01] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, San Diego, CA, USA, August 2001, pp 149-160.

[Sun03] Sun Microsystems Inc., "The JXTA Project", *Sun Microsystems JXTA Project*, see docs at http://www.jxta.org, 2003.

[Tal03] D. Talia, and P. Trunfio P, "Toward a Synergy Between P2P and Grids", *IEEE Internet Computing*, Volume 7, Number 4, July – August 2003, pp 94-96.

[Tan02] A. Tanenbaum, and M. vanSteen, "Distributed Systems, Principles and Paradigms", *Prentice-Hall, Inc.*, Upper Saddle River, New Jersey, USA, 2002.

[Tar02] P. Tarasewich, and P.R. McMullen, "Swarm Intelligence: Power in Numbers", *Communications of the ACM*, Volume 45, Number 8, August 2002, pp 62-67.

[The97] C. Thekkath, T. Mann, and E. Lee, "Frangipani: A Scalable Distributed File System", *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, St. Malo, France, October 1997, pp 224-237.

[Tho03] J. Thomson, D. Adams, P. Cowley and K. Walker. "Metadata's Role in a Scientific Archive", *IEEE Computer*, Volume 36, Number 12, December 2003, pp 27-34.

[Tou03] D. Tsoumakos, and N. Roussopoulos, "A Comparison of Peer-to-Peer Search Methods", *Proceedings of the 6th International Workshop on Web and Databases*, San Diego, CA, USA, June 2003, pp 61-66.

[Vah02] A. Vahdat, J. Chase, R. Braynard, D. Kostic, P. Reynolds, and A. Rodriguez, "Self-Organizing Subsets: From Each According to His Abilities, To Each According to His Needs", *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, MIT Faculty Club, Cambridge MA USA, March 2002.

[Van99] M. vanSteen, P. Homburg, and A. Tanenbaum, "Globe: A Wide-Area Distributed System", *IEEE Concurrency*, January – March 1999, pp 70-78.

[Van04] M. vanSteen, S. Voulgaris, E. Ogston et al, "Self-Made Systems", Proceedings of the International Workshop on Self-* Properties in Complex Information Systems, Bertinoro, Italy, June 2004, pp 93-96.

[Var02] P. Vargas, L.N. de Castro, and F.J. Von Zuben, "Artificial Immune Systems as Complex Adaptive Systems", *Proceedings of the International Conference on Artificial Immune Systems*, Canterbury, UK, September 2002, pp 115-123.

[Vos03] M. Voss, "Complex Adaptive Systems Research", *Documents and links available at http://www.casresearch.com/*, 2004.

[Wal83] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel, "The LOCUS Distributed Operating System", *Proceedings of the 9th Symposium on Operating System Principles*, Bretton Woods, New Hampshire, USA, October 1983, pp 49-70.

[Wan98] R. Wang, and T. Anderson, "Experience with a Distributed File System Implementation", *Technical Report CSD-98-986*, University of California at Berkeley, Berkeley, CA, USA, January 1998, pp 1-12.

[War98] M. Ward, "There's an Ant in my Phone", *New Scientist*, September 1998, pp 32-35.

[Wat02] S. Waterhouse, D. Doolin, G. Kan, and Y. Faybishenko, "Distributed Search in P2P Networks", *IEEE Internet Computing*, January – February 2002, pp 68-72.

[Wat98] D. Watts, and S. Strogatz, "Collective Dynamics of Small-World Networks", *Nature,* Volume 393, June 1998, pp 440-442.

[Wea01] H. Weatherspoon, C. Wells, P. Eaton, B. Zhao, and J. Kubiatowicz, "Silverback: A Global-Scale Archival System", *University of California in Berkeley Technical Report UCB/CSD-01-1139*, Berkeley, CA, USA, March 2001, pp 1-17.

[Whi97] T. White, "Swarm Intelligence and Problem Solving in Telecommunications", *Canadian Artificial Intelligence Magazine*, Number 41, Spring 1997, pp 14-16.

[Whi01] B. White, M. Walker, M. Humphrey, and A. Grimshaw, "LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications", *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, Denver CO USA, November 2001, pp 59-68.

[Whi02] T. White, B. Pagurek, and D. Deugo, "Management of Mobile Agent Systems using Social Insect Metaphors", *Proceedings of the International Workshop on Self-Repairing and Self-Configurable Distributed Systems (RCDS 2002)*, Osaka, Japan, October 2002.

[Wyl00] J. Wylie, M. Bigrigg, J. Strunk, and *et al*, "Survivable Information Storage Systems", *IEEE Computer*, Volume 33, Number 8, August 2000, pp 61-68.

[Yan02] B. Yang, and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks", *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002, pp 5-14.

[Yia01] P. Yianilos, and S. Sobti, "The Evolving Field of Distributed Storage", *IEEE Internet Computing*, Volume 5, Number 5, October 2001, pp 35-39.

[Zam04] F. Zambonelli, and M. Mamei, "Spatial Computing: A Recipe for Self-Organization in Distributed Computing Scenarios", *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems (Self-Star 2004)*, Bertinoro Italy, May-Jun 2004, pp 37-40.

[Zha01] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing", *Technical Report*

*UCB/CSD-01-1141*, University of California at Berkeley, Computer Science Department, Berkeley, CA, USA, April 2001.