ENABLING DISTRIBUTED RADAR DATA RETRIEVAL AND PROCESSING IN DISTRIBUTED COLLABORATIVE ADAPTIVE SENSING ENVIRONMENTS

By

Diego Mauricio Arias Velasco

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

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

March, 2007

Approved by:

José Colom Ustariz, Ph.D Member, Graduate Committee	Date
Rafael Rodríguez Solís, Ph.D Member, Graduate Committee	Date
Wilson Rivera Gallego, Ph.D President, Graduate Committee	Date
Pedro Vásquez Urbano, Ph.D. Representative of Graduate Studies	Date
Isidoro Couvertier Reyes, Ph.D Chairperson of the Department	Date

Abstract of Dissertation Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

ENABLING DISTRIBUTED RADAR DATA RETRIEVAL AND PROCESSING IN DISTRIBUTED COLLABORATIVE ADAPTIVE SENSING ENVIRONMENTS

By

Diego Mauricio Arias Velasco

March 2007

Chair: Wilson Rivera Gallego
Major Department: Electrical and Computer Engineering

This thesis describes the integration of radar network and grid computing technologies to provide a set of grid services to manipulate and store data from different radars, to execute algorithms on different platforms in a transparent way to end users, and to provide secure access to storage systems and instruments. The solution approach considered is a grid-based system which includes a Grid Portal Interface, a distributed storage to radar data management, and Grid services implementing distributed algorithms to process data radar information.

A major requirement of this system is data availability and reliability. Consequently, have been implemented two redundancy schemes to perform the data management of the network: A simple Replication Algorithm and the Information Dispersal Algorithm (IDA). Experimental results show that IDA provides better reliability and less storage spending than the traditional replication algorithm. Furthermore, under the same conditions, the redundancy in the replication technique is three times or

more than IDA, when the reliability required is over 90%. IDA is normally used to perform security deployments where message encryption is required. Thus, this algorithm has been modified to enable large file management in order to satisfy the large amount of data generated by radars.

Resumen de Disertación Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

RECUPERACIÓN Y PROCESAMIENTO DE DATOS DE RADAR DISTRIBUIDOS EN AMBIENTES DE MONITOREO DISTRIBUIDOS COLABORATIVOS ADAPTIVOS

Por

Diego Mauricio Arias Velasco

Marzo 2007

Consejero: Wilson Rivera Gallego
Departamento: Ingeniería Eléctrica y Computadoras

Esta tesis describe la integración de tecnologías grid computing y redes de radares metereológicos, con el objetivo de proveer un conjunto de servicios grid para manipular y almacenar datos de diferentes radares, ejecutar algoritmos sobre múltiples plataformas de manera transparente a los usuarios y proveer acceso seguro a los sistemas de almacenamiento e instrumentos. La solución propuesta es un sistema basado en servicios grid el cual incluye un Portal Grid, un sistema distribuido de almacenamiento de datos y servicios grid, implementando algoritmos distribuidos para procesar la información de los datos generados por los radares.

El mayor requerimiento de este sistema es la disponibilidad y la confiabilidad de los datos obtenidos. Consecuentemente, se han implementado dos esquemas de redundancia para el manejo de los datos de la red: Un algoritmo de replicación simple y el algoritmo de información dispersa (IDA). Resultados experimentales muestran que IDA provee mejor confiabilidad que el algoritmo de replicación. Adicionalmente,

bajo las mismas condiciones, la redundancia en la técnica de replicación es tres veces o más que la de IDA, cuando los requerimientos de confiabilidad están por encima del 90%. IDA es normalmente usado en aplicaciones de seguridad donde se requiere encriptación de mensajes. De esta manera este algoritmo ha sido modificado para poder manipular los archivos de gran tamaño generados por los radares.

Copyright © 2007

by

Diego Mauricio Arias Velasco

To María...

My light, my guide, my teacher, my sun, my heaven, my world..... my mother!

I LOVE YOU

ACKNOWLEDGMENTS

I would like to show my profound gratitude to my parents, María y Libardo. Your example and your constant support were my major source of motivation during my university studies. I would like to especially thank my friends (my family here in Puerto Rico) Natalia, Arjuna e Iveth. Natalia, without your patience, understanding and unconditional support I would not have been able to achieve my goals. Arjuna and Iveth, not only did you both welcome me to your home, but you provided me with the spirit to continue forward. My most sincere thanks to my advisor, Professor Wilson Rivera, who gave me the opportunity of belonging to his work team. Thank you for your patience and your support, thank you for allowing me to work in something which I like, but, most of all, special thanks for believing in me and my work. Being part of the PDC Lab was a very important and satisfying experience for me. I would like to give thanks to all the members of this magnificent work team, Wilson, Fernando, Gustavo, Mariana, John and Kennie. I would also like to thank all the members of the CASA Project, especially to the students, Manuel Vega and Victor Marrero and to all the people, who, in one way or another, contributed and gave support to my work. Many thanks to my committee, Dr. José Colom and Dr. Rafael Rodríguez for believing in the contribution of my work to the Student Testbed.

This work was supported by the NSF Engineering Research Center for Adaptive and Collaborative Sensing of the Atmosphere (CASA).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Overview

The National Science Foundation Engineering Research Center for Collaborative Adaptive Sensing of the Atmosphere (CASA) is focused on developing Distributed Collaborative Adaptive Sensing (DCAS) [1] as a systems technology to improve our ability to monitor the earth's lower atmosphere. Current approaches to sampling the first three kilometers of atmosphere are physically limited in their ability to provide the required resolution and coverage. For example, radar technology is currently limited by the focus on long range sensing by single instruments. Requiring radar to view distances up to 240km, as in the case of NEXRAD [2], introduces the problem of the earth's curvature [3]. As the range increases away from the radar, the earth's surface curves away under the radar beam. This causes the volume of atmosphere being observed to be located at an increasing height above the earth's surface. The radar is unable to observe the atmosphere close to the earth's surface where people live.

DCAS aims to radically alter the radar paradigm. Rather than relying on single radar to provide long range (hundreds of kilometers) coverage, DCAS proposes to mosaic the output of lower power shorter range (tens of kilometers) radars. It must be acknowledged that reducing the range would require an increase in the number of radars to cover the same land area. By directly comparing areas, reducing the maximum required range from 240 km to 30 km would require approximately 64 short range radars to cover the area of the single long range radar.

The island of Puerto Rico presents a unique set of challenges that accentuate the limitation of the current large radar paradigm. With a land area of approximately 9000 km^2, Puerto Rico is home to an estimated 3.8 million people, predominately located in the coastal lowlands, surrounding a central mountain range peaking 1.3 km above sea level. The island is located in the Tropic of Cancer and receives a 30-year average of 63 inches of rainfall per year [4]. The amount and nature of this rainfall, combined with the island's particular orography makes Puerto Rico prone to flash flooding events, which require Quantitative Precipitation Estimation (QPE) for accurate radar based forecasting. QPE forecasts are produced by the NWS forecast office in San Juan using, in part, the island's single NEXRAD WSR-88D, TJUA.

TJUA is located in Cayey on the eastern end of the central mountain range. The radar is installed at an elevation of 851 m, and is responsible for Puerto Rico and the U.S. Virgin Islands. Due to NWS regulations, the lowest elevation angle, a NEXRAD will use for sampling a 0.5°. As range is increased away from the instrument, so is the minimum altitude that the NEXRAD will sample. The University of Puerto Rico, Mayagüez (UPRM) campus, is located west of TJUA at a distance of approximately 100 km. Over the course of this 100 km the NEXRAD beam rises 871 m into the atmosphere. When combined with the base elevation of TJUA, the resulting minimum altitude of atmosphere being sampled is 1.7 km above sea level.

Sampling the atmosphere nearly 2 km above sea level can only provide an estimate of the ground-truth rainfall. Precipitation may be blown by wind or may evaporate before falling to the ground, which results in unavoidable errors in rainfall estimation introduced by the location where sampling takes place. Correctly estimating ground-truth rainfall requires sampling the atmosphere close to the earth's surface making the mission of QPE ideally suited to the DCAS paradigm of remote sensing. Utilizing radars with a maximum range of no more than 30 kilometers would, assuming an elevation angle of 0.5°, result in a minimum altitude of 261

m at maximum range. In addition, 30 km radars could be located in the coastal lowlands on tall buildings' roof tops allowing the beam minimum altitude to remain below 300 m. A total of four 30 km radars, properly located, would be required to provide complete coverage of the island. Varying the range of radars within a sensing network would allow the network to adapt to the requirements of the terrain, allowing radar's beam altitude to be kept below 1 km, impacting QPE measurements and the corresponding flood forecasts. To verify this hypothesis a QPE test-bed is being constructed in Puerto Rico. The infrastructure and technologies derived from this QPE test-bed will be used to build the base for the Puerto Rico Full DCAS System Testbed.

1.2 Problem Statement

Recent years have shown the utility of using many commodity computers networked to form a larger system instead of a single, more expensive, larger system. Similarly, the act of networking many inexpensive radars to cover the same area as a single high power radar introduces new capabilities into the system, such as fault tolerance and adaptability of the network sensing strategy, which the larger systems are currently not capable of performing.

A parallel development in the technology landscape is grid computing [5], which involves coordination, storage and networking of resources across dynamic and geographically dispersed organizations in a transparent way for users. The Open Grid Services Architecture (OGSA) [6], based upon standard Internet protocol, is becoming a standard platform for grid services and application development.

The integration of grid computing and radar network technologies enables the complementary strengths of these technologies to be achieved in an integrated platform. However, it poses significant computing challenges, such as the need to comply with emerging APIs for grid and Web services, the coordination of communication,

and the requirement of a more data-centric infrastructure focused on distributed services.

This thesis addresses the technical problems of integrating a weather radar network and grid computing technologies, focusing on data management issues.

1.3 Solution Approach

To perform the integration of data from the DCAS weather nodes into grid architecture while preserving data integrity, a grid-service based system is considered in order to access, manipulate and store radar data. The grid based system includes a Grid Portal interface, a distributed storage system to radar data management, and Grid services implementing data processing algorithms.

1.4 Objectives

The overall goal of the proposed research is to integrate radar network and grid computing technologies to provide a set of grid services to manipulate and store data from different radars, execute algorithms on different platforms in a transparent way to end users, and provide secure access to storage systems and instruments. The rationale of this research is that existing grid technologies will permit the fulfillment of some of the main design goals for a DCAS system.

The specific objectives are:

1. Design and develop a grid-service based system to access and manipulate radar data.
2. Develop and implement redundancy algorithm to enhance data management.

1.5 Contributions

The main contributions of this project can be summarized as follows:

- The Implementation and evaluation of redundancy schemes to perform data management of radar network. This evaluation demonstrates that the Information Dispersal Algorithm (IDA) presents higher access reliability than typical replication algorithm with less storage resources usage. The IDA, widely used in cryptographic techniques to encode messages, was enhanced to operate over large type of files of any kind, such as image, video, text and raw data.
- A grid based infrastructure which is fully functional and entirely configurable according to the primary goals of DCAS systems. The developed system provides transparency, security, reliability and expandable capabilities to allow users interaction with data and instruments.

1.6 Thesis Structure

This thesis is organized as follows: Chapter 2 presents the literature review including grid computing concepts and Galois field theory. Operations on Galois fields are the base for the implementation of the IDA. Chapter 3 describes the redundancy schemes implemented and the algorithms involved, as well as how the IDA was improved in order to allow large data files with an acceptable processing time. Chapter 4 describes the grid implementation and radar integration, listing the requirements and resources. Chapter 5 provides results of the experimentation, as well as the methodology and metrics used to evaluate the replication schemes and finally a summary of conclusions and future work are presented in chapter 6.

CHAPTER 2
BACKGROUND AND LITERATURE REVIEW

This chapter provides the necessary background to understand the weather radar technology and the nature of the acquired data from such radars. Grid computing technologies used in the course of this research are also discussed, particularly those related to grid computing implementations and portals. This chapter also provides information related to Galois field operations, which are the basis of the redundancy algorithm implementations proposed in this thesis. Finally, some related works to this thesis are also highlighted here.

2.1 Weather Radars

Modern meteorology leans on a well-developed radar-based technology as an effective strategy to provide warning capability for hazardous weather situations. Weather radars have become a very popular tool in atmospheric phenomena detection, tracking and forecasting, such as flash flooding, hurricanes, tornados, and thunderstorms. Radar networks have been established throughout entire countries in order to cover a wide area. And, to further compound the situation, information provided by these radars is widespread through mass media, such as the internet, in such a way that weather conditions can be verified in near real time.

Doppler radar [1] is the preferred approach for detecting and predicting thunderstorms due to its capability to detect motion. Point-measurement systems, such as surface weather-sensing systems, cannot rapidly measure the three-dimensional structure of the storms [7]. Additionally, satellite system measurements are not accurate. Doppler radar emits a burst of energy (electromagnetic pulse) which is scattered in all directions if an object is stricken within its path and a small fraction of that scattered energy is directed back toward the radar (See Figure 2–1).

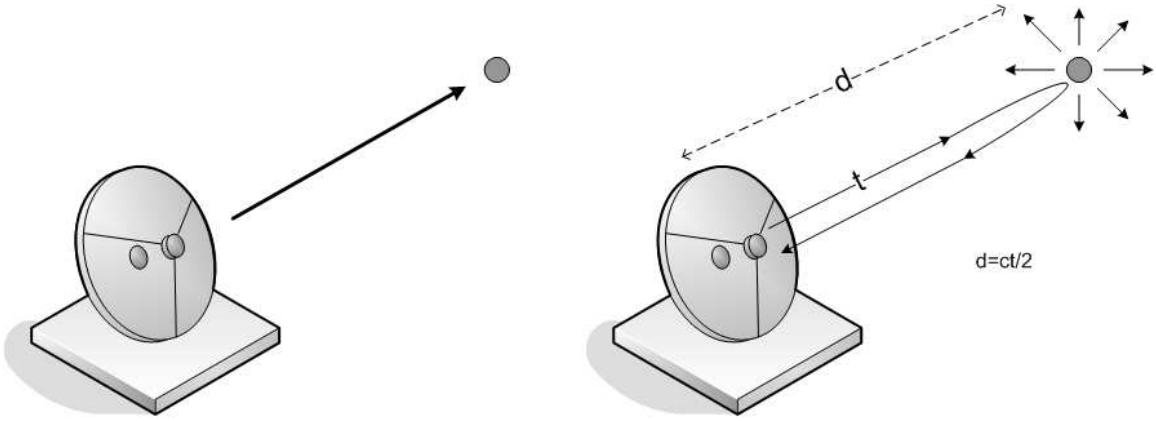

Figure 2–1: Basic Doppler radar operation

This reflected signal is then received by the radar during its listening period. Later, the computers will analyze the strength of the returned pulse, as well as the time it takes to travel to the object and return, and to also analyze the phase shift of the pulse. All this process which involves emitting a signal, listening for any returned signal, and emitting the next signal, occurs quite quickly up to around 1300 times each second.

[1] National Weather Service, "Doppler Radar", http://www.srh.noaa.gov/jetstream/remote/doppler.htm

2.1.1 Radar Measurements

Radars obtain weather information (precipitation and wind) based upon the distance from the target (range) and energy returned. The range is calculated when registering the elapsed time between emitted and received pulse. Since the pulse is an electromagnetic wave, it travels at the speed of light. The target's distance equation is:

$$d = \frac{ct}{2}$$

Weather radars register the strength of the signals returned from reflections. The display of these signals is called *reflectivity*. Reflectivity can be correlated to the intensity of the echo and return, the amount and the type of precipitation that is falling. It can be classified as follows:

- **Base Reflectivity** is measured in dBZ (decibels of Z, where Z represents the energy reflected back to the radar) and images which are available at several different elevation angles (tilts) of the antenna. If a base reflectivity image is from the lowest "tilt" angle (e.g. 0.5°), it means that the radar's antenna is tilted 0.5° above the horizon. The colors in the reflectivity image represent the reflectivity measured expressed in decibels (see Figure 2–2). These dBZ values equate to an approximate hourly rainfall rate as indicated in the Table 2–1.
- **Composite Reflectivity** is the display of maximum reflectivity from any elevation angle at every range from the radar. The Composite Reflectivity reveals important storm structure features and intensity trends of storms.

Relevant information also obtained from the radar, related to wind behavior is described as follows:

- **Base Velocity** is the radial velocity display of the overall wind field. The motion of the wind relative to the radar, along the path of the beam (the radial), is not the direction of the wind, but the portion of the motion of the wind that is moving either directly toward or away from the radar.

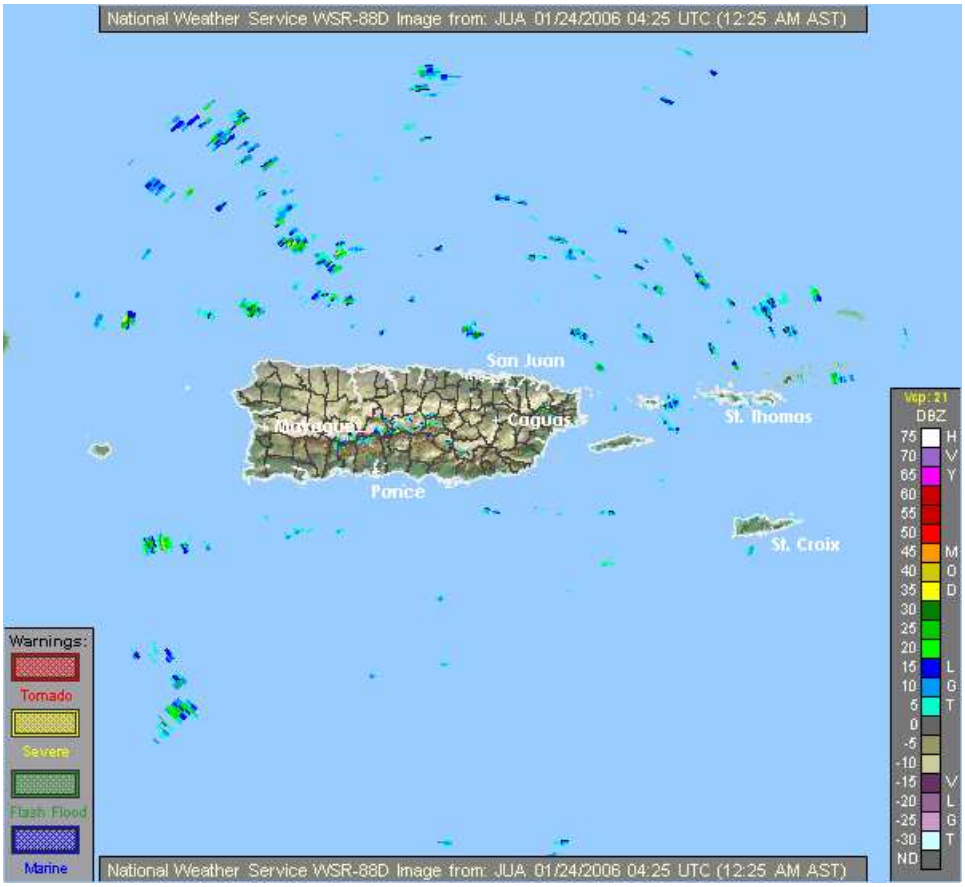

Figure 2–2: Base Reflectivity from the Doppler radar in San Juan, PR.

2.2 Weather Radar Data

Data from radars generally contains Data from radars generally contain information about reflectivity and wind velocity. This information is collected through radar scans. This is typically done with one axis scanning and the other axis sequencing through a number of fixed angles. Usually, two types of scans can be performed: PPI and RHI. In the PPI (plan position indicator), elevation is stepped through a sequence of fixed angles while the azimuth is scanned. The RHI (range height indicator) fixes the azimuth while the elevation angle is moving.

The information obtained from the radar data is stored in files which correspond to each scan procedure. The files are stored with a special name related to the radar, date and time. The file name begins with the name of the radar followed by

Table 2–1: Reflectivity and Rainfall Rate equivalencies

dBz	Rain Rate
65	16+
60	8.00
55	4.00
52	2.50
47	1.25
41	0.50
36	0.25
30	0.10
20	Trace
<20	No Rain

the year, month, day, hour, minute, and second at which the data file stored (e.g. SJU20060124_155702).

Parameters such as range, latitude, longitude, azimuth, elevation, radar and so on, are stored with the reflectivity information. NetCDF (network Common Data Form) is a common format used to store data from radars. NetCDF is an interface for array-oriented data access and a library that provides an implementation of the interface, supporting the creation, access, and sharing of scientific data. The following is an example of data Netcdf format:

```
dimensions:
        Azimuth = 367 ;
        Gate = 460 ;
variables:
        float Azimuth(Azimuth) ;
                Azimuth:Units = "Degrees" ;
        float BeamWidth(Azimuth) ;
                BeamWidth:Units = "Degrees" ;
        float GateWidth(Azimuth) ;
                GateWidth:Units = "Meters" ;
```

```
        float Reflectivity(Azimuth, Gate) ;
                Reflectivity:Units = "dBZ" ;


// global attributes:
                :TypeName = "Reflectivity" ;
                :DataType = "RadialSet" ;
                :Latitude = 32.573055267334 ;
                :Longitude = -97.3030548095703 ;
                :Height = 227.999999999916 ;
                :Time = 799875952 ;
                :FractionalTime = 0. ;
                :ExpiryInterval-unit = "Minutes" ;
                :ExpiryInterval-value = "15" ;
                :NyquistVelocity-unit = "MetersPerSecond" ;
                :NyquistVelocity-value = "53" ;
                :vcp-unit = "dimensionless" ;
                :vcp-value = "21" ;
                :radarName-unit = "dimensionless" ;
                :radarName-value = "KTLX" ;
                :Elevation = 0.46875 ;
                :ElevationUnits = "Degrees" ;
                :MissingData = -99900. ;
                :RangeFolded = -99901. ;
data:
*******Reflectivity values*********
```

In this example, the file above corresponds to a single elevation (sweep) of data. Gate is referred to as the regularly spaced samples and it depends on the pulse width.

A VCP (Volume Coverage Patterns) consists of the radar making multiple 360° scans of the atmosphere, sampling a set of increasing elevation angles.

Due to the size of the files, data from a set of scans is usually grouped and compressed in order to minimize the storage resources. However, weather radars generate large amounts of data which imply expensive procedures for storing and processing.

2.3 Grid Computing

The term *Grid Computing* was originated in the early 1990s as a metaphor for making computer power as easy to access as an electric power Grid. Many definitions can be found in literature on the grid concept. However, recently I. Foster [3] summarizes them as follows: A Grid is a system that coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service.

IBM defines Grid computing as the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity, to create a single system image, granting users and applications seamless access to vast information technology capabilities[2] .

As from these definitions, we can say that Grid Computing is a revolutionizing model proposed to solve challenging computational problems (protein folding, financial modeling, earthquake simulation, weather modeling, etc.), by taking advantage of unused resources from many networked computers, used as virtual computer architecture. Grid computing enables the distributed execution of expensive processes across a parallel infrastructure, performing as many computations as possible over the grid, than on a single computer.

[2] IBM Grid Computing, "What is grid", http://www-1.ibm.com/grid/

Grid computing is based on an open set of standards and protocols as Open Grid Services Architecture (OGSA), which enable communication across heterogeneous resources (based on different platforms, hardware and software architectures as well as computer languages) which are geographically dispersed.

2.3.1 Globus Toolkit

Many organizations have proposed and developed several new technologies that enable an effective interaction with the grid computing infrastructure. However, the Globus Toolkit [3] developed by the Globus alliance, has emerged as the preferred implementation tool to perform critical operations over the grid, becoming the de facto standard. This toolkit offers software to security, information infrastructure, resource management, data management, communication, fault detection, and portability. Globus toolkit is a fast growth technology, following the open source strategy, which avoids incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit is composed from the following set of components: Common runtime, security, data management, information services and execution management.

2.3.2 Gridsphere Framework

Grid portals are becoming very popular platforms to provide friendly and simplified interfaces to grid services and resources such as applications, data servers, application servers, clusters, sensing instruments, and so on. Grid portals, based on a web portal model, offer external developers, personalization and customization features through modular, extendable and reusable software components which may be developed independently from the purpose or the portal architecture [8]. These

[3] Globus Alliance, "Globus Tollkit", http://www.globus.org

modules (called portlets) are Java technology based web components, managed by a portal container which act as an intelligent dispatcher attending process requests and forwarding the proper response to the client through dynamic content. In addition portlets provide a presentation layer for Information (or Grid) Systems. The GridSphere [4] portal framework is a portlet JSR (Java Standard Resource) compliant container that allows portlet development through a set of classes and tools for web applications. Furthermore, this framework has included a collection of core services and portlets, providing easy environment to grid portals development. Figure 2–3 shows the portal architecture previously described.

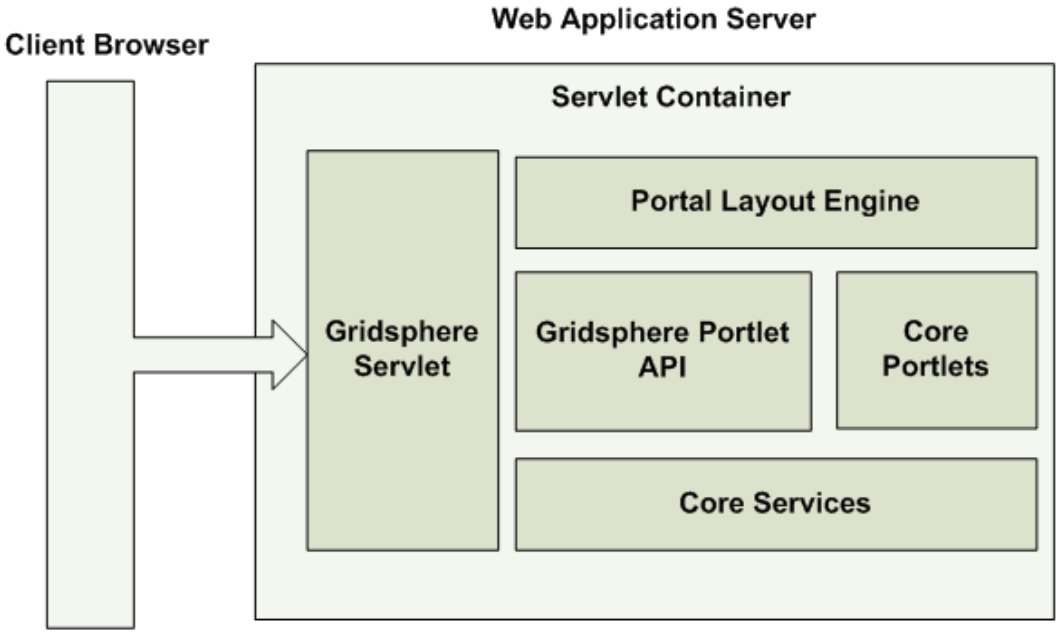

Figure 2–3: Gridsphere Portal Architecture.

2.4 Galois Field Theory

A finite field is any set of elements (that satisfies the field axioms) with a finite field order, also known as Galois field in honor of the French mathematician Évariste

[4] Gridsphere Project, "Gridsphere Framework", http://www.gridsphere.org

Galois. Galois field has many applications in number theory, algebraic geometry, cryptography, and coding theory.

The field axioms are a set of properties and rules to validate a finite field existence.

- There is a zero element (0) such that $a + 0 = a, \forall a$
- There is an identity element (1) such that $a \cdot 1 = a, \forall a$
- There is a multiplicative inverse (a^{-1}) such that such that $a \cdot a^{-1} = 1, \forall a \neq 0$
- There is an additive inverse ($-a$) such that $a + (-a) = 0, \forall a$

Addition and multiplication operations must be obey these rules. The field axioms can be summarized as follows:

Table 2–2: Field axioms summary

Name	Addition	Multiplication
Commutativity	$a + b = b + a$	$a \cdot b = b \cdot a$
Associativity	$(a + b) + c = a + (b + c)$	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$
Distributivity	$a \cdot (b + c) = a \cdot b + a \cdot$ c	$(a + b) \cdot c = a \cdot c + b \cdot c$
Identity	$a + 0 = a = 0 + a$	$a \cdot 1 = a = 1 \cdot a$
Inverses	$a + (-a) = 0 = (-a) + a$	$a \cdot a^{-1} = 1 = a^{-1} \cdot a, \forall a \neq$

The finite field order is the number of elements it contains. If $q > 1$ is an integer, then a finite field of order q exists if q is a prime power, otherwise it does not exist [9].

The structure theorem states that for finite fields [10] of a given order, there is exactly one (it is unique up to isomorphism) finite field of size q for each prime power number q. A Galois field with q elements often has many representations, however, it is traditionally denoted $GF(q)$ or F_q. Generally in a $GF(q)$, the term $a = b$ is equivalent to $a \equiv b(mod q)$.

Thus, if there is a $GF(16) = GF(2^4)$ of order 16, then every field with 16 elements is isomorphic to this field. Note that, a $GF(12)$ with 12 elements does not exist because 12 is not a prime power.

2.4.1 Classification of Galois Fields

The field theory establishes that a GF must exist for any prime power. Nonetheless, in practical applications such as cryptography only two kinds of field are widely used:

- The field $GF(p)$ is called a prime finite field of order p, if and only if p is a prime. The $GF(p)$ field is commonly represented as the set $Z_p = 0, 1, 2, 3, \ldots, p-1$ of integers modulo p.
- The field $GF(2^n)$ is called a binary field, when $q = 2^n$ for any positive integer n (also known as the degree of the field). The $GF(2^n)$ consists of the 2^n possible bit strings of length n

In a binary field, when $n = 1$ we obtain the simplest field $GF(2)$, which consists of the set $0, 1$ of integers modulo 2. The following addition and multiplication tables (Figure 2–4) must be satisfied as well:

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

Figure 2–4: Addition an multiplication tables in $GF(2)$.

2.4.2 Polynomial over Galois Fields

When $n > 1$, the finite field $GF(q^n)$ can be internally represented as polynomial rings to which coefficients belong to $GF(q)$ [9]. Arithmetic operations over $GF(q)$ (see Section 3.3) are defined as the same as in polynomial arithmetic except that the operations on the coefficients are performed in $GF(q)$. A polynomial over $GF(p)$ is usually known as polynomial modulo p. Multiplication and addition operations

are performed the same as for polynomials with integer coefficients, but the result coefficients are reduced modulo p.

2.4.3 Primitive Polynomials

Subfield and extension field concepts must be defined before discussing primitive polynomials. A subfield is a subset S of elements from a field F which satisfies the field axioms with the same operations of F. If a field F is a subfield of the field K, then K is an extension field (or extension, denoted K/F) of field F (i.e the complex numbers are an extension of the real numbers).

All elements of an extension field are generated from a field base, by a primitive polynomial. Furthermore, a polynomial over $GF(q)$ is reducible if it is the result of product of two smaller degree polynomial over $GF(q)$, thus, a polynomial is non-reducible, if and only if, it cannot be factored into a product of polynomials of a lower degree.

A polynomial over $GF(q)$ (excluding the zero polynomial, which has an indeterminate degree) has a unique representation as the product of powers of irreducible polynomials; hence a non-reducible polynomial of degree n over $GF(q)$ must exist for any prime or prime power q and any positive integer n.

For example, the field $GF(2^3)$ can be represented as polynomials with degrees less than 3, through a non-reducible polynomial. There are only two primitive polynomials of degree 3, over $GF(2^3)$; these are $P(x) = x^3 + x + 1$ and $P(x) = x^3 + x^2 + 1$.Using the modulus $x^3 + x^2 + 1$, the elements of $GF(8)$ can be written as the set $0, 1, x, x^2, x + 1, x^2 + x, x^2 + x + 1, x^2 + 1$, Table 2–3, shows the described example:

Note that, in general, $GF(2^n)$ is an extension field of degree n of $GF(2)$, where $GF(2)$ is called the base field of $GF(2^n)$, this implies that addition and multiplication tables are essentially the same, hence $x^m \pm x^m = -x^m \pm -x^m = 0, m = 0, 1, 2, \ldots, n - 1$.

Table 2–3: Polynomial representation of $GF(2^3)$

Elements & Polynomial representation $P(x) = x^3 + x^2 + 1$	Binary representation
$0 \mathrm{mod} P(x) \equiv 0$	000
$x^0 \mathrm{mod} P(x) \equiv x^0 = 1$	001
$x^1 \mathrm{mod} P(x) \equiv x$	010
$x^2 \mathrm{mod} P(x) \equiv x^2$	100
$x^3 \mathrm{mod} P(x) \equiv -x^2 - 1 = x^2 + 1$	101
$x^4 \mathrm{mod} P(x) \equiv -x^3 - x = x^3 + x = (x^2+1) + x = x^2 + x + 1$	111
$x^5 \mathrm{mod} P(x) \equiv -x^4 - x^2 = x^4 + x^2 = (x^2+x+1) + x^2 = x + 1$	011
$x^6 \mathrm{mod} P(x) \equiv -x^5 - x^3 = x^5 + x^3 = (x+1) + (x^2+1) = x^2 + x$	110
$x^7 \equiv x \cdot x^6 = x \cdot (x^2 + x) = x^3 + x^2 = (x^2+1) + x^2 = x^0 = 1$	–

2.5 Related Works

Current research efforts are being used to develop dynamically adaptive systems for analyzing and predicting the atmospheric conditions. Weather conditions often occur suddenly and evolve rapidly in such a way that an adaptive observation is necessary to perform accurate weather forecasting. Several research projects are focused on achieving this goal, in fact, multidisciplinary teams across multiple institutions are involved in this process. Researchers, students and technical staff from different areas such as meteorology, computer science, social science and other educational areas focus more attention of their work study efforts in this area to help mitigate the weather condition impacts through the analysis of current meteorological information. Thus, As a result of this growing interest in this area, weather forecasting is becoming one of the most important applications in computer science. An as such, Grid computing technologies are being considered as a promising solution in performing the most complex tasks involved in such a scientific challenge at this one. Grid technologies enable the virtualization of distributed hardware and software resources in such a way that the users and applications can take advantage of the available resources in a simple and transparent manner.

2.5.1 LEAD

Linked Environments for Atmospheric Discovery (LEAD) [11], is a Large Information Technology Research (ITR) Grant funded by the US National Science Foundation (NSF). LEAD is an IT framework for identifying, accessing, assimilating, forecasting, managing, analyzing, mining and visualizing a broad array of meteorological data and model output independent of format and physical location. LEAD is currently led by nine institutions and more than 100 scientists, students and different members from meteorology, computer science, social science and educational areas. The LEAD system is dynamically adaptable in terms of time, space, forecasting and processing:

Adaptation in time and space: Since weather events can appear and evolve fast, quick update to perform an accurate prediction is extremely important. Updating implies the running of successive forecasts more frequently than usual, in such a way that a prediction model can be adapted according to the evolution of the event. Beside this, models can also be adapted using nested grids. However, in this approach, large regions can be analyzed and then be divided into small regions in order to perform more detailed scans. LEAD automatically provides generation of nested domains in a grid context according to the available resources.

Ensemble Forecasting: Ensemble forecasting is a popular methodology in which several forecasts are generated, instead of just one. Forecasts are produced from different initial conditions, models different from each other, the same model started at distinct initiation times or when using different physic options within the same model or when using multiple models. Since ensemble forecasting requires high computational resources which are not desirable in all cases. In summary, LEAD offers an intelligent and automated adaptation.

Forecast Processing: Current tools used to perform forecasting and weather

events simulation consist of complex pieces of software. This type of software is generally developed over long periods of time by expert programmers. In addition, these programs generally must be configured using many variables, as well as managing different formats and data types. Thus, portability across computing platforms is limited, not scalable and very complex to deal with. One of the most important goals of LEAD is in helping to overcome these limitations by offering simple interaction method for the user when using the weather tools. The system capabilities of LEAD include; a complex array of services; applications; interfaces; and local and remote resources which are available to users to perform studies of mesoscale weather as it evolves. LEAD provides several foundational tools composed of a web portal (the primary user entry point), the ARPS Data Assimilation System (ADAS), a flexible metadata catalog service (myLEAD), the Weather Research and Forecast (WRF) and other important tools. LEAD is considered as a workflow orchestration for on-demand, real time, dynamically adaptive system (WOORDS).

The LEAD system consists of the following principal components:

User subsystem: Composed of the LEAD portal: it is the principal access point to LEAD technologies.

Data subsystem: Management of data and metadata, numerical model outputs and user generated information.

Tools subsystem: Composed of all meteorological and IT tools.

Orchestration subsystem: Permits users the management of data flows and model execution streams and permits them to create and mine output. It also provides a connection between the software and the processes for continuous or on-demand applications.

LEAD Grid: Consists of all distributed computing systems over six of the nine institutions involved in the project.

The LEAD system can be catalogued as a Service Oriented Architecture (SOA). SOAs are widely deployed in the commercial enterprise sector and they are the foundation for many scientific grid technologies. The LEAD SOA includes technologies and tools, such as, Globus toolkit, Unidata's Local Manager (LDM), Open-Source Project for a Network Data Access Protocol (OpenDap) and the OGSA Data Access and Integration (OGSA-DAI) service.

The LEAD portal is based on the NSF National Middleware Initiative Open Grid Computing Environment Portal toolkit (OGCE). This portal allows users authentication through proxy identity certificates, based on the Globus GSI model. Once user identity is verified, the LEAD grid, its services and resources will be made available for its users.

2.5.2 DCAS Network

Sensing of the atmosphere today, entails the uses of long range and high power radars in order to study meteorological phenomena. However, due to the earth's curvature, radars are limited to the observation of only approximately 72% of the troposphere below 1 Km, as shown in Figure 2–5. The meteorological conditions in the lower troposphere are under sampled. The observations of the lower atmosphere are important in estimating rainfall, as well as in the forecasting of flash floods.

The Research Center CASA [5] is operating to overcome the negative effects of the Earth's curvature and natures obstructions, such as, mountains and man-made physical obstacles like, buildings by employing low-cost networks of Doppler radars that operate at short range. In this manner, appearance and evolution of atmospheric phenomena at the lower atmosphere can be observed by the previously-mentioned systems. Rather than relying on a single radar to provide long range

[5] Gridsphere Project, http://www.casa.umass.edu

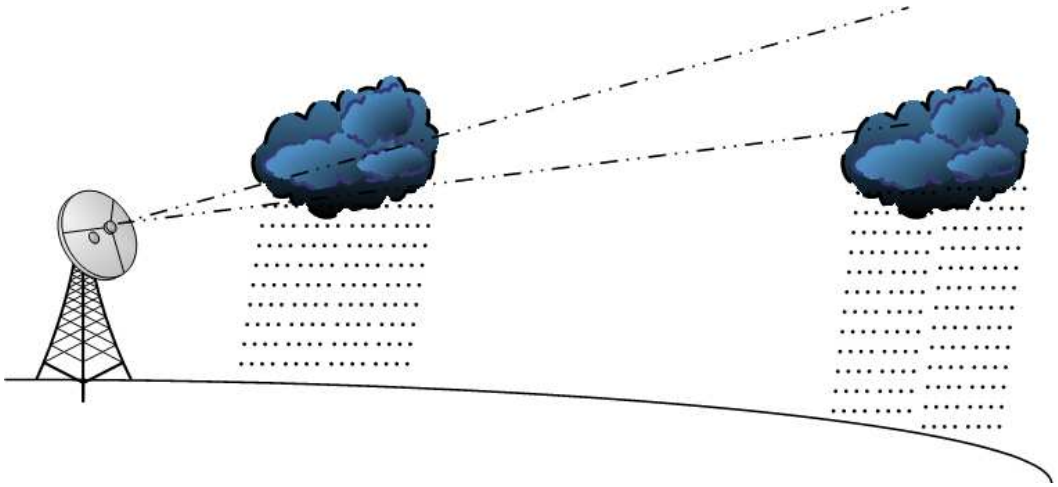

Figure 2–5: Current state of atmosphere sampling

(hundreds of kilometers) coverage, DCAS proposes to mosaic (see Figure 2–6) the output of lower power shorter range (tens of kilometers) radars [DPR]. This new approach is called DCAS, Distributed Collaborative Adaptive Sensing.

DCAS aims to radically change the radar paradigm, integrating remote sensing with multiple technologies, such as microwave engineering, electronics engineering, computer science and/or networking. DCAS network instruments can be located on building tops, communication towers and any suitable existing structure. These small radars are communicate between them, and rapidly can be easily adapted to their sensing mode in order to perform accurate tracking of weather changes.

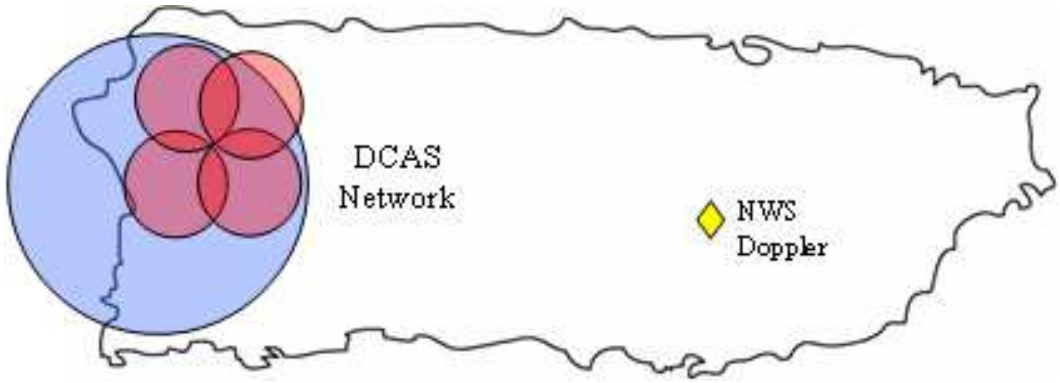

Figure 2–6: DCAS Network in Puerto Rico's western region.

Since the radars' range is reduced, the power peak required for operation is also reduced. Such reduction enables the usage of solid-state power amplifiers into the transmitter system. Additionally the radar frequency can be shifted from a lesser attenuated S-Band to a higher frequency such as X-Band. This change of frequency allows the reduction of the overall size of the radar antenna, while maintaining the same beamwidth. All these modifications lead to smaller, cheaper radars with low-power, which can be fed with batteries instead the connection to the local power grid ("off the grid" concept).

One of the most important features of the DCAS systems is the end-user interaction. The radars operate collaboratively within a dynamic IT infrastructure, adapting to changing atmospheric conditions in a manner that meets competing end user needs. These features introduce a dramatic change from current technologies.

2.5.3 Distributed Data Storage

Distributed data storage is a technique widely used for storing single data sets across multiple nodes from an organized architecture, such as, in cluster or grid form. Distributed data storage systems involve several techniques for the quick storage, search and retrieval of the more commonly used large sets of files. These systems aim to preserve the information, guaranteeing the availability of the data in event of a failure which could be caused by an application, a service, a single node, multiple nodes and/or even in case of the fault of the user.

Most of these systems improve availability by providing full replication. However, a few systems employ erasure-resilient correction codes require less space. A possible alternative to these approach systems is the redundancy schemes known as threshold schemes or information dispersal protocols. In these schemes, data sets are encoded, replicated and divided into multiple pieces of data which can be stored in

multiple nodes. Furthermore, a data set can only be retrieved if there are a defined number of pieces available.

Some examples of distributed data storage systems, including the previously mentioned features, are:

- ***PASIS***: Perpetually Available and Secure Information System [12] is a framework for demonstrating perpetually available information systems that guarantee the confidentiality, integrity, and availability of stored data even when some storage nodes fail to function correctly or when they are not available for use. PASIS is sponsored by the US Air Force and is led by the Carnegie Mellon University. The PASIS objective is creating survivable storage systems that are *perpetually available* (information availability in presence of failures), *perpetually secure* (integrity and confidentiality should always be enforced), and *graceful in degradation* (information access and performance should degrade gracefully, as system components fail). In the PASIS project many different techniques for encoding and distribution are experimented. Based on such tests, several models of performance, security, and availability of stored data, are proposed. These models enable to perform an engineering analysis and identify the encoding and distribution schemes which best meet the overall requirements for a particular storage system.
- ***Free Haven***: The Free Heaven [6] project aims to deploy a system for distributed, anonymous, persistent data storage which is robust, against attempts by powerful adversaries to find and destroy the stored data. This project began in 1999 and is being lead by several MIT students. It is oriented towards storing different kinds of documents while preserving the anonymity of the publisher, the readers and the storage servers. The main research goals of the project are *anonymity accountability* (without sacrificing anonymity), *persistence* (only the publisher determines the

[6] Free Haven Project, http://www.freehaven.net/"

document lifetime), and *flexibility* (the system continues to operate as peers dynamically join or leave). The system is based on a group of interconnected servers in which each server hosts data from another server instead of saving its own data in other servers. In this approach, a document is divided into shares, where a subset (any k of n) is sufficient to recovery of the document, and where each share negotiates for a server to publish that share onto the group of servers.

- ***Ocean Store***: OceanStore [13] is a global persistent data store designed to scale for billions of users. It provides a consistent, highly-available, and durable storage utility atop an infrastructure comprised of un-trusted servers. This project is led by John Kubiatowicz from the Computer Science Division at the University of California at Berkeley. Using this system, the unit of storage is the *data object* and all client data in the network is replicated and encrypted so that the hosting server cannot read the data object. To improve availability, each data object is divided into n fragments and recoded into kn fragments, and is further spread across many servers. Data size is increased by the factor of k, however, the original data can be reconstructed from any n fragments.
- ***GFS***: The Google File System is the distributed file system designed which supports large distributed data-intensive applications. GFS is widely deployed within Google as the storage platform and it can be extensive for research and development efforts that require large data sets. GFS provides high availability using two simple, yet effective strategies; fast recovery; and replication. The entire system is based on many clusters which are composed of a single master and multiple chunkservers. Files are divided into fixed-size chunks, which are identified by an unalterable and globally unique 64 bit chunk handle. Each chunk is stored as a plain Linux file on a chunkserver and in order to improve reliability; each chunk

is replicated on multiple chunkservers. The master maintains all file system metadata and clients interaction with the master for metadata operations. A technical paper with detailed information about GFS is presented in [14].

CHAPTER 3
RADAR DATA AVAILABILITY AND RELIABILITY

Implementation of redundancy schemes is a common strategy to enhance reliability in data storage [12]. Two different redundancy strategies have been implemented and analyzed: A simple replication scheme [15] and the Information Dispersal Algorithm (IDA) [16]. This chapter describes the details of the implementation of such redundancy strategies.

3.1 Replication Algorithm

The initial approach in order to provide reliability is known as splitting or striping technique [17][18]. Thereafter, the replication technique was added to overcome its limitations [19]. In the replication method, a file F is striped into m blocks of size $|F|/m$, where $|F|$ is the size of the original file. Blocks are labeled as $\{B_1, B_2, B_3, \ldots, B_m\}$. Each block $B_i (1 \leq i \leq m)$ is replicated r times in such a way that the total number of blocks is $n = m \times r$ and the storage spending is $r \times |F|$. To recover the original file, m blocks are required. However, in this procedure the label of the block must be taken into consideration. Thus, the redundancy $(r-1)\%$, is only effective if at least one block B_i exists. If all blocks with label i are damaged, corrupt or unavailable, the original file will not be recovered.

Assuming that blocks have independent probability of failure p, the recovery probability, also referred as access reliability, is determined by:

$$p(a) = \prod_{k=1}^{m} p_k(a) \tag{3.1}$$

Where $p_k(a)$ is the recovery probability for each blocks subset of r replicas with label i $\{B_{i1}, B_{i2}, B_{i3}, \ldots, B_{ir}\}$ and can be represented as

$$p_k(a) = \sum_{i=0}^{r-1} \binom{r}{i} p^i (1-p)^{r-i} \tag{3.2}$$

The replication algorithm is implemented as follows:

1. Divide the file size $|F|$ into m, in such a way that each block will be $|F|/m$ bytes. Let rd be the residue of the division operation. If $rd > 0$ then the last block will have rd extra bytes. Thus, padding is not necessary to make $|F|$ divisible by m. This feature inserts at least one block with a higher size, but if m is small in comparison with $|F|$, this effect is not significant.
2. Assign an ID label for each block and write them as separate files. The two first bytes in each file correspond to the label B_i, in such a way that, up to 2^{16} blocks can be permitted.
3. Write the labeled block r times in order to reach $n = m \times r$ blocks in total.
4. After the labeled files are ready, they must be distributed in n nodes or according to the established data distribution strategy. The complete path of these files will be registered in a log file.
5. In order to recover the file F, the existence of at least m blocks must be verified. When a file is found using the log file with the complete path, the two first bytes are read to identify the block label. If none of the blocks labeled as Bi is labeled, the recovery operation is not possible; otherwise, the first block Bi found is taken. The whole original file is recovered putting together these files in an organized manner.

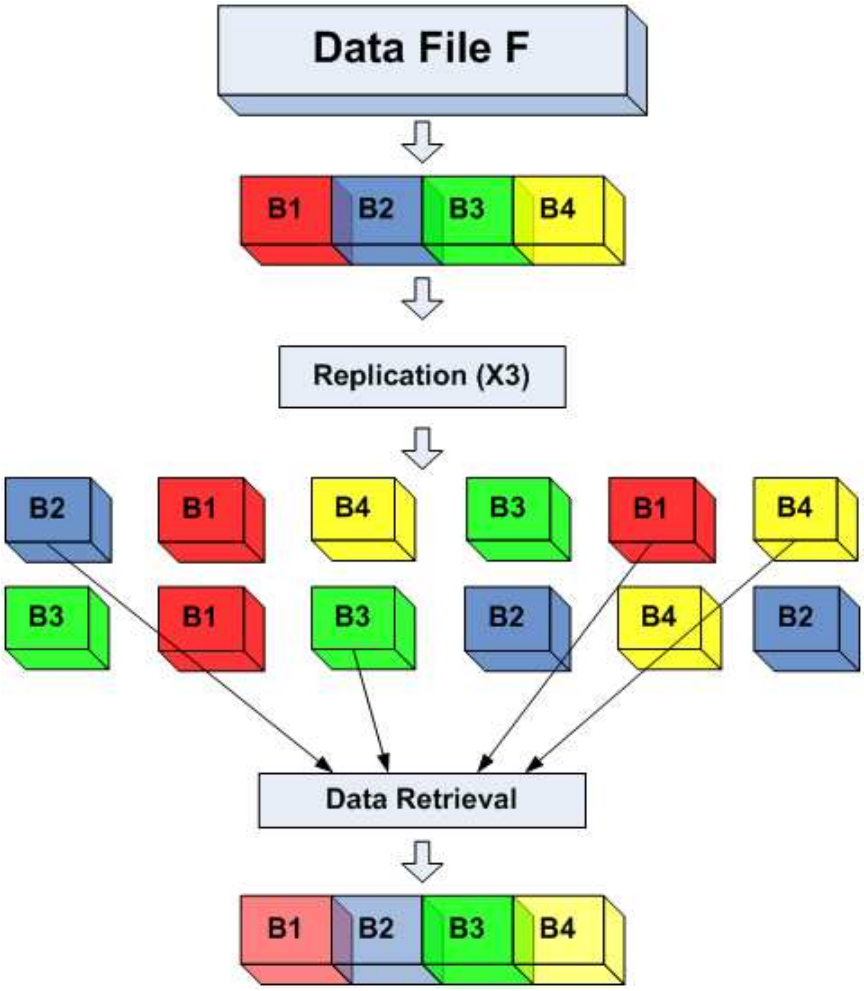

Figure 3–1: Replication example with $m = 4$ and $r = 3$

Figure 3–1 illustartes an example with four blocks an three replicas per block. In this case $p(a) \approx 0.77$ with a 200% redundancy, and $p = 0.4$.

3.2 Information Dispersal Algorithm

The information dispersal algorithm (IDA) was proposed as a fault-tolerance technique to be used in secure and reliable storage systems. In the basic approach, a file F is striped into n blocks of size $|F|/m$, where $|F|$ is the size of the file and m is the number of blocks required to recover the file F. A set of secret keys are used to disperse the file, providing confidentiality to the information. Since $m \leq n$, the redundancy level given by $(n/m - 1)\%$, can be selected to be smaller than the

replication technique. The storage spending is $|F| \times (n/m)$. An important feature of this technique is that any m blocks will reconstruct the file and, as a result, labels will not be necessary for each block. Additionally, IDA tolerates up to r failures, where $r = n - m$. Hence, IDA guarantees a higher availability.

When blocks have independent probability of failure p, the access reliability, is determined by:

$$p(a) = \sum_{i=0}^{n-m} \binom{n}{i} p^i (1-p)^{n-i} \tag{3.3}$$

Let F= b_1, b_2, b_3,... be a file, where b_i is an integer taken from a certain range $[0 \ldots (2^B - 1)]$. If bi is two bytes long, as in the actual implementation, then $0 \leq b_i \leq 65535$. Let q be a prime number greater than bi. Each bi is an element of the finite field Z_q where all arithmetic operations are done in mod q. Since $q > (2^B - 1)$, this implies an excess of one bit per byte when integers greater than $(2^B - 1)$ are obtained, this requires a storage space increment. In order to avoid the waste of space, all bi values are represented as polynomials with binary coefficients $(b_B x^B + b_{B-1} x^{B-1} + \ldots + b_1 x + b_0)$ and use a larger degree non-reducible polynomial $p(x)$ instead of the prime q [20]. The polynomial must suffice $(p(x) \in Z_2[x])$ in such a way that all operations can be done in the finite field $E = GF(2^B)$. GF refers to the "Galois Field".

In order to disperse F, a set of n vectors $a_1, a_2, a_3, \ldots, a_n \in E$ must be chosen, each of length m, such that every subset of m different vectors is linearly independent. These vectors are the keys that will be used to disperse every block of the file. Let $A_{n \times m}$ be a matrix whose i^th row is a_i. The file is divided into sequences of length m $(b_1, b_2, b_3, \ldots, b_m)$ and the dispersal operation is achieved mapping each sequence b_j into a new sequence of n elements using $A_{n \times m}$.

$$A_{n\times m} \cdot \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$$

Each resulting element c_i is stored in a separate block of file.

In order to reconstruct the file, m blocks are required $(s_1, s_2, s_3, \ldots, s_m)$ and the recovery operation is performed as follows: let $B_{m\times m}$ be a matrix whose rows are $(a_{s1}, a_{s2}, a_{s3}, \ldots, a_{sm})^T$. To recover the first m elements of F, the first element from each different block is needed. The whole file is obtained mapping sequences of m elements from each block into sequences of m elements using the inverse of $B_{m\times m}$.

$$B^{-1}_{m\times m} \cdot \begin{bmatrix} c_1 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

Note that the inverse of the $B_{m\times m}$ matrix is guaranteed since the rows of matrix A are mutually independent, which implies that any submatrix (in this case $B_{m\times m}$) is not singular and thus invertible by deleting $n - m$ rows of $A_{n\times m}$.

An $A_{n\times m}$ matrix $(n = m + r)$ with the properties above mentioned is the Vandermonde matrix. Let the rows of the Vandermonde matrix be indexed from 0 to $n - 1$. The i^th row of this matrix is defined as:

$$i^0, i^1, i^2, i^3, \ldots, i^{m-1}$$

By definition, this matrix has the property that any submatrix formed by deleting r rows of $A_{n\times m}$, is invertible. Additionally, any matrix derived from this matrix by a sequence of elementary matrix transformations, will maintain this property [21].

Finally, a non-reducible polynomial must be chosen, for the current implementation, the polynomial $p(x)$ of degree B over $GF(2^B)$, when $B = 16$ is

$$p(x) = x^{16} + x^{12} + x^3 + x + 1$$

The implementation of the IDA involves several operations over finite fields. In this case over $GF(2^{16})$. IDA is implemented as follows:

1. Create the dispersal matrix $A_{m \times m}$ which must obey the properties described above.
2. Divide the file F into sequences of m elements, where each element is 2 bytes of length. Note that $|F|$ must be divisible by m, therefore, padding must be added. In order to disperse the file, each sequence is multiplied by the matrix A to obtain the new sequences. The first block will have the 1^{st} element from the each new sequence. The second block will have the 2^{nd} element from that sequence and so forth.
3. A unique tag for each block must be established before these are written as separate files. This tag correspond to the i^{th} row of matrix A. This tag is necessary to choose the correct B matrix recovery.
4. After the tagged files are ready, they must be distributed in n nodes or according to the established data distribution strategy. The two first bytes of each file are used to identify the correspondent row. Thus a maximum of 2^{16} blocks are permitted. The complete path of these files will be registered in a log file.
5. In order to recover the file F, the existence of at least m blocks must be verified; this condition is necessary and will suffice in achieving the recovery operation. The two first bytes of each file are read to identify the row of the matrix A. The algorithm chooses the first m files and creates the recovery matrix B with the rows which were found. Then, the inverse of the B matrix is calculated.
6. Reconstruct the first sequence of m elements from the original file multiplying the matrix B^{-1} by the sequence formed by all the first elements from each file found.

Similarly, the second sequence from the original file is obtained, thus transforming the sequence containing all of the second elements from each file and so forth.

7. Finally, padding must be removed, if necessary, to obtain the original size of the file

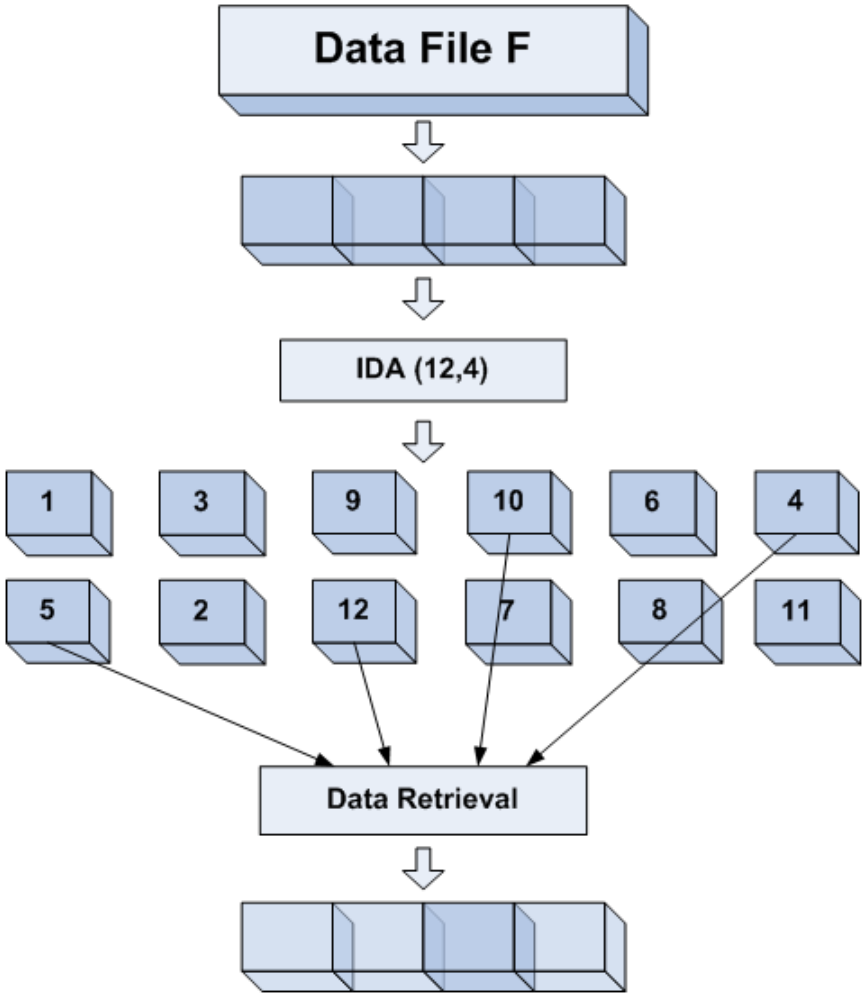

Figure 3–2: Replication example with $n = 12$ and $m = 4$

Figure 3–2 shows an example of the IDA behavior when $n = 12$ and $m = 4$. In this case, $p = 0.4$ and $p(a) \approx 0.98$, which has a better result than the reliability of the replication algorithm, with the same redundancy of (200%).

3.3 Implementation of Galois Fields Arithmetic

Arithmetic operations in finite fields are very different from classic arithmetic. Since a Galois field has a finite number of elements, the result of operations such as addition (+), subtraction, multiplication ($\cdot$) and division (except by zero), must be remaining within the field. As mentioned is section 2.4 a finite field must satisfy properties and rules of the field axioms.

3.3.1 Addition and Subtraction

In a finite field $GF(2^m)$, addition and subtraction operations are very simple; they are identical and can be calculated using the logical XOR operation.

Example:

Polynomial	Binary
$f(x) = x^3 + x + 1$ $g(x) = x^2 + 1$ $f(x) + g(x) = x^3 + x^2 + x$	$f = 1011$ $g = 101$ $f(x) \oplus g(x) = 1110$
$f(x) = x^3 + x + 1$ $g(x) = x^3 + x^2$ $f(x) + g(x) = x^2 + x + 1$	$f = 1011$ $g = 1100$ $f(x) \oplus g(x) = 0111$

Note that under traditional addition operations, the first equation outcome must be $x^3 + x^2 + x + 2$, but the x^0 coefficient becomes 0 and is dropped when the answer is modulo 2 reduced.

Since addition and subtraction can be performed using the exclusive OR operation, their implementation is reduced to the execution of the bitwise XOR operation between the binary representations of the polynomials.

```
/* Addition of two numbers in a GF(2^16)*/
word gfaddition(word a, word b){
return a^b;
}
/* Subtraction of two numbers in a GF(2^16)*/
```

```
word gfsubtraction(word a, word b){
return a^b;
}
```

Note: `word` is a 16-bits user defined type and is equivalent to `unsigned short` one.

3.3.2 Multiplication

The multiplication operation is more complex that of the addition in a $GF(2^m)$. Since the result of this multiplication must remain within the field, the operation must be calculated as a modular multiplication. Thus, it is executed in modulo primitive polynomial form, which is then used to define the finite field.

Modular multiplication is the most critical operation which affects the efficiency of the algorithms. In this document, two techniques to implement modular multiplication are discussed: the Rijndael's algorithm [22] and the logarithm look-up-tables.

Rijndael's Algorithm

Rijndael's Algorithm has been selected by the U.S. National Institute of Standards and Technology (NIST) as the candidate encryption system for the Advanced Encryption Standard (AES). In the description of the Rijndael block cipher, a technique to perform the multiplication procedure in Galois Fields is encountered. Rijndael uses a characteristic two finite field, also known as Rijndael's Galois field $GF(2^8)$, defined by the primitive polynomial $x^8 + x^4 + x^3 + x + 1$. The procedure for the multiplication in the GF is as follows:

- Take any two elements from the field, **a** and **b**, and define a product **p**.
- Set the product to 0.
- Make a copy of **a** and **b**.
- Run the following loop eight times:

1. If the low bit of **b** is set, execute the xor operation between the product **p** and **a**.
2. Assure that the MSB of **a** is set to 1 (one) or to 0 (zero).
3. Rotate **a**, one bit to the left, discarding the high bit, and set the low bit value to zero.
4. If the MSB of **a** is 1 (one) prior to this rotation, execute the xor operation between **a** and the hexadecimal number `0x1b`.
5. Rotate **b**, one bit to the right, discarding the low bit, and making the MSB has a value of 0.

- The product **p** now contains the product of **a** and **b**.

Note: MSB represents the most significant bit, defined as the eighth bit from left to right.

The method described above assumes operations in Galois Fields with 2^8 elements and it has a defined primitive polynomial. In the current project, the selected polynomial generator is set to $x^8 + x^4 + x^3 + x^2 + 1$ whose equivalent value is binary `1000101101` or decimal `285`. This polynomial is also a non-reducible polynomial order `255` over $GF(2^8)$. In order to scale this procedure to $GF(2^{16})$ with a different primitive polynomial, some modifications are required. The Rijndael's Algorithm (also known as AES) is a bit more complicated than addition and involves several operations. The procedure can be performed using the following code:

```
/* Multiplication of two numbers in GF(2^8) defined by the polynomial
x^8 + x^4 + x^3 + x + 1 */
byte gfmultiply(byte a, byte b) {
     byte p = 0;
     byte counter;
     byte hibit_set;
     for(counter = 0; counter < 8; counter++) {
```

```
        if((b & 1) == 1)
           p ^= a;
        hibit_set = (a & 0x80);
        a <<= 1;
        if(hibit_set == 0x80)
           a ^= 0x1b; /* x^8 + x^4 + x^3 + x + 1 */
        b >>= 1;
     }
     return p;
}
```

Multiplication using the Look-up Table Approach

Operations such as multiplication, division, exponentiation and inversion can be accelerated using the look-up table approach [23] [24].

Table look-ups are based on the idea of computed logarithm and inverse logarithm tables in the finite field $GF(2^n)$. These logarithm functions are described in their discrete sense. We denote this function as "gflog" and "gfantilog", respectively. This implementation is based of the fact that the inverse logarithm of an integer k is equal to the inverse logarithm of $(k \texttt{mod}(2^n - 1))$. Since all non-zero elements of $GF(2^n)$ can be represented as the power of a primitive element w, the gflog function is defined as follows:

$$k = \texttt{gflog}(w^k), w^k \in GF(2^n), k \in \{1, 2, \ldots, 2^n - 1\}$$

Similarly the gfantilog function can be defined as:

$$w^k = \texttt{gfantilog}(k) = \texttt{gfantilog}(\texttt{gflog}(w^k)), w^k \in GF(2^n), k \in \{1, 2, \ldots, 2^n - 1\}$$

Thus, the product of any two elements of $GF(2^n)$ can be reduced to a traditional addition operation and three table look-ups:

$$w_1 * w_2 = \texttt{gfantilog}(\texttt{gflog}(w_1) + \texttt{gflog}(w_2))\texttt{mod}(2^n - 1), w_1, w_2 \in GF(2^n)$$

To perform the modular multiplication, the appropriate integer values for `gflog` and `gfantilog` tables must be present. These tables for both 8 and 16 bits, can be generated using the following routines:

```
/* This routines generate the the logarithm and antilogarithm tables
for GF(2^8) and GF(2^16) */
#define PPoly8 0435 /* Octal Notation for X^8+X^4+X^3+X^2+1 */
#define GFs8 256 /* Define this GF size */
#define PPoly16 0210013 /* Octal Notation for X^16+X^12+X^3+X+1 */
#define GFs16 65536 /* Define this GF size */
static byte GFlog8[GFs8], GFantilog8[GFs8]; /* 8 bits arrays*/
static word GFlog16[GFs16], GFantilog16[GFs16]; /* 16 bits arrays*/
void logtables_8(){
int i,b;
for (i = 0; i < GFs8; i++) {
     GFlog8[i] = GFs8-1; /* Fill array with 255 */
     GFantilog8[i] = 0; /* Fill array with 0 */
     }
b=1;
for (i = 0; i < (GFs8 - 1); i++) {
     GFlog8[b] = j;
     GFantilog8[j] = b;
     b = b << 1;
     if (b & GFs8)
         b = (b ^ PPoly8) & (GFs8 - 1);
     }
}
void logtables_16(){
```

```
/* identical to logtables_8 function;
appropriate variables must be replaced */
}
```

These routines can be executed at the beginning of the process so they can used to perform the required GF operations. However, in practical experimentations, quick computing of the arithmetic operations is achieved using pre-computed tables instead of dynamically generate ones. Furthermore, In order to store the log arrays, the memory requirement is approximately 512 *bytes* for $n = 8$ and 256 $Kbytes$ when $n = 16$. Thus, with a moderated memory usage the product of the two elements in a GF, can be performed in a quick manner, and with a more consistent speed.

Once the log tables are obtained, the procedure to multiply two numbers is performed as follows:

```
/* Multiplication of two numbers in a GF(2^16) defined by
the primitive polynomial X^16+X^12+X^3+X+1 */
word gfmultiply(word a, word b){
unsigned int sum;
word c;
if (a == 0 || b == 0) {
    c = 0;
    }
else {
      sum = (word) (GFlog16[a] + GFlog16[b]);
      if (sum >= GFs16-1) /* Performs the mod operation */
          sum -= GFs16-1;
          c = GFantilog16[sum];
      }
return c;
```

```
}
```

3.3.3 Division

Like the multiplication process, the division operation is performed following the look-up table approach. The result from dividing any two numbers of a $GF(2^n)$ is obtained by an integer subtraction, instead of addition in the same multiplication procedure.

The modular division can be implemented as follows:

```
/* Division procedure in a GF(2^16) defined by the prime
polynomial X^16+X^12+X^3+X+1 */
word gfdivide(word a, word b){
unsigned int sum;
word c;
if (a == 0 || b == 0) {
    c = 0;
    }
else {
      sum = (word) (GFlog16[a] - GFlog16[b]);
      if (sum >= GFs16-1) /* Performs the mod operation */
          sum -= GFs16-1;
          c = GFantilog16[sum];
      }
return c;
}
```

3.3.4 Multiplicative Inverse

Multiplicative inverse operation plays an important role in cryptography and is often referred to as the reciprocal. Every number except zero (0) has a reciprocal. The reciprocal of a number $a \in GF(2^n)$ can be calculated using the log and antilog tables technique. The next procedure can be followed in order to find the reciprocal a^{-1}:

- Look up the `GFlog` of `a`.
- Subtract the value of `a` from 255.
- Look up the `GFantilog` of the previous result.

The multiplicative inverse can be implemented as follows:

```
/* Reciprocal for any number in a GF(2^16) */
word gfmulinv(word number) {
if (number == 0) /* 0 has not reciprocal and it is self inverting*/
    return 0;
else
    return GFantilog16[255 - GFlog16[number]];
}
```

3.4 Improving Computation Time

As previously discussed in section 3.3, multiplication is the most critical operation over GF, which can be accelerated using look-up tables. Therefore, the routine for implementing modular multiplication can be modified using different techniques. The modifications must be focused towards the memory management and the length of the string. These techniques are listed and described as follows:

1. **Memory Allocation (MMA)**: In this technique, the entire data file is copied into the memory, using a single system call to allocate memory. The manipulation

of the data is performed as if the data were local variables.

Advantage: bit string readings are preformed quite fast.

Disadvantage: the block of allocated memory must be as large as the size of the file.

2. **Memory Mapping (MAP)**: In this technique, the data file is mapped (i.e. not copied) into the memory. MM creates a one-to-one correspondence between data in the file and data in the mapped memory region [25]. Additionally, MM eliminates buffering in order to save memory space.

   Advantage: bit string readings are preformed quite fast.

   Disadvantage: memory maps can only be performed in UNIX based systems which affects the code portability across platforms.

3. **File-Read Loop (FRL)**: In this technique, the bit strings of the file are read using a file-read loop instead of copying or mapping the data into the memory. Thus, the block of allocated memory is as large as the length of the bit string.

   Advantage: memory resources can be used to perform other processes in the server.

   Disadvantage: bytes of the file are read in an iterative way, which implies many system calls.

These techniques were implemented using both, bit strings of 8 and 16 bits of length. This parameter is important because it determines the length of the vectors that are operated by the dispersion matrix (see section 3.2).

CHAPTER 4
GRID IMPLEMENTATION AND RADAR INTEGRATION

This chapter presents a detailed description of the Grid-service based system implementation and how radar network is integrated with the grid resources and services. We provide an overview of the available infrastructure as well as a description of the portlets and services developed.

4.1 Grid Infrastructure

The PDCLab Grid Testbed, deployed at the University of Puerto Rico-Mayaguez, is an experimental grid designed to address research issues, such as the effective integration of sensor and radar networks into grid infrastructures. The PDClab grid test-bed components run CentOS 4.2 and the Globus Toolkit 4.0.1. The Globus Toolkit 4.0.1 includes, among other components, services, such as a security infrastructure (GSI), data transport service (GridFTP), execution services (GRAM), and Information services (MDS).

- The Grid Security Infrastructure is used by the Globus Toolkit for authentication and secure communication. GSI is implemented using public key encryption, X.509 certificates, and the secure sockets layer (SSL) communication protocol and incorporates single sign-on and delegation.
- The Monitoring and Discovery Service (MDS) is used to discover, publish and access both static and dynamic information from different resources in a computational grid. MDS uses the Lightweight Directory Access Protocol (LDAP) to

access such information on the different grid components and provides a unified view of the disparate grid resources.

- The Globus Resource Allocation Manager (GRAM) is used for allocation and management of resources on the computational grid using a Resource Specification Language (RSL) to request resources. GRAM also updates the MDS with information as to the availability of grid resources. The GRAM API can be used to submit a job, query the status of a job, and cancel a job. A GRAM service runs on each resource that is part of the grid and that is responsible for interfacing with the local site resource management system (e.g. OpenPBS, Condor).
- GridFTP is a secure, high-performance and robust data transfer mechanism used to access remote data. In addition to GridFTP, Globus provides Globus Replica Catalog to maintain a catalog of dataset replicas so that, instead of duplicating large datasets, only necessary pieces of the datasets are stored on local hosts. The Globus Replica Management software provides the replica management capabilities for data grid by integrating the replica catalog and GridFTP.

The computational resources available on the Grid testbed include:

- An IBM xSeries Linux cluster with 64 nodes, dual-processor at 1.2GHz, 53GB of memory and 1TB of storage.
- Eight (8) IA-64 Itanium servers, dual processor at 900 MHz, each with 8GB of memory and 140GB of SCSI Ultra 320 storage.
- Two (2) IA-32 Pentium IV servers, dual processor at 3 GHz, each with 1GB of memory and 120GB of ATA-100 storage.
- One (1) IA-32 Pentium III server, dual processor at 1.2 GHz with 2GB of memory and 40Gb of SCSI Ultra 160 storage.
- One (1) IA-32 Xeon server, dual processor at 2.8 GHz, L2 Cache 1MB with 1GB of memory and one 230 GB RAID of storage (STB Server).

4.2 Grid-Service Based System Architecture

The grid-service based system which is used is composed of several components, such as data replication algorithms, data transportation protocols, and web components that must interact into the grid computing environment. In turn, the components of the system involve a variety of elements, including C routines, Java classes, scripts, descriptors and grid tools. Thus, an adequate configuration and synchronized operation of those components are required to enable a capable structure that supports the processes of the implemented applications. We can group these elements in four principal modules: The distributed storage system; the Grid portal interface; the Grid connection interface; and the Services for end-users.

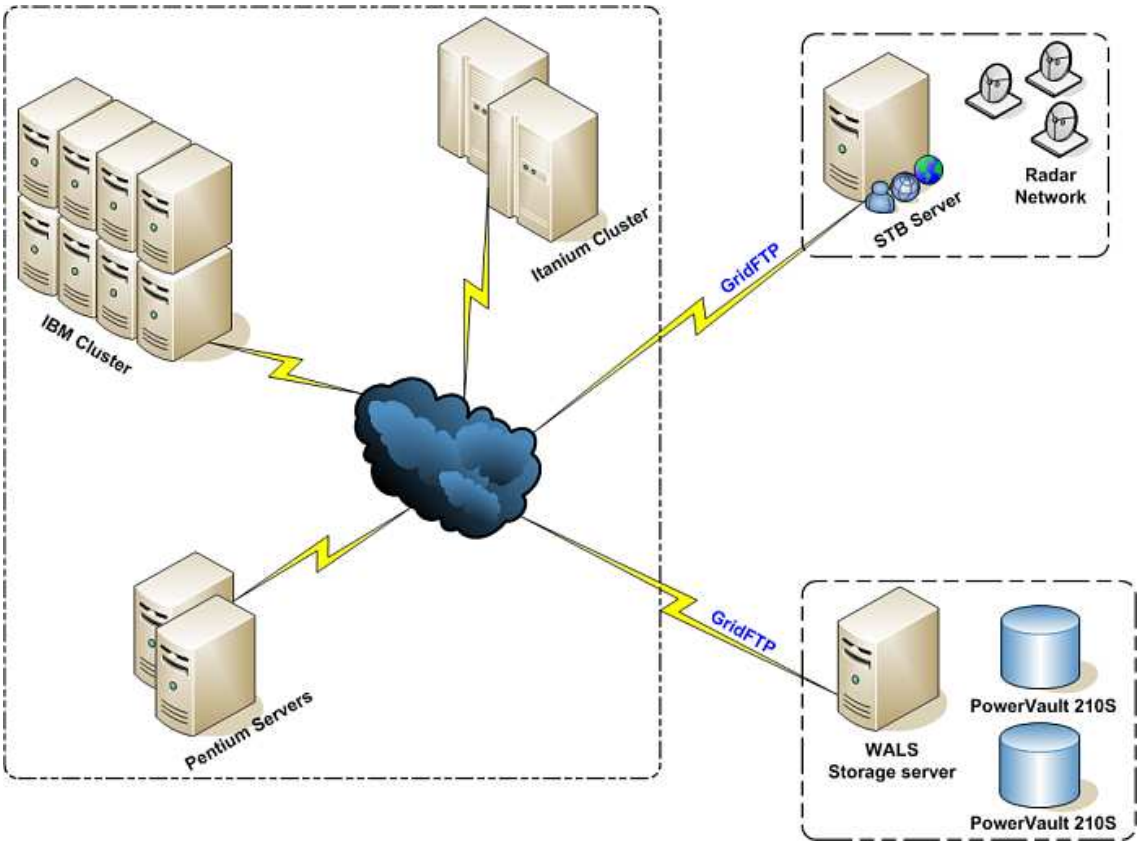

Figure 4–1: Proposed Grid-service based system

Figure 4–1 shows an overview of the Grid-Service based system structure. Raw data from radars are sent to a data server via wireless communication. GridFTP is

used to improve data transport from the data server to the PDCLab Grid Testbed. And data exchange between server and Grid testbed is authenticated using Grid Security Infrastructure (GSI).

Figure 4–2 shows the integration of the Mayaguez node with the grid testbed. Interconnections and the interfaces which can be identified.

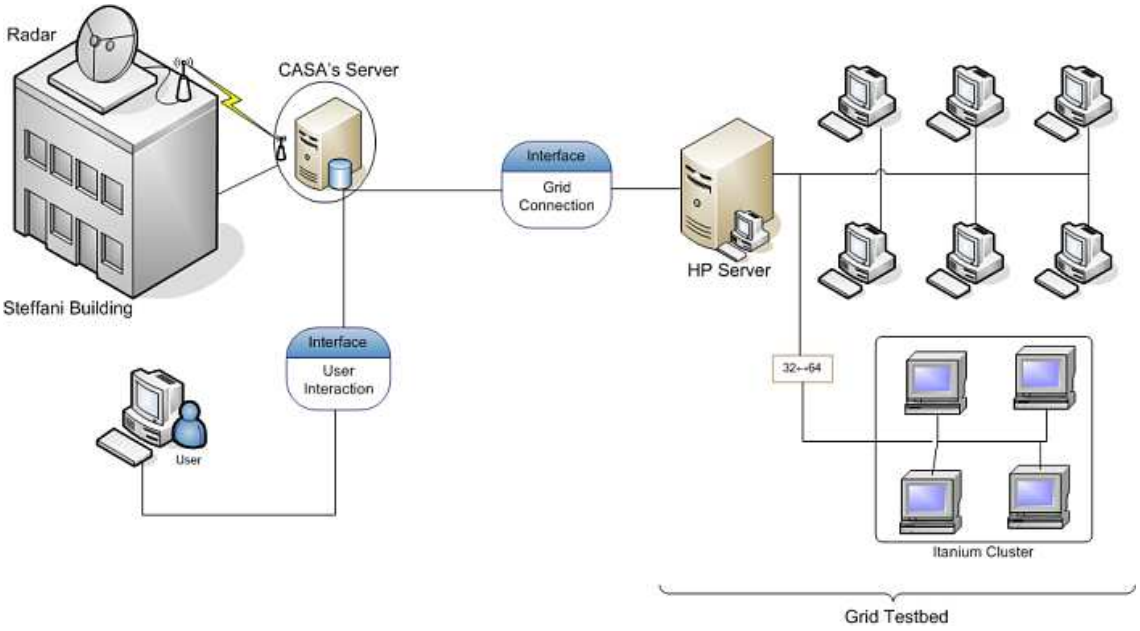

Figure 4–2: Mayaguez Radar Integration

4.3 Distributed Storage System

One of the main goals of the DCAS system is the radar data availability, meaning that an end-users may interact with the network by receiving and requesting meteorological data, where and when they require it. In order to achieve this goal, a distributed storage system was developed which is composed of a replication algorithm (previously discussed in chapter 3) to provide access reliability, and grid tools to improve data distribution into the grid.

As mentioned before, weather radars generate large amounts of data which imply expensive procedures for storing and processing. As an example, we can

consider the magnetron-based radar, located at the top of the Stefani building at the University of Puerto Rico - Mayaguez, the approximated data rate can be calculated from the radar specifications:

- Max Range: 30 km
- Resolution: 150 m
- Gates: $MaxRange/Resolution = 200$
- Sampling Average: $(1/2) * Beamwidth = 1deg$
- Total Samples: $Gates * 360 = 72000$

If the acquisition board has a 16 bits resolution, 7200 samples are multiplied by 2 bytes, add headers, labels and other information related to the NetCDF format, and the results are raw-data files of $\approx 200KB$ per sweep. TThe R.P.M of the radar is 3; this implies 3 sweeps per minute, this translates into $600KB$ per minute, $36MB$ per hour and $864MB$ per day. Using IDA to perform the data replication, data size can be multiplied by $1.25, 1.5, 2, , (n/m)$. With a redundancy of 100%, the data rate per minute is $12MB$, $72MB$ per hour, and therefore $\approx 1.7GB$ per day. Considering the storage resources of the server (see section 4.1), a distributed storage system, involving a redundancy scheme, provides a more reliable solution than a local storage system.

The distributed storage system function can be summarized as follows: Data files are dispersed using the Information Dispersal Algorithm; the redundancy level is determined by the number of nodes on the grid testbed and the desired access reliability (see section 5.2 for more details); blocks, generated by IDA, are sent to the grid testbed using gsiftp protocol (supported by the GridFTP tool), in a 1:1 distribution, meaning a block of file per node; original files are erased from the server to save storage resources and the data remains distributed in the grid; data is identified by the scan date, and daily sets are registered in log files which are kept in the server; information related to the dispersed data is registered in index files, for

the recovery process; when a specific file is required at the server, the system looks for the requested file in the index files to determine if the file exists; the required blocks are returned from the grid testbed to the server using GridFTP; and finally, the recovery operation can be performed and the request file will be made available.

4.4 Grid Portal Interface

The Grid Portal interface ensures a transparent mechanism for accessing resources and grid services. Gridsphere is the selected framework to which develops the STB Grid portal. The developed portlets provide a presentation layer for the manipulation of both, processed data and raw-data from radar, and for the services for end-users. Additionally, portlets make the visualization of weather information, such as reflectivity possible, in order to estimate rainfall rate over the western area of Puerto Rico.

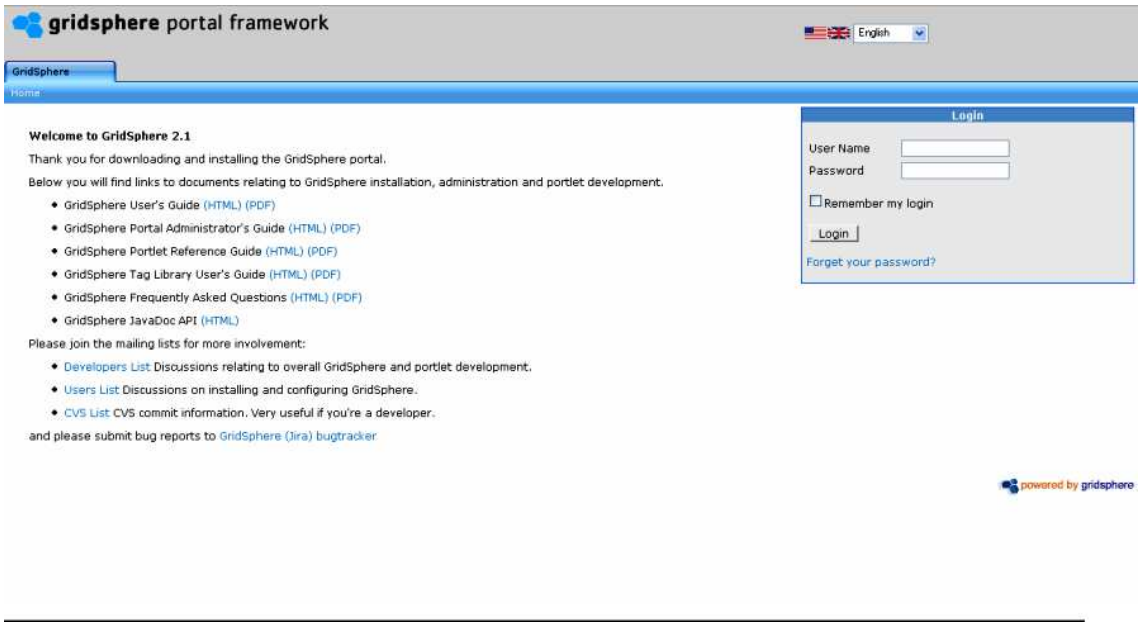

Figure 4–3: Gridsphere Portal Interface

The modification of the portal presentation layer and core portlets, included in the basic installation are made possible by Grisphere. Figures 4–3 and 4–4 show the basic gridsphere portal and the customized STB portal. Note that gridsphere

provides portlets to the user account manager which keeping managed in the STB portal design to improve controlled access to certain resources and services which will be explained further on.

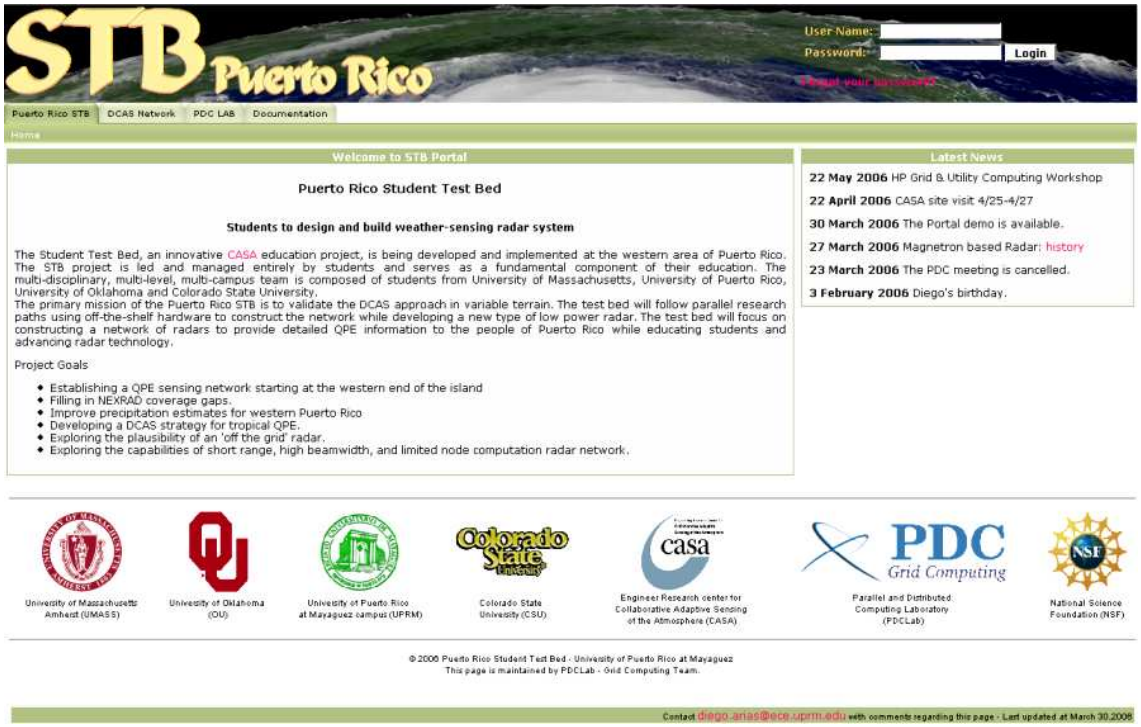

Figure 4–4: STB Grid Portal Interface

4.4.1 Data Management

Users can access raw-data from radars through the grid portal. Files containing the raw-data are stored using the Netcdf (as known as NC) format. The data management portlets permit end-users to download the data in such a way that they can obtain an exact copy of a file or a set of files. To avoid the server overload, raw data request is restricted to registered users only. Raw-data does not provide comprehensible information; it requires additional tools for extraction and processing. As a result, this feature is designed for advanced users (like students, teachers and researchers) who have the adequate software and previous knowledge of data

from radars. These kinds of users are able to request an account from the portal administrator.

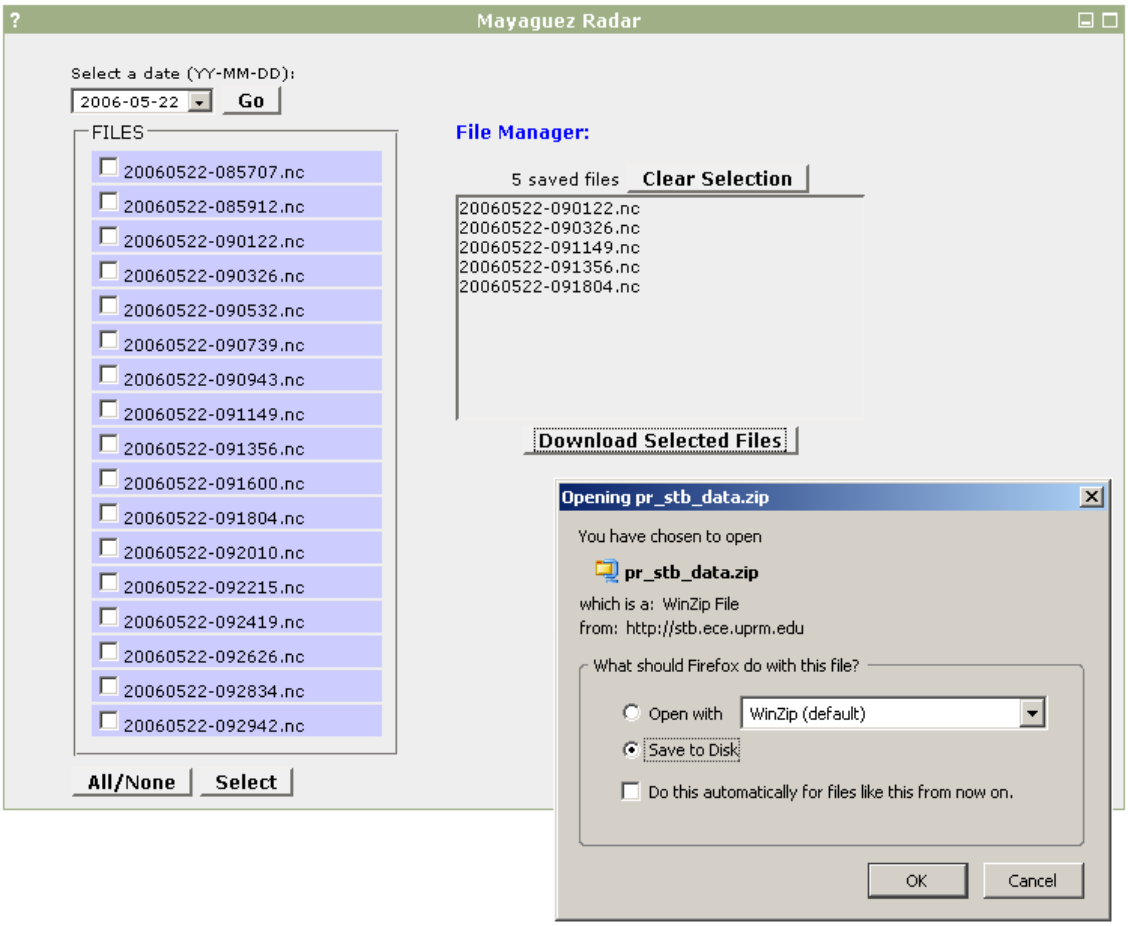

Figure 4–5: Data management portlet

Figure 4–5 shows the data management portlets. Once the user has been logged into the portal, the raw data request portlest are made available. The initial portlet shows a single selection form that permits the selection of the date of interest and then all available data is listed. Then, the data set selected can be downloaded as a compressed file.

4.4.2 Weather Information

The grid portal provides current rainfall estimates over the western area of Puerto Rico through reflectivity displays. This information is unrestricted and is

available for anyone who accesses the portal. Figure 4–6 shows how the base reflectivity information corresponding to a sweep is plotted over the Mayaguez area.

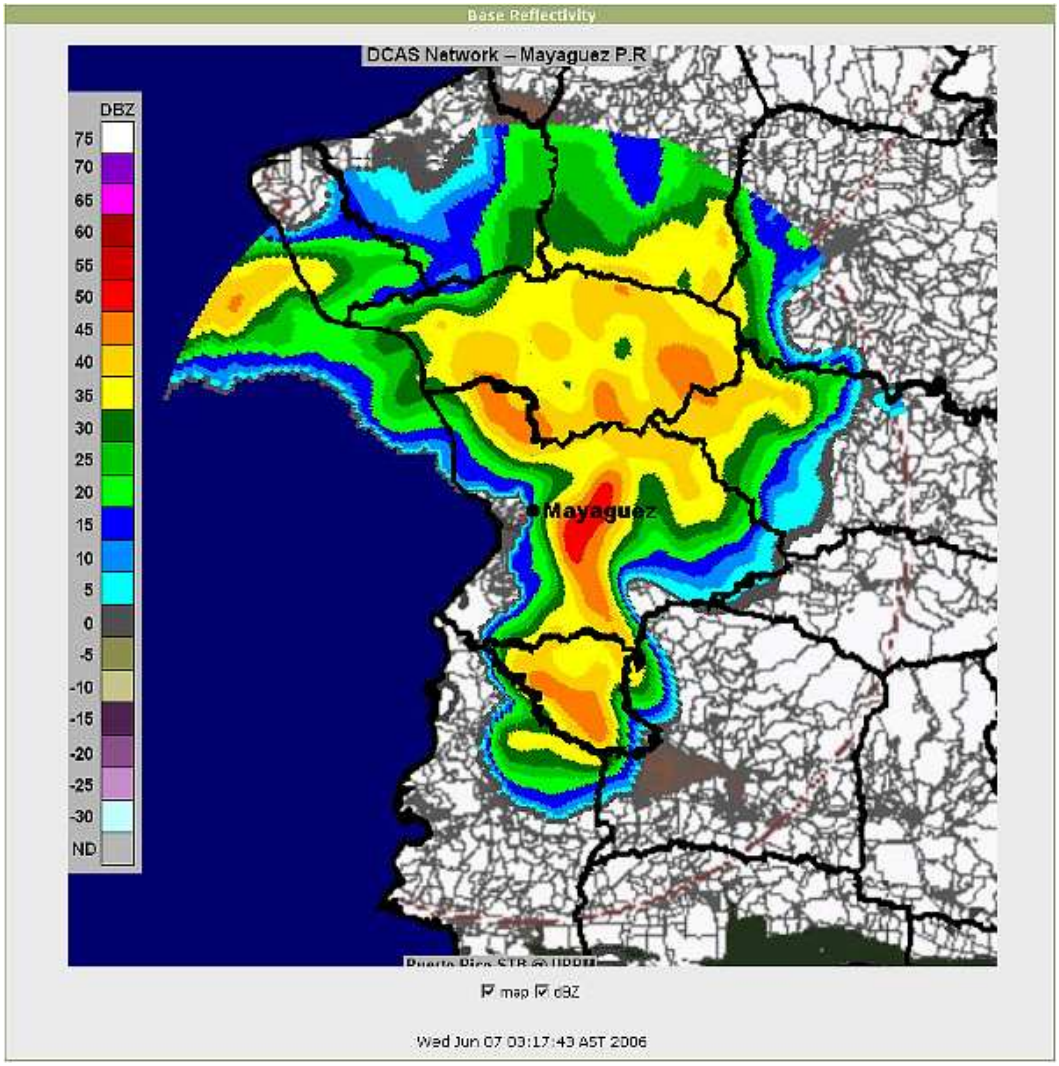

Figure 4–6: Base reflectivity portlet

Figure 4–7 shows a portlet used to display a set of base reflectivity over the Mayaguez area. This portlet performs the animation of the data set and includes loop controls and zooming. The base reflectivity loop is useful in facilitating the tracking of meteorological phenomena.

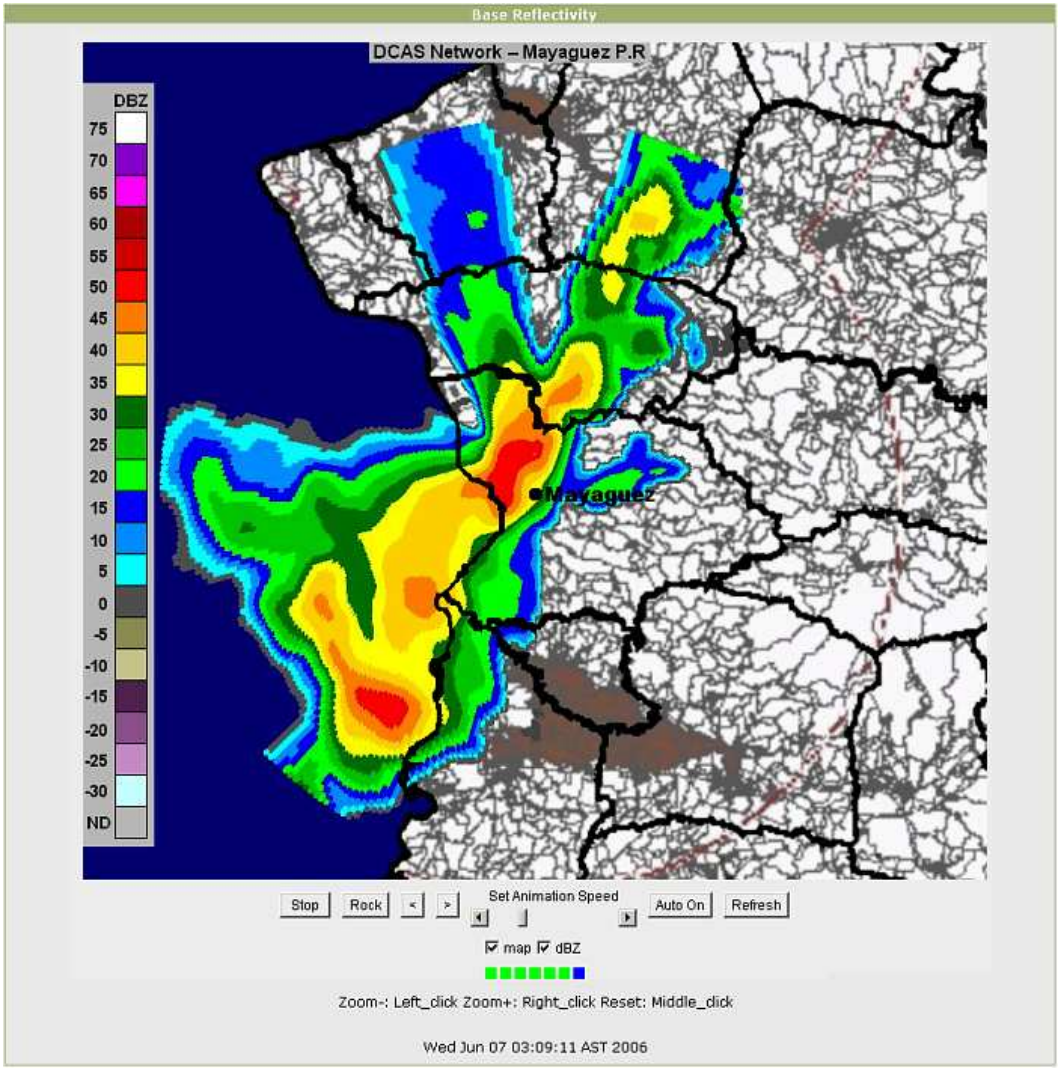

Figure 4–7: Base reflectivity animation portlet

4.4.3 Services for end-users

Netcdf data is written as binary files, thus, it can not be read by users as plain text, and specialized software is required for its interpretation. There are several libraries, plugins, programs and a variety of tools to manipulate NC files, however, installation, configuration and usage of these tools can be very complex for inexperienced users. Additionally, due to format flexibility, the structure of the files varies, depending on the implementation procedure. And so, to perform a specific task, one or more software tools are needed. For example, there is not

available software to generate the reflectivity plots from the radar raw-data. So, a Java class was developed using more basic classes and libraries for NC manipulation. Additionally, a similar class was developed to convert NC to ASCII.

To facilitate the manipulation of the raw-data from DCAS network nodes, two very useful services were implemented. These services allow end-users the execution of processes over the raw-data available in the storage system. Thus, users can upload its data sets from a local machine to the server, and process them. The available processes are:

- NCtoJPG: Rainfall rate plots are available in the Grid portal; but older plots are not maintained in to safe storage. Using the grid portal, users can send out from date data sets to the grid and, then, receive the corresponding reflectivity plots.
- NCtoASCII: Through use of the grid portal, users can convert the NC files to text files. This tool eliminates the utilization of extra software for data manipulation.

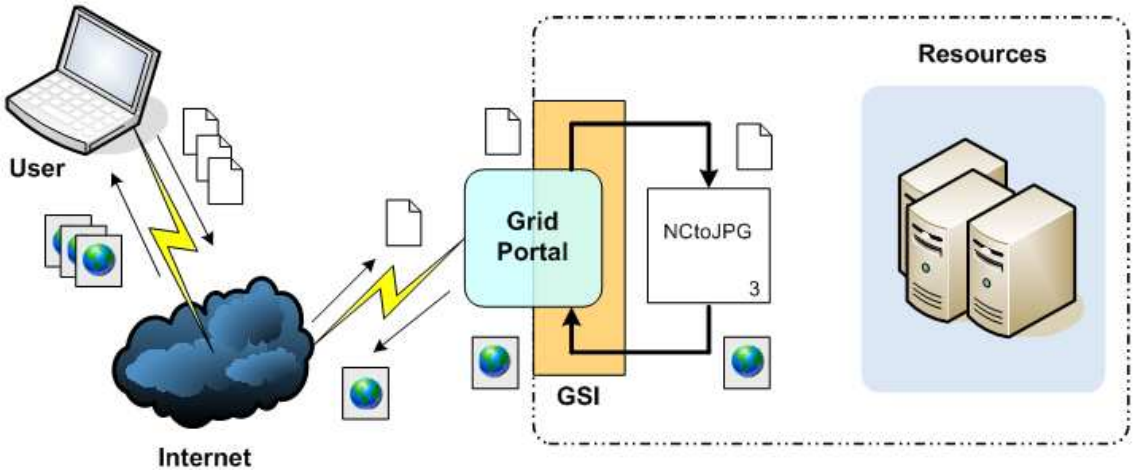

Figure 4–8: Services for end-users example

Figure 4–8 shows how raw-data is sent from a local machine to the server and how output files are sent back to the user.

4.5 Grid Connection Interface

Grid connection interface includes the necessary tools for the integration of the radar network to the grid computing technologies. In order to achieve this process, the first step included the installation of the Globus toolkit in the STB server to enable the utilization of the grid resources and services. The second step included the deployment of a gigabit link between server and PDC grid, to secure fast data transport. The following sections offer a description of the necessary tools and procedures to perform data management and service requests.

4.5.1 Data Management

The functions of the distributed data storage system were previously described in section 4.3, so they are skipped at this stage. The following procedure describes the manner in which to perform the data distribution on the PDC grid:

1. Execute the IDA (dispersal mode) over each raw-data file stored in the server, in such a way that $\boldsymbol{n}$ chunks are obtained.
2. Check if the required credential to access the grid is valid. If it has expired, then the renewal will be necessary. Select $\boldsymbol{n}$ nodes from all the available nodes on the grid.
3. Send the chunks to the grid using a 1:1 distribution, using gsiftp as the transport protocol. Use the globus-url-copy command, for example:

   ```
   globus-url-copy gsiftp://proc.uprm.edu/data/dcas/node1/
     file:///data/dcas/node1/20060512-080523.n-000.ida
   ```
4. Register in separate log files all the information related to the dispersed files, such as path, name, size and IDA settings.
5. Delete the original raw-data file.

Example: Log file content

```
232892 # File size in bytes
116446 # Added redundancy in bytes
0 #Padding in bytes
4 # n
2 # m
/data/dcas/node1/20060523-025818.nc-000.ida
/data/dcas/node1/20060523-025818.nc-001.ida
/data/dcas/node1/20060523-025818.nc-002.ida
/data/dcas/node1/20060523-025818.nc-003.ida
```

In reply user requests for distributed data on the grid testbed, the procedure is similar:

1. Check if the requested file(s) have been distributed into the nodes. This information is extracted from the log files.
2. Check for a valid credential. Select $\boldsymbol{m}$ nodes from the $\boldsymbol{n}$ nodes used in the dispersal operation.
3. Retrieve the chunk files from the using GridFTP

   ```
   globus-url-copy file:///data/dcas/node1/
     gsiftp://proc.uprm.edu/data/dcas/node1/20060512-080523.nc-000.ida)
   ```
4. Execute IDA (retrieve mode) over the retrieved chunk of files.
5. Allow user to download the reconstructed files and then delete them.

4.5.2 Services for end-users

Services for end-users involve execution of a process over a single file or over a set of files. For instance, a set of NC files can be uploaded with a NCtoJPG request. Data is processed and the output files are made available for downloading, using the grid portal. This entire procedure is transparent for the users, but may indicate issues which may need scrutiny. The server may process each file and then reply

to the output files; nevertheless, the server could be receiving data from the radar network or replying to other user requests at the same time.

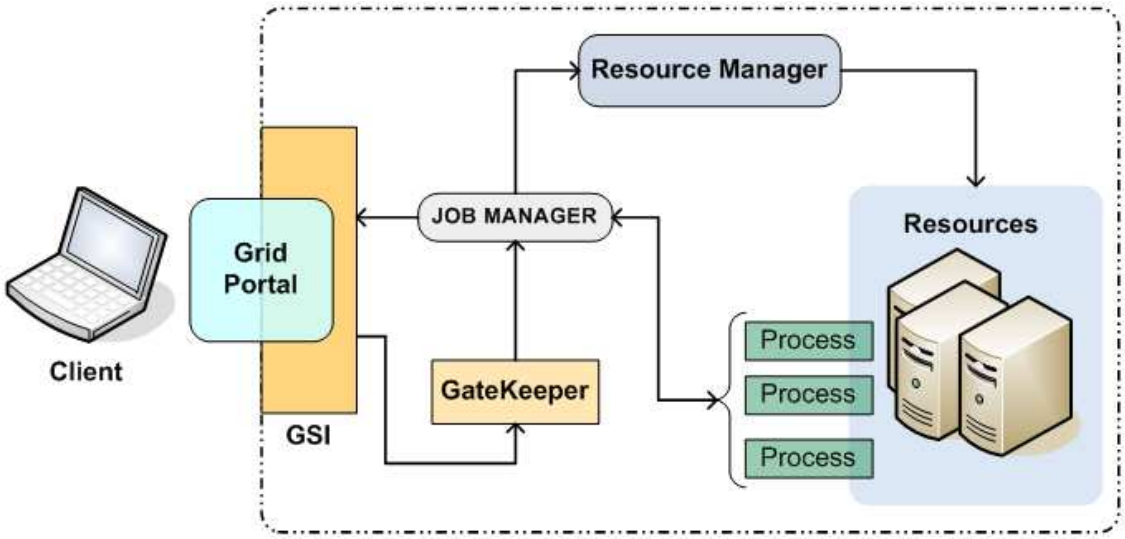

Figure 4–9: Job submission architecture

In order to avoid a crash due to an overload of simultaneous tasks, remote job execution is introduced. The server can submit a simple job or a multi-job to the grid testbed, instead the routine performance of simple local jobs only. Job submission is supported by Globus through GRAM. The Grid Resource Allocation Manager provides services for launching a job on a particular resource, check its status, and retrieves its results when it is complete. Additionally, PBS (Portable Batch System) is used as a job scheduler. Figure 4–9 shows the job submission architecture:

CHAPTER 5
RESULTS AND ANALYSIS

This chapter presents the results obtained after experimentation. Initially the methodology used for each set of experiments is described. A set of experiments have been carried out with the aim to compare replication and dispersal algorithms and thus determine the advantages and the disadvantages of each one. A comparison between local and remote job execution is also presented.

5.1 Methodology

Redundancy schemes implemented are evaluated in terms of reliability and execution. Initially, a study of performance is presented where the access reliability was the metric selected. Next, a set of data with various sizes is used to accomplish a complete analysis of the effect of data size on execution time. The elapsed time (process + user + system) measurements are used to make a comparison between different techniques used to minimize the execution time. Finally, the advantages and the disadvantages of local and remote job submission are discussed.

5.1.1 Redundancy Schemes Evaluation

For our performance A performance analysis of the redundancy algorithms provides a solid selection criterion to determine what is the suitable scheme to be integrated with the distributed storage system. According to the DCAS system goals, the more relevant results are the ones related to reliability as well as storage resource consumption and execution time.

Study of Reliability

Prior to the experimental results, a reliability study allows establishing the behavior of each algorithm when the nodes fail to occur. We take into consideration the total number of blocks after applying redundancy (TB), the size of each block (BS) and the added redundancy (AR) as parameters and measure the access reliability (R). In each case the storage spent (SS) required to perform redundancy.

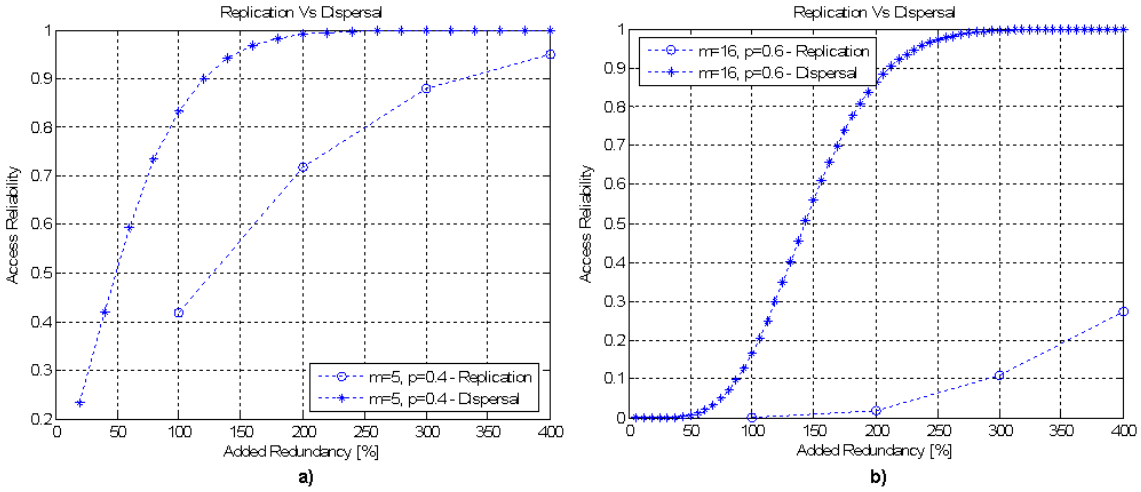

Figure 5–1: Reliability vs Added Redundancy comparison. a) $m = 5$, $p = 0.4$, b) $m = 10$, $p = 0.6$

Information dispersal algorithm shows a better access reliability than the replication algorithm. As a reference point, for an access reliability $R = 0.9$ when the probability of failure is $p = 0.4$, $m = 5$, the added redundancy for IDA is $AR = 120\%$, while in the replication approach the added redundancy must be approximately $AR \approx 300\%$ (Figure 5–1(a)). Note that, for replication algorithm, the AR increment is every 10%, because the redundancy is performed using multiplication with integer numbers.

Figure 5–1(b) shows the behavior of the algorithms when the probability $p = 0.6$ and $m = 16$. The reliability of replication approach is quite deficient if the probability failure increments.

Note that, as shown in Figure 5–1, the reliability for IDA is improved when m is incremented compensating a higher probability of failure. However, the reliability for replication is downgraded if the number of blocks is incremented and is worse still if p is higher. In contrast, a higher number of m involves a even higher number of total blocks (TB) and a reduction in the block size (BS). A small BS can be desirable to obtain weightless blocks to send them over a loaded network. In turn, a higher TB involves a higher number of nodes, if the node-block relationship is $1:1$.

Redundancy is an important feature to be taken into account when radar data must be manipulated, because the size of this data is usually large. Therefore, a proper redundancy must be selected to avoid storage overhead.

Data Size Vs Elapsed Time

Considering the calculations of data rate described before (section 4.3), next testing is achieved with a wide data size range that includes values between 1MB and 1000MB. In order to improve elapsed time measurements, a comparison point is established. Suppose that a minimum access reliability of 90 %. If $p = 0.4$, is required to provide data availability in the DCAS network an access reliability $R \geq 0.9$ can be obtained with the following settings:

- **Replication**: if $m = 8$, $r = 5$, therefore, $AR = 400\%$ and $R = 0.921$.
- **IDA**: if $m = 8$, $r = 10$, therefore, $AR = 125\%$ and $R = 0.942$.

Even though the added redundancy is lower for IDA than Replication Algorithm, the elapsed time required to complete dispersal and recovery operations in IDA is significantly higher than Replication approach. Figure 5–2 shows a comparison between dispersal and recovery operations for both algorithms with several data sizes. As is shown in Figure 5–2(b), the replication algorithm is a lot faster than IDA in both replication and recovery operations. Note that, when a file of size 1GB is required to be distributed, IDA takes long about 20 minutes and replication algorithm only takes 3.5 minutes.

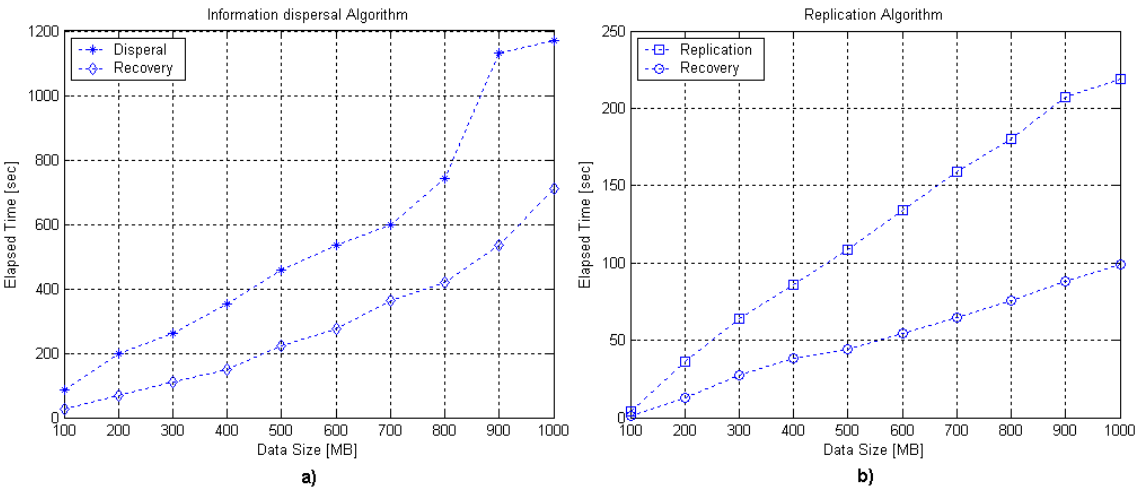

Figure 5–2: Data Size vs. Elapsed Time comparison. a)IDA, b)Replication

Results shown in Figure 5–2 are obtained using the look-up table approach, with log table generation in running time, memory allocation and bit strings of length 16 (labeled in Figure 5–3 as "IDA-OLD"). To reduce the IDA execution time, different techniques were considered previously in section 3.4. Additionally, routines to perform the operations over GF were modified. All modifications are related with the memory management and programming. For instance, the following function:

```
void testfunction(word* a, word* b, word* c ){
c = gfmultiply(a,b);
}
```

shows a lower computation time than the next function

```
word testfunction (word a, word b){
word c;
c = gfmultiply(a,b);
return c;
}
```

Figure 5–3 shows the results obtained with each technique, in the dispersal operation. The number 8 and 16 in the legend represents the bit string used.

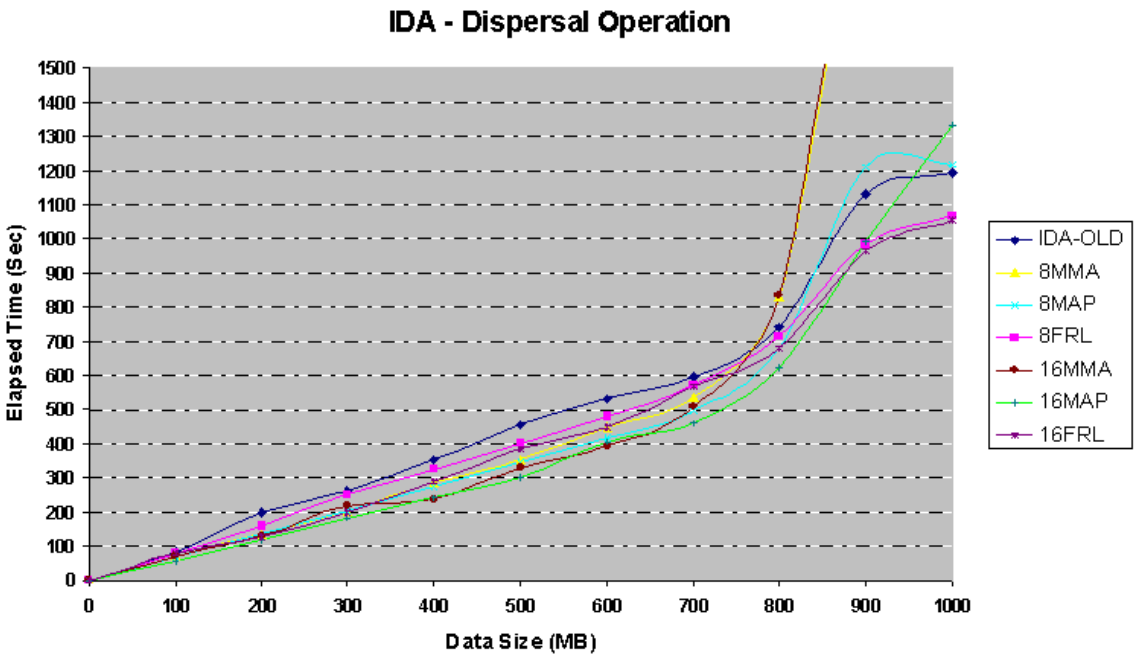

Figure 5–3: Techniques to improve IDA execution time

Note that, 8FRL is the enhanced version of IDA-OLD. They show a similar behavior with data size $< 800MB$, but the main difference occurs at $900MB$ and $1000MB$ where the elapsed time is reduced $\approx 2min$ and $\approx 2.5min$ respectively. Techniques using memory allocation present a good response, even better than FRL techniques. However, MMA produces an unacceptable time of computation. Memory mapping technique allows processing data size larger than $800MB$ without time reduction. MAP presents the better performance than MMA and FRL, whit data sizes smaller than $\approx 500MB$. The most representative reduction was obtained with the FRL technique. However, this reduction is not enough and some modifications are required.

Techniques to improve the IDA execution time can be enhanced using pre-computed look-up tables instead of dynamically generated tables. Figure 5–4 shows the results after performing such modifications.

The behavior of the different techniques is similar to the results shown in Figure 5–4. However, a relevant reduction of time is obtained with each technique. Again,

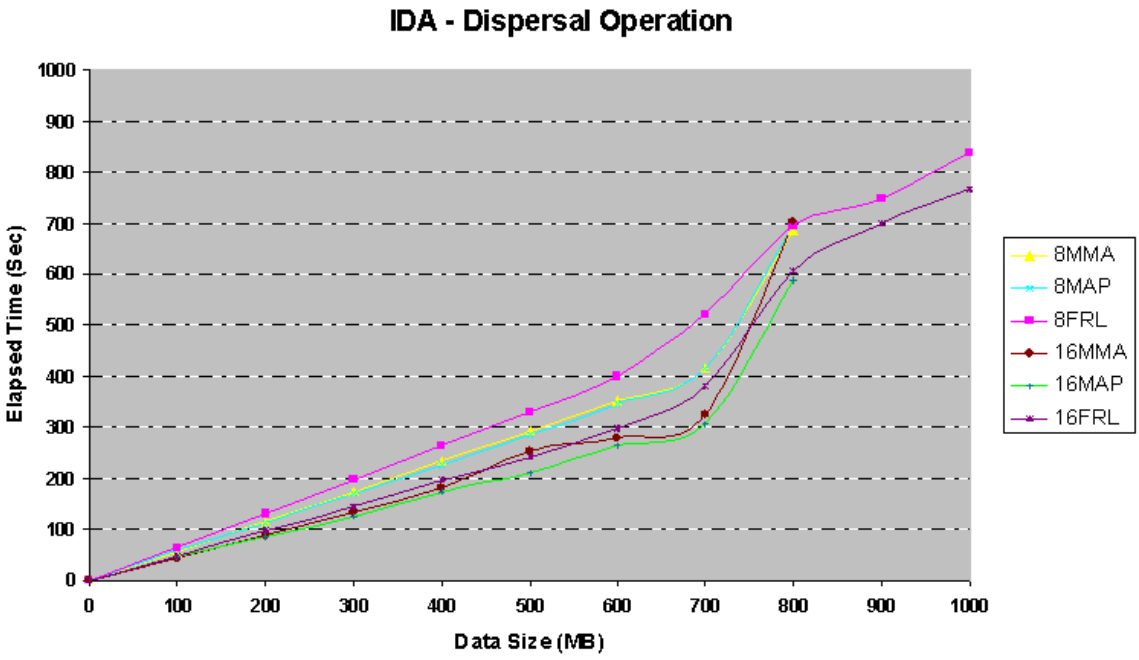

Figure 5–4: IDA with pre-computed tables

the most significant reduction was obtained with the FRL technique when the bit string is 2 bytes of length. For example, the introduction of pre-computed tables allows a reduction of ≈ 7 minutes in the dispersal operation for a $1000MB$ file.

An additional set of IDA experiments was performed for data sizes between $10MB$ and $100MB$, as well as data sizes lower than $10MB$. Figures 5–5 and 5–6 show the results for those tests. As shown in Figures 5–5 and 5–6 MAP and MMA techniques have a very similar behavior. This occurs because, with small files, copy or mapping operations use the same system resources. In general the 16 bits FRL offers better results than the other techniques, when data files are $> 500MB$. The MMA technique can be used for files $< 500MB$ instead of MAP, because MMA does not affect the code portability and it can be implemented in different platforms.

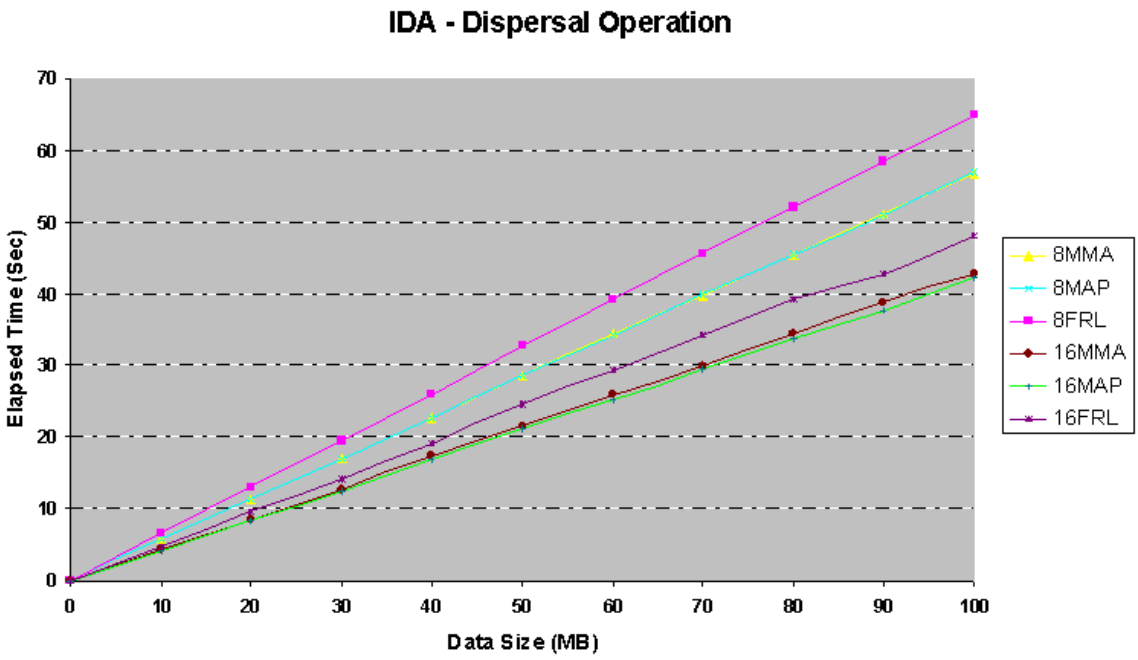

Figure 5–5: IDA, dispersal mode - Up to 100 MB

Finally, we present a comparison between the implemented IDA and the Reed-Solomon[1] (RS) algorithm. RS has similar properties as IDA and it requires operations over GF too. As shown in Figure 5–7 Reed-Solomon presents a better response in terms of execution time with files $< 800MB$. However, RS algorithm does not support large data files.

5.2 Local Job Vs Remote Job

On of the most important concepts On of the most important concepts related with the implemented services is the execution management. Process involvement in the current grid system implementation can be executed as local or remote jobs. Advantages and disadvantages of each one are discussed in this section. First, we consider a single job; this is for example a conversion from NC file to JPEG or to ACII file. Server could execute this process locally but it can be executed by another

[1] Plank J., "Reed-Solomon", http://www.cs.utk.edu/ plank/plank/gflib/

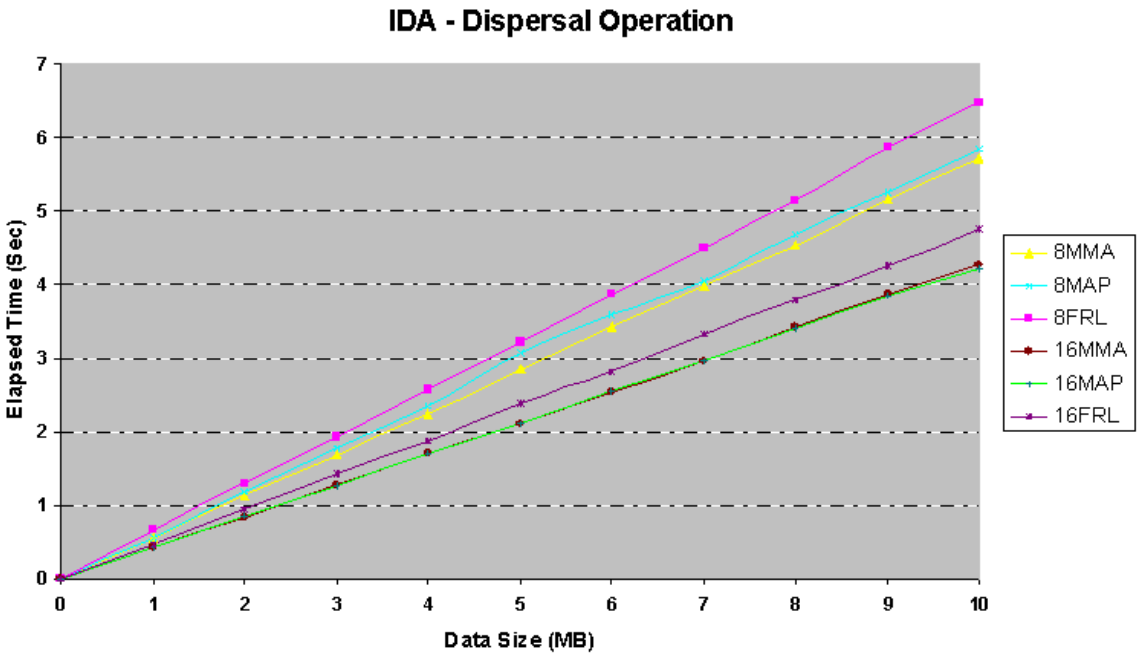

Figure 5–6: IDA, dispersal mode - Up to 10 MB

node in the grid. Figure 5–8 shows the resources used when NCtoJPG process is executed in the server and when it is executed in the grid. Figure 5–8(a) shows that the local execution is faster (3 times more) than remote execution if we are processing a single file. The multi-job test was performed with 6 files which imply 6 different jobs. Execution over the grid of the multi-job was achieved using GRAM, meanwhile in the local execution, each job ran sequentially. Upon executing the experiments, only four nodes were available, including the STB server. The total time used in a local submission is half the total time used in a distributed execution. In terms of process time, a multi-job executed in the server shows a better response than a multi-job executed over the grid. However, a local-job involves more resources usage than remote-jobs. Figure 5–8(b) shows the percentage of the CPU usage by the current job. Similar results are obtained for NCtoASCII process. Figure 5–9(a) shows the elapsed time and figure 5–9(b) shows the percentage of the CPU usage, for local and remote submission. Single job and multi-job are considered in the comparison.

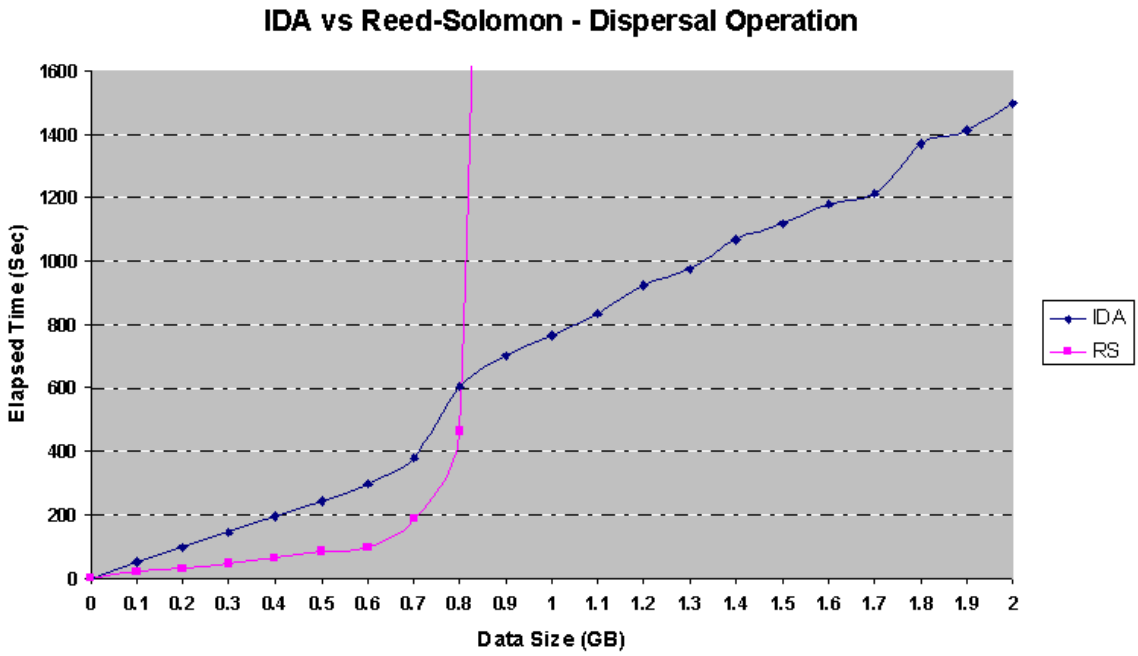

Figure 5–7: IDA vs Reed-Solomon comparison

A more general experiment was performed, using 4 nodes from the grid and a set of files from 1 to 10. The results show that the execution over the grid (for multiple files) takes approximately 2 times more than an execution on the server. This approximation remains true, even if when the amount of files to be processed is increased.

For example, in Figure 5–10(a) the time elapsed for five (5) files processed over the grid is ≈ two (2) times slower than a single job executed on the server. The process selected was NCtoJPG. Further studies show that a similar result is obtained when the required files are equal 8, 9, or 10. However, the CPU consumption (Figure 5–10(b)) is very quite high (≈ 97%) when a local job is executed, and is close to 1%, when a remote job is performed. In an attempt to reduce of the elapsed time for the multijob on the grid, limiting factors must be taken into account:

1. The total execution time is limited by the number of the nodes available in the grid.

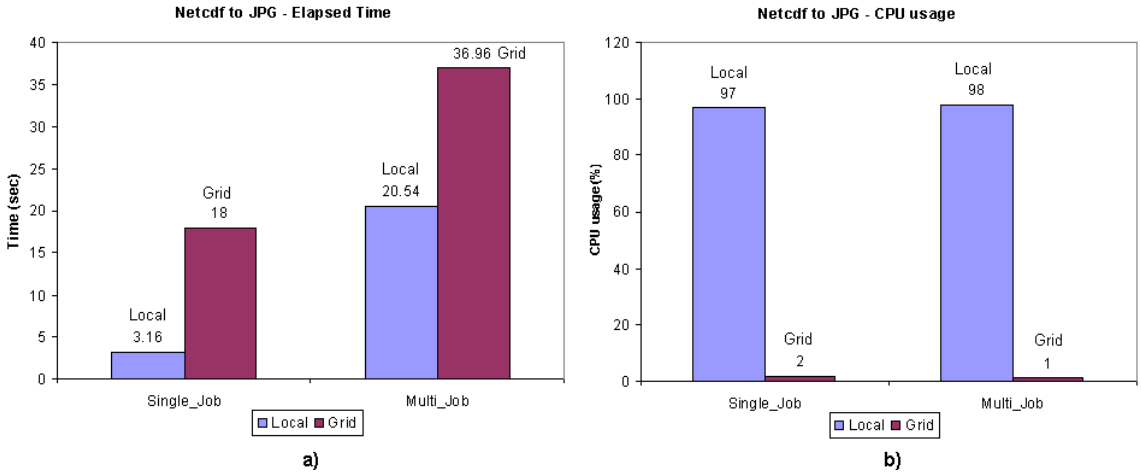

Figure 5–8: Resources used for NCtoJPG

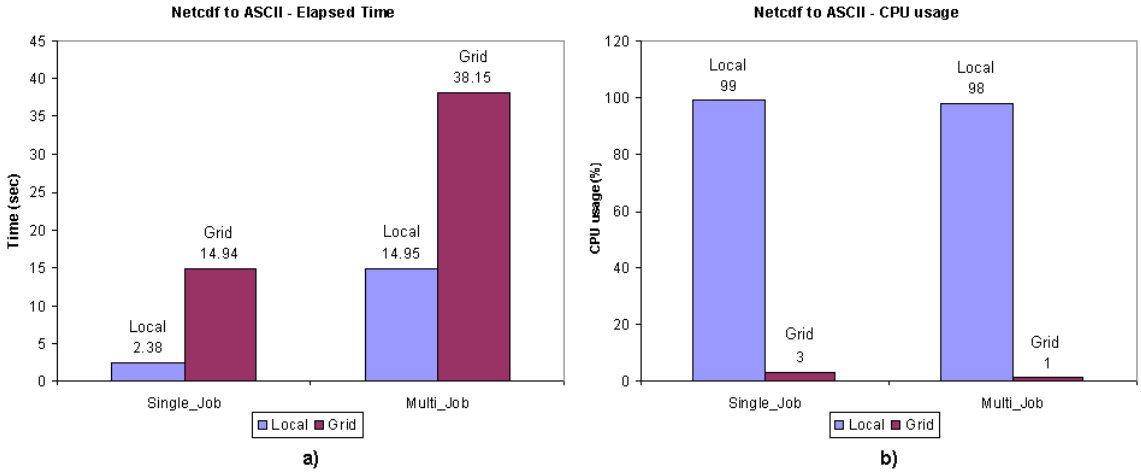

Figure 5–9: Resources used for NCtoASCII

2. The stage-in and stage-out procedures required to perform multijobs on the grid, due to the fact that, in these procedures, the input file is sent from the STB server to the node selected by the scheduler, and the output file is sent back from the node to the server. This communication time cannot be modified and depends of the amount of traffic and the load of the network.

In general, a local job can be submitted when the designated process involves a single execution (i.e. a single file). However, running the process on the grid is

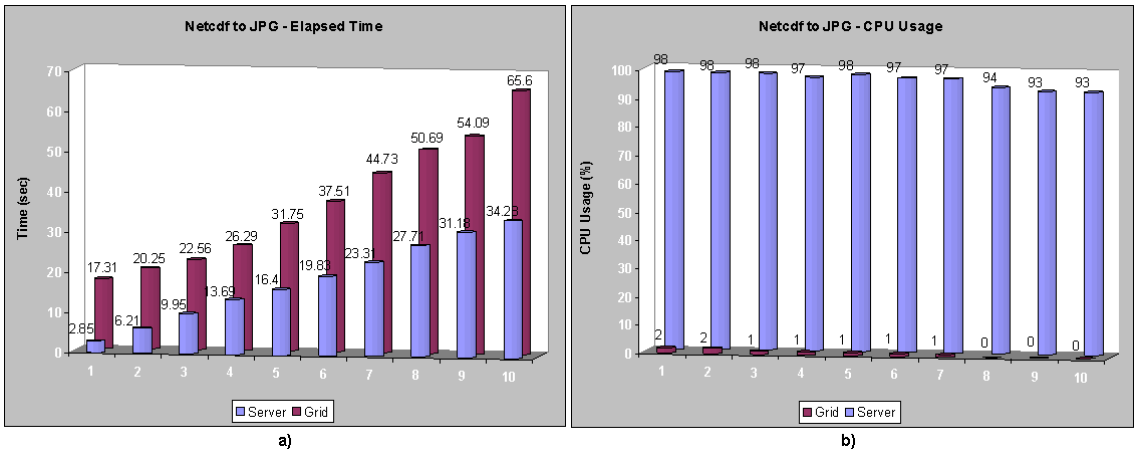

Figure 5–10: Resources used for NCtoJPG process. a) Elapsed Time, b) Percentage of CPU usage.

quite useful when the submitted task requires multiple or repetitive executions (i.e. processing of a data set) where a lower CPU usage is required.

CHAPTER 6
CONCLUSION AND FUTURE WORKS

In this thesis we presented the implementation of a grid-service based system that enables the integration of weather radar network and grid computing technologies. As an important requirement of this integration, a distributed storage system was developed using redundancy algorithm to improve data management. A grid portal to provide a simplified interface to developed applications and grid resources was implemented as well. Finally, services for end-users were designed in order to allow data manipulation.

6.1 Conclusions

The conclusions of this thesis can be summarized as follows:

- Implementations of two redundancy schemes to perform radar data management were presented. The reliability was the metric selected since it is an important parameter in the DCAS systems. At this stage, information dispersal algorithm shows a better data reliability than replication algorithm with less storage spending.
- Several techniques to enhance IDA execution time were discussed. Algorithms using file-read loop present an important reduction of the response times without affect to the portability. Techniques memory-based also show a time reduction, but with portability constraints.
- Radar system integration with grid computing technologies has been discussed as well. Experimental results demonstrate the feasibility of such interaction, when

independent and non grid based applications can be integrated to the grid infrastructure with minimum requirements.

- Tests over the distributed storage system included data exchange between server and grid-testbed, using IDA as redundancy scheme, and GridFTP as transport protocol with GSI support. Integrity of the radar data was preserved successfully.
- The grid-service based provides basic services for data manipulation. Processes for these services can be locally executed or executed over the grid. Experimental results demonstrate that remote or distributed execution; represent a suitable alternative to multi-job submission. In addition, it was shown that local multiple jobs submission implies a large resource usage.

6.2 Future Work

In order to achieve the main goals of the CASA research center, a revolutionary engineering prototype has been proposed, which involves weather-sensing networks operating collaboratively within a dynamic IT infrastructure based on distributed and adaptive computational resources. The grid-service based system presented in this thesis, offers a fundamental piece of the entire IT infrastructure required by the DCAS project. Additionally, this system allows not only integration of the grid resources with weather networks, but also it will become the window to show all the CASA staff efforts to modify the current weather sensing systems. Some modifications to the structural parts from the developed system could be relevant for further requirements; they can be summarized as follows:

- DCAS approach establishes that the end-users may interact with the radars network where and when they require it. In order to achieve this goal, a faster response time from the server to users' requests is required. Thus, one of the main research topics of the current project was the reduction time of the redundancy scheme algorithm, used in the distributed storage system. In section 5.2 several techniques

were proposed to reduce the IDA execution time, and the results show that it is fast enough to establish a good response time. However a similar algorithm (i.e Reed-Solomon) shows a better behavior in presence of files $< 800MB$. Therefore, a second redundancy scheme algorithm could be implemented into the storage system, in such a way that the grid-service base system can dynamically select the appropriate algorithm, in accord of the data size.

- One of the most important features of the grid-service based system developed is the manipulation of the radars data. Thus, either downloading or data processing will be available to end-users in order to extract meteorological information. Since weather algorithms can generate erroneous results when they process corrupted data, the integrity of such data must be maintained throughout any operation, such as transfer, storage, and retrieval. Preliminary tests show that the raw-data was preserved successfully with these operations; however, integrity verification is not automatically checked in the current deployment before recovery, so an appropriated technique to achieve this task is proposed as a future work.
- DCAS prototype operates over an information technology infrastructure, which involves distributed computation and grid computing technologies. Grid technologies enable the dynamic creation and destruction of services, through the Open Grid Services Architecture. It was shown that redundancy schemes can be used to improve reliability in distributed storage systems. Additionally the algorithm developed to perform the data distribution and retrieval can be extended to any kind of file more than data from radars. This algorithm is completely configurable and portable; so, its implementation as grid service can be useful to allow data management in current or future projects.

APPENDICES

APPENDIX A
INFORMATION DISPERSAL ALGORITHM

A.1 Galois Field Arithmetic Library

```
/* File: gf16lib.h Description: This library contains the
implementation of arithmetic operations over a Galois Field (2^16).
The functions available are:
    gf_single_multiply: Multiplication of two numbers in a GF(2^16), type int
    gf_single_multiplyw: Multiplication of two numbers in a GF(2^16), type word
    gf_single_divide: Division procedure in a GF(2^16)
    gf_make_vandermonde: Creation of the Vandermonde's Matrix (Dispersal matrix)
    gf_invert_matrix: Performs a matrix inversion, over a GF(2^16)
    get_alfam: Obtains the new vectors for the file dispersion using the dispersal matrix
    get_alfas: Obtains the original vector for file retrieval using the recovery matrix
    matsel: Creates the recovery matrix from the rows which correspond to each block found
These routines use the look-up table approach to improve the
computational time. Author:
        Diego M. Arias
        University of Puerto Rico at Mayaguez
        diego.arias@ece.uprm.edu
*/

#ifndef GF16LIB_H #define GF16LIB_H #include "luptable32.h" //Call
for the Look-up tables #define prim_poly_16 0210013 //Octal Notation
for X^16 + X^12 + X^3 + X + 1

typedef unsigned short word; size_t W_SIZE = sizeof(word); typedef
unsigned char unit; static int Modar_nwm1 = 65535;

int gf_single_multiply(int xxx, int yyy) {
  unsigned int sum_j;
  word zzz;
  if (xxx == 0 || yyy == 0) {
    zzz = 0;
  } else {
    sum_j = (int) (GFlog16[xxx] + (int) GFlog16[yyy]);
    if (sum_j >= Modar_nwm1) sum_j -= Modar_nwm1;
    zzz = GFantilog16[sum_j];
  }
  return zzz;
}

word gf_single_multiplyw(word xxx, word yyy) {
  unsigned int sum_j;
  word zzz;
  if (xxx == 0 || yyy == 0) {
    zzz = 0;
  } else {
    sum_j = (int) (GFlog16[xxx] + (int) GFlog16[yyy]);
    if (sum_j >= Modar_nwm1) sum_j -= Modar_nwm1;
    zzz = GFantilog16[sum_j];
  }
  return zzz;
```

```
}

int gf_single_divide(int a, int b) {
  int sum_j;
  if (b == 0) return -1;
  if (a == 0) return 0;
  sum_j = GFlog16[a] - GFlog16[b];
  if (sum_j < 0) sum_j += Modar_nwm1;
  return (int) GFantilog16[sum_j];
}

/* gf_make_vandermonde function was originally written by James S.
Plank plank@cs.utk.edu http://www.cs.utk.edu/~plank
*/
int *gf_make_vandermonde(int rows, int cols) {
  int *vdm, i, j, k;

  if (rows >= Modar_nwm1 || cols >= Modar_nwm1) {
    fprintf(stderr, "Error: gf_make_vandermonde: %d + %d >= %d\n",
       rows, cols, Modar_nwm1);
    exit(1);
  }
  vdm = (int *) malloc(sizeof(int) * rows * cols);
  if (vdm == NULL) {
    perror("Malloc: Vandermonde matrix");
    exit(1);
  }
  for (i = 0; i < rows; i++) {
    k = 1;
    for (j = 0; j < cols; j++) {
      vdm[i*cols+j] = k;
      k = gf_single_multiply(k, i);
    }
  }
  return vdm;
}

/* gf_invert_matrix function was originally written by James S.
Plank plank@cs.utk.edu http://www.cs.utk.edu/~plank
*/
int *gf_invert_matrix(int *mat, int rows) {
  int *inv;
  int *copy;
  int cols, i, j, k, x, rs2;
  int row_start, tmp, inverse;
  cols = rows;
  inv = (int *) malloc(sizeof(int)*rows*cols);
  if (inv == NULL) { perror("gf_invert_matrix - inv"); exit(1); }
  copy = (int *) malloc(sizeof(int)*rows*cols);
  if (copy == NULL) { perror("gf_invert_matrix - copy"); exit(1); }
  k = 0;
  for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
      inv[k] = (i == j) ? 1 : 0;
      copy[k] = mat[k];
      k++;
    }
  }
  /* First -- convert into upper triangular */
  for (i = 0; i < cols; i++) {
    row_start = cols*i;
    /* Swap rows if we ave a zero i,i element.  If we can't swap, then the
       matrix was not invertible */
    if (copy[row_start+i] == 0) {
      for (j = i+1; j < rows && copy[cols*j+i] == 0; j++) ;
      if (j == rows) {
        fprintf(stderr, "gf_invert_matrix: Matrix not invertible!!\n");
        exit(1);
```

```
      }
      rs2 = j*cols;
      for (k = 0; k < cols; k++) {
        tmp = copy[row_start+k];
        copy[row_start+k] = copy[rs2+k];
        copy[rs2+k] = tmp;
        tmp = inv[row_start+k];
        inv[row_start+k] = inv[rs2+k];
        inv[rs2+k] = tmp;
      }
    }
    /* Multiply the row by 1/element i,i */
    tmp = copy[row_start+i];
    if (tmp != 1) {
      inverse = gf_single_divide(1, tmp);
      for (j = 0; j < cols; j++) {
        copy[row_start+j] = gf_single_multiply(copy[row_start+j], inverse);
        inv[row_start+j] = gf_single_multiply(inv[row_start+j], inverse);
      }
    }
    /* Now for each j>i, add A_ji*Ai to Aj */
    k = row_start+i;
    for (j = i+1; j != cols; j++) {
      k += cols;
      if (copy[k] != 0) {
        if (copy[k] == 1) {
          rs2 = cols*j;
          for (x = 0; x < cols; x++) {
            copy[rs2+x] ^= copy[row_start+x];
            inv[rs2+x] ^= inv[row_start+x];
          }
        } else {
          tmp = copy[k];
          rs2 = cols*j;
          for (x = 0; x < cols; x++) {
            copy[rs2+x] ^= gf_single_multiply(tmp, copy[row_start+x]);
            inv[rs2+x] ^= gf_single_multiply(tmp, inv[row_start+x]);
          }
        }
      }
    }
  }
  /* Now the matrix is upper triangular.  Start at the top and multiply down */
  for (i = rows-1; i >= 0; i--) {
    row_start = i*cols;
    for (j = 0; j < i; j++) {
      rs2 = j*cols;
      if (copy[rs2+i] != 0) {
        tmp = copy[rs2+i];
        copy[rs2+i] = 0;
        for (k = 0; k < cols; k++) {
          inv[rs2+k] ^= gf_single_multiply(tmp, inv[row_start+k]);
        }
      }
    }
  }
  free(copy);
  return inv;
}

void get_alfas(int *a, word *b, word *prod, int rows, int cols) {
 int i, j;
 for(i=0;i<rows;i++)
     prod[i]=0;

 for(i=0; i<rows; i++)
      for(j=0; j<cols; j++)
           prod[i] ^= gf_single_multiplyw(a[i*cols+j], b[j]);
```

```
}

void get_alfam(int *a, word *b, word alfa[0], int rows, int cols) {
 int j;
 alfa[0]&=0x0;
 for(j=0; j<cols; j++)
      alfa[0]^=gf_single_multiplyw(a[rows*cols+j], b[j]);
}

int *matsel(int *mat, int *list, int cols) {
 int i,j;
 int *res;
 res = (int *) calloc(cols*cols,sizeof(int));
 for(i=0; i<cols; i++)
     {
      for(j=0; j<cols; j++)
          res[i*cols+j]=mat[list[i]*cols+j];
      }
 return res;
}
#endif
```

A.2 Information Dispersal Algorithm (IDA) Implementation

```
/* File: dispersal16.h Description:
            Procedures to implement the Information Dispersal Algorithm.
            This version is based on a GF(2^16).
The functions available are:
    DisperseFile: Disperses a file F into n chunks, using the "fread" function.
                  No memory allocation required.
    DisperseFile: Disperses a file F into n chunks, using memory allocation.
    RecoveryFile: Peforms the original file F retrieval.
Author:
        Diego M. Arias
        University of Puerto Rico at Mayaguez
        diego.arias@ece.uprm.edu
*/

#ifndef DISPERSAL16_H #define DISPERSAL16_H #include <stdio.h>
#include <stdlib.h> #include <sys/stat.h> #include <assert.h>
#include <time.h> #include <string.h> #include "gf16lib.h"

void DisperseFile(int n, int m, char *filename) {
  int *vdm,rows,cols,sz,padding;
  long i,k,blocksize,orig_size;
  char *buf_file, *log_file;
  struct stat buf;
  FILE *f,*flog,*fin;
  word *alfas,*bcol;
  word *ftemp;
  rows = n;
  cols = m;
  if (stat(filename, &buf) != 0) {
    perror(filename);
    exit(1);
  }
  sz = buf.st_size;
  printf("File Size: %d bytes -- Blocksize: %d bytes -- Residue: %d\n",sz,sz/m,sz%m);
  orig_size = buf.st_size;
  if (sz % (m*W_SIZE) != 0) {
    sz += m*W_SIZE - (sz % (m*W_SIZE));
  }
  blocksize = sz/m;
  padding=m*blocksize-(orig_size);
  printf("New Size: %ld bytes -- New Blocksize: %ld bytes -- Padding: %d bytes\n",m*blocksize,blocksize,padding);
```

```
   vdm=gf_make_vandermonde(rows, cols);
  bcol=(word *) calloc(cols,W_SIZE);
  alfas=(word *) calloc(1,W_SIZE);
  printf("\nReading data\n");
  f = fopen(filename, "rb");
  if (f == NULL) { perror(filename); exit(EXIT_FAILURE);}
  printf("\nData ready to process\n");
  buf_file = (char *) malloc(sizeof(char)*(strlen(filename)+50));
  if (buf_file == NULL) { perror("malloc - buf_file"); exit(EXIT_FAILURE); }
  word *drow=(word *) malloc(W_SIZE);
  if (drow == NULL) { perror("malloc - drow"); exit(EXIT_FAILURE); }
  ftemp = (word*) calloc(blocksize/2,W_SIZE);  //Array filled with zeros
  if (ftemp == NULL) { perror("malloc - ftemp"); exit(EXIT_FAILURE); }
  log_file = (char *) malloc(sizeof(char)*(strlen(buf_file)+50));
  if (log_file == NULL) { perror("malloc - log_file"); exit(EXIT_FAILURE); }
  sprintf(log_file, "%s-log.txt", filename);
  flog = fopen(log_file, "w");
  if (flog == NULL) { perror(log_file); exit(EXIT_FAILURE); }
  fprintf(flog, "%ld\n", orig_size);
  fprintf(flog, "%ld\n", blocksize);
  fprintf(flog, "%d\n", padding);
  fprintf(flog, "%d\n", n);
  fprintf(flog, "%d\n", m);
  printf("\nProcessing Data\n");
  /*** this procedure do not need memory allocation ***/
  for(k=0; k<rows; k++)
      {
       fseek(f, 0L, SEEK_SET);
       sprintf(buf_file, "%s-%04ld.ida", filename, k);
       fprintf(flog, "%s\n", buf_file);
       fin = fopen(buf_file, "wb");
       if (fin == NULL) { perror("Can not create block file"); exit(EXIT_FAILURE);}
       drow[0]=k;
       fwrite(&drow[0], W_SIZE, 1, fin);
       for (i=0; i<(blocksize)/2; i++)
            {
             fread(bcol,W_SIZE,cols,f);
        get_alfam(vdm,bcol,&ftemp[i],k,cols);
            }
       fwrite(ftemp, W_SIZE, blocksize/2, fin);
       fclose(fin);
      }
  fclose(f);
  free(vdm);
  free(bcol);
  free(alfas);
  free(ftemp);
  fclose(flog);
  free(buf_file);
  free(log_file);
  free(drow);
}

void DisperseFile2(int n, int m, char *filename) {
  int *vdm,rows,cols,sz,padding;
  long i,j,k, blocksize, orig_size;
  char *buf_file, *log_file;
  struct stat buf;
  FILE *f,*flog,*fin;
  word *alfas,*bcol;
  rows = n;
  cols = m;
  if (stat(filename, &buf) != 0) {
    perror(filename);
    exit(1);
  }
  sz = buf.st_size;
  printf("File Size: %d bytes -- Blocksize: %d bytes -- Residue: %d\n",sz,sz/m,sz%m);
```

```
  orig_size = buf.st_size;
  if (sz % (m*W_SIZE) != 0) {
    sz += m*W_SIZE - (sz % (m*W_SIZE));
  }
  blocksize = sz/m;
  padding=m*blocksize-(orig_size);
  printf("New Size: %ld bytes -- New Blocksize: %ld bytes -- Padding: %d bytes\n",m*blocksize,blocksize,padding);
 vdm=gf_make_vandermonde(rows, cols);
 bcol=(word *) calloc(cols,W_SIZE);
 alfas=(word *) calloc(rows,W_SIZE);
 printf("\nReading data\n");
 word *mydata = (word*) calloc(cols*blocksize/2,W_SIZE);  //Array filled with zeros
 if (mydata == NULL) { perror("malloc - mydata"); exit(EXIT_FAILURE); }
 f = fopen(filename, "rb");
 if (f == NULL) { perror(filename); exit(EXIT_FAILURE);}
 fread(mydata,1,orig_size,f); //Read all file
 fclose(f);
 printf("\nData ready to process\n");
 buf_file = (char *) malloc(sizeof(char)*(strlen(filename)+50));
 if (buf_file == NULL) { perror("malloc - buf_file"); exit(EXIT_FAILURE); }
 word *drow=(word *) malloc(W_SIZE);
 if (drow == NULL) { perror("malloc - drow"); exit(EXIT_FAILURE); }
 printf("\nProcessing Data\n");
 log_file = (char *) malloc(sizeof(char)*(strlen(buf_file)+50));
 if (log_file == NULL) { perror("malloc - log_file"); exit(EXIT_FAILURE); }
 sprintf(log_file, "%s-log.txt", filename);
 flog = fopen(log_file, "w");
 if (flog == NULL) { perror(log_file); exit(EXIT_FAILURE); }
 fprintf(flog, "%ld\n", orig_size);
 fprintf(flog, "%ld\n", blocksize);
 fprintf(flog, "%d\n", padding);
 fprintf(flog, "%d\n", n);
 fprintf(flog, "%d\n", m);
  word *ftemp = (word*) calloc(blocksize/2,W_SIZE);  //Array filled with zeros
  if (ftemp == NULL) { perror("malloc - ftemp"); exit(EXIT_FAILURE); }

/*this procedure require memory allocation*/
 for(k=0; k<rows; k++)
     {
      sprintf(buf_file, "%s-%04ld.ida", filename, k);
      fprintf(flog, "%s\n", buf_file);
      fin = fopen(buf_file, "wb");
      if (fin == NULL) { perror("Can not create block file"); exit(EXIT_FAILURE);}
      drow[0]=k;
      fwrite(&drow[0], W_SIZE, 1, fin);
      for (i=0; i<(blocksize)/2; i++)
           {
            for(j=0; j<cols; j++)
                {
                   bcol[j]=mydata[i*cols+j];
                }
        get_alfam(vdm,bcol,&ftemp[i],k,cols);
           }
      fwrite(ftemp, W_SIZE, blocksize/2, fin);
      fclose(fin);
     }
printf ("Iterations => %ld\n",k*i);
 free(vdm);
 free(mydata);
 free(bcol);
 free(alfas);
 free(ftemp);
 fclose(flog);
 free(buf_file);
 free(log_file);
 free(drow);
}
```

```
void RecoveryFile(char *filename) {
 int i, j, *vdm, *inv, fav, *list, *newvdm;
 int rows, cols, blocksize, orig_size, padding;
 char *log_file, *line, *out_file;
 char (*nfiles)[128];
 FILE *f, *flog, *fout;
 word *myid, *alfas, *bcol;
 const char *suffix = "-log.txt";
 size_t log_file_len = strlen(filename) + strlen(suffix) + 1;
 log_file = malloc(log_file_len);
 if (log_file == NULL) { perror("malloc - log_file"); exit(EXIT_FAILURE); }
 strcpy(log_file, filename);
 strcat(log_file, suffix);
 line = (char *) malloc(sizeof(char)*(128));
 if (line == NULL) { perror("malloc - line"); exit(1); }
 fav=0;
 myid=(word *) malloc(W_SIZE);
 flog = fopen(log_file, "r");
 if (flog == NULL) { perror(log_file); exit(1); }
 fscanf(flog, "%d\n", &orig_size);
 fscanf(flog, "%d\n", &blocksize);
 fscanf(flog, "%d\n", &padding);
 fscanf(flog, "%d\n", &rows);
 fscanf(flog, "%d\n", &cols);
 nfiles=calloc(rows,sizeof(*nfiles));
 if (nfiles == NULL) { perror("malloc - nfiles"); exit(EXIT_FAILURE); }
 for(i=0;i<rows;i++)
     sprintf(nfiles[i]," ");
 list=(int *) malloc(rows*sizeof(int));
 if (list == NULL) { perror("malloc - list"); exit(EXIT_FAILURE); }
 for(j=0; j<rows; j++)
     list[j]=-1;
 for(i=0; i<rows; i++)
     {
       fscanf(flog,"%s\n",line);
       f = fopen(line, "rb");
       if (f == NULL) { perror(" is unavailable"); continue;}
       else
           {
            fread(myid,W_SIZE,1,f);
            list[fav]=i;
            sprintf(nfiles[fav],line);
            fav++;
           }
      fclose(f);
     }
 fclose(flog);
 printf("Files: %d\n",fav);
 if(fav<cols)
    { perror("Number of files insufficient"); exit(EXIT_FAILURE); }
 printf("O_Size: %d - Blk_Size: %d - Padd: %d - Rows: %d - Cols: %d\n",orig_size, blocksize, padding, rows, cols);
 word *mydata = (word*) calloc(cols*blocksize/2,2);  //Array filled with zeros
 if (mydata == NULL) { perror("malloc - mydata"); exit(EXIT_FAILURE); }
 printf("\nCollecting Data: %d\n",cols);

/*Begin - Collect mydata*/
 for(i=0; i<cols; i++)
     {
       f = fopen(nfiles[i], "rb");
       if (f == NULL) { perror("File unavailable");exit(EXIT_FAILURE);}
       else
           {
            fread(myid,W_SIZE,1,f);
            fread(mydata + i * (blocksize/2), sizeof (word), blocksize/2, f);
           }
      fclose(f);
     }
/*End - Collect mydata*/
```

```
 vdm=gf_make_vandermonde(rows, cols);
 newvdm=matsel(vdm, list, cols);
 inv=gf_invert_matrix(newvdm, cols);
 free(vdm);
 bcol=(word *) calloc(cols,W_SIZE);
 alfas=(word *) calloc(cols,W_SIZE);
 out_file = (char *) malloc(sizeof(char)*(strlen(filename)+30));
 if (out_file == NULL) { perror("malloc - out_file"); exit(EXIT_FAILURE); }
 sprintf(out_file, "%s.rida", filename);
 fout = fopen(out_file, "wb");
 if (fout == NULL) { perror(out_file); exit(EXIT_FAILURE); }
 printf("\nProcessing Data\n");
 for(i=0; i<(blocksize/2)-1; i++)
    {
     for (j=0; j<cols; j++)
         {
          alfas[j]=0;
          bcol[j]=mydata[i+(blocksize*j)/2];
         }
    get_alfas(inv, bcol, alfas, cols, cols);
    fwrite(alfas, W_SIZE, cols, fout);
    }

 /*To remove padding if it exists*/
 i=(blocksize/2)-1;
 for (j=0; j<cols; j++)
      {
       alfas[j]=0;
       bcol[j]=mydata[i+(blocksize*j)/2];
      }
 get_alfas(inv, bcol, alfas, cols, cols);
 fwrite(alfas, 1, (cols*2)-padding, fout);
 free(mydata);
 free(myid);
 free(bcol);
 free(alfas);
 free(inv);
 free(fout);
 free(nfiles);
}
#endif
```

A.3 User Interface for IDA Program

```
/* File: idacodec16.h Description:
            User interface for the Information Dispersal Algorithm implementation.
            This version is based on a GF(2^16).
Author:
        Diego M. Arias
        University of Puerto Rico at Mayaguez
        diego.arias@ece.uprm.edu
*/
#include <stdlib.h> #include "dispersal16.h"

void message() {
 printf("*************************************************************\n");
 printf("* Error: Incorrect Number of Parameters                     *\n");
 printf("* Dispersal: dispersal -d n m file_input  -->To Disperse *\n");
 printf("* Recovery: dispersal -r file_input  --> To Recovery     *\n");
 printf("* Written by: Diego M. Arias -- 2005                        *\n");
 printf("*************************************************************\n");
}

int main(int argc, char *argv[]) {
  char option;
```

```
  if (argc < 3 )  //Minimal 3 args. to the Recovery Operation
    option = 'h';
  else if ((argc > 3 && argc < 5) || argc >5) //Minimum 5 args. to the Dispersal Op.
    option = 'b';
  else
    option = argv[1][1];

  switch (option)
     {
      case 'd':
             DisperseFile(atoi(argv[2]), atoi(argv[3]),argv[4]);
             printf("\nDisperse done successfully\n");
       break;
      case 'e':
             DisperseFile2(atoi(argv[2]), atoi(argv[3]),argv[4]);
             printf("\nDisperse done successfully\n");
       break;
       case 'r':
             RecoveryFile(argv[2]);
             printf("\nRecovery done successfully\n");
       break;
       case 'h':
             message();
       break;
       case 'b':
             message();
       break;
       default:
             message();
       break;
     }
 return 0;
}
```

APPENDIX B
DISTRIBUTED STORAGE SYSTEM

B.1 NC Files Generator & Dispersal Operation

```
#!/bin/bash
# File: datemul.sh
# Description: Emulates the data radar files and data rate, performs dispersal operation using IDA
#               and distributes the chunks generated over the grid in a distribution 1:1  using GridFTP
# Functions:
#   gendata: Copy randomly a file from the seeds source to the raw data directory.
#            Performs Dispersal operation using IDA with redundancy = 100
#            Copy each chunk to the available nodes, 1 chunk per node
#            Each NC file is generated and distributed every Tr seconds
# Author:
#         Diego M. Arias
#         University of Puerto Rico at Mayaguez
#         diego.arias@ece.uprm.edu

## raw data path, node1 = Mayaguez Radar
## Data from node1 must be
here fpath="/data/dcas/node1/"

## creates a log file which contains all dispersed files per day
(creates an index)
logfile="${fpath}logs/‘date +%Y%m%d‘.log"

## if the log file does not exist, create it if [ -e $source ]; then
    sleep 0
else
    touch $logfile
fi

## n: number of chunks
## m: minimal number of chunks required to
reconstruct the original file
## Tr seconds
n=4
m=2
Tr=20

## Checking Grid-Proxy-Init
/data/dcas/./checkproxy.sh 2>/data/dcas/stblogs/gprxerr.txt

function gendata {
    ## Generates random number between 0 and 9
    RANGE=9
    FLOOR=0
    number=0   #initialize
    while [ "$number" -le $FLOOR ]
    do
      number=$RANDOM
      let "number %= $RANGE"
    done
```

```
    ## random source: data00 - data02 - data03 ... data09
    source="/stbradar/data0${number}.nc"
    ## new filename in format yy/mm/dd-hh/mm/ss
    datename="`date +%Y%m%d`-`date +%H%M%S`.nc"
    newfilename="/data/dcas/node1/${datename}"
    ## new file from a random source
    if [ -e $source ]; then
         cp $source $newfilename
         cp $source ${fpath}sourcenc/first.nc
    fi
    targetfile="${newfilename}"
    ## List of available nodes
    server1="proc.ece.uprm.edu"
    server2="devzero.ece.uprm.edu"
    server3="pdcgrid-32-01.ece.uprm.edu"
    server4="/data/dcas/node1/" ## local server
    ## Dispersal Operation using IDA
    ${fpath}./dispersal16 -d $n $m $targetfile
    echo "$targetfile" >> $logfile

    ## Distribution over the grid using gsiftp protocol
globus-url-copy -vb file:///${targetfile}-0000.ida gsiftp://${server1}:2811/data/dcas/node1/${datename}-0000.ida;
globus-url-copy -vb file:///${targetfile}-0001.ida gsiftp://${server2}:2811/data/dcas/node1/${datename}-0001.ida;
globus-url-copy -vb file:///${targetfile}-0002.ida gsiftp://${server3}:2811/data/dcas/node1/${datename}-0002.ida;
    ## remove original file and *.ida
    rm ${targetfile}
    rm ${targetfile}-0000.ida
    rm ${targetfile}-0001.ida
    rm ${targetfile}-0002.ida
    rm ${targetfile}-0003.ida
}

while : do
   gendata
   sleep Tr
done
```

B.2 Data Generator Interface

```
#!/bin/bash
##
# File: gendata.sh
# Description:  Start  / Stop datemul.sh
# Author:
#         Diego M. Arias
#         University of Puerto Rico at Mayaguez
#         diego.arias@ece.uprm.edu
##

MINPARAMS=1 EXECUTABLE=datemul.sh

if [ -e $EXECUTABLE ] then
    echo "Data Generator manager"
else
    echo "File: $EXECUTABLE not found!"
    exit 1
fi

if [ "$1" = "start" ] ## Start datemul.sh
then
     echo "Starting Data Generator"
     exec ./$EXECUTABLE &
elif [ "$1" = "stop" ] ## Stop datemul.sh
then
     echo "Stopping Data Generator"
     killall -e $EXECUTABLE
```

```
else
     echo "Command not recognized"
fi
if [ $# -lt "$MINPARAMS" ]
then
    echo "datemul start -- To Start Data Generator"
    echo "datemul stop -- To Stop Data Generator"
    echo
fi
echo "--Written by DMA"
```

B.3 Data Retrieval Using IDA & GridFTP

```
#!/bin/bash
##
# File: updater.sh
# Description: Performs retrieval operation using GridFTP
#              and reconstruct the original file from the found chunks using IDA.
# Functions:
#  checkf:  Performs a search for the chunks that correspond to the requested file(s).
#           Copy found chunks to the raw data directory.
#           Performs Retrieval operation using IDA
#           Copy the reconstructed file to the download directory
# Author:
#         Diego M. Arias
#         University of Puerto Rico at Mayaguez
#         diego.arias@ece.uprm.edu
##

## raw data path, node1
mpath="/data/dcas/node1/"

## List of the requested files
rfile="/data/dcas/node1/downs.out"

## Checking Grid-Proxy-Init
/data/dcas/./checkproxy.sh 2>/data/dcas/stblogs/gprxerr.txt

## List of the available nodes
server[0]="proc.ece.uprm.edu"
server[1]="devzero.ece.uprm.edu"
server[2]="pdcgrid-32-01.ece.uprm.edu"
server[3]="/data/dcas/node1/" #local server

function checkf {
             cat $rfile | while read line
             do
             ind=0
             while [  $ind -lt 3 ]; do
             ## Search for chunks over the grid using gsiftp protocol
             globus-url-copy gsiftp://${server[${ind}]}:2811${line}-000${ind}.ida file:///${mpath}
             let ind=ind+1
             done
             ## Retrieval Operation using IDA
             /data/dcas/node1/./dispersal16 -r "${line}"
             cp "${line}.rida" "/data/dcas/node1/download/${line: -18}"
             ## Removes the chunks in the local node
             rm ${line}.rida
             rm ${line}-0000.ida
             rm ${line}-0001.ida
             rm ${line}-0002.ida
            done
}
checkf
```

APPENDIX C
SERVICES FOR END-USERS

C.1 NetCDF to ASCII Conversion

```
/*
 File: NCtoASCII.java
 Description:
            NC to TXT conversion.
Author:
          Diego M. Arias
          University of Puerto Rico at Mayaguez
          diego.arias@ece.uprm.edu
*/

import java.io.IOException; import ucar.ma2.*; import ucar.nc2.*;
import java.io.*;

public class NCtoASCII {

     static String fileName = "example.nc";

    /**
     * Prints schema (structure) of an existing netCDF file with a
     * specified file name.
     *
     * @param args name of netCDF file to be read.  */
    public static void main(String[] args) {
        if (args.length == 1)
        fileName = args[0];
    String fileNameout=fileName.substring(0,fileName.lastIndexOf("."))+".txt";
   try
   {
     DataOutputStream fout = new DataOutputStream(new FileOutputStream( fileNameout));
     NCdump.print(fileName+" -vall",fout);
     fout.close();
   }
   catch ( IOException iox )
   {
     System.out.println("Problem with TXT");
   }
  }
}
```

C.2 Reflectivity Plots from NetCDF Files

```
/* File: PlotNetcdf.java Description:
            Procedures to generate reflectivity plots from NetCDF (NC) files
The functions available are:
    DisperseFile: Disperses a file F into n chunks, using the "fread" function.
                  No memory allocation required.
    DisperseFile: Disperses a file F into n chunks, using memory allocation.
```

```
    RecoveryFile: Peforms the original file F retrieval.
Author:
        Alexis Perez & Diego M. Arias
        University of Puerto Rico at Mayaguez
        diego.arias@ece.uprm.edu
 */

//package image;
import javax.swing.*; import java.awt.*; import java.awt.geom.*;
import java.net.*; import java.applet.*; import java.awt.image.*;
import java.io.*; import javax.imageio.ImageIO;
//NetDCF
import ucar.ma2.*; import ucar.nc2.*;

import java.lang.Math;

public class PlotNetcdf {
    static String fileName = "test/example.nc";
    static String backimage = "test/mayaguez.jpg";
    /** Creates a new instance of PlotNetcdf */
    public PlotNetcdf() {
    }

    public static void main(String[] args) {
        boolean doConstraints = true;
        boolean doConsistency = true;
        boolean doHeuristics = true;
        boolean doSoft = true;
        boolean showStates = false;
        //Check for arguments
        String arg = args[0];

        if (arg != null) {
            fileName=arg;
        }
        arg = args[1];

        if (arg != null) {
            backimage=arg;
        }
        try {
            //Open NC file
            NetcdfFile nc = NetcdfFile.open(fileName);

            //Read Azimuth from the file
            Variable azimuth = nc.findVariable("Azimuth");
            Array az = azimuth.read();
            float[] azimuthArray = (float[]) az.copyTo1DJavaArray();

            //Read Elevation from the file
            Variable elevation = nc.findVariable("Elevation");
            Array el = elevation.read();
            float[] elevationArray = (float[]) el.copyTo1DJavaArray();

            //Conversion of Azimuth to Degrees and Radians
            double[] azimuthRadiansArray = new double[azimuthArray.length];
            for(int i=0;i<azimuthArray.length;i++){
                azimuthRadiansArray[i] = Math.toRadians(azimuthArray[i]);
            }
            ucar.nc2.Dimension gate = new Group(nc, null, "gate").findDimension("maxCells");
            int gasize = gate.getLength();

            //Read Reflectivity from the file
            Variable zh = nc.findVariable("ZH");
            Array z = zh.read();
            short[] zhArray = (short[]) z.copyTo1DJavaArray();

            Attribute scale = zh.findAttribute("scale_factor");
```

```
        int k = 0;
        double[][] zhDoubleDimensionArray = new double[azimuthRadiansArray.length][gasize];
        for(int i=0;i<azimuthRadiansArray.length;i++){
            for(int j=0;j<gasize;j++){
                zhDoubleDimensionArray[i][j] = zhArray[k]*.01;
                k++;
            }
        }
        try{
            nc.close();
        }
        catch(java.io.IOException e){
            System.out.println("Error closing input file");
        }

        // Set a new image - 800px x 800px
        Draw t = new Draw(800, 800, backimage);
        t.setScale(0, 0, 800, 800);
        double x = 0.0;
        double y = 0.0;
        int spot = 0;

        for(int i=1;i<=azimuthArray.length;i++){
            for(int j=1;j<=gasize;j++){
                //A conversion from Cartesian Coordinates to Spherical Coordinates is performed.
                x = (j*1.1) *Math.sin(azimuthRadiansArray[i-1])* Math.cos(elevationArray[i-1] * (Math.PI/180));
                y = (j*1.1) *Math.cos(azimuthRadiansArray[i-1])* Math.cos(elevationArray[i-1] * (Math.PI/180));
                t.moveTo(x+400, y+400);
                if (zhDoubleDimensionArray[i-1][j-1] > 0 && zhDoubleDimensionArray[i-1][j-1] < 77.5){
                    t.setColor(setReflectivityColor(zhDoubleDimensionArray[i-1][j-1]));
                    if(j<50){
                        t.spot(2);
                    }
                    else if(j>=50 && j<100){
                        t.spot(3);
                    }
                    else if(j>=100 && j<150){
                        t.spot(4);
                    }
                    else
                        t.spot(6);
                }
            }
        }
        //save as JPG image
        t.save(fileName.substring(0,fileName.lastIndexOf('.') + 1)+"jpg");

    }catch(java.io.IOException e){
        System.out.println("Error reading netcdf file");
    }
}

//Relation between Reflectivity values and the Pixel Color
public static Color setReflectivityColor(double val){
    float[] hsb = new float[3];

    if (val>72.5 && val<=77.5){
        hsb = Color.RGBtoHSB(255, 255, 255, hsb);
    }
    else if (val>67.5 && val<=72.5){
        hsb = Color.RGBtoHSB(148, 0, 211, hsb);
    }
    else if (val>62.5 && val<=67.5){
        hsb = Color.RGBtoHSB(255, 0, 255, hsb);
    }
    else if (val>57.5 && val<=62.5){
        hsb = Color.RGBtoHSB(184, 0, 0, hsb);
```

```
        }
        else if (val>52.5 && val<=57.5){
            hsb = Color.RGBtoHSB(216, 0, 0, hsb);
        }
        else if (val>47.5 && val<=52.5){
            hsb = Color.RGBtoHSB(255, 0, 0, hsb);
        }
        else if (val>42.5 && val<=47.5){
            hsb = Color.RGBtoHSB(255, 140, 0, hsb);
        }
        else if (val>37.5 && val<=42.5){
            hsb = Color.RGBtoHSB(255, 215, 0, hsb);
        }
        else if (val>32.5 && val<=37.5){
            hsb = Color.RGBtoHSB(255, 255, 0, hsb);
        }
        else if (val>27.5 && val<=32.5){
            hsb = Color.RGBtoHSB(0, 128, 0, hsb);
        }
        else if (val>22.5 && val<=27.5){
            hsb = Color.RGBtoHSB(0, 204, 0, hsb);
        }
        else if (val>17.5 && val<=22.5){
            hsb = Color.RGBtoHSB(0, 255, 0, hsb);
        }
        else if (val>12.5 && val<=17.5){
            hsb = Color.RGBtoHSB(0, 0, 255, hsb);
        }
        else if (val>7.5 && val<=12.5){
            hsb = Color.RGBtoHSB(0,153, 255, hsb);
        }
        else if (val>2.5 && val<=7.5){
            hsb = Color.RGBtoHSB(0, 255, 255, hsb);
        }
        else if (val>-2.5 && val<=2.5){
            hsb = Color.RGBtoHSB(102, 102, 102, hsb);
        }
        else if (val>-7.5 && val<=-2.5){
            hsb = Color.RGBtoHSB(153, 153, 102, hsb);
        }
        else if (val>-12.5 && val<=-7.5){
            hsb = Color.RGBtoHSB(204, 204, 153, hsb);
        }
        else if (val>-17.5 && val<=-12.5){
            hsb = Color.RGBtoHSB(102, 51, 102, hsb);
        }
        else if (val>-22.5 && val<=-17.5){
            hsb = Color.RGBtoHSB(153, 102, 153, hsb);
        }
        else if (val>-27.5 && val<=-22.5){
            hsb = Color.RGBtoHSB(204, 153, 204, hsb);
        }
        else if (val>-32.5 && val<=-27.5){
            hsb = Color.RGBtoHSB(204, 255, 255, hsb);
        }
        else{
            hsb = Color.RGBtoHSB(255, 255, 255, hsb);
        }
        return Color.getHSBColor(hsb[0], hsb[1], hsb[2]);
    }
}
```

REFERENCE LIST

[1] McLaughlin D.; Brotzge J.; Chandresakar V.; Droegemeier K.; Kurose J.; Philips B.; Preston M. and Sekelsky S. Distributed collaborative adaptive sensing for hazardous weather detection, tracking, and predicting. *International Conference on Computational Science*, 2004.

[2] Committee on Weather Radar Technology Beyond NEXRAD National Research Council. *Weather Radar Technology Beyond NEXRAD*. National Academy Press, Washington D.C., 2002.

[3] Doviak R.J. and Zrnic D.S. *Radar and Weather Observations*. National Academy Press, San Diego, 2 edition, 1993.

[4] Daz P. L.; Aquino Z.; Figueroa-Alamo C.; Garca R. and Sánchez A. V. Water resources data puerto rico and the u.s. virgin islands water year 2001. Technical report, US Geological Survey, 2001.

[5] Foster I. and Kesselman C. *The grid: blueprint for a future computing infrastructure*. Morgan Kaufmann Publishers, Apr 1998.

[6] Foster I.; Kesselman C.; Nick J. and Tuecke S. The physiology of the grid: An open grid services architecture for distributed systems integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, Jun 2002.

[7] Evans J. E.; Weber M. E. Weather radar development and application programs. *MIT Lincoln Laboratory Journal*, 10:1033–1036, Apr 1985.

[8] Novotny J.; Russell M.; Wehrens O. Gridsphere: an advanced portal framework. In *30th EUROMICRO Conference*, pages 412–419. IEEE Computer Society, 2004.

[9] Microprocessor and Microcomputer Standards Committee. Ieee standard specifications for public-key cryptography. Technical Report IEEE Std 1363-2000, IEEE-SA Standards Board, 2000.

[10] Roman S. *Field Theory*. Springer, 2nd edition, 1994.

[11] Droegemeier K.K.; Gannon D.; Reed D.; Plale B.; Alameda J.; Baltzer T.; Brewster K.; Clark R.; Domenico B.; Graves S.; Joseph E.; Murray D.; Ramachandran R.; Ramamurthy M.; Ramakrishnan L.; Rushing J.A.; Weber D.; Wilhelmson R.; Wilson A.; Xue M.; Yalda S. Service-oriented environments for dynamically interacting with mesoscale weather. *Computing in Science & Engineering*, 7(6):12–29, Nov.-Dec. 2005.

[12] Wylie J. J.; Bigrigg M.W.; Strunk J. D.; Ganger G. R.; Kiliççöte H. and Khosla P. K. Survivable information storage system. *IEEE Computer*, 33(8):61–68, August 2000.

[13] Rhea S.; Eaton P.; Geels D.; Weatherspoon H.; Zhao B. and Kubiatowicz J. The oceanstore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies - FAST'03*, pages 1–14, Berkeley, California, United States, 2003. USENIX Association.

[14] Ghemawat S.; Gobio H. and Leung S. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP'03*, pages 29–43, New York, New York, United States, 2003. ACM Press.

[15] Santos J. R.; Muntz R. R. and Ribeiro-Neto B. A. Comparing random data allocation and data striping in multimedia servers. In *ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 44–45, 2000.

[16] Rabin M. O. Efficient dispersal of information for security, load balancing, and fault tolerance. *ACM Journal*, 36(2):335–348, 1989.

[17] Brinkmann A.; Salzwedel K. and Scheideler C. Efficient, distributed data placement strategies for storage area networks. In *Proceedings of the Twelfth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 119–128, Bar Harbor, Maine, United States, July 2000. ACM Press.

[18] Patterson D. A.; Gibson G. and Katz R. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD international Conference on Management of Data*, pages 109–116, Chicago, Illinois, United States, June 1988. ACM Press.

[19] Santos J. R.; Muntz R. R. and Ribeiro-Neto B. Efficient dispersal of information for security, load balancing, and fault tolerance. *ACM*, 36(2):335–348, April 1989.

[20] Bestavros A. Seth: A vlsi chip for the real-time information dispersal and retrieval for security and fault-tolerance. In *Proceedings of ICPP'90, The 1990 Intemational Conference on Parallel Processing*, Chicago, Illinois, United States, August 1990.

[21] Plank J. S. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Software-Practice and Experience (SPE)*, 27(9):995–1012, September 1997. Correction in Plank J. S. and Ding Y., Technical Report UT-CS-03-504, U. Tennessee, 2003.

[22] Daemen J. and Rijmen V. The block cipher rijndael. In *Proceedings of the The International Conference on Smart Card Research and Applications - CARDIS'98*, pages 277–284, London, United Kingdom, 2000. Springer-Verlag.

[23] Koc C. and Acar T. Montgomery multiplication in gf(2k). *Design, Codes and Cryptography*, 14(1):57–69, April 1998.

[24] Win E.; Bosselaers A.; Vandenberghe S.; Gersem P.D. and Vandewalle J. A fast software implementation for arithmetic operations in gf(2n). In *Proceedings of Advances in Cryptology - ASIACRYPT'96*, pages 65–76, 1996.

[25] Johnson M.K. and Troan E.W. *Linux application development.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

ENABLING DISTRIBUTED RADAR DATA RETRIEVAL AND PROCESSING IN DISTRIBUTED COLLABORATIVE ADAPTIVE SENSING ENVIRONMENTS

Diego Mauricio Arias Velasco
(787) 415-2020
Department of Electrical and Computer Engineering
Chair: Wilson Rivera Gallego
Degree: Master of Science
Graduation Date: March 2007

This thesis describes the integration of radar network and grid computing technologies to provide a set of grid services to manipulate and store data from different radars, to execute algorithms on different platforms in a transparent way to end users, and to provide secure access to storage systems and instruments. The solution approach considered is a grid-based system which includes a Grid Portal Interface, a distributed storage to radar data management, and Grid services implementing distributed algorithms to process data radar information.

A major requirement of this system is data availability and reliability. Consequently, have been implemented two redundancy schemes to perform the data management of the network: A simple Replication Algorithm and the Information Dispersal Algorithm (IDA). Experimental results show that IDA provides better reliability and less storage spending than the traditional replication algorithm. Furthermore, under the same conditions, the redundancy in the replication technique is three times or more than IDA, when the reliability required is over 90%. IDA is normally used to perform security deployments where message encryption is required. Thus, this algorithm has been modified to enable large file management in order to satisfy the large amount of data generated by radars.