Grid Based Pervasive Distributed Storage

Diego Arias, John Sanabria, and Wilson Rivera
Parallel and Distributed Computing Laboratory
University of Puerto Rico at Mayaguez
{diego.arias, john.sanabria, wrivera}@ee.uprm.edu

Abstract— **This paper *describes a tool that implements a set of services to manipulate and store data from a sensor network in a transparent way to end users. A major requirement of this system is data availability and reliability. Consequently, we have implemented a replication schema based on the Information Dispersal Algorithm (IDA). Preliminary results show that the IDA based replication provides better reliability and less storage spending than traditional replication. The storage scheme has been deployed on top of a Globus based infrastructure.***

Index Terms—**Distributed storage, information dispersal, grid computing, service oriented architecture.**

I. Introduction

THE Wide Area Large Scale Automated Information Processing (WALS-AIP) project, funded by the US National Science Foundation, aims at developing an infrastructure for the treatment of signal-based information arriving from physical sensors in a wide-area, large scale environment. The proposed model accentuates a distributed space-time processing format. This approach demands efficient data and resource management techniques.

We have deployed the prototype of a grid-service based system to access and manipulate data from a sensor network: a grid portal interface provides transparent access to end-users; raw data from sensors are sent to a data server via wireless communication; GridFTP is used to improve data transport from the data server to a grid infrastructure; data exchange between server and the Grid testbed is authenticated using Grid Security Infrastructure (GSI); and finally a replication strategy based on the information dispersal algorithm (IDA) is incorporate into the tool to manage the distributed storage of the data.

The information dispersal algorithm (IDA) [1] 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 recovery 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 replication technique. The storage spending is $|F|*(n/m)$. An important feature of this technique is that any m blocks will reconstruct the file and labels are not necessary for each block. Additionally IDA tolerates up to r failures, where $r = n - m$. Hence, IDA guarantees a higher availability.

This paper is organized as follows. Section II describes in detail the implementation of the information dispersal algorithm. Section III presents experimental results. Section IV describes the implementation and deployment of the information dispersal service in a grid infrastructure. Finally, section V draws conclusions and future work.

II. Information Dispersal

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

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

Let F= $b_1, b_2, b_3, \ldots$ be a file, where b_i is an integer taken from a certain range $[0 \ldots (2^B -1)]$. If b_i is two bytes long, as in the actual implementation, then $0 \leq b_i \leq 65535$. Let p be a prime number greater than b_i. Each b_i is an element of the finite field Z_p where all arithmetic operations are done in mod p. Since $p > (2^B -1)$, 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 b_i 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 the prime p [2]. 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_{nxm} be a matrix whose ith 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_{nxm}.

This material is based upon work supported by the US National Science Foundation under Grants No 0313747 and No 0424546.

$$A_{nxm} \cdot \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$$

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, …, s_m) and the recovery operation is performed as follows: let B_{mxm} be a matrix whose rows are (a_{s1}, a_{s2}, a_{s3}, …, a_{sm}). 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_{mxm}.

$$B^{-1}_{mxm} \cdot \begin{bmatrix} c_1 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

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

An A_{nxm} matrix with the properties above mentioned is the Vandermonde matrix. The ith row of this matrix is defined as:

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

By definition, this matrix has the property that any submatrix formed by deleting its *m* rows, is invertible. Additionally, any matrix derived from this matrix by a sequence of elementary matrix transformations, will maintain this property [3].

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 *n*x*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 1st element from the each new sequence. The second block will have the 2nd 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 corresponds to the ith 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 1 shows an example of the IDA behavior when *n* = 12 and *m* = 4.

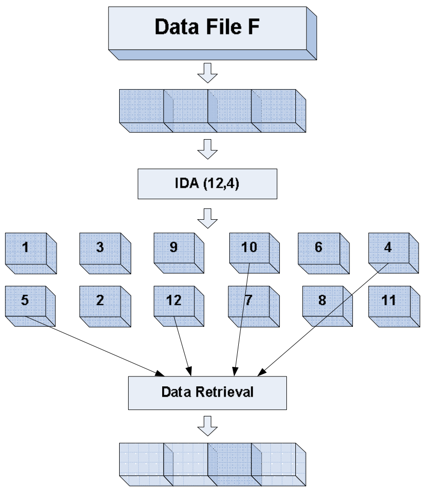


Figure 1: Replication example with *n* = 12 and *m* = 4.

III. EXPERIMENTAL RESULTS

For our performance analysis we consider 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 spending (SS) required to perform redundancy.

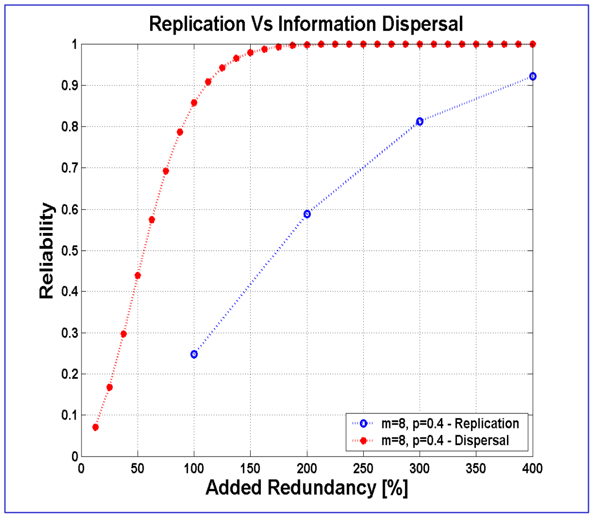


Figure 2: Standard Replication and IDA

Information dispersal algorithm shows a better access reliability than the simple replication algorithm as shown in Figure 2. As a reference point, for an access reliability $R = 0.9$ when the probability of failure is $p = 0.4$ and $m = 8$, the added redundancy for IDA is $AR = 120$ %, while in the replication approach the added redundancy must be approximately AR ≈ 300 %. Note that, for replication algorithm, AR increment is every 100%, because the redundancy is performed using multiplication with integer numbers.

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 even worse if p is higher. In contrast, a higher number m involves an 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 taking into account when sensor data must be manipulated, because the size of this data is usually large. Therefore, a proper redundancy must be selected to avoid storage overhead.

Experiments involving time measurements vs. data size, take as reference data size from National Climatic Data Center. This data is a Level II base data [4], available in compressed tape archive format. It contains data per day from specific NEXRAD Level II radar. The compressed data for a day is about 150 MB, while uncompressed is about 2.3GB. Note that this is the data mount for 24 hours of continuous scan. For rain fall measurements and precipitation estimation, the primary implementation of DCAS network requires less than 8 hours of continuous scan. Considering all the exposed before and the limited transmission due to wireless communication as mentioned, testing is achieved with data size range from 100MB to 1GB.

Different memory management strategies for the IDA have been studied to reduce execution time (Figure 3). We have been able to reduce significantly the operation time of the IDA approach. In addition the new IDA based implementation of replication supports large amount of data as required in a radar network

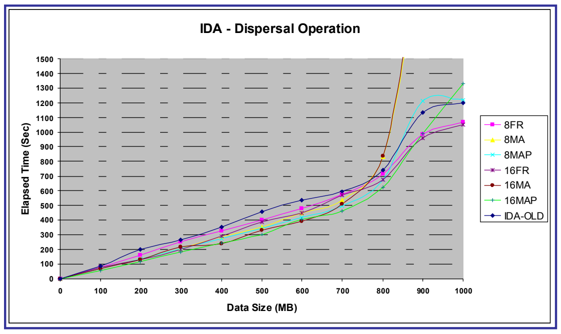


Figure 3: IDA Operation and Time Reduction

Note that, 8FRL is the enhanced version of IDA-OLD. They show a similar behavior with data size < 800 MB, but the main difference occurs at 900 MB and 1000 MB where the elapsed time is reduced ~2 min and ~2.5 min 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 800 MB without time reduction. MAP presents the better performance than MMA and FRL, with data sizes smaller than 500 MB. 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 4 shows the results after performing such modifications.

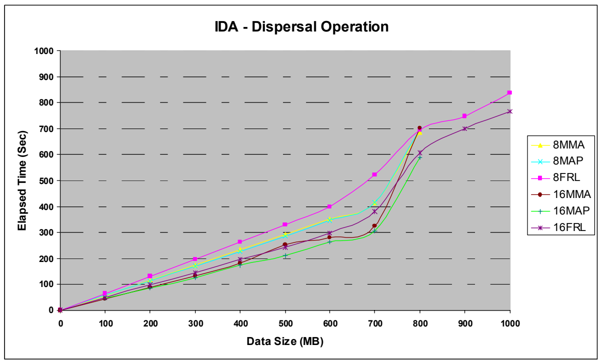


Figure 4: IDA with pre-computed tables

The behavior of the different techniques is similar to the results shown in Figure 3. However, a significant reduction of time is obtained. Again, 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 1000 MB file.

An additional set of IDA experiments was performed for data sizes between 10MB and 100 MB, as well as data sizes lower than 10 MB.

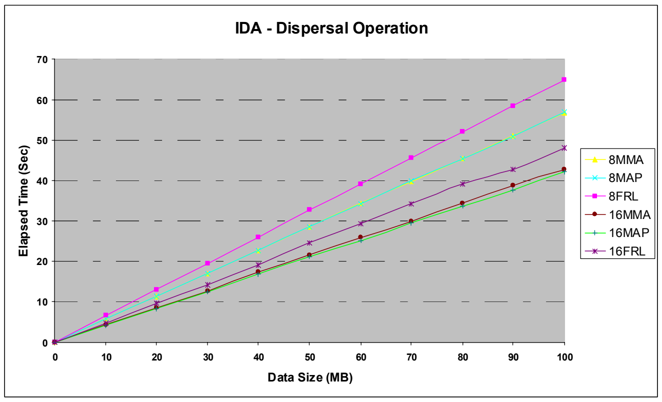


Figure 5: Data Size vs. Elapsed Time (10 to 100 MB)

As shown in Figure 5, 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 <500 MB instead of MAP, because MMA does not affect the code portability and it can be implemented in different platforms.

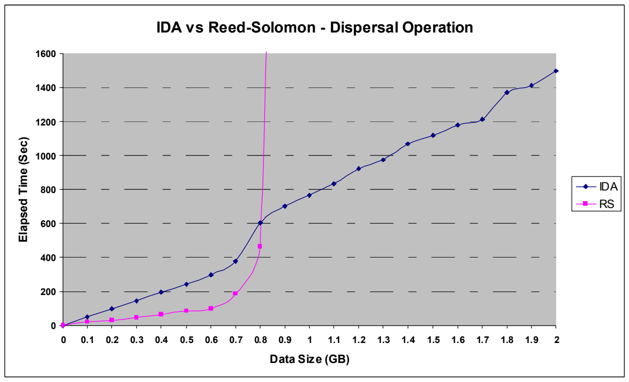
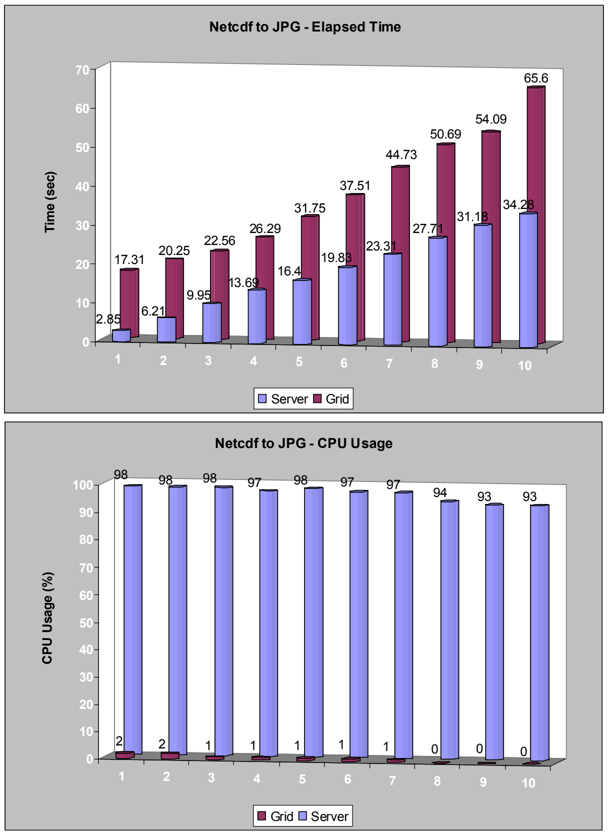


Figure 6: IDA vs Reed-Solomon comparison

Finally, we present a comparison between the implemented IDA and the Reed-Solomon (RS) algorithm. RS has similar properties as IDA and it also requires operations over GF. As shown in Figure 6 Reed-Solomon presents a better response in terms of execution time with files < 800MB. However, RS algorithm does not support large data files.

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.



	1	2	3	4	5	6	7	8	9	10
Server (sec)	2.85	6.21	9.95	13.69	16.4	19.83	23.31	27.71	31.13	34.28
Grid (sec)	17.31	20.25	22.56	26.29	31.75	37.51	44.73	50.69	54.09	65.6
Server CPU (%)	98	98	98	97	98	97	97	94	93	93
Grid CPU (%)	2	2	1	1	1	1	1	0	0	0

Figure 7: Resources used for NCtoJPG process. a) Elapsed Time, b) Percentage of CPU usage.

For example, in Figure 7(a) the time elapsed for five (5) files processed over the grid is ≈ two times slower than a single job executed on the server. Further studies show that a similar result is obtained when the required files are equal 8, 9, or 10. However, the CPU consumption (Figure 7(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. First, the total execution time is limited by the number of the nodes available in the grid. Second, the stage-in and stage-out procedures required to perform multi-jobs 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 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.

IV. SERVICE DEPLOYMENT

The IDA based replication strategy has been deployed as a composite service on top of the Globus toolkit following the WSRF specification. The components of the IDA service include a service to select the adequate places to store the file pieces (adapter service) and a service to split and recover files (IDA based service). Each service is an independent functional entity so it can be re-used on other service deployment. In our deployment GridFTP is used to transfer files among resources, and GSI (Globus Security Infrastructure) services are used to certificate management and authentication.

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 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.

GKrellM, a third party monitor tool, is used to monitor and collect data related to memory and disk space in resources. The adapter service takes into consideration information related to both disk space and GridFTP service availability on specific resources to determine the number of file chunks and places to store pieces of files. The IDA-based service then implements the partition and store of the pieces of files. Figure 8 illustrates the interaction among the different services.

The deployment of the services has been tested on the PDCLab Grid Testbed, deployed at the University of Puerto Rico-Mayaguez. This testbed 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 computational resources available on the grid include an IBM xSeries Linux cluster with 64 nodes, dualprocessor 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 140Gb 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); and two (2) PowerVault storage with 8TB.

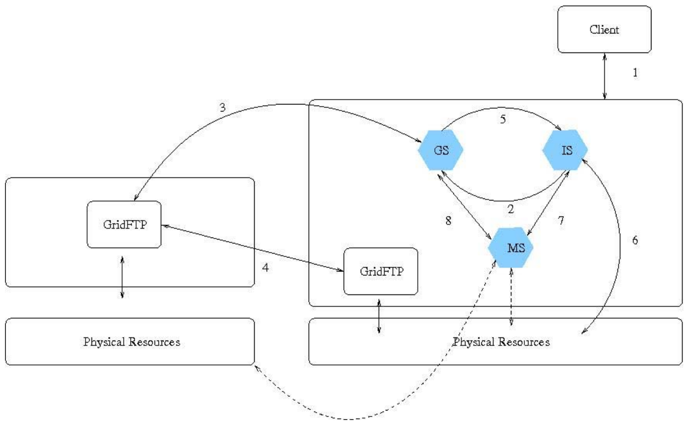


Figure 8: IDA Service Deployment Schema

V. CONCLUSIONS AND FUTURE WORK

A replication schema based on the Information Dispersal Algorithm (IDA) has been presented in this paper. It was shown that the proposed redundancy scheme and its subsequent deployment as a grid service improve reliability in distributed storage.

The work in this paper is considered as a initial proof of concept for a more complex project related to the design and implementation of adaptive resource allocation and migration. In the proposed service oriented architecture, several instances of adapters are allowed to deal with local administrative domain management. Consequently, the adapter module will implement mechanisms to monitor and react to local conditions via active/reactive services. We are working on developing new adaptive resource allocation mechanisms using the functionalities of both the adapter module, partially described in this paper, and active/reactive services.

REFERENCES

[1] Rabin, M. O., “Efficient dispersal of information for security, load balancing, and fault tolerance*”, Journal ACM.* 36, 2 (Apr. 1989), 335-348.

[2] 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 International Conference on Parallel Processing,* Chicago, Illinois, August 1990.

[3] 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, Sept. 1997. Correction in James S. Plank and Ying Ding, Technical Report UT-CS-03-504, U. Tennessee, 2003.

[4] National Climatic Data Center, “Data documentation for DSI-6500 NEXRAD Level II”, *National Climatic Data Center*, Asheville N.C., April 11, 2005