D

Data Replication and Encoding

Behrooz Parhami
Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

Synonyms

Big-data availability; Incorruptible big data

Definition of Entry

Conventional or well-established redundancy methods for preventing data loss, unavailability, or corruption can be used to protect big data, but they need to be updated in order to make them efficiently applicable to large data sets.

Overview

Data stored in memory devices, in storage networks, on the Web, or in the Cloud must be protected against loss, accidental contamination, or deliberate adulteration. Data are valuable assets that can be lost to negligence or theft (for illicit use or to exchange for ransom). Over the years, many methods of data protection have been devised by researchers in the field of dependable and fault-tolerant computing (Jalote 1994; Parhami 2018), all of which entail introducing redundancy to make data robust and recoverable in the event of loss or corruption. As data assumes ever-more important roles in the proper functioning of systems that affect our daily lives, greater protection, often involving higher degrees of redundancy, becomes necessary. Yet, the age of big data (Hilbert and Gomez 2011) makes naive redundancy methods costlier and less convenient, given the exponentially growing data volume, which has seen aggregate data storage capacity in the world rise from exabytes in 1986 to zettabytes at the turn of the century (Hilbert and Gomez 2011).

Data Unavailability, Corruption, and Loss

As valuable resources, data assets are subject to theft (can be worse than loss, if the entity stealing it abuses the resource), unavailability (not as bad as loss or corruption but inconvenient nonetheless), corruption, or permanent loss. Theft for illicit use can be prevented by encryption, a topic that is outside the scope of this article (Hankerson et al. 2000).

Data is lost or becomes unavailable for use if the storage device or host site holding the data crashes or is otherwise unresponsive. Data unavailability can be avoided by keeping multiple copies of the data on different devices/sites or by means of distributed encoding.

S. Sakr, A. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-63962-8_174-1

The term "error" refers to any change of bit or symbol values from the original or intended values. Errors can be classified by their type (inversion, erasure, substitution, and so on) and extent (the number and positions of the bits/symbols affected). Accordingly, error codes can be single- or multiple-error detecting/correcting. In the case of multiple errors, they can be arbitrary or correlated, important examples of the latter type being unidirectional and burst errors.

The most immediate threats to data corruption are inadvertent changes to bit or symbol values due to hardware failures (during the storage or processing of data) or transmission errors (as the data is moved from one place to another). The corruption extent ranges from one bit-flip ($0 \rightarrow 1$ or $1 \rightarrow 0$) to complete erasure via all bits being changed to 0s, say. Data corruption can be dealt with in ways similar to data loss or unavailability. Corruption that is limited in scope can be handled by error codes applied to locally stored data. We thus devote two sections to error-detecting and error-correcting codes, before describing schemes capable of dealing with more widespread data corruption.

Error codes are means of representing data so as to detect corrupting changes or to fix the corrupted values automatically (Rao and Fujiwara 1989). Of course, detection by itself is inadequate for avoiding data loss. It must be combined with a data reconstruction or recovery scheme in order to regain the original data. The recovery strategy is often based on data replication, where backup copies of data are kept in a reasonably up-to-date manner for use during recovery.

Error codes are assessed by their extent of redundancy, the total number n of bits/symbols required to represent k data bits/symbols, with $r = n - k$ being the absolute redundancy and r/k constituting the redundancy ratio, and by their detection/correction capabilities (how many and what types of errors are detected or corrected).

Error-Detecting Codes

The simplest error-detecting codes (Wakerly 1978) are based on parity, that is, the evenness or oddness of the number of 1 s in a code consisting of 0s and 1s. A simple parity code has code distance of 2, meaning that any two code words are different in at least two positions, ensuring the detection of any single-bit error. Multiple parity bits can be used in a variety of ways, leading to a wide class of codes (Garrett 2004), including Hamming error-detecting and error-correcting codes (Hamming 1950).

Information flow in coded protection of data is typically as follows:

Input ➔ Encode ➔ Send ➔ Store ➔ Send ➔ Decode ➔ Output

Such a coding framework protects the middle three stages of sending to memory, residence in memory, and sending to the final destination. If data processing is also included, it corresponds to a feedback path from the decode stage back to the encode stage through a processing unit. In other words, the redundant information is stripped, and data in its original nonredundant form is subjected to processing, with the result re-encoded before being stored. Communication application of coding corresponds to the linear path from input to output depicted above, while computing applications also include the feedback path just discussed.

Protection against storage errors has a lot in common with schemes of circumventing transmission errors, so the same codes are often applicable. However, there are specific codes that can exploit the data storage pattern, say, on two-dimensional recording surfaces, to provide greater protection at the same or lower cost (e.g., Parhami and Avizienis 1973).

With codes that are closed under processing, the feedback path can be shortened to span only the middle three, rather than five, stages, thus obviating the need for stripping and reinstating the code. Processing coded information directly provides stronger protection against accidental or hardware-induced data corruption. For example, arithmetic error-detecting codes (Avizienis 1971; Avizienis 1973; Garner 1966) allow direct addition and subtraction of coded numbers, yielding coded results. Multiplication with arithmetic-coded operands is a bit more involved but still

feasible, whereas division is quite difficult, thus limiting the scope of applicability.

Popular examples of error-detecting codes include checksum codes (McAuley 1994); weight-based and Berger codes (Berger 1961; Knuth 1986); cyclic codes (Peterson and Brown 1961); linear codes (Sklar and Harris 2004), which include Hamming codes (Hamming 1950) as special cases, balanced codes (Knuth 1986); arithmetic codes (Avizienis 1971); and many other types developed for specific applications.

Error-Correcting Codes

Error-correcting codes (Arazi 1987, Peterson and Weldon 1972) are used extensively in storing data on disk memory, whose raw reliability would be dismal without the detecting and correcting capabilities of such codes (Petascale Data Storage Institute 2012; Schroeder and Gibson 2007), as well as in many other application contexts. The use of such codes within individual data blocks is often augmented by inter-block redundancy schemes that provide an extra layer of protection in what is known as RAID architecture, acronym for redundant array of independent disks (Chen et al. 1994; Feng et al. 2005), or redundant disk array, for short.

Popular examples of error-correcting codes in practical use include cyclic codes (Lin and Costello 2004), which include the widely used cyclic redundancy check or CRC codes as special cases; linear codes (Sklar and Harris 2004), which include Hamming codes (Hamming 1950) as special cases; Reed-Solomon codes (Feng et al. 2005; Reed and Solomon 1960); BCH codes (Pless 1998); and many other types developed for specific applications. Turbo codes (Benedetto and Montorsi 1996) and low-density parity-check or LDPC codes (Gallager 1962) are examples of widely studied and used modern codes that achieve very low redundancy ratios.

The theory of error-detecting and error-correcting codes is quite advanced, and progress is still being made (Guruswami and Rudra 2009). What remains missing is a methodology for choosing an optimal or near-optimal code for a given set of application requirements.

Replication Codes

Replication codes are the conceptually simplest codes but imply high redundancy (100% in the case of duplication, 200% for triplication). Their advantages include the fact that any unrestricted error in one copy is detectable (duplication) or correctable (triplication). Such detection and correction capabilities require that data copies be stored on distributed nodes whose failures are independent of each other (Jalote 1994).

In addition to allowing error detection or correction, replication facilitates access to data by improving availability and access latency. For example, with two copies of a particular piece of data stored on two different computers, extreme slowness or even crashing of one computer will not hinder access to the data. The prevailing nomenclature here is that we have a primary site for each piece of data and one or more backup sites (Budhiraja et al. 1993; Guerraoui and Schiper 1997; Jalote 1994). In addition to high redundancy in such primary-backup schemes, a key challenge is to keep the multiple copies up-to-date and consistent with each other (Dullmann et al. 2001).

Data Dispersion

An alternative to replication is a scheme known as data dispersion (Iyengar et al. 1998; Rabin 1989), originally devised by Michael O. Rabin, which entails much less redundancy but requires some computation to reconstruct the data.

Let us present this method through an example. Consider the three numbers a, b, and c as the data of interest to be protected. Instead of storing the three numbers, we might store the four values $f(1)$, $f(2)$, $f(3)$, and $f(4)$, where f is the polynomial function $f(x) = ax^2 + bx + c$. We can lose any one of the four f values and still be able to derive the original data values a, b, and c from the three remaining f values by solving a

system of three linear equations. This example, which entails 33% redundancy to protect against loss of any single piece of data (compared with 100% for duplication), is readily understood, but it is not a particularly efficient way of dispersing data.

Data dispersion is actually a kind of encoding (Feng et al. 2005), where k pieces of data (say, k numbers) are encoded into n pieces (with $n > k$), in such a way that any k pieces suffice to derive the original data. Besides being useful for data protection, the method has security and load-balancing applications. In security, imagine devising a scheme so that at least three officers of a bank must cooperate to gain access to a safe, whereas no two officers can gain access. Such an arrangement is sometimes referred to as "secret sharing" (Beimel 2011). In load balancing, consider that the k pieces retrieved for reconstructing the original data can come from any of the stakeholders, thus making it possible to avoid congested sites. In such a case, data dispersion can be viewed as a performance-enhancing strategy, as well as a reliability improvement method.

Future Directions

The field of dependable computing has advanced for several decades to provide more reliable processing, storage, and communication resources. It is well known that reliability is a weakest-link phenomenon, in the sense that any unprotected or weakly protected part of the system can doom the entire system. The exponential reliability law (Parhami 2018) suggests that reliability declines exponentially with an increase in the number of system parts. In the context of big data, all aspects of the system (processing, storage, and communication) will expand in complexity, challenging the system designer to seek more advanced methods of reliability assurance.

Design of computer systems and their attendant attributes in light of new application domains and technological developments in the twenty-first century is under review by many researchers (Chen and Zhang 2014; Hu et al. 2014; Stanford University 2012). Since resources and services for big data will be provided through the Cloud, directions of cloud computing (Armburst et al. 2010) and associated hardware acceleration mechanisms become relevant to our discussion here (Caulfield et al. 2016).

A promising direction is provided by network coding (Dimakis et al. 2011). This new field essentially expands on the idea of data dispersion by considering more general data distribution schemes while also taking the impact of "repair traffic" on system performance into account.

Cross-References

- ▸ Data Longevity and Compatibility
- ▸ Hardware Reliability Requirements
- ▸ Parallel Processing with Big Data

References

Arazi B (1987) A commonsense approach to the theory of error-correcting codes. MIT Press, Cambridge, MA

Armburst M et al (2010) A view of cloud computing. Commun ACM 53(4):50–58

Avizienis A (1971) Arithmetic error codes: cost and effectiveness studies for application in digital system design. IEEE Trans Comput 20(11):1322–1331

Avizienis A (1973) Arithmetic algorithms for error-coded operands. IEEE Trans Comput 22(6):567–572

Beimel A (2011) Secret-sharing schemes: a survey. In: Proceedings of international conference coding and cryptology, Springer LNCS no. 6639, Berlin, pp 11–46

Benedetto S, Montorsi G (1996) Unveiling turbo codes: some results on parallel concatenated coding schemes. IEEE Trans Inf Theory 42(2):409–428

Berger JM (1961) A note on error detection codes for asymmetric channels. Inf Control 4:68–73

Budhiraja N, Marzullo K, Schneider FB, Toueg S (1993) The primary-backup approach. Distrib Syst 2:199–216

Caulfield AM, et al (2016) A cloud-scale acceleration architecture. In: Proceedings of 49th IEEE/ACM international symposium on microarchitecture, Taipei, Taiwan, pp 1–13

Chen CLP, Zhang C-Y (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf Sci 275:314–347

Chen PM, Lee EK, Gibson GA, Katz RH, Patterson DA (1994) RAID: high-performance reliable secondary storage. ACM Comput Surv 26(2):145–185

Dimakis AG, Ramachandran K, Wu Y, Suh C (2011) A survey on network codes for distributed storage. Proc IEEE 99(3):476–489

Dullmann D et al (2001) Models for replica synchronization and consistency in a data grid. In: Proceedings of 10th IEEE international symposium on high performance distributed computing, San Francisco, CA, pp 67–75

Feng G-L, Deng RH, Bao F, Shen J-C (2005) New efficient MDS array codes for RAID – part I: Reed-Solomon-like codes for tolerating three disk failures. IEEE Trans Comput 54(9):1071–1080; Part II: Rabin-like codes for tolerating multiple ($\geq$ 4) disk failures. IEEE Trans Comput 54(12):1473–1483

Gallager R (1962) Low-density parity-check codes. IRE Trans Inf Theory 8(1):21–28

Garner HL (1966) Error codes for arithmetic operations. IEEE Trans Electron Comput 5:763–770

Garrett P (2004) The mathematics of coding theory. Prentice Hall, Upper Saddle River

Guerraoui R, Schiper A (1997) Software-based replication for fault tolerance. IEEE Comput 30(4):68–74

Guruswami V, Rudra A (2009) Error correction up to the information-theoretic limit. Commun ACM 52(3):87–95

Hamming RW (1950) Error detecting and error correcting codes. Bell Labs Tech J 29(2):147–160

Hankerson R et al (2000) Coding theory and cryptography: the essentials. Marcel Dekker, New York

Hilbert M, Gomez P (2011) The World's technological capacity to store, communicate, and compute information. Science 332:60–65

Hu H, Wen Y, Chua T-S, Li X (2014) Toward scalable systems for big data analytics; a technology tutorial. IEEE Access 2:652–687

Iyengar A, Cahn R, Garay JA, Jutla C (1998) Design and implementation of a secure distributed data repository. IBM Thomas J. Watson Research Division, Yorktown Heights

Jalote P (1994) Fault tolerance in distributed systems. Prentice Hall, Englewood Cliffs

Knuth DE (1986) Efficient balanced codes. IEEE Trans Inf Theory 32(1):51–53

Lin S, Costello DJ (2004) Error control coding, vol 2. Prentice Hall, Upper Saddle River

McAuley AJ (1994) Weighted sum codes for error detection and their comparison with existing codes. IEEE/ACM Trans Networking 2(1):16–22

Parhami B (2018) Dependable computing: a multi-level approach. Draft of book manuscript, available on-line at: http://www.ece.ucsb.edu/~parhami/text_dep_comp.htm

Parhami B, Avizienis A (1973) Detection of storage errors in mass memories using arithmetic error codes. IEEE Trans Comput 27(4):302–308

Petascale Data Storage Institute (2012) Analyzing failure data. Project Web site: http://www.pdl.cmu.edu/PDSI/FailureData/index.html

Peterson WW, Brown DT (1961) Cyclic codes for error detection. Proc IRE 49(1):228–235

Peterson WW, Weldon EJ Jr (1972) Error-correcting codes, 2nd edn. MIT Press, Cambridge, MA

Pless V (1998) Bose-Chaudhuri-Hocquenghem (BCH) codes. In: Introduction to the theory of error-correcting codes, 3rd edn. Wiley, New York, pp 109–222

Rabin M (1989) Efficient dispersal of information for security, load balancing, and fault tolerance. J ACM 36(2):335–348

Rao TRN, Fujiwara E (1989) Error-control coding for computer systems. Prentice Hall, Upper Saddle River, NJ

Reed I, Solomon G (1960) Polynomial codes over certain finite fields. SIAM J Appl Math 8:300–304

Schroeder B, Gibson GA (2007) Understanding disk failure rates: what does an MTTF of 1,000,000 hours mean to you? ACM Trans Storage 3(3):Article 8, 31 pp

Sklar B, Harris FJ (2004) The ABCs of linear block codes. IEEE Signal Process 21(4):14–35

Stanford University (2012) 21st century computer architecture: a community white paper. On-line: http://csl.stanford.edu/~christos/publications/2012.21stcenturyarchitecture.whitepaper.pdf

Wakerly JF (1978) Error detecting codes, self-checking circuits and applications. North Holland, New York