Improving the Performance of Data Servers Using Array Codes *

Lihao Xu   Jehoshua Bruck
California Institute of Technology
Mail Code 136-93
Pasadena CA 91125
Email: {lihao,bruck}@paradise.caltech.edu

Abstract

This paper discusses improving performance (throughput) of data server systems by introducing proper data redundancy into the system. General performance properties of a server system with redundant data are described. We show that *proper* data redundancy in a server system can significantly improve the performance, in addition to the reliability of the system. Two problems related to the performance together with their solutions are proposed, namely, the problems of efficient *data distribution* scheme for the *servers* and *data acquisition* scheme for the *client*. Both schemes utilize array codes, a class of error-correcting codes whose encoding and decoding procedures only use simple binary exclusive-OR operations, which can be implemented efficiently in software and/or hardware. Construction of general MDS array codes suitable for the both schemes is discussed. A new property of MDS array codes, called the *strong MDS* property, is also defined to improve the data acquisition performance. A method for modeling data server performance and the related experimental results are presented as well.

Keywords: Distributed Data, Information Dispersity, Data Efficiency, Array Codes, strong MDS

*Supported in part by the NSF Young Investigator Award CCR-9457811 , by the Sloan Research Fellowship, and by DARPA through an agreement with NASA/OSAT.

1 Introduction

It has become a trend to use a cluster of distributed computing nodes in server systems, such as image servers, video servers, multimedia servers, and web servers, all of which can be generally regarded as data servers. Distributed server systems can improve both data *reliability* (or availability) and *performance* (or efficiency, i.e., data throughput of the system). Much research has been done to improve the reliability by introducing *data redundancy* or *information dispersity*[9] into the systems, such as the well known k-out-of-n systems[1].

In a real system, while the redundant data enables the system to provide continuous service when certain failures (communication link failure, server node failure) happen, most of the time the system works in a normal mode, i.e., there is no failure in the system; in this case the data redundancy can be used to improve the performance of the system. It was first shown in [5] that redundant data can make packet routing more efficient by reducing the mean and variance of the routing delay. Recently, more scalable and efficient reliable multicast schemes have also been proposed based on the data redundancy in the messages to be multicast[6]. In database community, it also has been known that data redundancy can help improve data throughput [8]. However, these previous work only dealt with simple redundancy schemes, i.e., replication using only mirroring. While it is conventional wisdom that data redundancy is helpful to improve the performance of a data server system, it has not been well studied that how much data redundancy will *optimize* the performance of a data server system when the total *resource* (number of servers) of the system is fixed. This optimization problem will be investigated in this paper, called the *data distribution problem*. Another novelty of this paper is to discuss how the *client* should utilize the data redundancy in the server system to optimize the data throughput to the client. This problem is called the *data acquisition problem* in this paper. Our solutions to the problems are based on array codes, a class of error-correcting codes, which provide a more general way of distributing redundant data than the simple mirroring scheme and the simple parity scheme, and thus give more flexibility to both the data distribution and data acquisition schemes. This will be explained in details later. We will also explore ways of constructing more general array codes and a nice property of array codes, called the *strong MDS* property, which provides more solutions to the data acquisition for the client.

Our system setup is shown in Figure 1: a cluster of servers are connected via some reliable communication network. In addition, broadcast is supported over the network so that a client can broadcast its request for certain data to some or all of the n servers in the system. The data is distributed over the servers in such a way that a client can obtain the complete requested data after it gets data from at least *any* k of the n servers. Such a distributed data server systems is called an (n, k) server system in this paper. Implementation of (n, k) systems can be achieved by using error-correcting codes.

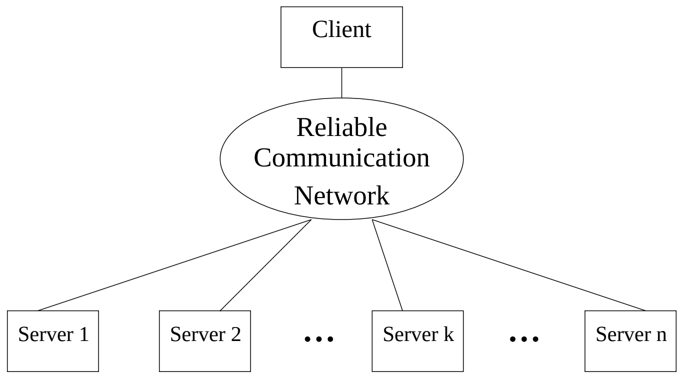


Figure 1: An (n, k) server system

1.1 Array Codes

An *error-correcting* code is a method to place data in such a correlated way that if there are some faulty data symbols (*error*), they can be detected and/or corrected. In particular, if some data symbols are missing (these symbols are then called *erasures*), they can be reconstructed (this reconstruction procedure is called the *erasure-correcting*)[7]. To achieve this goal, some redundant data (*parity*) must be introduced so that when some data are missing, the amount of the remaining data is still no less than that of the original raw (*information*) data, which makes it possible for a full reconstruction of the information. In order to make maximum use of the redundant data, it should be possible to reconstruct the original information data from any remaining data as long as its amount equals the amount of the original data. Mapping it to an (n, k) server system, the ideal size of the data stored on each server should be only $\frac{1}{k}$ of that of the whole requested data. This can be achieved by a special class of error-correcting codes: MDS (*Maximum Distance Separable*) codes, such as the *Reed-Solomon codes*[7].

In a real system, however, in addition to the MDS property, the computations needed for constructing and distributing the redundant data (or *encoding*) and for assembling the data from k servers into a whole data (or *decoding*) should be as simple as possible. Array codes are ideal for this goal[3][4]. The only operations needed for encoding and decoding are simple binary exclusive-or (*XOR*) operations, which can be implemented efficiently in hardware and/or software. Some MDS array codes have been found. For example, the EVENODD code[2] can be used for a general (n, k) system. Recently two classes of MDS array codes with optimal number of encoding and decoding operations have also been found, which can be applied to an $(n, n-2)$ or $(n, 2)$ server system[10][11]. Table 1 shows such a code that can be used for a data placement scheme for a $(6, 4)$ system: the original data is equally partitioned into 12 pieces (*symbols*), which are represented as a through f, and A through F, where the addition (+) operation is just the binary exclusive-OR (XOR) operation. The symbols in column i (where $i = 1, \cdots, 6$) represent data stored on the ith server. It is easy to verify that the original 12 data symbols can be *reconstructed* from 12 data symbols of *any* 4 columns (or servers). Notice that the number of data symbols used to reconstruct the original data symbols *exactly* equals the number of the original data symbols, i.e., the array code is an MDS code and the redundant data is used to the maximum extent.

a	b	c	d	e	f
A	B	C	D	E	F
$B+D+e+f$	$C+E+f+a$	$D+F+a+b$	$E+A+b+c$	$F+B+c+d$	$A+C+d+e$

Table 1: A data placement scheme for a (6,4) system

Example 1 *For the above (6,4) system, the original data that a client needs is 12-bit long:* 111010101010, *then using the code in Table 1, the data placement array for the 6 servers is as follows:*

1	1	1	1	1	1
1	0	0	0	0	0
0	0	0	1	0	1

The ith server stores the bits in the ith column of the above array, i.e., the 1st server stores 110, *and the 2nd server stores* 100, *and so on.* □

1.2 Performance Improvements

For a data server system, since the I/O speed of the local disks is much slower than the CPU speed of the servers, the whole performance of the system is dominated by the bottleneck of the the disks. To overcome this bottleneck, distributing data over multiple servers can help improve the data throughput of the whole system, since the data can be accessed in parallel from multiple servers at the same time. For a server system with n servers, the system performance can further be improved by introducing proper redundant data into system, i.e., an (n, k) system should be chosen instead of naively distributing the raw data over all the n servers.

Now suppose $n = 6$ and the total amount of the data that a client requests is 12 Kbytes, then in a $(6, k)$ system, the data stored on each server is $\frac{12}{k}$ Kbytes using MDS array codes, where k can range from 1 to 6.The delivery time model is as follows: the time of each server to deliver m Kbytes of data to the client is $bm + 0.8$ msecs, where for simplicity, b is a random variable uniformly distributed over $[0.3, 0.5]$ msecs/Kbytes. The following examples show the performance for different k.

Example 2 *For $k = 6$, then each server stores 2 Kbytes data, and there is* no *redundant data in the whole system. Suppose the delivery time of the 2 Kbytes of data from each server is $1.50, 1.45, 1.65, 1.75, 1.55, 1.70$ msecs respectively, then the client has to wait for 1.75 msecs until it can obtain the whole 12 Kbytes of data it requested.*

Now choose $k = 5$, then the 12 Kbytes of data is even distributed over 5 servers, and the rest server stores the parity of the data stored on the first 5 servers. In this case, each server has 2.4 Kbytes data, and the total amount of redundant data in the system is 2.4 Kbytes. Now suppose the delivery time of the 2.4 Kbytes of data from each server is $1.55, 1.65, 1.60, 1.72, 1.70, 1.90$ msecs respectively. Since now it can obtain the 12 Kbytes data from any 5 servers, the client only needs to wait for 1.72 msecs, which is shorter than the $k = 6$ case. □

This shows that data redundancy can improve the system's performance, as is already known intuitively. But the following question is less obvious: can more redundancy provide more performance improvement when the total number of the servers in the system is fixed?

Example 3 *Now choose $k = 4$. Then use the placement scheme in Table 1 , each server stores 3 Kbytes of data, and the total amount of the redundant data in the system is now 6 Kbytes. Say now the delivery time of the 3 Kbytes data from each server is respectively $1.72, 1.78, 2.20, 1.82, 2.10, 1.85$ msecs, then the total time the client needs to wait is 1.85 msecs, which is worse than the choices in the previous example.* □

So in the above examples, with the above data delivery time model, a $(6, 5)$ system gives the best performance. What is the *proper* redundancy when the total number of the servers is given? Or how to determine k when the n is given in order to achieve the best system performance? This is the so-called *data distribution* problem at the servers' side, which will be discussed in detail later in this paper.

We will also discuss another problem called *data acquisition* at the client's side, that once data redundancy is properly distributed among the servers, optimal mean service time can be reached by choosing matching read approaches. The following example gives a flavor of this problem.

Example 4 *A client requests* 12 *Kbytes of data from a* $(6, 4)$ *system. The distribution scheme in Table 1 is used, and the delivery time model remains the same as in the above two examples. Now each server stores* 3 *Kbytes of data,* 1 *Kbyte of which is the redundant data as in the last row of Table 1. The client has two options to read the* 12 *Kbytes of data from the servers: 1) request* 2 *Kbytes of data from each of the* 6 *servers, i.e., the* 12 *original data symbols in the top 2 rows in Table 1. In this case the client needs to gather data from all the* 6 *servers; 2) request* 3 *Kbytes of data from all the servers, then the client only needs to wait for data from any* 4 *servers. Suppose the data delivery times for* 2 *Kbytes and* 3 *Kbytes of data from each server are the same as in the above two examples, then the client needs* 1.75 *msecs if it chooses option 1 and* 1.85 *msecs if it chooses option 2. So in this case, option 1 gives better performance.* □

1.3 Organization of This Paper

The main contribution of this paper are 1) propose the data distribution and the acquisition schemes for a given server system, to improve the system performance. The schemes are based on both analytical and experimental results; 2) explore ways of constructing general MDS array codes that are suitable to the data distribution and the data acquisition schemes; 3) propose a new property for MDS array codes that gives flexibility to the data acquisition scheme; and 4) propose a probability model for a real data server system.

This paper is organized as follows. Section 2 first describes a performance measurement parameters for distributed server systems, then experimental results are utilized to create a probability model for service time in a practical disk-based data server system. Some analysis results related to general distributed server systems are given in the appendix. The data distribution and data acquisition schemes are discussed in more detail in Section 3. Section 4.2 further discusses constructing more general MDS array codes from the well-known Reed-Solomon codes, and proposes a new property for MDS array code, called the *strong MDS* property, which is very useful for the data acquisition scheme. Section 5 concludes the paper, and proposes a future research direction.

2 Server Performance Model

Define the service time T_i of the server i $(1 \leq i \leq n)$ to be the elapsed time from when the client sends its request to the server i to when it receives data from the server i. Notice that T_i does *not* include the time needed at the client's side to do any necessary *computations* to recover the final data, since here we assume that the computations of array codes are rather simple and thus take much less time compared to the data delivery time through the local disk and communication media. We model T_i as a continuous random variable with *probability density function* (pdf) $f_i(t)$. For the simplicity of analysis, we assume that all T_is are i.i.d (*independent identically distributed*) random variables, i.e., $f_i(t) = f(t)$, $1 \leq i \leq n$. Now let $T(n,k)$ be the elapsed time from when the client broadcasts its data request to the servers to when it receives data from at least k out of the n servers. Then $T(n,k)$ is another random variable, which is a simple function of all T_is:

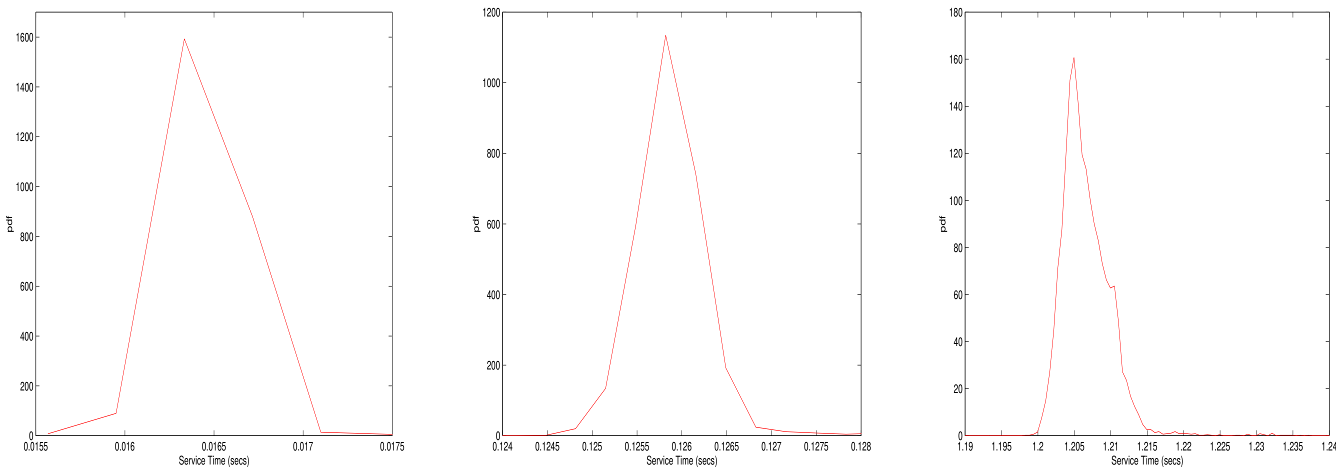
$$T(n,k) \geq T_i, \quad where \ \|\{i\}\| \geq k$$

The mean $E[T(n,k)]$ of $T(n,k)$ is a good measurement of the server system's performance. It is a function of the pdf $f(t)$ of an individual server's data service time. The calculation of $E[T(n,k)]$, together with some mathematical relations among $E[T(n,k)]$, n, k and $f(t)$ are discussed in the appendix. The goal of the data distribution and data acquisition is to reduce $E[T(n,k)]$ under various conditions. Before we propose the data distribution and acquisition schemes, it is necessary to establish a model for $f(t)$.

2.1 Abstraction from Experiments

The data service time T depends on many factors in a practical server system, such as computing power (i.e., CPU speed) of the servers, local disk I/O speed of the servers, bandwidth and latency of the communication medium (usually including a reliable communication software layer) connecting the servers and the client, the client's computing power. A model considering all the factors will be fairly complex. In this paper, we will try to model a simple probability distribution of the data service time, that can be analyzed rather easily, and yet can approximate the real data service time closely. Such a model will be abstracted from experimental results of a real data server system.

Our practical server system consists of several servers, which are PCs running Linux. Each server has data stored on its local hard disk. Data is accessed via the Linux file system. A client is a PC running the same Linux as well. The nodes are connected via Myrinet switches. A *sliding window* protocol is used to ensure reliable communication. Experiments are conducted in such a real system to measure the service time for data of different sizes. The procedure of the experiment is as follows: (1) the client sends a request for a certain amount of data to a server; (2) the server reads the data from its local disk and sends it to the client through the reliable communication layer; (3) the data is delivered to the client through the reliable communication layer. The data service time is measured from the instant that the client finishes sending its request to the instant that the client gets the data. We run the above procedure a few thousand times for data of a certain size, and get its pdf according to the frequencies of different ranges of service time. Figure 2 shows such empirical pdfs of the service time for data of the size (a) 32 Kbytes, (b) 320 Kbytes and (c) 3200 Kbytes.



(a) *data size* $= 32Kbytes$ (b) *data size* $= 320Kbytes$ (c) *data size* $= 3200Kbytes$

Figure 2: Empirical pdfs of service time for data of different sizes

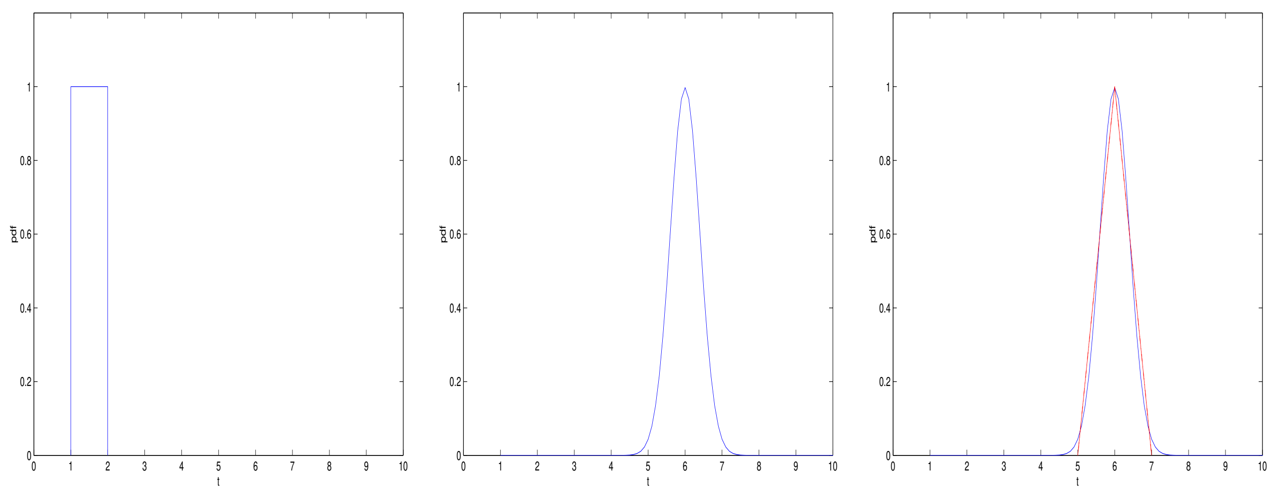
The effective data bandwidths in this experiments are quite low, since they are the concatenation of the local disk bandwidth and the reliable communication layer bandwidth. But the shape of the bandwidth pdfs is more interesting. The experiment results show that the shape of empirical pdfs of different data size can be approximated by the same distribution. A closer look shows that the width of the distribution base is approximately *proportional to* the data size. More complex distributions, such as Gamma distribution and Beta distribution, might give more accuracy. But to simplify the analysis, that follows, we will regard the data service time T as a random variable defined on $[a, b]$ (a and b are two parameters of a real system), whose pdf is a triangular distribution, denoted as $Tr[a, b]$:

$$f(t) = \begin{cases} \frac{4(t-a)}{(b-a)^2} & a \leq t \leq \frac{a+b}{2} \\ \frac{4(b-t)}{(b-a)^2} & \frac{a+b}{2} < t \leq b \end{cases} \tag{2.1}$$

or its cdf (*cumulative distribution function*) is

$$F(t) = \begin{cases} \frac{2(t-a)^2}{(b-a)^2} & a \leq t \leq \frac{a+b}{2} \\ 1 - \frac{2(b-t)^2}{(b-a)^2} & \frac{a+b}{2} < t \leq b \end{cases} \tag{2.2}$$

One explanation for this model can be as follows: in a real system, data is delivered in packets of some small size. The delivery time of ith packet is a random variable t_i, whose probability distribution can be characterized by a uniform distribution over some time span, and also model the t_is as i.i.d. random variables. Then the service time T of the whole data is: $T = s + \sum_i t_i$, where s is another random variable of uniform distribution describing the setup (or overhead) time for sending a certain amount of data. Thus the pdf of T is a *Gaussian-like* function, whose base width is approximately proportional to the number of the packets in the data, which in turn is proportional to the data size. For simplicity, we approximate the Gaussian-like function by a suitable triangular function. The distributions are shown in Figure 3.



(a) (b) (c)

Figure 3: Probability distributions of data service time of (a) single packet, (b) whole data, and (c) approximation with $Tr[a, b]$

2.2 Verification with T(n,1)

Intuitively, more redundant servers should provide more performance improvement when the amount of the data stored on each server is fixed, i.e., $E[T(n, k)]$ decreases as n increases and/or k decreases. More accurate relations among $E[T(n, k)]$, n and k are discussed in the appendix. Now by applying the service time distribution Eq.(2.1), Eq.(2.2) and the calculation of $T(n, k)$'s pdf Eq.(A.2), we can get pdfs of the $T(n, k)$ for a data server system. Figure 4(a) shows the pdfs of $T(n, 1)$, where $1 \leq n \leq 3$ and T is of the triangular distribution $Tr[1, 2]$. Here we can see the pdf of $T(n, k)$ shifts left as n increases.

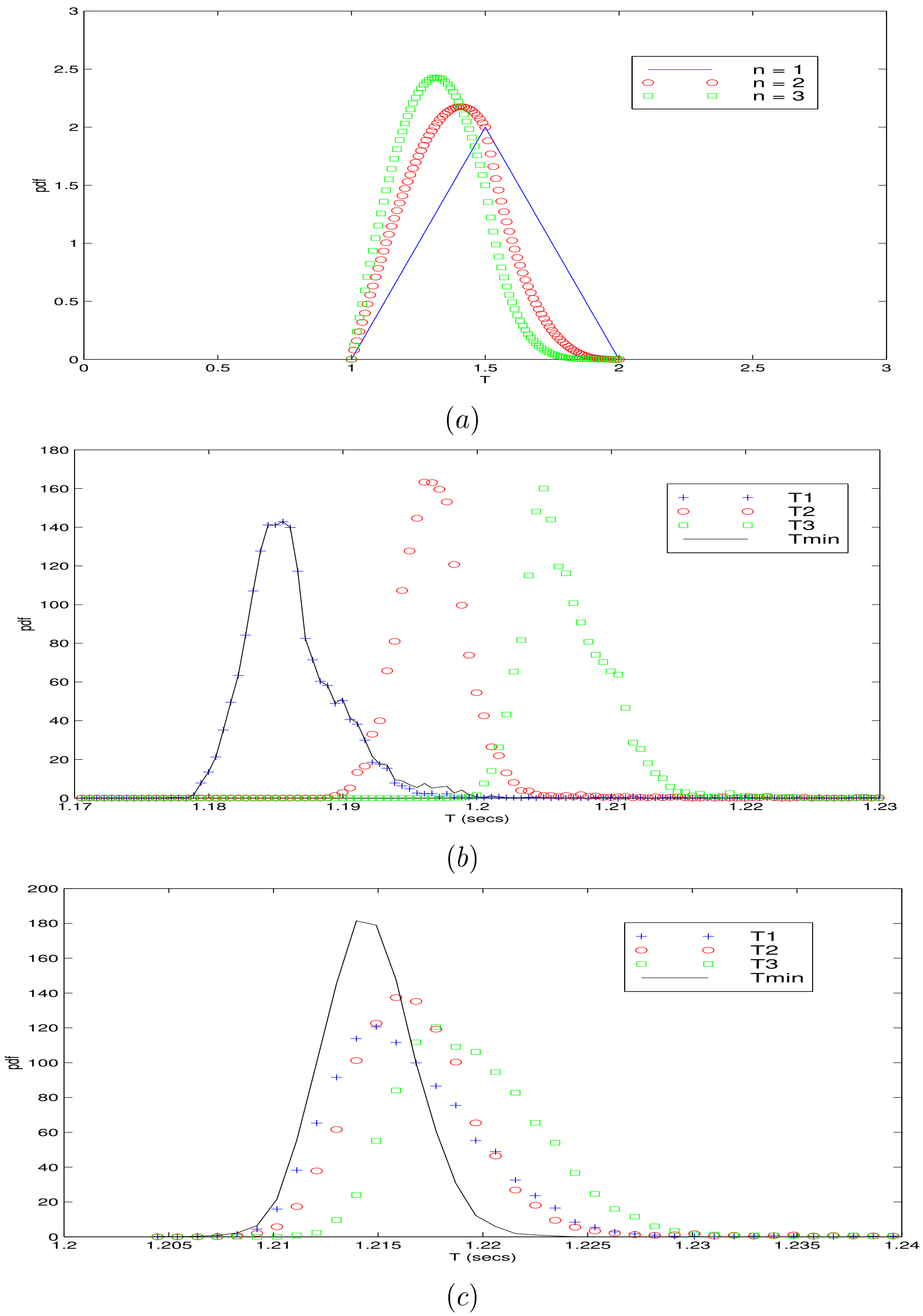


Figure 4: pdfs of $T(n, 1)$: (a) analytical result, where the pdf of T is $Tr[1, 2]$, and experimental service time for data of size 3200 Kbytes, where (b) no other loads on the servers, and (c) other random loads on the servers

To further verify the properties of $E[T(n,k)]$, simple experiments of $T(n,1)$ are done in a real server system as described in previous subsection. The system consists of three servers. In order to remove other factors which also affect data the service time, such as the contention in the communication medium (including the reliable communication layer, which is a bottleneck if we only use one client which communicates with the three servers), we use three clients, each of which is served by a separate server. Conceptually the three clients are regarded as only a single client, thus the whole data service time is the minimum of the three individual service time of the server-client pairs. Figure 4(b) shows service time (T_1, T_2, and T_3) of the three individual server-client pairs for 3200 Kbytes data each. Since the variance among the three pairs is bigger than the variance within each pair, intuitively the whole service time (T_{min}), which is the minimum of the three, is determined by the service time of the best pair, and the experimental result shows so. In this case, the pdf of T_{min} is very close to that of T_1. To make the experimental results more interesting, some random loads are added to each server so that the variance among the three pair is less than the variance within each pair, i.e., each pair behaves more similarly. The service time of three individual pairs (T_1, T_2, and T_3) and the whole service time (T_{min}) are shown in Figure 4(c). Of the four pdfs (T_{min}, T_1, T_2 and T_3), that of T_{min} is the leftmost, which supports the analytical properties of $T(n,k)$ and the pdf model of T.

3 Design An Efficient System

The performance of a server system with redundancy can be improved in two dimensions: (1) given the total number of the servers in the system, data can be distributed wisely among them so that the service time using all the servers is minimized. This is the data distribution problem, which consists of determining k and choosing proper MDS array codes accordingly; (2) once the data distribution is set, data should be read out from servers wisely so that the whole service time is minimized. This is the data acquisition problem.

The data distribution problem is at the servers' side, while the data acquisition problem is at the client's side. Because of the properties of $E[T(n,k)]$, the two problems are uncorrelated, thus they can be dealt with separately.

3.1 Data Distribution Scheme

In a server system, given the total number of the servers, n, we need to determine the number of the servers k which store the *raw* data in order to maximize the performance of the whole system (i.e., to minimize the mean service time of the client's data request); given k, the rest of the servers can store the *redundant* data. When n and the pdf $f(t)$ are fixed, $E[T(n,k)]$ monotonically decreases as k decreases. This means k should be as small as possible in order to make $E[T(n,k)]$ small. On the other hand, however, the smaller the k is, the more data needs to be stored on each server, which means higher service time from each server, since the total amount of the data a client needs is always fixed. Our goal is to find such a k that the $E[T(n,k)]$ is minimized when the both sides of the problem are considered.

After the parameter k is determined, in order to achieve optimal performance in terms of $E[T(n,k)]$, we can always use MDS array codes to distribute the redundant data so that data from *any* k servers can be assembled to form the whole requested data, as was already shown in the introduction. Now the remaining problem is to determine k to minimize $E[T(n,k)]$. Use the pdf model of each server's service time T and MDS codes for distributing the redundant data, if the pdf of T is $Tr[a,b]$ when k=1, then for general k, the corresponding pdf is $Tr[\frac{a}{k}, \frac{b}{k}]$, since the base width of the pdf is proportional to the data size. Theoretically, the optimal k can be calculated as follows:

$$k_{min} = argmin_k \int_0^\infty k \binom{n}{k} F(t)^{k-1}[1-F(t)]^{n-k} t f(t) dt \qquad (3.1)$$

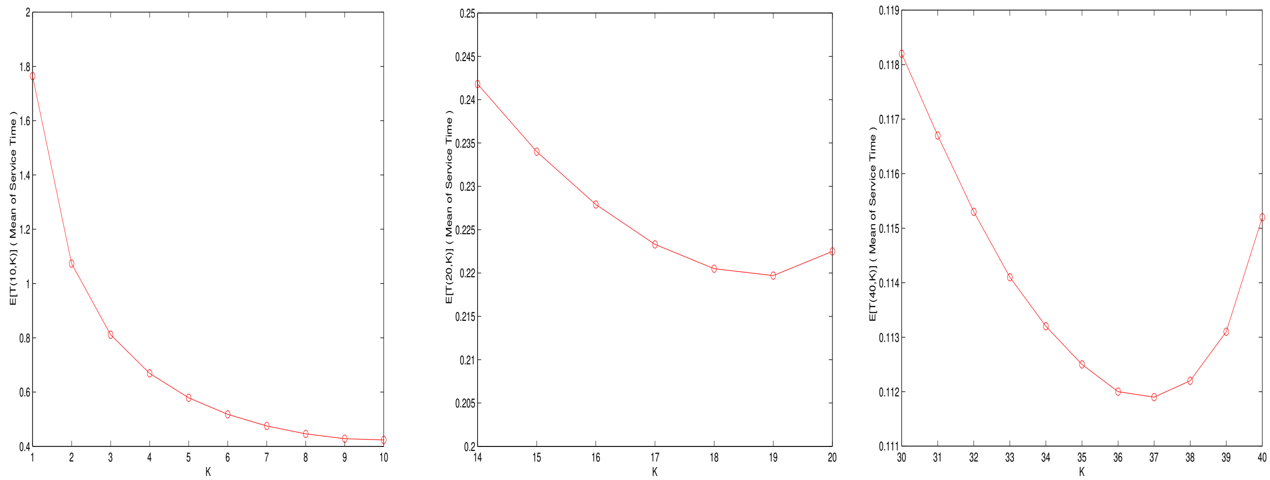
where $f(t)$ and $F(t)$ are as in Eq.(2.1) and Eq.(2.2) except that a and b should be replaced with $\frac{a}{k}$ and $\frac{b}{k}$ respectively. Notice that k_{min} is a function of the pdf $f(t)$, not only the mean $E(T)$ and the variance $Var[T]$.

Even for a simple pdf such as the $Tr[a,b]$, the above equation can not be solved in a closed form. But for a practical system, the system parameter a and b can be determined by experiments, then the above equation can be solved numerically. Figure 5 gives several examples of solving the above equation. In the examples, $a = 1$ and $b = 5$. For $n = 10$, 20, and 40, $E[T(n,k)]$ is calculated for $1 \le k \le n$. The results are shown in Figure5(a)(b)(c), where (b) and (c) only show the last a few numbers for k, since for small k, $E[T(n,k)]$ monotonically decreases as k increases. From the results, we can see (a) $k_{min} = 10$, when $n = 10$, (b) $k_{min} = 19$, when $n = 20$, and (c) $k_{min} = 37$, when n = 40.

Even though the above examples use specific pdfs, the same method also apply with other pdfs by plugging suitable $f(t)$ into Eq.(3.1). Thus for a given practical server system, such a k_{min} can always be found. Proper MDS array codes can then be used based on the (n,k) pair. Thus we get an optimal data distribution scheme for a given server system.

3.2 Data Acquisition Scheme

Once the data distribution scheme is set, i.e., k is determined and the proper MDS array code is chosen, the client needs to decide the way of requesting (or reading) its data. In general, a



$(a)\ n = 10$ $(b)\ n = 20$ $(c)\ n = 40$

Figure 5: $E[T(n, k)]$ vs. k for different n, where $a = 1$ and $b = 5$

client should send its request to as many servers as possible, and also makes the amount of data it needs from *each* server as small as possible, since the properties of $E[T(n, k)]$ show that more redundancy brings more performance. For a specific distribution scheme, the client needs to calculate the pdfs of all possible data read schemes, and then choose an optimal read scheme. Since the read schemes are closely related to the MDS array code being used, here we will give an example using a specific code to show the guidelines of choosing an optimal read scheme.

In this example, the server system has $2n$ servers and client's requested data can be assembled from any $2n-2$ servers, i.e., this is a $(2n, 2n-2)$ system. B-Code[11] can be used for implementing this system. The data distribution using the B-Code is as follows: (1) the whole raw (*information*) data is partitioned into $2n(n-1)$ blocks of equal size (some paddings are added if necessary); (2) each of the $2n$ servers stores $n-1$ blocks of the data; (3) $2n$ blocks of redundant (or *parity*) data are calculated according to the encoding rules of the B-Code, i.e., each parity block is an XOR sum of suitable $2n-2$ raw data blocks, and then each server stores 1 parity block. The structure of the B-Code is shown in Figure 6:

The MDS property of B-Code gives 3 schemes of constructing the whole raw data from the data stored on $2n$ servers, each of which has $n-1$ blocks of raw data and 1 block of parity data: (1) read from all of the $2n$ servers, each of which sends its $n-1$ blocks of raw data; (2) read from any $2n-2$ servers, each of which sends all of its n blocks of data (including raw and parity data); (3) read from all of the $2n$ servers, each of which sends all of its n blocks of data. The 3

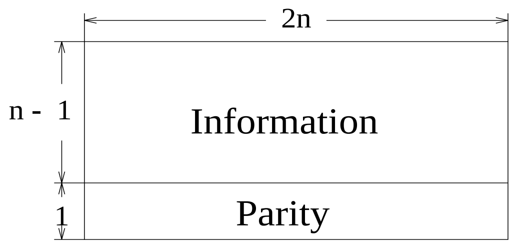
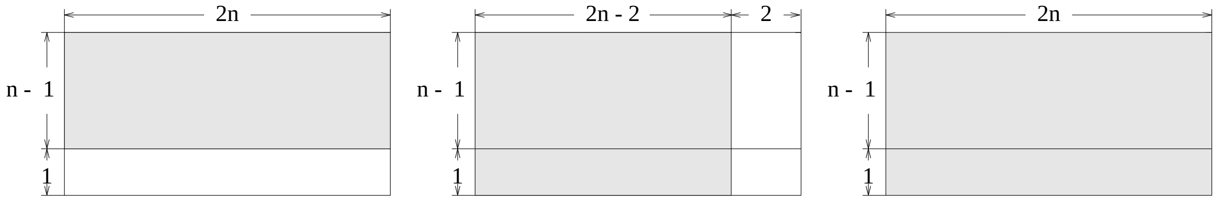


Figure 6: Structure of the B-Code

schemes are shown in Figure 7 where the shaded parts are the data to read.



(a) *Scheme* 1 (b) *Scheme* 2 (c) *Scheme* 3

Figure 7: Three read schemes using the B-Code

Notice that there is no redundant data in scheme (1) and scheme (2), the client must wait until it receives all the data from the requested servers. But in scheme (3), there is redundant data, then the client only needs to receive all data from any $2n-2$ of the $2n$ requested servers. Let the mean data service time of the three schemes to be $E[T(2n,2n)]_{n-1}$, $E[T(2n-2,2n-2)]_n$ and $E[T(2n,2n-2)]_n$ respectively. From Property 1 of Theorem 1 in the appendix, $E[T(2n-2,2n-2)]_n > E[T(2n,2n-2)]_n$. But the relation between $E[T(2n,2n)]_{n-1}$ and $E[T(2n-2,2n-2)]_n$ or $E[T(2n,2n-2)]_n$ is not so obvious, since in scheme (1) while it needs to wait for more servers, the client also needs less data from each server, thus less service time from each individual server. So to determine which scheme is the best scheme for a given system, we need to calculate the pdf of the whole service time for all the schemes.

Assume that the pdf of service time T for each server sending n blocks of data to the client is $Tr[a,b]$, then the pdf of T in scheme (1) is $Tr[\frac{n-1}{n}a, \frac{n-1}{n}b]$, since each server only needs to send $n-1$ blocks of data, and the pdf of T in scheme (2) or (3) is $Tr[a,b]$. Now the pdfs of the whole service time in the different schemes can be calculated according to Eq.(A.2), Eq.(2.1) and Eq.(2.2). Figure 8 shows the pdfs of different n, where $a=1$ and $b=10$.

Then using Eq.(A.3), the mean of the whole service time in different schemes can also be calculated. They are listed in Table 2, when $a=1$ and $b=10$.

n	3	7	10
$E[T(2n,2n)]_{n-1}$	**5.2195**	7.3128	7.8857
$E[T(2n-2,2n-2)]_n$	7.4089	8.4207	8.6976
$E[T(2n,2n-2)]_n$	5.8910	**7.2466**	**7.6786**

Table 2: Mean service time of different data read schemes, where $a = 1$, and $b = 10$

The above calculations show that the performance of the three schemes depend on the system parameter n (when a and b are fixed). In a small server system (6 servers when $n = 3$), scheme (1) is the best. As n increases, scheme (3) becomes better. For the system of 14 servers ($n = 7$) and 20 servers ($n = 10$), scheme (3) is the best.

Though quite simple, the above example shows that after the data distribution is set at the servers' side, the client has different ways of reading data from the servers. For a given system (i.e., the pdf of T, (n, k) pair and codes used), there always exists an optimal read scheme for the client, choice of which needs careful calculation.

4 Further Discussions on Array Codes

The data distribution and the data acquisition schemes need MDS array codes to support a general (n, k) system. However, all the known MDS array codes can only support (n, k) systems, where $k = 1, 2, n-1, n-2$. Thus it is very useful to obtain more MDS array codes for general k.

4.1 Reed-Solomon Codes as Array Codes

Reed-Solomon codes[7] are a class of well-known MDS codes, which work can any (n, k) combinations, thus are more flexible. But their shortcoming is their relatively complex encoding and decoding procedures which use operations over finite fields, since usually they are described as 1-dimensional codes instead of 2-dimensional array codes. Because array codes are 2-dimensional, they are generalizations of 1-dimensional codes, and this generalization applies to any arbitrary 1-dimensional code. Thus Reed-Solomon codes can also be described as array codes. The following example shows an array code representation of a (7,2,6) Reed-Solomon code.

Example 5 *Array representation of (7,2,6) Reed-Solomon code*

Let α be a root of the primitive binary polynomial $x^3 + x + 1$ that generates the Galois field

GF(8). Using 1, α and α^2 as a basis, the 8 elements of GF(8) can be represented as vectors:

0	α	α^2	α^3	α^4	α^5	α^6	1
000	010	001	110	011	111	101	100

Now a (7,2,6) Reed-Solomon code can be constructed using the following *generator polynomial* [7]:
$g(x) = (x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4)(x+\alpha^5) = x^5+\alpha^2x^4+(1+\alpha)x^3+(1+\alpha^2)x^2+(\alpha+\alpha^2)x+\alpha$
Any information to be encoded can be represented by an information polynomial of degree 1, i.e.,

$$m(x) = (a_1 + a_2\alpha + a_3\alpha^2)x + (b_1 + b_2\alpha + b_3\alpha^2)$$

where the information bits, the a_i's and the b_i's, are in GF(2), i.e., they are binary. The above information polynomial can then be described as an array of size 3×2 over GF(2):

a_1	b_1
a_2	b_2
a_3	b_3

The code polynomial is then of degree 6:

$$c(x) = m(x)g(x)$$

After calculating and simplifying $c(x)$, a codeword of the (7,2,6) Reed-Solomon code can then be represented by an array of size 3×7, in a way similar to the representation of the information polynomial:

a_1	$a_2 + b_1$	$a_1 + a_3 + b_2$	$a_1 + a_2 + b_1 + b_3$	$a_2 + a_3 + b_1 + b_2$	$a_3 + b_2 + b_3$	b_3
a_2	$a_2 + a_3 + b_2$	$a_1 + a_2 + a_3 + b_2 + b_3$	$a_3 + b_1 + b_2 + b_3$	$a_1 + a_2 + b_3$	$a_1 + a_3 + b_1 + b_2$	$b_1 + b_3$
a_3	$a_1 + a_3 + b_3$	$a_2 + a_3 + b_1 + b_3$	$a_1 + b_2 + b_3$	$a_1 + a_2 + a_3 + b_1$	$a_2 + b_1 + b_2 + b_3$	b_2

Table 3: An array representation of a (7,2,6) Reed-Solomon code. Total number of additions: 39.

$\square$

Representing Reed-Solomon codes using arrays alone does *not* simplify encoding operations. Further simplifications need to be done. For the (7,2,6) Reed-Solomon code in Table 3, the last column certainly can be simplified to (b_3, b_1, b_2) instead of $(b_3, b_1 + b_3, b_2)$, without changing the code's MDS property, since the two vectors span the same space. We can simplify all other columns in a similar way, so that the density of each column is reduced to its minimum while the space spanned by the column remains unchanged. The following array is a simplified form of the (7,2,6) Reed-Solomon code, derived from its array form in the above example:

a_1	a_2+b_1	$a_1+a_3+b_2$	$a_2+a_3+b_3$	$a_2+a_3+b_1+b_2$	$a_3+b_2+b_3$	b_3
a_2	$a_2+a_3+b_2$	a_2+b_3	$a_1+a_3+b_1$	$a_1+a_2+b_3$	$a_1+b_1+b_3$	b_1
a_3	$a_1+a_3+b_3$	a_3+b_1	$a_1+b_2+b_3$	a_1+b_2	$a_1+a_2+b_2$	b_2

Table 4: A simplified array representation of a (7,2,6) Reed-Solomon code. Total number of additions: 27.

Though the above array form has been simplified a lot, i.e., its encoding operation is simpler than the original Reed-Solomon code, it can be further simplified as in Table 5 without changing the update complexity, i.e., some intermediate parity bits (s_i's) are calculated once and then reused in calculating other parity bits.

a_1	s_2	a_1+s_3	a_3+s_5	s_2+s_3	b_3+s_3	b_3
a_2	a_2+s_3	s_5	a_1+s_4	a_2+s_1	b_1+s_1	b_1
a_3	a_3+s_1	s_4	b_2+s_1	s_6	a_2+s_6	b_2

Table 5: A further simplified array representation of a (7,2,6) Reed-Solomon code, where $s_1 = a_1+b_3$, $s_2 = a_2+b_1$, $s_3 = a_3+b_2$, $s_4 = a_3+b_1$, $s_5 = a_2+b_3$, and $s_6 = a_1+b_2$. Total number of additions: 17.

So this gives a way to design more MDS array codes suitable for any (n, k) system, based on Reed-Solomon codes. Of course, this method should apply with *any* other linear code that is not MDS, i.e., any linear code can be described by a simplified array code with simple encoding operations.

Even though the array in Table 5 has been simplified a lot, it is still not clear whether it is the *optimal* form in terms of encoding operations, since we can simplify multiple columns at the same time as long as these columns span the same space. There are many research problems to solve, related to representing arbitrary codes as array codes. To name a few: 1) given a linear code, what is its optimal array code representation in terms of encoding complexity? or, how does one determine whether an array description is optimal in terms of its component density, i.e., total number of information bits appearing? 2) how does one design an efficient erasure-correcting algorithm once an array code is derived from a Reed-Solomon code or from anther code? 3) how does one design efficient *multiple* error-correcting algorithms for an array code?

4.2 Strong MDS Codes

For an MDS (l,k) array code of size $n \times l$, its MDS property means that the nk original information bits can be recovered from any k columns with each containing n bits. As shown in the previous section, if an (l,k) MDS array code is used in a data distribution scheme, a matching data acquisition scheme may read nk bits from m $(m \geq k)$ servers, with the ith server sending a_i bits such that $\sum_{i=1}^{m} a_i = nk$, where of course $0 \leq a_i \leq n$ and $1 \leq i \leq m$. Mapping to array codes, this leads to a new MDS property, which is called the *strong MDS property*, as defined as follows:

Definition 1 *(strong MDS property) An array code of size $n \times l$ with nk raw information bits has the* strong MDS property*, if, for any given m columns and any given series of positive integers a_i, where $k \leq m \leq l$, $0 \leq a_i \leq n$, $1 \leq i \leq m$ and $\sum_{i=1}^{m} a_i = nk$, there* always *exist a_i bits from the ith column such that the nk raw information bits can be reconstructed from these nk bits from the m columns.*

The interested reader can verify that the Reed-Solomon code in Table 4 has the strong MDS property. For example, if $m = 3$, and we are given the first 3 columns, let $a_1 = a_2 = a_3 = 2$, then we can choose either of the following 6 bits from the the 3 columns with 2 bits from each column:

a_1	$a_2 + b_1$	$a_2 + b_3$
a_2	$a_2 + a_3 + b_2$	$a_3 + b_1$

a_1	$a_2 + b_1$	$a_2 + b_3$
a_3	$a_2 + a_3 + b_2$	$a_3 + b_1$

Codes with the strong MDS property can present more choices of which data to read from multiple servers in an optimal data acquisition scheme. They can also provide more flexibility between redundancy and efficiency when distributing data. So codes with the strong MDS property will have useful roles in many applications in distributed storage systems. From the definition, any code with the strong MDS property is of course MDS.

A final question is how to construct codes with the strong MDS property? Since any code can be represented in array form, we answer this question by the following conjecture:

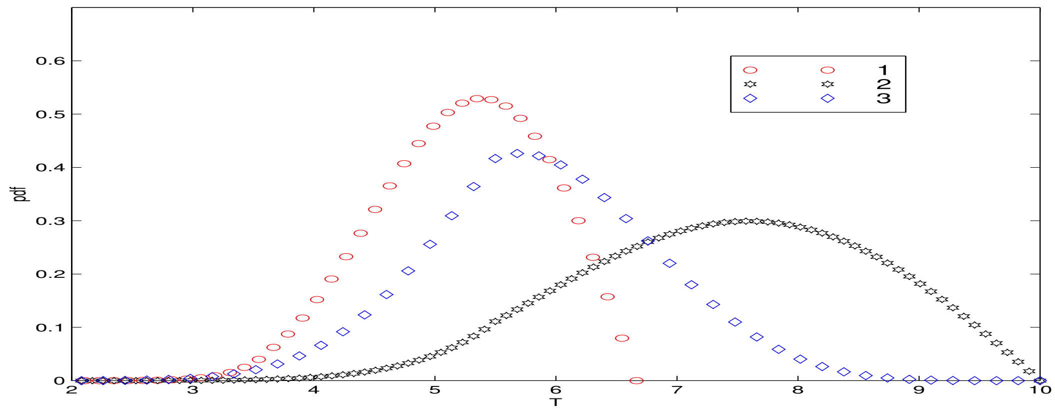
Conjecture 1 *(Strong MDS Conjecture)*

All MDS codes have the strong MDS property.

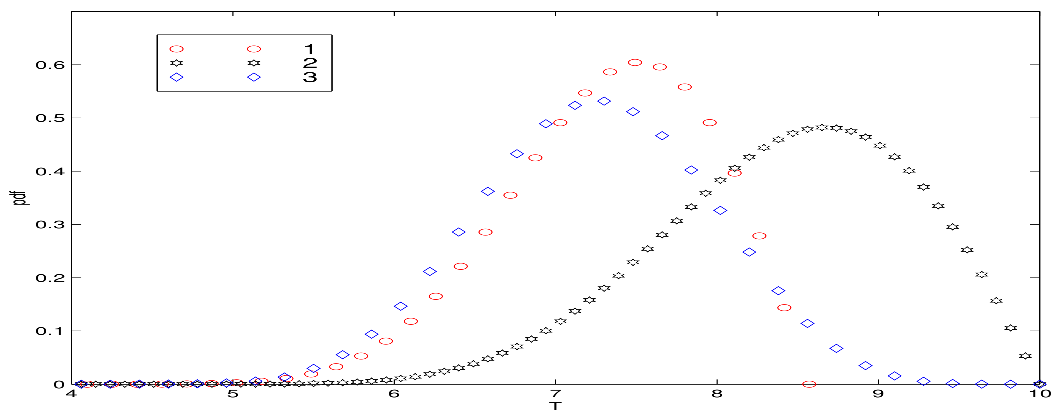
5 Conclusions

We have explored improving performance of data server systems by introducing data redundancy based on array codes. We have proposed methods of achieving the performance at both the servers' side and the client's side of the systems, namely finding an optimal data distribution scheme and an optimal data acquisition scheme. We have discussed constructing more general MDS array codes, and proposed the strong MDS property for the array codes. We also have given a simple probability model for a practical data server system, based on experimental results. Our further experimental results also qualitatively verify our analysis results.

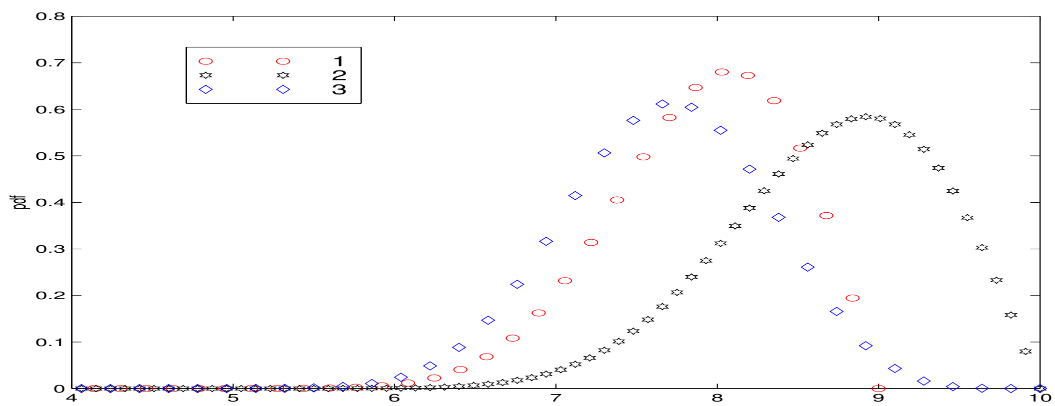
In this paper, the system model is rather simple: data requests from one client or multiple clients are processed sequentially, i.e., at one time, all the servers only handle one data request. But in a more practical system, such as web servers, multiple data requests can come in forms of some stochastic processes, and they can be processed by the servers in such a parallel fashion that the number of requests processed per unit time is maximized. Optimizing such systems is an interesting and useful research problem, which might need some sophisticated scheduling schemes among the servers, that are based on the results in this paper.



$(a)\ n = 3$



$(b)\ n = 7$



$(c)\ n = 10$

Figure 8: PDFs of different data read scheme, where $a = 1$, $b = 10$; 1, 2 and 3 represent scheme (1), (2) and (3) respectively.

Appendix

A Preliminary Analysis

Let $F_i(t)$ be the *cumulative distribution function* (cdf) of T_i, i.e.,

$$F_i(t) = Probability(T_i \leq t) = \int_0^t f_i(x)dx$$

Now let $T(n,k)$ be the time needed that the client has received from at least k out of the n servers since it sends its data request. Then $T(n,k)$ is another random variable, which is a simple function of all T_is:

$$T(n,k) \geq T_i, \quad where \ \|\{i\}\| \geq k$$

Let $f_{(n,k)}(t)$ and $F_{(n,k)}(t)$ be the pdf and cdf of $T(n,k)$, then it is easy to get relation between $F_{(n,k)}(t)$, $f_{(n,k)}(t)$ and $F(t)$, $f(t)$ [5]:

$$F_{(n,k)}(t) = \sum_{i=k}^{n} \binom{n}{i} F(t)^i [1 - F(t)]^{n-i} \tag{A.1}$$

or

$$f_{(n,k)}(t) = \frac{dF_{(n,k)}(t)}{dt} = k \binom{n}{k} F(t)^{k-1} [1 - F(t)]^{n-k} f(t) \tag{A.2}$$

The mean $E[T(n,k)]$ can be calculated once the $f_{(n,k)}(t)$ is known:

$$E[T(n,k)] = \int_0^\infty t f_{(n,k)}(t)dt \tag{A.3}$$

Though it is usually hard to get a clean closed form of $E[T(n,k)]$ for a general pdf $f(t)$, it is still possible to get some of its properties with respect to n and k. Intuitively, for a fixed pdf $f(t)$, bigger n and/or smaller k lead to smaller $E[T(n,k)]$, which can be proven mathematically.

Lemma 1 *For two continuous random variables X and Y defined on [a,b] with their cdf's as $F_X(t)$ and $F_Y(t)$ respectively, if $F_X(t) \geq F_Y(t)$, for $a \leq t \leq b$, and $F_X(t) \neq F_Y(t)$, i.e., the pdf of X is* left *to that of Y, then $E[X] < E[Y]$.* $\square$

Proof: Notice that $F_X(b) = F_Y(b) = 1$ and $F_X(a) - F_Y(a) = 0$, then
$E[X] - E[Y] = \int_a^b t dF_X(t) - \int_a^b t dF_Y(t) = [tF_X(t)|_a^b - \int_a^b F_X(t)dt] - [tF_Y(t)|_a^b - \int_a^b F_Y(t)dt]$
$= t[F_X(t) - F_Y(t)]|_a^b - \int_a^b [F_X(t) - F_Y(t)]dt = -\int_a^b (F_X(t) - F_Y(t))dt < 0.$ $\square$

It has been shown in [9] that

Lemma 2 *For a random variable T with a fixed pdf $f(t)$, for $1 \leq k \leq n$, the following inequalities hold (for $0 < F(t) < 1$):*

1. $F_{(n,k)}(t) < F_{(n+m,k)}(t)$, *for* $m \geq 1$*;*
2. $F_{(n,k)}(t) > F_{(n,k+m)}(t)$, *for* $m \geq 1$*;*
3. $F_{(n,k)}(t) > F_{(n+m,k+m)}(t)$, *for* $m \geq 1$*;*
4. $F_{(i,j)}(t) \leq F_{(n,k)}(t)$, *if* $n \geq i$ *and* $k \leq j$, *equality holds only when* $n = i$ *and* $k = j$*;*
5. $F_{(i,j)}(t) > F_{(n,k)}(t)$, *if* $n \geq i$, $k > j$ *and* $n - k \leq i - j$.

□

Using the two lemmas, it is straight forward to get the following properties of the mean of the service time:

Theorem 1 *For a random variable T with a fixed pdf $f(t)$, for $1 \leq k \leq n$, the following inequalities hold:*

1. $E[T(n,k)] > E[T(n+m,k)]$, *for* $m \geq 1$*;*
2. $E[T(n,k)] < E[T(n,k+m)]$, *for* $m \geq 1$*;*
3. $E[T(n,k)] < E[T(n+m,k+m)]$, *for* $m \geq 1$*;*
4. $E[T(i,j)] \geq E[T(n,k)]$, *if* $n \geq i$ *and* $k \leq j$, *equality holds only when* $n = i$ *and* $k = j$*;*
5. $E[T(i,j)] < E[T(n,k)]$, *if* $n \geq i$, $k > j$ *and* $n - k \leq i - j$.

□

One would hope that the *variance* of random variables also have the similar property. Unfortunately, however, the above property does *not* hold for the variance. We will show one example about the variance and one more property of the $E[T(n,k)]$ as well.

Lemma 3 *For two continuous random variables X and Y defined on [a,b] with their pdf's as $f(t)$ and $g(t)$ respectively, if $f(t) = g(a+b-t)$, for $a \leq t \leq b$, i.e., $f(t)$ is the* reflection *of $g(t)$ about the line $t = \frac{a+b}{2}$, then $E(X) = a + b - E(Y)$, and $Var(X) = Var(Y)$, where $Var(X)$ and $Var(Y)$ are the* variances *of X and Y.* □

Proof: Omitted.

Lemma 4 *If the pdf $f(t)$ of a random variable T defined on [a,b] is* symmetric *or* self-reflective *about the line $t = \frac{a+b}{2}$, i.e., $f(t) = f(a+b-t)$, then $f_{(n,k)}(t) = f_{(n,n+1-k)}(a+b-t)$.* $\square$

Proof: First it is easy to get that if $f(t) = f(a+b-t)$, then

$$F(t) = 1 - F(a+b-t) \tag{A.4}$$

Using Eq.(A.2) and Eq.(A.4), and also notice that $k\binom{n}{k} = (n+1-k)\binom{n}{n+1-k}$, we can get

$$f_{(n,n+1-k)}(a+b-t) = (n+1-k)\binom{n}{n+1-k} F(a+b-t)^{n-k}[1-F(a+b-t)]^{k-1} f(a+b-t)$$

i.e.,

$$f_{(n,n+1-k)}(a+b-t) = k\binom{n}{k}[1-F(t)]^{n-k}F(t)^{k-1}f(t) = f_{(n,k)}(t)$$

$\square$

This lemma shows that if T's pdf is symmetric, then there also exists symmetry between $T(n,k)$ and $T(n,n+1-k)$: their pdfs are reflective to each other. The above two lemmas lead to following theorem:

Theorem 2 *If the pdf $f(t)$ of a random variable T defined on [a,b] is* symmetric *about the line $t = \frac{a+b}{2}$, i.e., $f(t) = f(a+b-t)$, then $E[T(n,n+1-k)] = a+b-E[T(n,k)]$, and $Var[T(n,k)] = Var[T(n,n+1-k)]$.* $\square$

Here we see an example where the monotonicity of the $E[T(n,k)]$ with respect to n or k does *not* hold for the $Var[T(n,k)]$.

References

[1] A. Behr and L. Camarinopoulos, "Two Formulas for Computing the Reliability of Incomplete k-out-of-n:G systems", *IEEE Trans. on Reliability*, 46(3), 421-429, Sep. 1997.

[2] M. Blaum, J. Brady, J. Bruck and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures", *IEEE Trans. on Computers*, 44(2), 192-202, Feb. 1995.

[3] M. Blaum, P. G. Farrell and H. C. A. van Tilborg, "Chapter on Array Codes", Handbook of Coding Theory, edited by V. S. Pless and W. C. Huffman, to appear.

[4] P. G. Farrell, "A Survey of Array Error Control Codes", *ETT* , Vol.3, No.5, 441-454, 1992.

[5] M. N. Frank, "Dispersity routing in Store-and-Forward Networks", Ph.D. thesis, University of Pennsylvania, 1975.

[6] J. Gemmell, "Scalable Reliable Multicast Using Erasuring-Correcting Re-Sends", Technical Report MSR-TR-97-20, Microsoft Research, June, 1997.

[7] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, Amsterdam: North-Holland, 1977.

[8] C. A. Polyzois, A. Bhide and D. M. Dias, "Disk Mirroring with Alternating Deferred Updates", Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.

[9] H. M. Sun and S. P. Shieh, "Optimal Information-Dispersal for Increasing the Reliability of a Distributed Service", *IEEE Trans. on Reliability*, 46(4), 462-466, Dec. 1997.

[10] L. Xu and J. Bruck, "X-Code: MDS Array Codes with Optimal Encoding", *to appear in IEEE Trans. on Information Theory*, Jan., 1999.

[11] L. Xu, V. Bohossian, J. Bruck and D. Wagner, "Low Density MDS Codes and Factors of Complete Graphs", Proceedings of 1998 IEEE Symposium on Information Theory, Aug. 1998.