

Data Assignment in Fault Tolerant Uploads for Digital Government Applications: A Genetic Algorithms Approach*

Yan Yang
Computer Science
Department
University of Southern
California
yangyan@usc.edu

Leslie Cheung
Computer Science
Department
University of Southern
California
lccheung@usc.edu

Leana Golubchik
Computer Science &
EE-System Departments,
IMSC, and ISI
University of Southern
California
leana@cs.usc.edu

ABSTRACT

This paper investigates a data assignment problem in a fault tolerance protocol of Bistro, a wide area upload framework. Uploads correspond to an important class of applications, whose examples include a large number of digital government applications. Specifically, government at all levels is a major *collector* and provider of data, and there are clear benefits to disseminating and collecting data over the Internet, given its existing large-scale infrastructure and wide-spread reach in commercial, private, and government domains. In this project we focus on the *collection of data over the Internet*. By data collection, we mean applications such as Internal Revenue Service (IRS) applications with respect to electronic submission of income tax forms.

In Bistro, clients upload their data to intermediaries, known as bistros, to reduce the traffic to the destination around a deadline. The destination server then computes a schedule for pulling the data from bistros after the deadline. In the Bistro framework, bistros can be unavailable or malicious. Thus, a fault tolerance protocol is a vital and fundamental component of the Bistro framework. In this paper, we are particularly interested in a data assignment problem in the Bistro fault tolerance protocol. We formulate this problem into a non-linear optimization problem and develop a genetic algorithm heuristic as an approximation. We evaluate our approach using simulations and compare the results of our heuristic with other simple heuristics as well as an optimal solution obtained from a brute-force approach.

Categories and Subject Descriptors

*This work is supported in part by the NSF Digital Government Grant 0091474. This work made use of Integrated Media Systems Center Shared Facilities supported by the National Science Foundation under Cooperative Agreement No. EEC-9529152. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation. <http://bourbon.usc.edu/iml/bistro>.

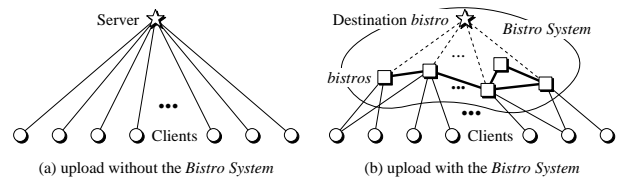


Figure 1: Traditional Uploads and Uploads with Bistro

D.4.5 [Operating Systems]: Reliability—*Fault-tolerance*;

J.1 [Computer Applications]: Administrative Data Processing—*Government*

Keywords

Data collection, uploads, many-to-one communication

1. INTRODUCTION

High demand for some services or data creates hot spots, which is a major hurdle to achieving scalability in Internet-based applications. In many cases, hot spots are associated with real life events. There are also real life deadlines associated with some events, such as submissions of papers to conferences, electronic submissions of income tax forms, submissions of proposals to granting agencies, homework submissions in distance education, and online shopping for limited-time bargain products. The demand of applications with deadlines is potentially higher when the deadlines are approaching.

Previous work attempted to relieve hot spots in one-to-one communications, such as email and instant messaging, one-to-many communications, such as web downloads, and many-to-many communications, such as chatrooms and video conferencing. A number of techniques have been introduced including service replication (e.g., replication of DNS servers), data replication (e.g., web caching and Akamai), and data replacement (e.g., different streaming rates for audio and video streaming). To the best of our knowledge, however, there are no research attempts to relieve hot spots in many-to-one applications, except for Bistro [1].

Many-to-one applications, or *upload* applications, correspond to an important class of applications, and particularly digital government applications. Specifically, government at all levels is a major *collector* and provider of data, and there are clear benefits to disseminating and collecting data over the Internet, given its existing large-

scale infrastructure and wide-spread reach in commercial, private, and government domains. In this project, we focus on the *collection of data over the Internet*, and specifically, on the *scalability* issues which arise in the context of Internet-based massive data collection applications. By data collection, we mean applications such as Internal Revenue Service (IRS) applications with respect to electronic submission of income tax forms. Briefly other such applications are as follows. The Integrated Justice Information Technology Initiative facilitates information sharing among state, local, and tribal justice components. An integrated (global) information sharing system involves collection, analysis, and dissemination of criminal data. Clearly, in order to facilitate such a system one must provide a *scalable* infrastructure for collection of data. Furthermore, a number of government agencies (e.g., NSF, NIH) support research activities, where the funds are awarded through a grant proposal process, with deadlines imposed on submission dates. The entire process involves not only submission of proposals, which can involve fairly large data sizes, but also a review process, a reporting process (after the grant is awarded), and possibly a results dissemination process. All these processes involve a data collection step. Lastly, digital democracy applications, such as online voting during federal, state, or local elections, constitute another set of massive upload applications. Of course, there are numerous other examples of digital government applications with large-scale data collection needs. In all these upload applications, hot spots are due to a demand for a popular service. The real-life event which causes hot spots often imposes a hard deadline on the data transfer service.

Bistro is a wide-area upload architecture built at the application layer, and previous work [3] has shown that it is scalable and secure. Figure 1 compares the upload data flow in traditional applications and using the Bistro framework. In traditional upload applications, every client sets up a one-to-one communication channel with the destination server, and transfers data through that channel. It is simple yet not scalable and creates hot spots. In Bistro, an upload process is broken down into three steps [1]. First, in the timestamp step, clients send hashes of their files to the server, and obtain timestamps. These timestamps clock clients' submission time. In the data transfer step, clients send their data to intermediaries called bistros. In the last step, called the data collection step, the server coordinates bistros to transfer clients' data to itself. The server then matches the hashes of the received files against the hashes it received directly from the clients. The server accepts files that pass this test, and asks the clients to resubmit otherwise. This completes the upload procedure in the original Bistro architecture. (See [1, 3] for full details of the protocol.)

Let us give an example of a large-scale digital government upload application. Let us consider a collection of income tax forms in the US. Shortly before the April 15 deadline, we can expect a large number of individuals to be trying to submit their tax forms to the IRS server.

Without the Bistro framework, each client sets up a connection to the server, and sends his/her form via that connection. However, since there are a large number of clients, the server or the resources between the client and server (e.g., the network), are likely to saturate. That is, the server is likely to be too busy to handle all requests. Also, the tax forms' size is typically fairly large (on the order of 100 KB [15]), which means that it may take a while for the client to transfer the form to the IRS server. The high demand for service around the deadline time and the relatively large size of files cause "traffic jams" and result in very long transfer times.

Hence, clients may not be able to submit their forms before the deadline. Alternatively, instead of using a single server, the IRS could use a cluster of servers, i.e., clients can submit to a number of servers providing the same service (in our case, a tax form submission service). If we locate the cluster in the IRS domain, it is still possible to saturate network resources in the IRS domain. On the other hand, if we spread the cluster over different domains, we have to trust the other domains since they could have access to sensitive information. Nevertheless, the cost for maintaining a cluster of servers is high. This is especially the case for a IRS type application where the submission event is relatively rare. Hence an investment in a cluster of servers might be a fairly expensive one.

With the Bistro framework, a client first sends a checksum of his/her tax form, instead of the tax form itself, to the IRS server, and obtains a timestamp. A checksum is typically much smaller than the size of a tax form (20 bytes for a SHA1 checksum versus 100KB for a tax form), so the problem mentioned above is alleviated. After the client receives a timestamp, s/he cannot change the content of his/her tax form without IRS detecting it. This is because it is practically impossible to modify a tax form in a meaningful way and have it result in the same checksum. At this point, since we are essentially guaranteed that the client cannot change the form, we can transfer the form at a later time. The client then sends the entire tax form to one of the bistros. Since we assume that anyone can install a bistro server, bistros are not trusted; hence a secure protocol was developed to prevent bistros from accessing content of uploaded files [3]. The cost of maintaining a Bistro system is much lower, because we can use the same set of intermediate bistros in many other upload events, e.g., online voting in federal, state, or local elections. Finally, the IRS server will contact the bistros to transfer the form from the chosen bistros to itself.

It is possible that when the destination server tries to pull clients' data, some intermediaries are not available due to, for example, power failure or network problems. Then all data on the unavailable bistros is not accessible and can be considered lost. In addition, since one of our design goals is to deploy Bistro in a public infrastructure, such as the Internet, intermediaries are not trusted. As a result, clients encrypt their data to ensure data privacy. Yet, even though malicious bistros cannot read the data, they can modify it. The destination server can detect this, but it has no way of recovering the original data, which results in data losses. In case of data losses, the original Bistro protocol requires the destination server to request client resubmissions, which can be a slow process. Hence, unavailable bistros and malicious behavior can result in degraded system performance.

To improve system performance in the face of these failures and malicious behavior, we developed a fault tolerance protocol in [4]. In this protocol, we use forward error correction techniques to recover corrupted or lost data. We add redundancy to clients' data and modify the data transfer in the original protocol to allow clients to stripe their data to multiple intermediate bistros. In particular, we use erasure codes to recover the lost data. An (n, k) erasure code encodes a FEC (forward error correction) group of k packets into n packets. As long as we receive any k packets from a FEC group during transmission, the receiver can recover the file. In addition to forward error correction techniques, we employ striping techniques to further improve system reliability. In our fault tolerance protocol, we stripe clients' data across a number of bistros, in contrast to putting all data from a client on the same bistro in the original protocol. This is done to reduce the risk of losing all clients' data

and hence, improve system reliability and performance.

In this paper, we are interested in a data assignment problem within the context of the Bistro fault tolerance protocol. In the data assignment problem, we need to determine how much data goes to each bistro and how this affects the probability that the final destination can successfully receive the data from intermediate bistros or recover it. Our goal is to maximize the probability for the destination to receive or be able to recover the data. We believe the data assignment problem is an important problem in the Bistro fault tolerance protocol: (1) since intermediate bistros are not trusted, and a well designed data assignment strategy can make the system more reliable; and (2) since every intermediate bistro has different reliability characteristics, different assignments affect system reliability. In this problem, a client needs to decide how much data it should stripe to each bistro. If the destination server receives at least k out of n packets from a FEC group, it can recover the original data.

Let us go back to the IRS example. Suppose we have 3 intermediate bistros, b_1, b_2 , and b_3 . Assume that their failure probabilities are 0.1, 0.2, 0.3, respectively. Also assume that the client uses a (3,2) erasure code to encode his/her tax form, and the size of the encoded tax form is 3 packets. That is, the IRS server needs to collect at least 2 out of 3 packets. Two possible (out of several) ways to assign these packets to the intermediate bistros are: (1) assign all of them to b_1 , the most reliable bistro, or (2) spread them among all bistros. We are interested in the probability that the IRS server receives the tax form, i.e., receives at least 2 out of 3 packets successfully. In the first scheme, the probability is 0.9. In the second scheme, the probability is 0.902.

From this example, we can see that the assignment of packets affects the reliability of the Bistro system. Our task is to find an optimal assignment such that the destination has a maximal probability of collecting enough packets of the file in order to be able to reconstruct it. In this paper, we use a genetic algorithm heuristic to approximate an optimal solution of this data assignment problem. The detailed problem formulation can be found in the next section.

The contributions of this work are as follows. We formulate the data assignment problem in the Bistro fault tolerance protocol. We then propose a genetic algorithm heuristic to approximate an optimal solution to this problem which is more accurate than several simple heuristics used for comparison, and efficient, as compare to a brute-force approach.

The remainder of this paper is organized as follows. We give the problem formulation in Section 2. In Section 3, we describe our genetic algorithm heuristic. We give validation and evaluation of our heuristic in Section 4. Section 5 describes related work. Finally, we conclude in Section 6.

2. PROBLEM FORMULATION

In this section, we formulate an optimization problem corresponding to the assignment problem described before.

2.1 Notation

Our notation is based on Figure 2. The original file is divided into K FEC groups, where each group contains k packets. After encoding, each FEC group is expanded to n packets. Each FEC group is further divided into G checksum groups. The total number of striping units is then KG . The number of intermediate bistros to which we can stripe is B . In this paper, we use a simple model that

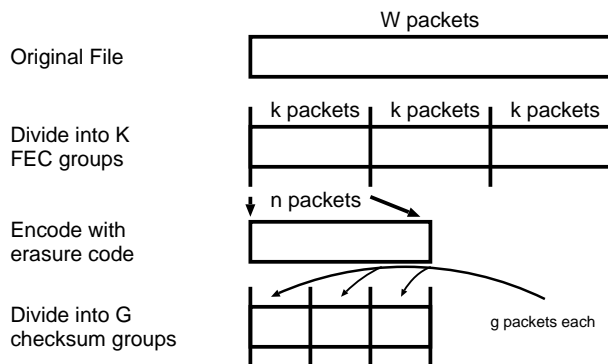


Figure 2: Graphical Representation of Notation

assumes all bistros fail independently. Let us assume the probability that bistro i fails be p_i ($i = 1, 2, \dots, B$). In practice, we can estimate p_i by monitoring bistro i for a period of time. Investigating more sophisticated reliability models is part of our future work. We believe that using more sophisticated models does not change our framework significantly.

2.2 Definitions

Let x_i ($i = 1, 2, \dots, B$) denote the number of checksum groups we transfer to an intermediate bistro B_i . This data assignment forms a set X , where

$$X \triangleq \{x_1, x_2, \dots, x_B\}.$$

Let \mathbb{S}_X denote the maximal cardinality subset of the power set of X , such that all the elements $S_i \in \mathbb{S}_X$ satisfy the condition that the sum of all the elements in S_i is no more than the maximal number of checksum groups that the destination server can lose and still be able to reconstruct the entire file. Intuitively, \mathbb{S}_X denotes all the intermediate bistro state patterns corresponding to data assignment X where the final destination can still successfully reconstruct the entire file. Therefore, we have,

$$\mathbb{S}_X \subseteq 2^X \quad \text{and} \quad S_i \in \mathbb{S}_X \quad \text{for} \quad i = 1, 2, \dots, |\mathbb{S}_X|. \quad (1)$$

$$\sum_{x_j \in S_i} x_j \leq \lfloor \frac{n-k}{n} KG \rfloor \quad \text{for} \quad i = 1, 2, \dots, |\mathbb{S}_X|. \quad (2)$$

A special case for \mathbb{S}_X is the empty set \emptyset ; we can easily verify from Equations (1) and (2) that $\emptyset \in \mathbb{S}_X$ holds true for every X . Here, the empty set \emptyset corresponds to no intermediate bistro failures. Hence, \mathbb{S}_X denotes all intermediate bistro state patterns corresponding to a data assignment X where the final destination can successfully reconstruct the original file.

Let P_i denote the probability of obtaining the failure pattern S_i corresponding to assignment X ; then we have

$$P_i \triangleq \prod_{x_j \in S_i} p_j \cdot \prod_{x_m \notin S_i} (1 - p_m). \quad (3)$$

We can use Equation (3) to calculate P_i for each S_i . Thus, the probability that the destination server can collect enough checksum groups to reconstruct the original file under assignment X , denoted

by P_X , is given by

$$P_X = \sum_{i=1}^{|\mathbb{S}_X|} P_i. \quad (4)$$

2.3 Optimization Problem

We would like to find an assignment $\{x_1, x_2, \dots, x_B\}$, where x_i ($i = 1, 2, \dots, B$) is a non-negative integer, such that the probability P_X is maximized. Thus, our assignment problem can be written formally as:

$$\max \sum_{i=1}^{|\mathbb{S}_X|} \left(\prod_{x_j \in S_i} p_j \cdot \prod_{x_m \notin S_i} (1 - p_m) \right).$$

subject to

$$\begin{aligned} \mathbb{S}_X &\subseteq 2^X; \\ S_i &\in \mathbb{S}_X & i = 1, 2, \dots, |\mathbb{S}_X|; \\ \sum_{x_j \in S_i} x_j &\leq \lfloor \frac{n-k}{n} KG \rfloor & i = 1, 2, \dots, |\mathbb{S}_X|; \\ \sum_{i=1}^B x_i &= KG; \\ x_i &\in \mathbb{N}. \end{aligned}$$

A straightforward approach here would be to divide our optimization problem into two steps: (1) computing \mathbb{S}_X , and (2) calculating P_X . In the first step, to compute \mathbb{S}_X , we need to determine each S_i which satisfies Equation (2). This is a subset-sum problem and is NP-Complete [9]. Once we have \mathbb{S}_X , calculating P_X takes linear time. Therefore, the total time complexity for our formulation above is determined by the time complexity of the first step.

3. GENETIC ALGORITHM HEURISTIC

In this section, we develop an approximation to an optimal solution of the optimization problem formulated in the previous section. We have already shown that the time complexity of our optimization problem is determined by its first step, which is NP-Complete. Therefore, we are interested in developing a heuristic. In particular, we will develop a genetic algorithm based heuristic. We first give a brief introduction to genetic algorithms, and then we show how to apply a genetic algorithm approach to our problem.

3.1 Genetic Algorithms

Genetic algorithms (GAs) [11] are stochastic search techniques guided by the principles of evolution and natural genetics. They are modeled loosely on the principles of evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation-inducing operators such as mutation and recombination (cross-over). A fitness function is used to evaluate individuals, and reproductive success varies with fitness. It generally proceeds as follows [6]:

- Step 1. Randomly generate an initial population;
- Step 2. Compute and save the fitness of each individual in the current population;
- Step 3. Define selection criteria for each individual such that the good gene is likely to be inherited;
- Step 4. Generate a new generation by inheriting good genes via genetic operators;
- Step 5. Repeat steps 2 to 4 until a satisfactory solution is obtained.

A typical genetic algorithm uses three genetic operators: selection, cross-over, and mutation to direct the population over a series of time steps or generations toward convergence at the global optimum. Selection attempts to apply pressure upon the population in a manner similar to that of natural selection found in biological systems. Poorer performing individuals (evaluated by a fitness function) are weeded out and better performing, or fitter, individuals have a greater than average chance of promoting the information they contain to the next generation. Crossover allows solutions to exchange information in a way similar to that used by a natural organism undergoing reproduction. Mutation is used to randomly change (flip) the value of single bits within individual strings to keep the diversity of a population and help a genetic algorithm to get out of a local optimum. It is typically used sparingly [6].

An effective GA representation, a meaningful fitness evaluation, and genetic operations are key to a successful GA application. The appeal of GAs comes from their simplicity as robust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems. GAs are useful and efficient when: (1) the search space is large, complex, or poorly understood; (2) domain knowledge is scarce or expert knowledge is difficult; (3) no mathematical analysis is available; and (4) traditional search methods fail [11, 12]. One of the advantages of the GA approach is the ease with which it can handle arbitrary types of constraints and objectives [6].

3.2 GA in Our Problem

As mentioned above, a good design of cross-over and mutation techniques can make GA powerful. We now develop a GA solution for our problem formulated in Section 2. In our formulation, we encode the data assignment into a vector $\{x_1, x_2, \dots, x_B\}$. Each element of the vector is the number of checksum groups we will transfer to bistro i .

3.2.1 Cross-over

Cross-over always happens between two vectors. In our GA design, we use a two-point cross-over mechanism. Two cross-over positions are randomly selected. Once we have chosen two positions, we swap elements between them, using the two vectors. This gives us two new vectors. This cross-over scheme is similar to that of [6] which is used to solve the traveling salesman problem. The difference lies in the way we maintain the feasibility of our solution. Our selection always picks the best half of a generation and allows them to cross-over to generate the next generation. For example, let $B = 6$. Then, given two assignments, $\{10, 10, 10, 10, 10, 10\}$ and $\{20, 0, 20, 0, 10, 10\}$, we randomly choose 2 positions, say 2 and 5; then all the elements between these two indices are swapped. Then, we obtain two children from the original two vectors. They are $\{10, 0, 20, 0, 10, 10\}$ and $\{20, 10, 10, 10, 10, 10\}$. For each child, we calculate their fitness using Equation (4) in Section 2.

One side effect of our cross-over operation is that a new assignment may break one of our constraints, i.e., $\sum x_i = KG$ ($i = 1, 2, \dots, B$). That is, the total number of checksum groups of a new assignment may not equal to the total number of checksum groups we want to stripe. In the previous example, the total number of checksum groups to stripe is $KG = 60$, but after cross-over, the sum of checksum groups in the newly generated assignments are 50 and 70, respectively. However, these assignments are not feasible. To solve this problem, we add/deduct checksum groups from some positions of the assignment. These positions are chosen randomly until the total number of checksum groups satis-

fies the assignment constraint again. For example, in assignment $\{20, 10, 10, 10, 10, 10\}$, we need to deduct 10 checksum groups to satisfy the constraint. To do this, we generate 10 random numbers and each number represents a chosen position in the assignment. For each of these randomly chosen positions, we deduct 1 checksum group if the number of checksum groups in this position is greater than zero. Otherwise, we randomly choose another position to deduct 1 checksum group. Similarly, if in a newly generated assignment, the total number of checksum groups is less than the assignment constraint, we randomly add checksum groups until the constraint is satisfied. These positions are randomly chosen as well.

3.2.2 Mutation

Another important element in our GA design is mutation. Although it is infrequently used, it keeps the diversity of a population and can help our GA heuristic escape from a local optimum. In our scheme, mutation is implemented by introducing a random assignment after a certain number of generations, if we find that our GA heuristic is stuck at a local optimum. In our simulations, we invoke a mutation if we observe the average fitness of a population does not change for 5 generations. We choose 5 generations, as we find in our test cases, the GA heuristic usually converges in less than 10 generations. We consider it a good indication for either a local optimum or a global optimum if the average fitness does not change in 5 generations. In our previous example, assignment $\{20, 10, 10, 10, 10, 10\}$, after adjusting it to satisfy an assignment constraint, may give a valid assignment $\{17, 9, 8, 10, 10, 6\}$. If it happens to keep crossing over with assignments without 0 elements, we may end up with assignments in our population all with non-zero values. Using this population, we are very unlikely to obtain an assignment with an element with 0 since there is no 0 gene in the population. This will prevent us from reaching a global optimum if it contains 0 elements. In this case, a mutation is needed to intentionally introduce some assignments with 0 elements.

Once we have completed the above operations, we are done with one generation. We run this process iteratively. The average fitness will keep improving if our GA heuristic is not stuck at a local optimum or reaches a global optimum. It stops after T_{long} generations so that a satisfactory assignment is found. We will discuss how to set T_{long} in the next section.

From the above description, a sketch of our GA heuristic is as follows:

- Step 1. Generate an initial population by encoding randomly generated assignments;
- Step 2. Calculate fitness of each individual assignment using Equation (4);
- Step 3. Pick the best half of the population and perform cross-over;
- Step 4. Adjust the newly generated assignments to satisfy $\sum_{i=1}^B x_i = KG$;
- Step 5. Repeat steps 2 to 4 until a satisfactory assignment is found.

4. VALIDATION AND EVALUATION

In this section, we present a small set of simulation results to illustrate the potential of our GA heuristic. These simulations are done

4	4	4	4	4	4
7	6	5	3	2	1
6	2	6	2	6	1
1	7	1	7	1	7
7	1	7	1	7	1
24	0	0	0	0	0
0	12	0	0	12	0
8	0	8	0	8	0

Table 1: Initial Population for Test Cases 1-4

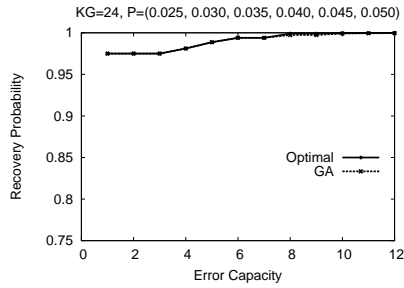
	b_1	b_2	b_3	b_4	b_5	b_6
1	0.025	0.030	0.035	0.040	0.045	0.050
2	0.020	0.050	0.080	0.110	0.140	0.170
3	0.150	0.250	0.350	0.450	0.550	0.650
4	0.250	0.250	0.250	0.250	0.250	0.250

Table 2: Bistro Failure Probability Settings for Each Test Case

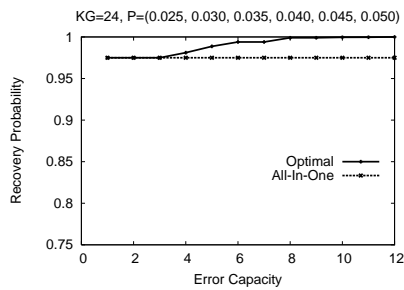
on an Intel Celeron 733MHZ PC with 256MB memory. In these simulations, we create a scenario with $KG = 24$ and $B = 6$. We assign a failure probability to each server. We use a brute-force search program to traverse the whole search space to compute a global optimal assignment. This is done for comparison purposes. We use 6 bistros as we found it feasible for a brute-force approach to compute a global optimum in this case. For more than 6 bistros, the brute-force approach takes an extremely long amount of time to compute the solution. We simulated a number of other test cases with $B > 6$, but omit them here due to lack of space. For each test case, we stop the iterations after T_{long} generations. We set $T_{long} = 50$ in the simulation as we found that in most cases, our GA heuristic converges to a global optimum in less than 10 generations. We believe it is a reasonable choice as 50 generations are considered to be a long convergence time in many GA applications [6].

To also demonstrate the strength of our GA heuristic, we compare it to three simple heuristics. The first simple heuristic, which we term the “all-in-one assignment” strategy, puts all the checksum groups on the most reliable bistro. The second simple heuristic spreads all the checksum groups evenly among the bistros, and we term it the “even assignment” strategy. In the third simple heuristic, we spread the checksum groups among bistros in proportion to their failure probabilities. The more reliable a bistro is, the more checksum groups it will receive. For example, if the failure probability of bistro a is half of the failure probability of bistro b , bistro b will receive half as many checksum groups as bistro a . We term this the “proportional assignment” strategy.

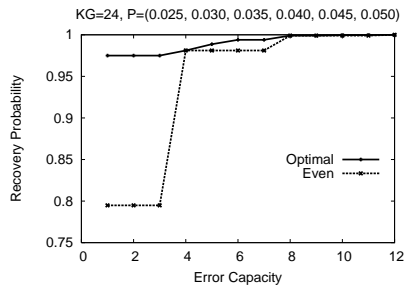
For test cases 1-4 (described below), we use the initial population which is given in Table 1. Table 2 gives the bistro failure probability settings for these test cases. We present our simulation results in Figures 3 - 6. Each figure plots the recovery probability as a function of *error capacity*. Error capacity is defined as the maximum number of checksum groups that the destination server can tolerate to lose and still be able to reconstruct the original file. In each figure, we compare the results of our GA heuristic, “all-in-one assignment”, “even assignment”, and “proportional assignment” with an optimal solution obtained by the brute-force approach.



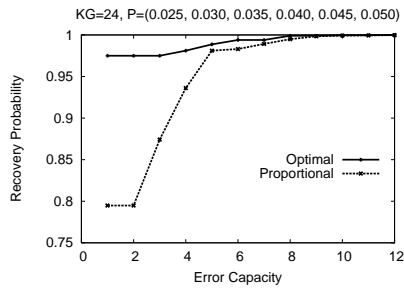
(a) Optimal vs GA Heuristic



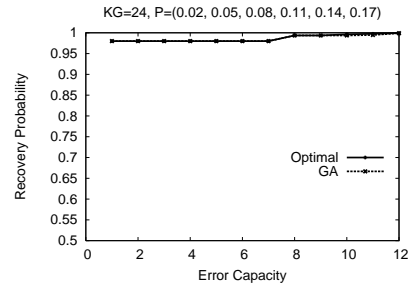
(b) Optimal vs All-in-One Assignment



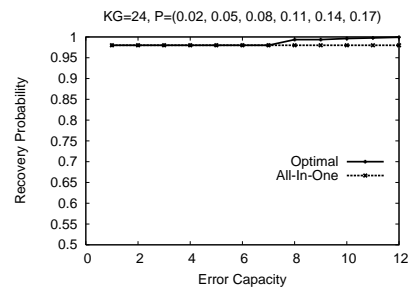
(c) Optimal vs Even Assignment



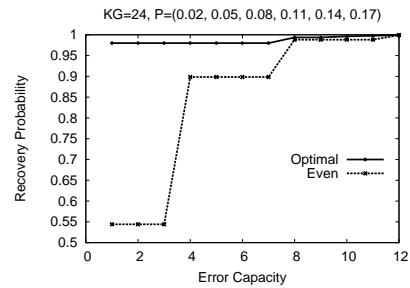
(d) Optimal vs Proportional Assignment



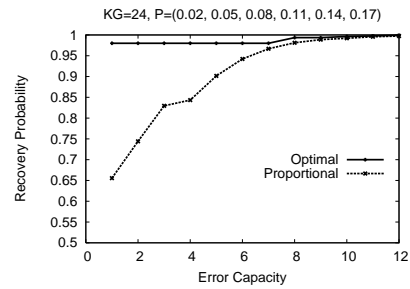
(a) Optimal vs GA Heuristic



(b) Optimal vs All-in-One Assignment



(c) Optimal vs Even Assignment



(d) Optimal vs Proportional Assignment

Figure 3: Test Case 1: A Scenario with Reliable Conditions

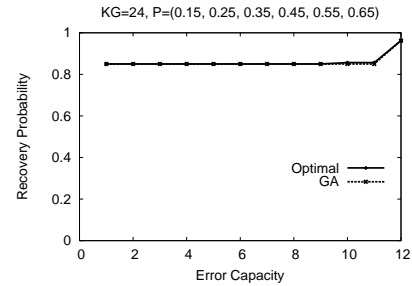
Figure 4: Test Case 2: A Scenario with Reliable Conditions

In test case 1 and test case 2 (Figures 3 and 4, respectively), we assign to each server a small failure probability. Here we try to simulate reliable conditions. Meanwhile, in test case 1, we make the difference between failure probabilities of bistros small while in test case 2, we make it large. In our simulations, we found that our GA heuristic can reach a global optimum in: 7 out of 12 cases in test case 1 (Figure 3(a)) and 10 out of 12 cases in test case 2 (Figure 4(a)). For those cases where our GA heuristic is stuck at a local optimum, it approximates the global optimum by at least 99.5%. We have also found that in these two cases, the three simple heuristics perform worse than the GA based heuristic. The “all-in-one” strategy reaches a global optimum in 3 out 12 cases in test case 1 (Figure 3(b)) and 7 out of 12 cases in test case 2 (Figure 4(b)). The “even” strategy reaches a global optimum in 5 out 12 cases in test case 1 (Figure 3(c)) and 4 out of 12 cases in test case 2 (Figure 4(c)). The “proportional” strategy reaches a global optimum in 4 out of 12 cases in test case 1 (Figure 3(d)) and 3 out of 12 cases in test case 2 (Figure 4(d)). In addition, “even” and “proportional” assignment strategies perform quite poorly in cases where error capacity is less than 4 checksum groups. Their best approximation to the optimal solution is only about 80% (refer to Figures 3 and 4).

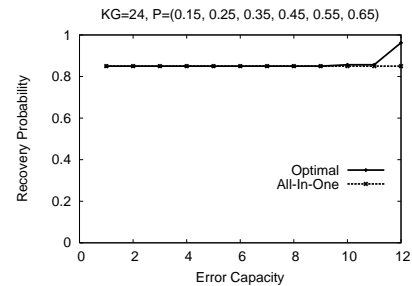
In test case 3 and test case 4 (Figures 5 and 6, respectively), we assign each server a high failure probability. We try to simulate error-prone conditions in which bistros are unreliable. In test case 3, we make the difference in failure probabilities of bistros large while in test case 4 they are the same. In the simulation, we found that our GA heuristic can reach a global optimum in most cases: 10 out of 12 in both test cases (Figures 5(a) and 6(a)). For those cases where our GA heuristic is stuck at a local optimum, it approximates the global optimum by at least 94.1%. We have also found that in these test cases, the “all-in-one” strategy can obtain a fairly good performance while the other two simple heuristics perform poorly. The “all-in-one” strategy reaches a global optimum in 8 out 12 cases in test case 3 (Figure 5(b)) and 7 out of 12 cases in test case 4 (Figure 6(b)), which is close to our GA heuristic. However, the “even” strategy and “proportional” strategy reach a global optimum only in 1 out of 12 cases in both test cases (Figures 5(c)-5(d) and Figures 6(c)-6(d)). In addition, in both test cases, their performance is far below that of our GA heuristic. For example, the “even” and “proportional” assignment strategies perform quite poorly in both cases with error capacity is less than 4 checksum groups. In both cases, their approximations to a global optimum is at most 30%. All these results clearly indicate that the “even” and the “proportional” assignment strategies may not be very useful in an error-prone environment. Our experiments also indicate that the “all-in-one” assignment strategy is not as good as our GA heuristic. It reaches a global optimum fewer times and the achieved approximation is not as good as our GA heuristic in a number of cases.

We also note that when the error capacity is small, an optimal assignment tends to choose the most reliable bistro. With the increase in error capacity, more bistros will be involved in an assignment. It can be explained intuitively as follows. When the error capacity is small, the gain of putting checksum groups on multiple bistros is smaller than the failure risk of those bistros which have a larger failure probability than the most reliable one. However, with an increase in error capacity, this risk can be “reimbursed” by the error capacity and can yield a higher final probability of reconstructing the original file.

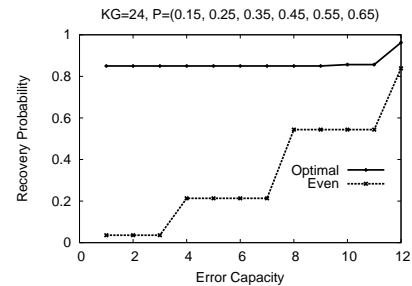
We now consider convergence characteristics of our GA heuristic.



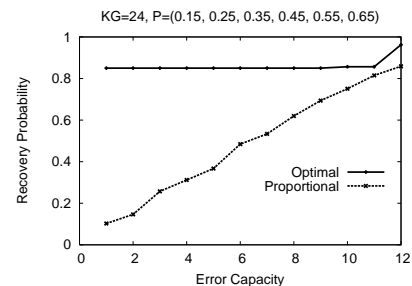
(a) Optimal vs GA Heuristic



(b) Optimal vs All-in-One Assignment

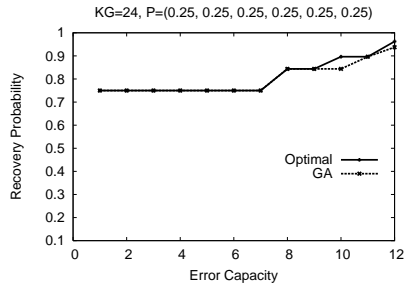


(c) Optimal vs Even Assignment

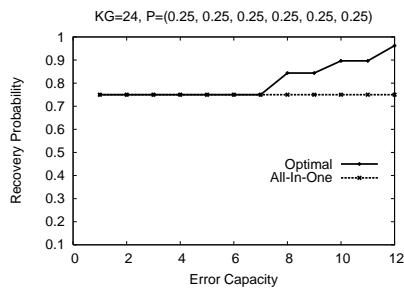


(d) Optimal vs Proportional Assignment

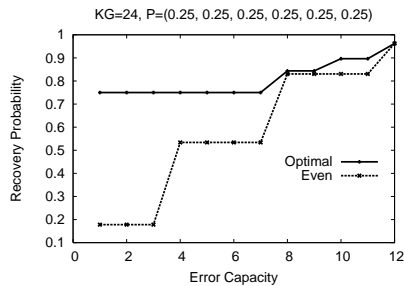
Figure 5: Test Case 3: A Scenario with Error-prone Conditions



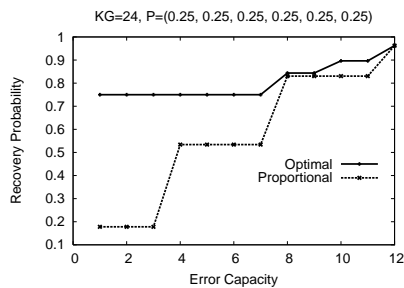
(a) Optimal vs GA Heuristic



(b) Optimal vs All-in-One Assignment



(c) Optimal vs Even Assignment



(d) Optimal vs Proportional Assignment

Figure 6: Test Case 4: A Scenario with Error-prone Conditions

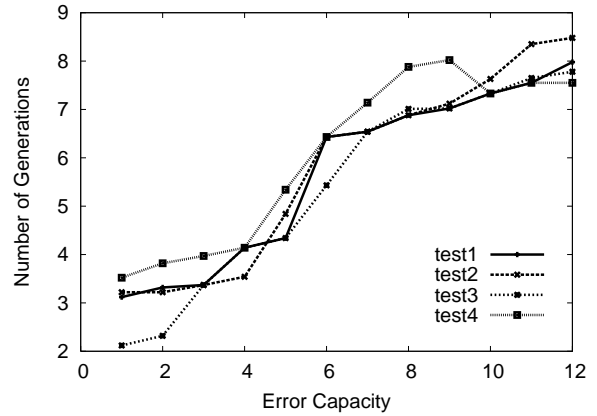


Figure 7: Number of Generations to Converge with Random Initial Population

To this end we generated a number of initial populations by randomly selecting 8 assignments from the entire search space. In each test case, we continue the computation until the GA heuristic reaches the same result as it did in the previous test cases. We record the average number of generations in 100 randomly generated initial populations and present this average number in Figure 7.

From Figure 7, we can find that our GA heuristic converges reasonably fast. In all test cases, the average number of generations to reach the same result as in a fixed initial population case is less than 9. Considering that the whole search space is 24^6 , our GA heuristic only needs to check at most $8 \times 9 = 72$ assignments. According to [6], by introducing good genes into the initial population, we can further speed up convergence of a GA. In the future, we can apply a heuristic in selecting the initial population. In fact, as an initial attempt, we introduced our “all-in-one” and “even” heuristic assignments into our initial population in tests presented in Figures 3 - 6 and we found that our GA heuristic can converge in less than 5 generations in all cases.

From the above results, we can see that our GA heuristic reduces the running time of an NP-hard problem efficiently while still achieving a good approximation to a optimal solution. Although we only presented the results of test cases for 6 servers in this paper, we observed similar results in test cases for more servers. Thus, we believe this genetic algorithms approach is feasible in realistic settings.

5. RELATED WORK

This section briefly discusses related work in the context of data assignment in distributed systems.

RAID [22] is commonly used in distributed storage systems, to provide better fault tolerance and performance. RAID spreads information across several disks, using techniques such as disk striping, disk mirroring, and erasure codes to achieve redundancy, lower latency and/or higher bandwidth for reading and/or writing, and recoverability from hard-disk crashes. In the Bistro fault tolerance protocol, we also stripe data with erasure codes. We can think of RAID as employing the “even” assignment strategy as described above. We have shown that this is not a good approximation for

our application.

An important issue in data striping over the Internet is the data placement problem. That is, which data to place where. With the deployment of large content distribution networks (Inktomi [10], Exodus [14], Digital Island [13]) which provide hosting services to multiple content providers, data placement issues become more and more important. A number of problem formulations are used to characterize and improve different objectives.

In [18], the k-median formulation is used to address the problem of data placement in order to reduce network bandwidth consumption. In our problem, by increasing the probability for the destination server to reconstruct the original file, we can reduce the chance of retransmitting the file and improve reliability and performance. The k-median problem [20] is a well-known NP-hard problem. The k-median formulation is used to address the problem of distributing a single replica over a fixed number of hosts. Some works use a k-median formulation to try to address a performance metric in replica placement [16, 17]. The solution is usually network topology dependent. [21] proposes a bin packing formulation to achieve load balancing in distributing documents in a cluster of web servers. [21] also proposes an algorithm for the initial distribution and network flow formulations in cases where either access patterns change or there is a server failure. This is similar to our problem in that intermediate bistros can fail and we need to maximize the availability of the file. [5] studies the formulation of a file allocation problem which is proved to be NP-Complete in [7]. The file allocation problem is similar to our problem in that it is also a data placement problem and it needs to store N files on M servers in order to optimize a performance parameter, with respect to the storage capacity available at each server.

However, it is difficult to apply the above formulations directly to our problem. Our problem differs from those in the following aspects: (1) in the Bistro system, intermediate bistros are not trusted, while in replication, the replicas are usually put in a trusted server; (2) in the Bistro framework, there is no difference between intermediate bistros while in replication, there is usually a primary replication server which is more important than other servers; (3) in file allocation problems, there are normally storage capacity constraints but in our model, we do not focus on this constraint; and (4) our problem is modeled on the application layer and is network topology independent.

Genetic algorithms have been used to solve various optimization problems including graph partitioning [2], multiprocessor document allocation [8], and file allocation [19]. We took advantage of their ability to explore, fast and efficiently, the solution space of a problem in order to design our heuristic for the data assignment problem. To the best of our knowledge, we are the first to apply genetic algorithms to the data assignment problem in many-to-one applications. The genetic operations in our GA heuristic are novel and can explore the search space quickly.

6. CONCLUSIONS

In this paper, we formulated a data assignment problem, in the context of the Bistro fault tolerance protocol as a non-linear optimization problem. We also proposed a genetic algorithm heuristic to solve this problem. We use simulations to evaluate our approach and compare the results with other simple heuristics as well as with an optimal solution obtained through a brute-force approach. Our results indicate that the proposed GA heuristic is efficient to pro-

vide a good approximation. We believe that the proposed approach is feasible and can result in a better fault tolerance framework.

There are still several open research issues in the Bistro fault tolerance protocol. One such problem is how to integrate performance metrics with the data assignment problem. In this paper, we address the data assignment problem from a reliability perspective. However, performance is another perspective we need to consider. For example, clients have different bandwidth constraints to intermediate bistros and will get different response times in transferring the same number of checksum groups. We are currently investigating these issues.

7. REFERENCES

- [1] S. Bhattacharjee, W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. Bistro: a framework for building scalable wide-area upload applications. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):29–35, September 2000.
- [2] R. Chandrasekharam, S. Subhranian, and S. Chaudhury. Genetic algorithm for node partitioning problem and applications in VLSI design. *IEEE Proceedings*, 140(5):255–260, September 1993.
- [3] W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. A secure and scalable wide-area upload service. In *Proceedings of 2nd International Conference on Internet Computing*, volume 2, pages 733–739, June 2001.
- [4] L. Cheung, C.-F. Chou, L. Golubchik, and Y. Yang. A fault tolerance protocol for uploads: Design and evaluation. In *Second Int. Symposium on Parallel and Distributed Processing and Applications*, December 2004.
- [5] W. Chu. Optimal file allocation in a multiple computer system. *IEEE Trans. on Computers*, c-18(10), 1999.
- [6] D. A. Coley. *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific Publishing Company, 1997.
- [7] K. P. Eswaran. Placement of records in a file and file allocation in a computer network. *Information Processing*, pages 304–307, 1974.
- [8] O. Frieder and H. T. Siegelmann. Multiprocessor document allocation: A genetic algorithm approach. *IEEE Trans. on Knowledge and Data Engineering*, 9, no.4, July/Aug 1997.
- [9] M. Garey and D. Johnson. *Computers and Intractability*. W.H. Freeman, 1979.
- [10] J. Heidemann and V. Visweswaraiiah. Automatic selection of nearby web servers. In *1998 SIGMETRICS/Performance Workshop on Internet Server Performance*, June 1998.
- [11] J. H. Holland. Robust algorithms for adaptation set in a general formal framework. In *Proc. IEEE Symposium on Adaptive Processes-Decision and Control*, pages 5.1–5.5, 1970.
- [12] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [13] <http://www.digitalisland.com>. In *Digital Island*.

- [14] <http://www.exdous.com>. In *Exodus*.
- [15] IRS. *Fill-in Forms*.
<http://www.irs.gov/formspubs/lists/0,,id=97817,00.html>, 2005.
- [16] S. Jamin, C. Jin, Y. Jin, D. Riaz, Y. Shavitt, and L. Zhang. On the placement of Internet instrumentation. In *Proc. of the IEEE INFOCOM'00 Conf.*, March 2000.
- [17] S. Jamin, C. Jin, T. Kurc, D. Riaz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proc. of the IEEE INFOCOM'00 Conf.*, April 2001.
- [18] V. P. L. Qiu and G. Voelker. On the placement of web server replicas. In *Proc. of the IEEE INFOCOM'01 Conf*, April 2001.
- [19] T. Loukopoulos and I. Ahmad. Static and adaptive data replication algorithms for fast information access in large distributed systems. In *Proc. of the 20th IEEE Int. Conf. on Distributed Computing Systems*, 2000.
- [20] P. Mirchandani and R. Francis. *Discrete Location Theory*. John Wiley and Sons, 1990.
- [21] B. Narendran, S. Rangarajan, and S. Yajnik. Data distribution algorithms for load balanced fault-tolerant web access. In *Proc. of the 16th Symposium on Reliable Distributed Systems (SRDS '97)*, October 1997.
- [22] D. A. Patterson, G. Gibson, and R. H. Katz. 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. ACM Press, 1988.