International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 3 (2015) pp. 6901-6917

http://www.ripublication.com

Dual Quorum-Dispersal, A Novel Approach for Reliable and Consistent ECC based Storages

C.K. Shyamala and N.V. Vidya

Department of Computer Science and Engineering
Amrita School of Engineering
Amrita Vishwa Vidyapeetham(University)
Amrita Nagar PO 641112
ck_shyamala@cb.amrita.edu
Department of Computer Science and Engineering
Amrita School of Engineering
Amrita Vishwa Vidyapeetham(University)
Amrita Nagar PO 641112
nv.vidya@gmail.com

Abstract

Consistency, Availability, and Fault tolerance are the primary requirements at storage in a distributed environment. Besides these requirements storage efficacy and security in terms of confidentiality are also equally important. The traditional approach to achieve primary requirements is replication for data redundancy. Redundancy using replication provides high rates of availability at the cost of security and storage efficacy. This paper explores the possibility of incorporating dual quorum protocol, which provides availability, strong consistency and reasonable disk access time, with dispersal, to satisfy the additional requirements of security and storage efficiency at nominal cost. The performance of Dual quorum dispersal with respect to availability, errors, erasures and disk space is evaluated in comparison with Dual quorum incorporated for replication. Dual quorum dispersal outperforms Dual quorum replication in several functionalities like disk access time, security storage efficiency etc. On the other hand Dual quorum replication performs better when availability is considered. For applications that require space efficient, secure, error and erasure tolerant system which makes use of interleaved read and write workloads, Dual quorum- dispersal is an attractive choice and this is reinstated with the analyzed results.

Keywords : Dual Quorum, Availability, Consistency, Fault tolerance, Replication, Dispersal, Quorum system, RS coding.

1. Introduction

With the explosion of digital information in the computing world, it is an absolute necessity to have a convenient storage facility which is reliable, secure and space efficient. Reliability itself encompasses consistency, availability and fault tolerance [1]. A logical choice is a distributed storage system. A distributed system is vulnerable to numerous faults due to its public nature. A reliable storage in distributed environment approach should embrace fault tolerance which can be achieved through redundancy. Incorporation of redundancy in a storage system is a matter of concern as the wide spread of distributed storage and enormous amount of requests from a global audience pose challenges to its functionality [1]. Another major concern regarding a reliable distributed system is consistency. Reflection of an update operation in the entire storage, as and when it is made, assures consistency. Availability of data at the instant of access is the prime concern for a client and hence it receives high priority from the system [2]. A system is said to be available when it responds correctly to a request placed at any instant of time [9] and is said to be unavailable when it fails to respond positively to pings by a client or a periodic health checking monitor [2]. Nodes in a distributed storage system become unavailable for various reasons in the likes of overload in the network, hardware error etc. And the challenge is addressing these issues in tandem.

A classic approach for data storage in a distributed environment is replication [3] where multiple copies of a file (information) are stored at several servers across a network. This in turn results in the decline of storage efficiency along with the increase in the bandwidth consumption [3]. Applications that prefer low cost storage schemes should look for the alternative option, dispersal. Data is dispersed by initially splitting it into k fragments and adding parity to obtain a total of n fragments, and then by using dispersal techniques in the likes of Reed Solomon encoding technique[17], IDA[11] etc. These fragments are allotted to servers such that each server holds a unique fragment. A file is reconstructed by using any k of these n fragments using decoding processes [11]. Dispersal can be deployed for archiving data as *write* operation seldom takes place in such storages. Data is written once in such storages and there are multiple attempts to read the same. When the workload is in the form of interleaved reads and writes dispersal technique alone will not suffice to provide consistency. In such workloads, the consistency model should be lenient enough to incorporate changes without affecting availability but also be strict enough to service the request correctly [1].

Quorum system, where there is an essential intersection between read set of servers and write set of servers, is used to achieve high rates of availability [6]. The latest updated value, which is available in the intersection of servers, is propagated across the other servers. When read and write take place simultaneously, availability rate of the system is put at stake. To tackle the situation, we bring in dual quorum protocol(DQ), where there are two separate quorum systems in which one is bestowed upon the task of serving read requests and another is dedicated to serve write requests independently. Architecture of dual quorum is shown in the figure Fig 1.

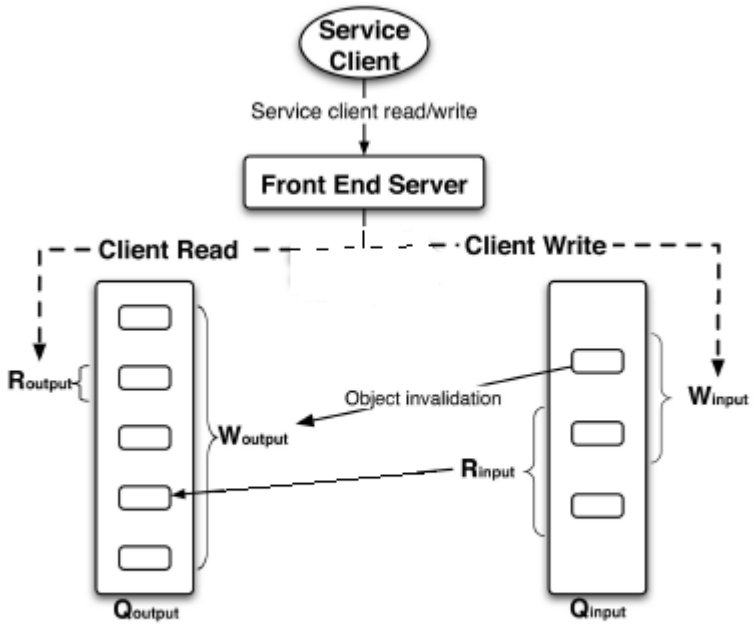

Fig 1 Dual quorum architecture [4]

As depicted in the figure Fig 1 the output quorum serves the read request and the input quorum serves the write request. Security is assured by incorporating dispersal which needs a minimum of *k* servers out of the *n* servers to rebuild the information. This is implemented with the help of Reed Solomon codes[17], a special case of error correcting codes, which is capable of detecting *2t* errors and correcting *t* of them, where *2t= n-k*. RS coding scheme also ensures that the information is reconstructed in the presence of *2t* erasures as well [17].

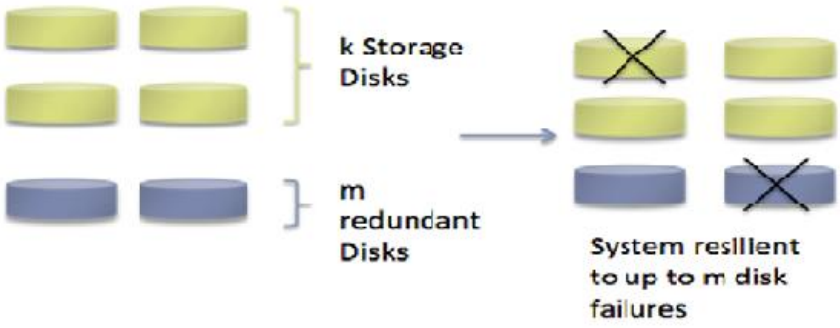

Fig 2 Basic dispersal technique [4]

Incorporation of dispersal technique in dual quorum protocol caters a distributed system to achieve essential requirements like consistency, availability, fault tolerance, security and space efficiency for workloads involving interleaved reads and writes.

The main contribution of the work in the paper is in the novel incorporation of data dispersal in Dual quorum protocol, intended for workloads with interleaved reads and writes. Here the contributions are as follows 1) Encoding and decoding technique for enhanced security and fault tolerance using RS coding technique 2) Design of Dual quorum protocol incorporated with dispersal for read and write operations which ensures availability 3) Design of consistency model for Dual Quorum- Dispersal for a consistent data retrieval 4) Extensive research and comparison of Dual quorum designed for replication with that designed for dispersal has aided the formation of recommendations for building a consistent and reliable ECC based storage.

The paper is organized as follows; Section 2 presents the proposed system design. In section 3 a comparison of the proposed design with the replication approach to dual quorum is given. Section 4 discusses about related works and with section 5 the paper concludes.

2. System model and protocol design

A representative architecture is illustrated in the figure fig 1[4]. This achieves reliability at storage for applications that involves interleaved reads and writes, where the environments consists of a set of servers that can play as Front End Servers (FES), servers in the output quorum or the servers in the input quorum. Requests are redirected to the input quorum or the output quorum by any of the FES which has smaller queue. In order to achieve consistency that leads to regular semantics, there are two separate quorums namely Q_{output} and Q_{input}. Here we assume fail-stop failures where if a system crashes it does not sent any more faulty messages that disrupt the functioning. The prototype developed transmits requests by randomly selecting the read servers and write servers. Read or write request is serviced if and only if sufficient number of servers (minimum k for reads, n for writes) are available to perform the same. Failure to pool minimum of k servers result in read failure or write failure. The data fragments for the read or write operations are obtained with Error correcting codes (ECC).

ECC is used to transmit data over a noisy channel or to retrieve the stored data [12]. The basic idea behind ECC is to add redundancy in the form of message symbols prior to transmission or storage. There are ECC based storages that are used for its fault tolerance capacities. In order to model the proposed design, Reed Solomon (RS) coding scheme a special case of ECC is used [18]. Data is encoded (systematic form) and written to all n servers. Read request is serviced with a minimum of k servers. Data is decoded with errors, erasures and their combinations that are detected and corrected.

2.1 Encoding and decoding

(n, k) Reed Solomon coding scheme is a popular Maximum Distance Separable (MDS) Code [18]. RS coding scheme is followed so as to improve the performance.

And high code rates (*code rate= k/n*) indicate faster encoding and decoding [15] [12]. At machine level every file is maintained as a combination of bits. These bits are fragmented and are considered as data words (*message u*) to facilitate encoding and corresponding decoding.

2.1.1 Encoding of RS codes

The codeword c (a n-symbol entity) is the code vector $(c_0,c_1,....c_{n-1})$, the corresponding code polynomial being $c(x) = c_0+c_1x+....c_{n-1}\,x^{n-1}$ with degree $n-1$. $c(x)$is in RS code if and only if $\alpha,\alpha^2,....\alpha^{2t}$are its roots, where α is a primitive in the multiplicative group of F_2^m. Every code polynomial has these $2t$ consecutive powers of α as roots. The minimum distance of such a code is $2t+1$. The message u(a k-symbol entity) is a vector $(u_0,u_1,....u_{k-1})$, and the corresponding polynomial $u(x) = u_0+u_1x+....u_{k-1}\,x^{k-1}$is of degree $k-1$. u can be encoded into $c(x)$ using the generator $g=(g_0,g_1,....g_{n-k})$– the corresponding polynomial $g(x) = g_0+g_1x+....g_{n-k}\,x^{n-k}$ is of degree $(n-k)$.

There are two forms of encoding (non-systematic and systematic); placement (location) of the redundant symbols in the encoded word varies. Encoding in the systematic form appends redundant symbols as a contiguous block to the original message; whereas in the non-systematic form the redundant symbols are scattered in the encoded word.

Encoding in non-systematic form:

$$\boldsymbol{c(x) = u(x).\, g(x)} \tag{2.1}$$

Encoding in systematic form:

$$\boldsymbol{c(x) = u(x)x^{n-k} + u(x)x^{n-k} \bmod g(x)} \tag{2.2}$$

where, $u(\mathbf{x})x^{n-k}$ shifts the message symbols $(n-k)$ places to the left and $u(\mathbf{x})x^{n-k}$ mod $g(x)$ is the remainder polynomial of degree $(n-k-1)$ or less, representing the $(n-k)$ redundant symbols that are to be appended to the shifted message symbols. The code symbols of **c(x)** are then uniquely assigned to each server as a file share.

2.1.2 Decoding of RS codes

Generally decoding is represented as :

$$\hat{\boldsymbol{c}}\boldsymbol{(x)= r(x) + e(x)} \tag{2.3}$$
and $\hat{\boldsymbol{c}}\boldsymbol{(x)= c(x)}$

where, the received vector r (a n- symbol entity with errors possibly) is a polynomial $r(x) = r_0+r_1x+....r_{n-1}\,x^{n-1}$of degree $n-1$; $r(x)= c(x) + e(x)$; and $e(x) = e_{j_1}x^{j_1} + ... + e_{j_{n-k}}x^{j_{n-k}}$.

Decoding of RS codes involves determination of error locations in the received polynomial and finding the error magnitudes in those locations [18]. The basic idea is to compute the 'syndromes' from the received polynomial and use those syndromes to form an error location polynomial.

The reciprocal of the roots of the error location polynomial represent the error location numbers. Once the positions of errors are known their magnitudes are implied with binary codes; in contrast the magnitudes are to be determined with RS codes [15][17]. There are well established algorithms that are used for implementing the above steps.

The decoding steps are as mentioned below [18]:

1 Computation of syndrome

$$\boldsymbol{S_i = r(x)|_{x=\alpha^i} = r(\alpha^i)\ ;\ i = 1, \dots, n-k = 2t} \tag{2.4}$$

Error is implied if S_i evaluates to a non-zero value.

2. Determination of error location polynomial $\boldsymbol{\sigma(x)}$

$\sigma(x)$ is defined as : $\boldsymbol{\sigma(x) = (1-\beta_1 x)(1-\beta_2 x) \dots (1-\beta_v x)}$

$$\boldsymbol{= 1 + \sigma_1 x + \sigma_2 x^2 + \cdots \sigma_v x^v} \tag{2.5}$$

Relating the coefficients of $\boldsymbol{\sigma(x)}$and $\boldsymbol{S_i}$using (2.4) and (2.5) we have

$$\boldsymbol{S_i + \sigma_j S_i + \cdots . \sigma_{n-k} S_1 = 0\ ;\ i = n-k+1\ , \dots , 1 \text{and} j = 1, \dots , n-k} \tag{2.6}$$

Solving (2.6) gives σ_i for the error polynomial $\boldsymbol{e(x)}$.

3. Evaluation of error location numbers

The error-location numbers of $\boldsymbol{e(x)}$ are the reciprocals of the roots of $\boldsymbol{\sigma(x)}$ computed in step 2.

4. Evaluation of error values

Error values $e_i$are computed using $S_i$and$\boldsymbol{\beta_i}$from steps 1 and 3 respectively.

$$\boldsymbol{S_i = r(\alpha^i) = \hat{e}(x) = e_1 \beta_1^i + \dots + e_j \beta_j^i}\ ;\ \text{j is the no. of error locations computed in step 3.}$$

5. Error correction

$$\boldsymbol{\hat{c}(x) = r(x) + \hat{e}(x)}$$

If decoding is successful then $\hat{c}(x) = c(x)$. Decoding algorithms are capable of correcting up to $\lfloor d_{min}/2 \rfloor$symbol errors but fail to correct errors beyond$\lfloor d_{min}/2 \rfloor$.

Decoding (Hard Decision Decoding–HDD) algorithms such as the Berlekamp - Massey and the Euclidian algorithms are widely used for correcting errors

2.2 Consistency

The basic encoding decoding scheme alone [vide section 2.1] does not provide consistent reads. It is essential to maintain consistency in distributed environments as it is one of the functional requirements of a storage system. A system is said to be in consistent state if it continues to generate correct after the occurrence of concurrent events [1]. In static storages where updates seldom take place, assurance of consistency is not a big problem. Whereas, the active storages (eg: storage for shopping site data) are subjected to dynamic updates and the changes have to be propagated to all the servers holding redundant data in the form of dispersed information. The issue of consistency pops up here. There are wide varieties of consistency models viz. optimistic consistency, strong consistency, eventual consistency. Strong consistency (Pessimistic consistency) ensures that the data beholds ACID properties and hence is consistent [6]. But the level of concurrency achieved and availability of data gained are very low. Weak consistency (optimistic consistency) model, allows a system to function despite inconsistencies. It allows multiple reads and writes without a central locking mechanism [6]. This does not ensure that the client is serviced with data that is not stale [4]. In order to achieve regular semantics in the designed protocol we assume to provide an eventual consistency model. Regular semantics for the design here is defined as follows:

- A read operation should return the latest completed write.
- A read operation should not return a value older than the previous read.

In Eventual consistency all the replicas eventually become identical without losing any of the writes. Here file synchronization is gradually performed at a later stage [4].

2.3 Dual quorum design for dispersal

Performance of traditional Quorum system is put at stake when there are simultaneous reads and writes [7]. Consistency is not guaranteed in large scale distributed environment .To enhance the performance during concurrent execution of reads and writes, dual quorums are exploited. Dual quorum as shown in the figure Fig 1 uses two set of servers namely input quorum and output quorum to service writes and reads respectively, instead of forming a single quorum in the intersection of read and write set of servers. This separation of quorum system enhances the availability [4].

2.3.1 Read

When a read request arrives, Front End Servers (FES) redirect the request to output quorum. A file is reconstructed using Reed Solomon decoder when *k* out of *n* fragments is valid. This case is called as a *read hit*. A *read miss* occurs when the

fragments in the output quorum are not valid. In this case, fragments of the requested data with highest time stamp is retrieved from the input quorum. The retrieved fragments are made valid at the output quorum and then decoded using Reed Solomon decoding [17]; file is read. File read (read miss) is illustrated in the figure Fig 3.A request arrives to read a file o, where o is fragmented and encoded into $o_{1..j}$. If the fragments are valid in the output quorum it returns the decoded output else the new updated value retrieved from the input quorum (R_{input}) is decoded and then returned. Here read servers are shown as R1.. Ri and write servers are shown as R1.. Rj

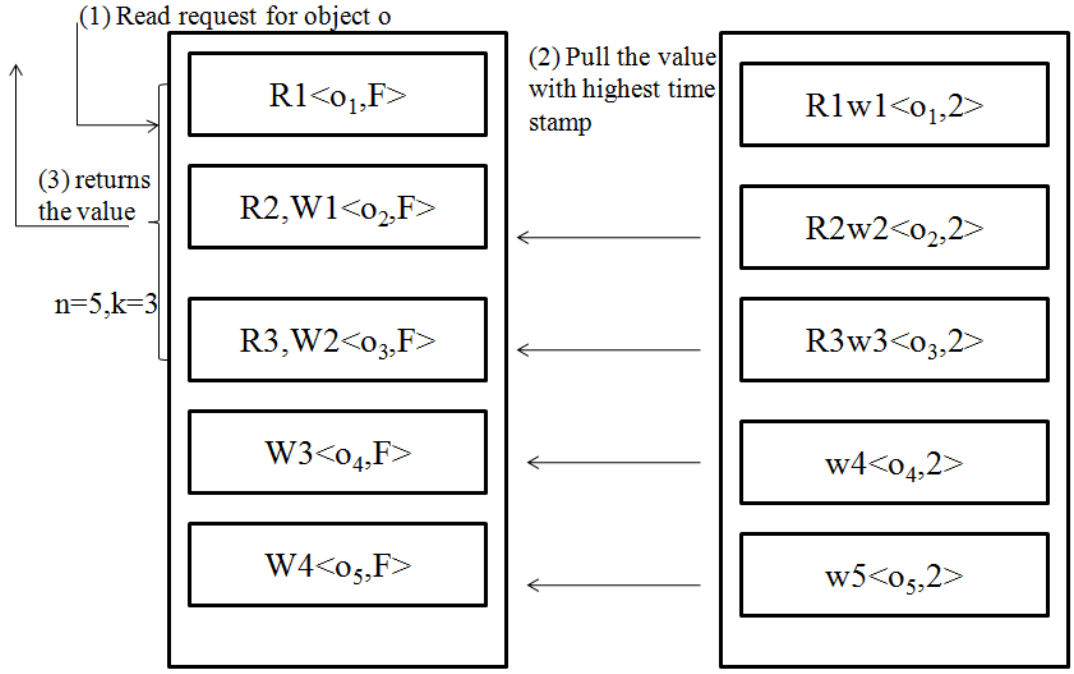

Fig 3:Read miss in DQ

2.3.2 Write

There are two scenarios in the case of write; *write through* and *write suppress [4]*.In *write through*, fragments have to be invalidated at the output quorum before performing the write operation. If successive writes are performed over the same file, it is quite obvious that the fragments in the output quorum have been invalidated for the file. In such cases where the invalidation is performed beforehand write is performed without communicating with the output server. This scenario termed as write suppress case, is less tedious when compared to the write through. Invalidation of data in the output quorum ensures that the system remains in the consistent state in spite of the update operations. Stale data is not returned, instead the latest value is pulled from the input quorum as and when required. File write (*write through*) is illustrated in the figure Fig 4.

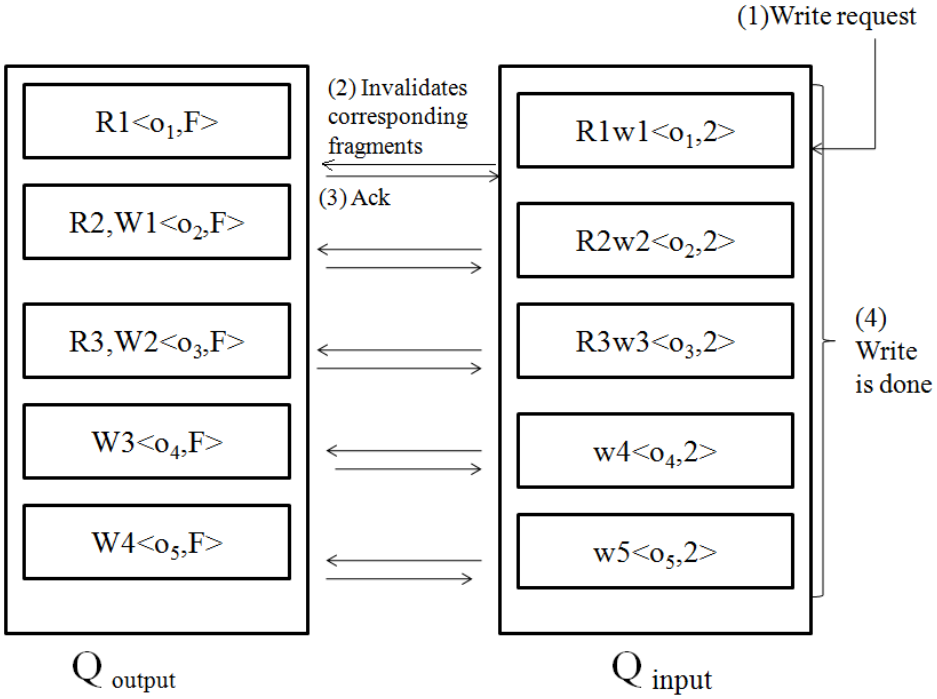

Fig 4: Write through in DQ-D

Dual quorum dispersal (DQ-D) performs the best when there is a maximum combination of read hits and write suppresses. This is confirmed with analytical and experimental results.

Data structures in the dual quorum protocol designed for dispersal are enlisted in the Table [1].

Table 1 Data Structure for DQ design.

Data structure	Meaning
Valid $_{o,i}$	Is true if *j* has a valid fragment from Q_{input} server *i*
Value$_{o,j}$	Value of the fragment at the server *j*
Lc	Logical clock for generating version numbers
Ack $_{o,j}$	Invalidation Acknowledgement received for object *o's* fragment *j* at server *i*

3. Results and analysis

Through Analytical and experimental evaluations we compare features viz. availability, disk access time, fault tolerance capacity (errors and erasures) of dispersal technique incorporated with Dual Quorum (DQ-D) against replication

incorporated in Dual Quorum (DQ-R) technique. Here we show that the results are competitive to the read and write performances showcased by DQ-R and hence DQ-D is a reasonable choice of data storage for applications involving interleaved read and write operations.

3.1 Availability

Availability (*av*) is defined as the number of requests successfully processed over the total number of requests [4] [2]. Availability of quorums is analytically modeled using the equation 3.1 [4], where av_{quorum} is given by the equation 3.2 where *w* is the write ratio, which is the ratio of the number of write to the total number of read and write operations, *n* is the size of the total size of Q_{input} (or) Q_{output}, *s* is the size of the read (or) write quorum and *p* is the probability of node failure .Unavailability (*1-av*) is measured against various write ratios; It is interesting to observe that the unavailability of DQ-R for lower write ratios (Read dominant workloads) are relatively low as the size of R_{output} is 1 in majority of cases. The availability of quorum on read operation, takes into consideration a minimum of *k* servers (R_{output}) in case of DQ-D and hence affects performance, which is a tradeoff for security and storage efficiency. The unavailability for larger write ratios (Write dominant workloads) is the kind of similar for both DQ-R and DQ-D as the size of the quorums in Q_{input} is the same. The study has been conducted for DQ-R with 6 replicas and using (6,3) RS code for DQ—D. The per server failure probability *p* is 0.001 and assumption here is that the failures are independent of each other.

$$av_{DQ} = (1 - w) \times \min\left(av(R_{output}), av(R_{input})\right) + w \times \min\left(av(W_{output}), av(W_{input})\right) \quad (3.1)$$

$$av_{quorum} = \sum_{i=0}^{n-s} \frac{n}{s} (1-p)^{i+s} p^{n-s-i} \quad (3.2)$$

Where *w* is the write ratio, *av* is the availability, n is the size of the quorum, *s* is the size of the read set of servers or the write set of servers and *p* is the probability of failure of single server (node).

The graphical illustration in the figure Fig 5 shows that DQ-D is competitive to DQ-R even though DQ –R outperforms DQ-D to a small extent; this is due to the size of the output quorum at the cost of security and storage efficiency. It is also observed that as the number of servers (*n*) increases availability also increases.

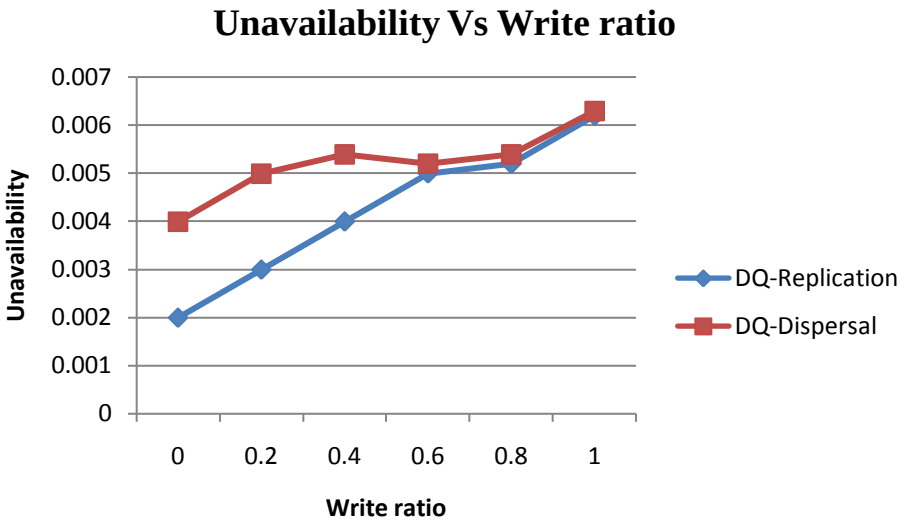

Fig 5: ***Unavailability Vs write ratio***

3.2 Disk access time

Disk access time is measured as the time taken to process a request from the processor and retrieve data as requested by the user. It is calculated as per equation 3.3 and is a combination of several other factors viz. seek time, rotational delay, transfer time, controller overhead and queuing delay .Here we ignore the controller overhead and queuing delay as they are negligible in the considered set up.

$$DiskAccesstime_{avg} = Average\ seek\ time + Average\ rotational\ delay + Transfer\ time \quad (3.3)$$

All factors except transfer time are the same for both DQ-R as well as DQ-D. Transfer time is directly proportional to block size rotational speed and density of the track. Rotational speed and density of the track for DQ-Rand DQ-D is similar as the setup is done over the same systems, where as the size of the block that is to be transferred is very less for DQ–D when compared to DQ-R. Transfer time with respect to DQ-R is relatively large as the transfer of entire block takes place from a single system and blocks need to be transferred sequentially. DQ-D is at an advantage as the time taken to transfer blocks of *k* data fragments (smaller in size when compared to entire file size) simultaneously from *k* parallel systems is comparatively less. With today's high end system and high speed processors encoding and decoding is never an overhead as a result of which the Disk access time for DQ-D is much lesser when compared to DQ-R. The graph in figure Fig 6 shows the disk access time for DQ-R and DQ-D for varying file sizes 10 KB, 100KB, 1MB, 100MB,1 GB etc. The (n, k) combination used for RS coding is (14,10).

Average seek time(T_s)	Average rotational delay(T_r)	Block transfer time(T_t)
Constant for DQ-D& DQ-R	Constant for DQ-D& DQ-R	Varies according to size of blocks transferred

Time taken for 1 rotation $T_{\text{rotation}} = (60\ seconds/R_{\text{speed}})$

Rotational Delay (As we are seeking only a single disk) $T_r = (1/2) \times T_{\text{rotation}}$

File size =|F|, Number of stripe units in each file, $N_{stripes}$ = (|F|/8) (As 1 stripe unit is 8KB)

Number of sector, $N_{sec} = N_{stripes} \times 16$ (As 1 stripe unit is 16 sectors)

Transfer time $T_t = (N_{sec}/T_{total}) \times T_{rotation}$ where T_{total} total number of sectors per track

Here we assume $R_{\text{speed}} = 1000\ RPM$, |F|=10 KB, 100KB, 1MB, 100MB, 1 GB etc, T_{total}=200,We obtain different disk access time ($T_s+T_r+T_t$) corresponding to each file size assuming seek time T_s=.4.9 ms. This is plotted for DQ-D and DQ-R and corresponding values to each file size is shown in the table below.

Table 2 : Disk access time for DQ-R and DQ-D

File size |F|	DQ-D (in ms)	DQ-R (in ms)
10 KB	7.906	8.5
100 KB	8.5	13.9
1 MB	13.9	67.9
100MB	67.9	6007.9
1 GB	6007.9	60007.9

When the file size is large the disk access time increases heavily for DQ –R by a huge factor.

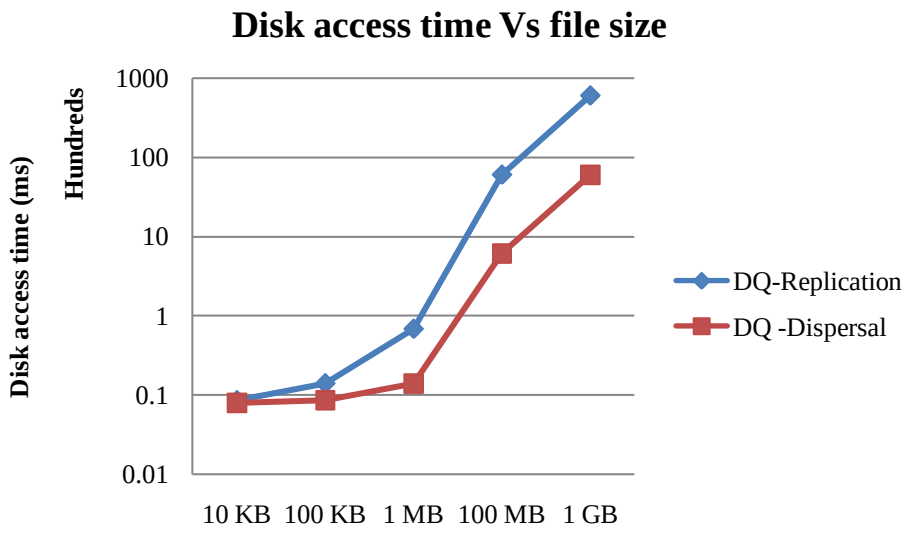

Fig 6: Disk access time Vs file size

The graph in the figure fig 6 demonstrates the relationship between Disk access time and various file sizes. It illustrates the average disk access time for both DQ-D and DQ-R for five file sizes. DQ-D has relatively lower disk access time when compared to DQ-R. This makes DQ-D efficient in terms of disk access time. This difference is clearly visible when the file size is relatively high. As the fragment access takes place simultaneously from independent systems the time consumed for block transfer is considerably reduced in DQ-D. Whereas in DQ-R sequential access of file needs to be performed from a single system as the R_{output} is a single server in Q_{output} server.

3.3 Errors and erasures

Erasures indicate the absence of a fragment and errors indicate the introduction of unnecessary bits into the data stream which affects the storage and reconstruction in usual cases. Errors and erasures are inevitable at storage. When storage becomes public, where it has to serve a global audience, it should be resistant to errors, erasures, or a combination of both. (*n*, *k*) RS encoding technique ensures that the resultant decoded code word is built accurately in spite of errors and erasures. RS code ensures that a maximum number of *2t (= n-k*) errors are detected and *t* of them are corrected. As in the case of erasures *2t* of the known erasures (known location) can be tolerated. The detection and correction of errors and erasures are done with the help of syndromes while decoding the data. Analysis of the designed model brings out the clear picture of the number of errors and erasures that can be tolerated [16].This is shown in table 3.

Table 3 : Illustration of fault tolerance

(n,k)	Errors(Detected)	Errors(Corrected)	Erasures
(6,3)	3	1	3
(10,6)	4	2	4
(14,10)	4	2	4
(31,11)	20	10	20
(255,64)	191	95	191

3.4 Storage efficiency

Storage efficiency is measured in terms of the space consumed for storing the data along with redundancy. For replication, it is expressed as the ratio of one file to the total number of replicas used inorder to maintain redundancy. In dispersal, storage efficiency is measured as the ratio of the number of fragments (*k*) to the total number of encoded fragments (*n*). We consider three specific cases for DQ-D and DQ-R for different values of *n* and *k.* In case of DQ-R, we consider three cases as in the figure fig 7 where *n* is the number of replicas (n=6, 10, 14,31). Here *Storage efficiency* $_{rep}$ = *1/ n* [5] where 1 indicates the file to be replicated (original file) and *n* indicates the number of replicas at storage. For DQ-D *Storage efficiency* $_{disp}$ = *k/n* [5] where *k* is the number of initial fragments and *n* is the total number of fragments for storage of

data. We have considered 3 (*n, k*) combinations {(6,3),(10,6),(14,10),(31,11)}. As in the normal case dispersal outperforms replication in terms of storage efficiency, DQ-D outperforms DQ-R and this is depicted in the figure Fig 7.

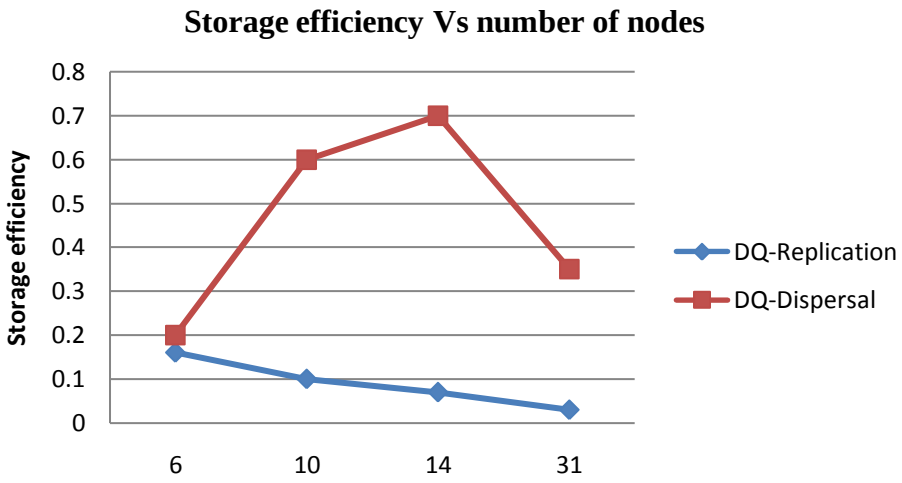

Fig 7. Storage efficiency Vs number of nodes

3.5 Security

DQ-R generates multiple copies of original file and stores, which makes it easy for the adversary to acquire the file since he is required to gain control over any one of the *n* servers holding the replica. This makes data vulnerable to attacks [13]. In case of DQ-D the adversary need to acquire control over a minimum of *k* servers to obtain the entire file. This becomes a tedious task for the malicious entity and thereby confidentiality is maintained up to *k-1* nodes [10]. It is also interesting to observe that collusion attack, in which a group of malicious entities decide to attack on a single node, does not guarantee the malicious entity the entire file or data, as the fragments are not bound to servers for all writes which takes security to a higher level [14] [15].

4. Related work

There are several works done related to storage in a distributed environment. Key features that were taken into consideration were consistency, availability and fault tolerance.

ROWA (Read Once Write All) protocol [4] which is either synchronous or asynchronous provides good read availability but the tradeoff here is write availability. Writes cannot be completed if any of the servers is unavailable in case of synchronous ROWA protocol. Hence this can be deployed efficiently only in a read dominant environment. In ROWA-A writes propagates asynchronously which ensures that a majority of the servers are available and write can be performed [4] [1] [7].The issue to be addressed here is the inconsistency of data. When the writes are

propagated asynchronously there are chances of inconsistency since this does not guarantee latest updated value in all the servers.

Quorum based protocols tolerates network partitions as long as they form quorum where there is a non-empty intersection of the read and write servers. Here, the read request will have to query a larger number of servers when compared to ROWA hence the read response time increases and the probability of reduced rates of availability increases. Hence for read dominated workloads the traditional quorum protocols are not preferred [4]. In spite of the availability issues the fault tolerance capability of quorum system is remarkable [3].

An improvised version of quorum protocol is dual quorum protocol that enables the servicing of read and write request by separate set of servers. The read requests are serviced by a set of servers in the output quorum and writes are serviced by set of servers in the input quorum. This provides high availability and guarantees consistency [4]. Since basic replication protocol requires just a single server for servicing a read, there are chances of compromising security. This paper incorporates the dual quorum protocol with dispersal technique which enhances security and reduces the special requirement for storage.

Archival storages seldom perform write operation. Once data is written, it is read multiple times over time. Here consistency is not a matter of concern. Information dispersal Algorithm [11] [8] holds good for such archival storages. Here data of length L=|F| is fragmented to n fragments, each of size L/m where only m fragments are required to reconstruct the original data. Each fragment becomes a file share and is uniquely allocated to servers. Here the ration of n: m is approximately 1 even though the values of n and m can be tuned dynamically. This enhances security and storage efficiency. This method is used for reliable transmission of data as well. But when it comes to consistency IDA does not guarantee consistency, also availability of data all the time is not guaranteed.

5. Conclusion

In this paper, we have presented the Dual Quorum-Dispersal technique, a novel approach to data storage in a distributed environment. The naïve dispersal approach ensures data security and storage efficiency at storage for archival data, but the same technique would not suffice to provide consistency and availability in applications that demand regular updates. In such situations a quorum based approach, which takes care of availability, becomes vital even though consistency is kept at stake. Dual quorum approach, an enhanced version of the quorum, ensures consistency and availability of data by making use of two separate quorum systems for servicing reads and writes independently. The two quorums achieve regular semantics by communicating with each other. Incorporation of dispersal with the Dual quorum approach ensures that data at storage is available, consistent, fault tolerant, secure and space efficient. The highlighted contribution of this paper is the protocol design for the Dual Quorum-Dispersal. The proposed model is implemented and it is found that the performance, in terms of disk access time, security, fault tolerance and space

efficiency, is better than the Dual Quorum Replication. At the same time, availability is found competitive to the Dual Quorum Replication. According to our inference, Dual Quorum-Dispersal is a best choice for data storage at distributed environment for all applications except that highly prioritize availability.

6. References

[1] Martin Placek and Raj Kumar Bhuyya, *"*A Taxonomy of Distributed Storage Systems" The university of Melbourne revision 1.148, Reporte técnico, Universidad de Melbourne, Laboratorio de sistemas distribuidos y cómputo grid ,2006.

[2] Daniel Ford, Francois Labelle, Florentina I. Popovici, Murray Stokely, Van-AnhTruong, LuizBarroso, Carrie Grimes, and Sean Quinlan, "Availability in Globally Distributed Storage Systems", *OSDI*,. 2010. pp. 61-74

[3] Dongfang Zhao, Kent Burlingame, Corentin Debains, Pedro Alvarez-Tabio, Ioan Raicu, "Towards High-Performance and Cost-Effective Distributed Storage Systems with Information Dispersal Algorithms", Cluster Computing (CLUSTER), 2013 IEEE International Conference on. IEEE, 2013

[4] Lei Gao, Mike Dahlin, JiandanZheng,ArunIyengar, "Dual-Quorum: A Highly Available and Consistent Replication System for Edge Services" ,IEEE transactions on dependable and secure computing,VOL. 7,NO. 2,APRIL-JUNE 2010

[5] Debains, Corentin, et al. "IStore: Towards High Efficiency, Performance, and Reliability in Distributed Data Storage with Information Dispersal Algorithms." under review at IEEE MSST , 2013.

[6] Andrew S Tanenbaum, Maarten Van Steen, Distributed Systems Principles and paradigms, Pearson and Princeton Hall,2nd Edition

[7] David Peleg, Avishai Wooly, "The Availability of Quorum Systems*",* Information and Computation 123.2 (1995): 210-223

[8] Kevin D. Bowers, AriJuels, Alina Oprea ,"HAIL: A High-Availability and Integrity Layer for Cloud Storage*"* , Proceedings of the 16th ACM conference on Computer and communications security. ACM, 2009

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-TakLeung, "The Google File System", *ACM SIGOPS* Operating Systems Review. Vol. 37. No. 5. ACM, 2003.

[10] Nihar B. Shah, K. V. Rashmi, Kannan Ramchandran, and P. Vijay Kumar, "Privacy-preserving and Secure Distributed Storage Codes", Presented at Globecom 2011 and ISIT 2012

[11] Michael O Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance", Journal of the Association for Computing Machinery, Vol. 36, No. 2, April 1989.

[12] Ranjan Bose, "Information Theory Coding and cryptography", Tata McGraw-Hill Publishing Company Limited, New Delhi, 2002

[13] Toni Ernvall, Salim El Rouayheb, Camilla Hollanti, and H. Vincent Poor "Capacity and Security of Heterogeneous Distributed Storage Systems" .IEEE journal on selected areas in communications, vol. 31, no. 12, December 2013

[14] Sameer Pawar, Salim El Rouayheb, and Kannan Ramchandran, "Securing Dynamic Distributed Storage Systems Against Eavesdropping and Adversarial Attacks", IEEE transactions on information theory, vol. 57, no. 10, October 2011

[15] Gadiel Seroussi and Ron M. Roth, "On MDS Extensions of Generalized Reed-Solomon Codes", IEEE transactions on Information Theory Vol.IT 32,No. 3 May 1986

[16] K. V. Rashmi, Nihar B. Shah, Kannan Ramchandran, and P. Vijay Kumar, "Regenerating Codes for Errors and Erasures in Distributed Storage", IEEE International Symposium on Information Theory Proceedings,2012

[17] Reed I., and Solomon G, "Polynomial Codes Over Certain Finite Fields." Journal of the Society for Industrial and Applied Mathematics, 1960, 8(2):300–304.

[18] Costello, D., and Shu Lin. Error control coding. Pearson Higher Education, 2004, chapter7.