Large-scale Decentralized Storage Systems Used by Volunteer Computing Systems

Iuliia Proskurnia, Arinto Murdopo, Enkhjin Bayarsaikhan and Diego Montero
Facultat Informatica de Barcelona
Universitat Politecnica de Catalunya
Barcelona, Spain

***Abstract*—Recently, Volunteer Computing (VC) is becoming quite popular for providing its resources for large-scale computational problems in scientific researches. Centralized control of the VC systems is the main bottleneck, since it introduces asymmetry to the system. Several systems are already benefitting from VC in terms of CPU sharing, e.g. SETI@home. Another side of the problem is storage sharing, where volunteer nodes will dedicate their free memory space to the system.**

Introduction of Decentralized Storage System that is suitable for the VC is appropriate, mainly, to reduce possible overhead from the centralized part of the VC and provide available and eventually consistent decentralized storage system with minimal cost.

The focus of this paper is to thoroughly survey currently existing decentralized storage systems, evaluate them according to its suitability for VC system, and provide a state-of-the-art solution for this issue.

***Index Terms*—Decentralized Storage Systems, Volunteer Computing, Scalability, Replication.**

I. Introduction

Large scale decentralized storage systems' goals are decentralization, reduced cost and fault tolerance. At the same time decentralized storage systems provide inherent scalability and availability of resources compared to its centralized counterpart [4], [7], [8], [9], [10], [13], [14], [15].

Main design issues of decentralized storage systems are the followings [6]:

- *Decentralization.* Decentralized storage systems, as the name suggests, are decentralized by their nature. They could support distributed storage, processing, information sharing etc.
- *Fault-tolerance.* System should be resilient to removal and failure of nodes at any moment.
- *Fast Resource Location.* Efficient mechanism for resource location is an important point.
- *Load Balancing.* System should make optimal distribution of resources based on nodes' capability and availability.
- *Churn Protection.* Denial of service attack from excessive node churns should be handled by the system.
- *Anonymity.* Resistance to censorship is important characteristic to be introduced by the system.
- *Security.* Security from attacks and system failure should be ensured by the system.
- *Scalability.* Supporting millions of users are essential for decentralized storage systems.

Achieving all these goals is still a huge challenge for researchers. Existing decentralized storage system solutions only could achieve some of them while sacrificing the others.

The most popular techniques to achieve all these properties among large-scale decentralized storage systems are the following:

- *Consistent Hashing.* In consistent hashing, the output range of a hash function is treated as a fixed circular space or "ring". Each node is assigned a random value within this space. Each data item identified by a key is assigned to a node by hashing the data item's key to yield its position on the ring.
- *Active or passive replication.* In active replication each client request is processed by all the servers. In passive replication there is only one server (called primary) that processes client requests and subsequently propagates the updates to back-up nodes.
- *Gossip-based protocol for failure handling.* The protocol is based on the gossip/virus-based information distribution, including random destinations to spread the information.
- *Logging read/write operations.* The main function of logging is to store all the changes made which are reads and writes by all nodes during the object life.
- *Ring locality for load balancing.* This technique should be applied to deal with non-uniform data, load distribution and heterogeneity of nodes performance.

Most of these properties can be found in Cassandra and Dynamo systems and they are partly covered by other systems like Ivy [4], Squirrel [5], Pastis [18], PAST [12], Riak [4], Voldemort [8], OceanStore [7], [19], Farsite [11].

Volunteer Computing (VC) uses the free resources in Internet and Intranet for some computational or storage purposes. It is important to discover the endless options for its application. One of the differences between VC and P2P systems is behavior of their nodes. Analysis of the real traces from SETI@home [37] project proved clients contribution consciousness. For example, SETI@home follows a typical model of a volunteer computing project, in which an "agent" is installed on the user's machine after they register to participate. All the registered participants are contributing with their CPU to complete some important computational problem: biological, chemical etc.

However, current architectures are based on the client-server architecture. In such VC systems, cluster or group of central servers assigns jobs to voluntarily-contributed-machines/volunteers. This configuration may lead to bottleneck in the group of central servers, in terms of task distribution or data movement between server and clients. Harvard TONIC project tried to reduce the chance of bottleneck by splitting centralized servers into central storage system and lookup service. However, one point of failure still exists and TONIC does not support storage sharing between its users. P2P-Tuple [21] solution could be used to alleviate aforementioned TONIC'S problems.

Until now VC popularity is focused on CPU sharing area. As the volume of existing data and knowledge is growing rapidly, the necessity of new approaches for storage is critical. One of the solutions is employing scalable decentralized storage systems in Volunteer Computing.

The main goal of this survey is to provide a comprehensive description of the current decentralized storage systems and the extent to which these systems could be combined with VC, the last part of the paper will include proposal of a state-of-the-art system that fits Volunteer Computing storage needs.

The rest of the paper is organized as follows. Section II introduces background information about Decentralized Storage Systems and Volunteer Computing. Section III covers a review of the existing storage systems with their suitability for Volunteer Computing, which is referred as a Volunteer Computing Extension. Section IV provides a proposal and design of the most suitable state-of-the-art decentralized storage system for VC. Finally the survey will end up with conclusion and list of references.

II. Background and Related works

A. *Decentralized Storage Systems*

Decentralized storage systems (DSS) are growing far more attractive over the years, as they are being employed by the popular networking corporations like Facebook [13], Amazon [15], LinkedIn [8] and etc. Behind the growing popularity is the rising number of researches and developments in the field of DSS. The coordination between nodes can be symmetric as in P2P systems [12] where nodes have same capabilities or responsibilities, or in some implementation it could be asymmetric. Figure 1 illustrates a comparison of system architectures in DSS and centralized storage systems. The data in DSS are stored in several coordinating nodes, whereas in centralized storage system, the data are stored in a single server.

The goals of decentralized storage system are an ad hoc connectivity between nodes, a decentralization of the system, reduced cost of ownership, and anonymity [6]. In this paper we surveyed number of different decentralized storage systems with their key features and pros-and-cons, and evaluated them according to their suitableness in storage system for volunteer computing. The characteristics that are important for storage system in volunteer computing are availability, scalability, eventual consistency, performance and security.

The design issues that associate with these characteristics are implementation of read/write access, replication handling among nodes in the system for fast resource location, symmetry design of the storage for load balancing, fault and

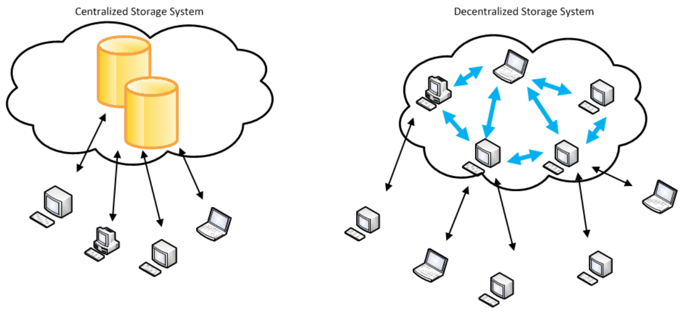


Fig. 1. Decentralized Storage System

security handling for reliable storage, and the scalability feature for handling thousands of nodes. Multi-writer designs face numerable issues not found in read-only systems, such as maintaining consistencies among replicas, handling concurrent updates, and preventing malicious attacks [18]. Similarly, one can find multiple approaches in replication techniques within the storage systems, for high reliability, churn protection, and fast resource location. The same goes to fault tolerance; there are multiple techniques to achieve certain tolerance level against faulty nodes. For example Dynamo's sloppy quorum and hinted hand-off [15] as well as Total Recall's redundancy management [10]. In addition, security is equally important in peer-to-peer storage systems for volunteer computing, considering the importance of data integrity and computing resources of the volunteers in volunteer computing systems.

B. *Volunteer Computing*

Volunteer Computing (VC) is a distributed computing paradigm based on computing and storage capabilities of computers donated by volunteer individuals via the Internet. The volunteer members donate their personal computers to be used as hardware resources, when the computers are in idle status. Through this arrangement, VC environments have been very successful in harnessing idle hardware resources in personal computers.

Another definition of Volunteer Computing is a type of Desktop Grid (DG) computing, which is one of the largest distributed systems in the world. DG aims to bring large numbers of donated computing systems together to form computing communities with vast resource pools. They are well suited to perform highly parallel computations that do not require any interaction between network participants.

VC is currently being used by many scientific research projects. A well-known example of this is a project called SETI@home, whose main purpose is to analyze radio-telescopic signals in an attempt to detect extra-terrestrial activities. In this project, SETI@home employed a volunteer computing platform called BOINC.

BOINC, which stands for Berkeley Infrastructure for Open Network Computing, relies on a publicly open system that allows private individuals (i.e., "volunteers") to install a software program, which takes advantage of their computers' processing power when it would otherwise be idle. In addition, BOINC can also be used in multiple projects at once; hence a single volunteer-computer can share its resources with different projects using this platform [46].

Another example of Volunteer Computing platforms is the XtremWeb project. The main principle of this open source research project is not only that any participant can volunteer his computing resources, as in BOINC, but also can use other participants' computing resources or storage. In this scenario, each participant has the ability to register new applications and data and to submit new computational jobs [45].

Architecture of a VC middleware - BOINC

From a general point of view, The BOINC architecture is based on a strict master/worker model, as shown on the Figure 2:

- *Client:* allows platform users to interact with the platform by submitting stand-alone jobs and retrieving results.
- *Server:* a coordination service which connects clients and workers. It receives jobs submissions from clients and distributes them to workers according to a scheduling policy.
- *Worker:* the component running on the PC which is responsible for executing jobs.

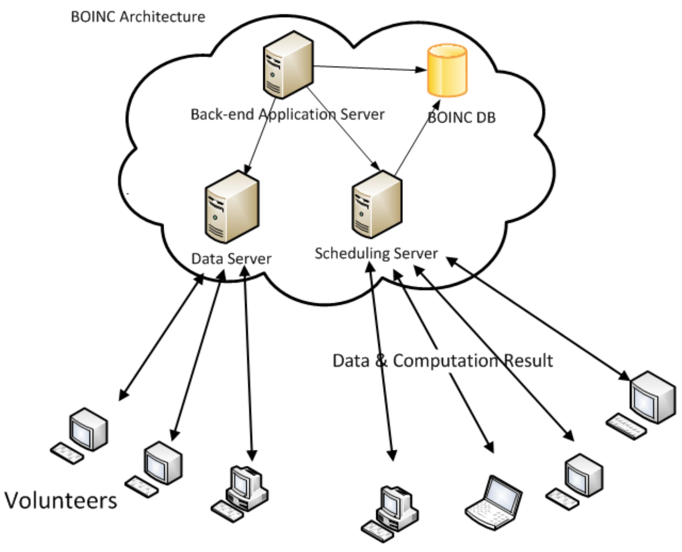


Fig. 2. VC BOINC Architecture

This architecture defines a Pull Model (workers pull tasks from the server). The centralized server never initiates communication with worker nodes. All the communications are instantiated from the worker simplifying some communication issues due to NAT and Firewalls. Besides, the central servers do not have to keep track of the workers connections.

Critical scaling issues with VC may arise from the fact that both VC and DG use centralized data storage, thus creating a potential bottleneck in the system when tasks share large input files or when the central server has limited bandwidth. In this paper, we surveyed the use of decentralized data sharing techniques that can be further introduced to VC and DG data distribution system. The functionalities of decentralized data sharing techniques may range from BitTorrent-style networks where all participants share equally, to more constrained and customizable unstructured networks where certain groups are in charge of data distribution and discovery. Figure 3 shows decentralized data sharing technique with its relation to Volunteer Computing.

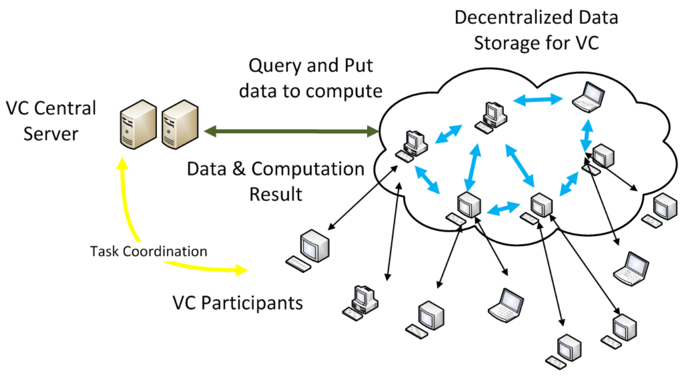

Decentralized Data Storage for VC
Query and Put data to compute
VC Central Server
Data & Computation Result
Task Coordination
VC Participants

Fig. 3. Decentralized Data Storage for VC

III. Existing System Review

In the quest for defining the state-of-the-art decentralized storage system for volunteer computing, the following well-known distributed storage systems were thoroughly surveyed: FarSite, IVY, Overnet, PAST, PASTIS, Voldemort, OceanStore, Glacier, Total Recall, Cassandra, Riak, Dynamo, and Attic. The systems are arranged in the order of least useful to most useful. The last two systems, TFS and Squirrel, are special type of decentralized storage systems, however their implementations and characteristics make them equally important for our research.

The survey is based on analysis of characteristics, implementations and the system architecture of each storage systems. The characteristics, on which we focused, in this survey are Availability, Scalability, Eventual Consistency, Performance and Security. Each distributed storage system was evaluated by its advantages, disadvantages and VC compatibility.

In the end of this section, Table I shows short description for each system together with main characteristics of each system, based on five aforementioned characteristics previously (AV = Availability, SC = Scalability, ECO = Eventual Consistency, P = Performance and SC = Security). Appendix contains more comprehensive comparison table for surveyed storage systems according to their read/write access, replication management, symmetry, fault tolerance and security characteristics.

A. Farsite

FARSITE (Federated, Available, and Reliable Storage for an Incompletely Trusted Environment) is designed to harness collective resources of loosely coupled, insecure, and unreliable machines to provide logically centralized, secure, and reliable file-storage services. It tries to solve the problem of high cost in maintaining centralized server and its vulnerability to geographically localized faults.

The foundation of FARSITE design is built on the fact that there is a general increase in unused disk capacity and a decrease in computational cost of cryptographic operation relative to I/O.

To achieve file system capability, FARSITE should have hierarchical directory namespace. FARSITE allows the flexibility of having multiple roots (regarded as name of virtual file server). Each root has designated set of machines to manage it and these machines form a Byzantine-fault-tolerant group to enhance their security. FARSITE manages trust in its system by using public-key-cryptography, which is done by introducing several types of certificates. Symmetric encryption is used to encrypt users' private keys of their certificates. Note that in FARSITE, once encrypted, the private key of the certificate is stored in globally readable directory in FARSITE and it only can be accessed upon login.

In relation with Volunteer Computing, the notion of directory group is a good idea. It allows the protocol to distribute the management job as well as the data into several machines. Security is also handled well with encryption and public-key-cryptography-based access list control. However, it is designed to handle up to 10^5 nodes and it may not be sufficient in Volunteer Computing context. Another drawback is its design which is only intended for University or Workplace environment (relatively slower churn rate and high machine availability, as what have assumed by the writer of the paper), which has different characteristics with Volunteer Computing environment (relatively higher churn rate and low machine availability).

Advantages	**Disadvantages**
Quantified scalability. Scalable up to 10^5 nodes. Security is really prioritized in the design. Mimic the centralized file system.	Limited uses, current design and implementation is suitable for desktop machines in academic and corporate settings.
Volunteer Computing Extension	
Scalability up to 10^5 nodes may not enough for volunteer computing environment. Volunteer computing environment has different characteristics to FARSITE intended environment (University or Workplace environment).	

B. IVY

Ivy is a read/write distributed storage system based on log files of its clients. Each user takes periodic snapshots of all other users' logs, containing history of uploads and updates to all files in the system. When a user makes updates to any file, its log is saved in his/her own log. This setup makes the Ivy storage system suitable for smaller group of cooperating peers. In addition, use of client logs slow down the system, despite that IVY uses periodic snapshots to mitigate this issue. Non-conflicting concurrent file update logs are merged, and conflicting files are resolved in application level. Ivy exposes the whole operation history of files to every client. Hence, security in Ivy is achieved by peers reviewing the log files of faulty users, and choosing not to trust those bad users. In other words, the security measures take place after the disaster. In a large scale distributed storage system, clients expect global security and stability, without having to choose which clients to trust based on their user logs. This feature of Ivy makes it less attractive as a potentially large scale distributed storage system. Finally, [4] mentions that more work is required, in order to improve IVY's conflict resolution techniques.

Advantages	**Disadvantages**
Decentralized storage – no central point of failure.	Security measures happen after the disaster. Not so scalable.
Volunteer Computing Extension	
IVY is not scalable, does not provide enough security tolerance. Therefore, it is not a recommended storage system for volunteer computing.	

C. Overnet/Kademlia

Overnet used to be popular decentralized peer-to-peer computer network in 2006. It is based on Kademlia DHT; hence there is no scientific paper that discuss about Overnet, we will discuss Kademlia instead.

Kademlia is a symmetric DHT based on XOR-based metric to perform the routing functionality. It treats nodes as leaves in binary tree; with each node position is determined by ID prefix. Kademlia ensures that every node knows of at least one node each of its subtrees. Node ID is represented as 160-bit number.

XOR metric is used to introduce the notion of distances between nodes in Kademlia. However, it doesn't consider the actual location of the nodes since it uses the node ID to determine the "distance" between them. XOR is unidirectional.

Kademlia nodes store contact information about each other to route query message with maximum list of k. There are four RPC protocols in Kademlia: PING, STORE, FIND_NODE and FIND_VALUE. In Kademlia, the most important procedure for each participant is called node lookup, where by a node in Kademlia tries to find the k closest nodes to some given node ID. Kademlia employs recursive algorithm for node-lookup. The initiator is able to send parallel, asynchronous FIND_NODE RPCs to the nodes it has chosen. This parallelization is able to accelerate the lookup process although it increases the number of messages that are circulating in the network. Nodes in Kademlia also need to periodically broadcast (key, value) pairs to keep them alive. The routing table in Kademlia is in the form of binary trees and it is always maintained to avoid unbalanced tree. In addition, some optimization is performed in Kademlia to efficiently republish key.

Advantages
The notion of symmetry using XOR metrics is good. The concept of learning about neighbor when receiving queries is interesting. It is resistant to certain degree of DoS attack.
Disadvantages
Kademlia's notion of distance does not consider the actual network distance. In the current era of geolocation, this distance notion can be improved by considering the actual network distance. Routing table in Kademlia is in the form of binary trees and it needs to maintain it periodically. This maintenance potentially has lots of message to exchange in the network.
Volunteer Computing Extension
High churn rate causes lot of message overhead in Kademlia and makes not really suitable for Volunteer Computing system. The message overhead here is caused by maintenance of binary tree and parralelization of node lookup message.

D. PAST

PAST is a large-scale archival storage system based on Pastry P2P overlay network. Its main intended purpose is an archival storage; hence the design of PAST was focused on global balance of storage load while providing locally available replicas. The overall load balance is achieved through replica diversion, in which before a replica is stored to a node, it considers the storage load surrounding the itself. If the node is located (in terms of nodeIDs) in a heavily packed area, the replication is diverted to less loaded area. In addition, by storing highly requested files in users' available caches, PAST achieves a graceful degradation of global performance when the total storage reaches its limit. Therefore, in addition to stability, PAST offers fast resource locating. Security in PAST is achieved by issuing smart cards for writing users, and public key cryptography is used in the overall system [12]. PAST became a base overlay storage system to Pastis.

Advantages	Disadvantages
Read and Write access. Highly scalable and overall reliable storage system.	Writing users must have a physical smart card, which is a big liability. In addition PAST is intended as an archival storage, which limits resource management.
Volunteer Computing Extension	
Although PAST is highly scalable, in terms of storage nodes, the use of smart cards strongly limit the physical scalability, in terms of clients. In addition, state-of-the art distributed storage system requires more resource management service than solely archiving.	

E. Pastis

It's a distributed storage system built on top of PAST. It's features of locality, scalability, and replication maintenance are similar to those of PAST. Additional design features are conflict-resolution mechanism, consistency, and security. The main conflict arises when multiple users write to a file concurrently, in which case Pastis' conflict resolution mechanism works by comparing each updates' version number and the unique ID of users who issued the updates. PASTIS supports two consistency models: Close-to-open model, which is also used in both Andrew File System (AFS) and Network File System (NFS), and a more relaxed consistency model called Read-Your-Writes model. Pastis uses write access control and data integrity, and does not provide read access control. Write access control works by by allowing the owner of the file to issue write certificates to certain trustees, who must properly sign the new version of inode and provide his own write certificate when modifying content. This model assumes that all users allowed to write to a given file trust one another [18].

Advantages	Disadvantages
Provides high scalability and locality. Read and write access. Up to twice as fast as Ivy or Oceanstore. Supports multiple consistency models.	Security based on trust between parties. Conflict resolution mechanism is not sophisticated enough for very large scale storage system.
Volunteer Computing Extension	
Locally controlled write access to storage nodes requires each storage nodes to fully trust volunteer computing members. This may create security concerns that are not suitable for distributed storage system used by volunteer computing.	

F. Project Voldemort

Voldemort can be described as a big, distributed and fault tolerant hashtable. It promises high horizontal scalability and higher availability in the cost of lesser convenience in using it due to its primitive API.

Voldemort consists of several layers and each layer in Voldemort implements a simple storage interface that does put, get, and delete operations. Generally, each layer is focused in performing one function such as TCP/IP network communication, serialization, version reconciliation, inter-node routing. Voldemort keeps each of these layers separate, that means they can be configured (by mixing and matching) at runtime to meet different needs.

In Voldemort, anyone can derive location of a value by knowing its key, and lookup is performed in peer-to-peer fashion. Consistent hashing is used to compute the location of each key in the cluster and to prevent data shift between servers when a server is added or removed from the cluster.

Voldemort supports pluggable serialization. We can choose among these kinds of serialization procedure: json, string, java-serialization, protobuf, thrift, avro-generic/avro-specific/avro-reflective, and identity.

Voldemort tolerates the possibility of inconsistency, and resolves inconsistencies at read time (versioning and read-repair). Versioning scheme in Voldemort uses vector clock to guarantee partial order over values.

Advantages	**Disadvantages**
Integrity of data is preserved using versioning and without compromising system availability. Data replication, data partition and server failure are handled transparently. Voldemort layer is built based on plug-in-architecture, and we can configure these layers to suit application needs.	No data found for its performance analysis. However, Voldemort has been effectively proven in production environment through LinkedIn. Only support key-value store, that means if we want to improve the usability (such as a file system), we need to add additional layer on top of Voldemort to perform the job
Volunteer Computing Extension	
We will need to fine-tune the module in each layer to support volunteer computing, which initially can be complicated, but once it is fine-tuned, it will be suitable for volunteer computing.	

G. OceanStore

OceanStore is a two-tiered, fully read/write distributed storage system for any type of files and applications. The inner tier consists of well-connected servers for primary replications – Byzantine Fault Tolerant (BFT) algorithm is used in this tier. The outer tier consists of loosely connected client computers for storage and archives – aggressive replication is used for client replicas and erasure coding is used for reliable archives. In addition to being able to support all types of files and applications in the system, OceanStore provides high fault tolerance through the combined use of BFT and erasure coding. With this design, they've achieved speedup of 4.6 over NFS in Read-only version, and 7.3 slower than NFS in read-write version. For security, it uses threshold cryptography, which allows host computers store files without knowing the contents. They assume that all user infrastructures are untrusted [19].

Advantages	**Disadvantages**
Provides read and write access. High fault tolerance and security is assured, since none of the nodes are trusted.	Writing access slows down the system, significantly.
Volunteer Computing Extension	
High fault tolerance and security is achieved through the use of complex algorithm - BFT. This comes with the cost of limited scalability.	

H. Glacier

Glacier is a highly durable read-only distributed storage system. It is designed to provide availability during cases large scale, correlated, Byzantine failures of storage nodes. As a tradeoff for high durability, Glacier does not support remote write-access. A prototype of Glacier is built on top of Pastry overlay, and makes use of PAST distributed hash table.

Glacier uses a combination of full replicas and erasure coding. A few numbers of full replicas of each files is stored for short-term availability. A redundant amount of erasure-coded fragments are stored among nodes for long term archiving. The nodes containing fragments communicate with each other, and create extra fragment copies when needed. Durability is ensured by spreading redundant amount of fragments across randomly selected nodes in the system. Glacier uses erasure codes and garbage collection for low storage cost, and loosely coupled fragment maintenance protocol to reduce the message costs among nodes.

A security in Glacier is achieved simply because there is no remote write or update access to users. Since users can only overwrite or modify their own local nodes, any malicious activity cannot harm the storage system. Data integrity is achieved by excessive amount of erasure-coded fragment replications across the system.

Similar to Tapestry and CFS, Glacier [14] uses leasing to control the lifetime of a stored data. Objects must be renewed periodically, in order to keep them alive in the archive. Experimental results of its prototype show that Glacier is able to manage a large amount of data with low maintenance, while ensuring high scalability in data size and system size. [14]

Advantages	**Disadvantages**
High durability. Far less vulnerable to large scale failures.	No remote write/update. Excessive Redundancy within the storage system, in order to ensure high fault tolerance. Lifetimes of stored data are limited.
Volunteer Computing Extension	
Due to its high reliability, and security, this storage system is suitable for Volunteer computing. However since there's no remote write and update, this system could be used purely as an archival storage of computed data.	

I. Total Recall

Total Recall is a peer-to-peer storage system that is capable of automatically managing its availability in dynamic changing environment. It is able to perform availability prediction based on empirical results, and to repair itself (dynamic repair) automatically when hosts leave the system and decreasing redundancy. It has redundancy management feature so that Total Recall intelligently chooses between replication, erasure code, and hybrid techniques based on system condition. However, Total Recall is not symmetric. It introduces three types of host in its implementation: master host, storage host and client host. Total Recall's lazy repair with erasure code is suitable for Volunteer Computing environment since it is claimed to have good performance for highly dynamic and highly unavailable environment. However, [10] does not really mention about the security aspect of this storage system except of the usage of erasure code to protect the data. Overall, it has potential for Volunteer Computing environment if the security aspect of Total Recall is designed and implemented fully.

Advantages
Peer-to-peer nature of Total Recall implies high scalability. It offers flexibility for the system administrator to specify availability target. Able to handle highly dynamic and highly unavailable environment by using lazy repair with erasure code technique.
Disadvantages
Security aspect of Total Recall is not really designed and implemented fully.
Volunteer Computing Extension
Total Recall's lazy repair with erasure code is suitable and has potential to be used in Volunteer Computing system as long as the security aspect of Total Recall is designed and implemented fully.

J. Cassandra

Cassandra is a distributed storage system for managing very large amount of structured data spread out across many servers with high availability. The system doesn't support a full relational data model.

DHT in Cassandra is a distributed multi-dimensional map indexed by a key. The row key in a table is a string with no size restrictions. Every operation under a single row key is atomic per replica. Columns are grouped into column families: simple and super column families. Columns could be sorted by time or by name.

API in Cassandra consists of three simple methods: insert, get, delete. Distributed system techniques used in the system:

- *Partitioning.* It is done by means of consistent hashing. To deal with non-uniform data and load distribution and heterogeneity of nodes performance, developers applies nodes moving in the ring, according to their load.
- *Replication.* Each data replicated in N hosts, where N is a replication factor per instance. Coordinator is in charge of replication. Replication policies: Rack Unaware, Rack Aware, Datacenter Aware. For the last two policies Cassandra uses *Zookeeper.*
- *Membership* is based on *Scuttlebutt*, a very efficient anti-entropy Gossip based mechanism.
- *Failure handling.* Cassandra uses a modified version of *φ Accrual Failure Detector* with Exponential Distribution for inter-arrival time for other nodes gossip messages. φ is a value which represent a suspicion level for each of monitored nodes. The more φ the less likelihood % of mistake we will do in the future about its failing.
- *Bootstrapping.* For the fault tolerance, mapping is persisted to disk locally and also in Zookeeper. Then the token is gossiped around the cluster. When a node needs to join a cluster, it reads its configuration file with a few contact points (seeds) within the cluster.
- *Scaling.* Each new node is assigned with a token to alleviate a heavily loaded nodes.
- *Read/write requests.* Write is a write to into a commit log. Writing to in-memory is performed only after successful write to commit log. When in-memory is full it dumps itself to disk. Read operation queries the in-memory data structure before looking into the file in disk. Lookup a key could be done through many data files. But here developers use bloom filter to summarize the keys in the file.

Advantages	Disadvantages
Linear scalability and fault-tolerance. No single point of failure. Map/reduce possible with Apache Hadoop.	Reads are slower than writes. Limitation for column and row size in the system. Cassandra is a bad fit for large objects.
Volunteer Computing Extension	
The main feature of the system is to perform writes faster that reads. It could be a first barrier for VC application, as the main idea of storage systems in general to perform reads more that writes. Also, as the system is highly scalable, VC availability feature could be solved through replication techniques. Finally, fault-tolerance could be handled quite well even with gossip algorithm application. From the other point of view, Cassandra has limitation on the object size, hence, big object can't be stored in the system. That's why all files should be partitioned. Overall, due to good replication policy and fault-tolerance techniques the system could be quite good platform for future VC application.	

K. Riak

Riak is a Dynamo-inspired database, which scales predictably and easily. It is a highly fault-tolerant, scalable and higly available distributed database system with no single point of failure. One important feature of Riak is the data organization. Buckets and keys are the only way to organize data inside of Riak. User data is stored and referenced by bucket/key pairs and Riak uses a backend key/value storage system. This feature allows the use of any kind of key/value storage system as a backend such as Bitcask (Pluggable Backends). Buckets are used to define a virtual keyspace and to provide some per-bucket configurability, such as, replication factor. Riak's client interface speaks of buckets and keys. Internally, Riak computes a 160-bit binary hash of the bucket/key pair, and maps this value to a position on an ordered "ring" of all such values. This ring is divided into partitions and each Riak node is responsible for a partition. Replication is fundamental and automatic in Riak, providing security that the data will still be there if a node on the cluster goes down. All data stored in Riak will be replicated to a number of nodes in the cluster according to the "N" property set on the bucket. It is based on the N,R,W Dynamo values. Another feature is the object versioning. Each update to a Riak object is tracked by a vector clock. Vector clocks determine causal ordering and detect conflicts in a distributed system. Each time an object is created or updated, a vector clock is generated to keep track of each version and ensure that the proper value can be determined in the event that there are conflicting updates. Besides, when there are update conflicts on objects, Riak allows the last update to automatically "win" or returns both versions of the object to the client who has to take a decision to resolve the conflict.

Advantages	Disadvantages
A distributed key/value storage based on Dynamo. N, R and W configurable values. Support of pluggable backend storage systems.	Riak aims for Availability, and Partition (failure) tolerance. The problem is the eventually consistency camp
Volunteer Computing Extension	
Riak could be applicable to the volunteer computing, but there should be defined the size of files that the volunteer computer system uses in order to pick a suitable backend storage.	

L. Dynamo

Dynamo is a distributed storage system developed by Amazon to support its online retail operation. Its main goals are reliability at massive scale, and scalability. It should be able to treat failure handling as a norm.

Dynamo satisfies eventual consistency and target design space of an "always writable" data storage (highly available for write). It allows the flexibility of deciding who performs the conflict resolution, whether it is application layer or the data store layer. It is able to perform incremental scalability (scale out one storage host at a time). It maintains symmetry in the network, which means there is no central server that manages everything. It should be decentralized and able to handle heterogeneity of the nodes.

Dynamo uses consistent hashing to handle partitioning problem; it uses vector clocks for data reconciliation during reads. Sloppy quorum and hinted handoff are used to handle temporary failures. Merkle trees are used to provide data integrity during failure recovery. Dynamo also it uses a gossip-based membership protocol and failure detector.

Advantages
Optimized for storage system usage, whereby we need highly availability for write (with consistency trade off of course). Flexibility on local persistence component, which implies that we can easily changes the storage engines implementation (Berkeley DB, MySQL, Berkeley DB Java edition and in-memory buffer with persistent backing store). Dynamo is highly configurable based on R, W, and N values (number of replicas needed). It has been used, deployed, tested and fine-tuned in Amazon's production environment.
Disadvantages
It has no security feature implemented because it is run in Amazon's internal non-hostile environment (no malicious node). Gossip protocol to update group membership status in Dynamo may not be scalable when used in thousands of nodes, due to overhead in maintaining routing table.
Volunteer Computing Extension
Dynamo works well if the Volunteer Computing system needs high write availability. However, it has no security feature implemented since it is assumed to be used in non-hostile environment. Therefore, current Dynamo implementation is not suitable for Volunteer Computing system unless the security feature is implemented and tested well.

M. Attic

Attic [31] is a software project that creates a secure data distribution network applicable to projects like BOINC that employs P2P data sharing practices to share data across the data distribution layer. This alternative provides a more lightweight and dynamic environment for distributing data. Four main elements: *i)* A data serving application that replicates data on the network. *ii)* Data Centers (data cacher) that cache data, providing the distributed data source overlay. *iii)* A Look up service that keeps track of which Data Centers have individual data items. *iv)* Client applications that download data from Data Centers on the network.

Data Centers are interim storage facilities that provide a buffer between the data serving application and the client applications. This buffer is particularly important for volunteer computing environments because it ensures that the data sources can be trusted by clients. Trust plays a crucial role in volunteer computing environments. If it is broken then the project will fail to attract volunteers. Therefore, in controlled environments, Data Centers are typically issued with certificates signed by a trusted certificate authority allowing client applications to verify the identity of the Data Center when downloading. Attic is based on the Peer-to-Peer Architecture for Data-Intensive Cycle Sharing (P2P-ADICS) research project.

Advantages
It has been designed with volunteer computing in mind, and does not require all participants to share data, or share equally, therefore differing it from other types of P2P file-sharing systems. Secure decentralized approach and BitTorrent-like file swarming techniques to serve data and manage load. It uses HTTP for the transfer data transfer layer (multiple concurrent downloads). It can be integrated with BOINC projects. “Trusted” peers: secure data centers are a way of implementing a super-peer topology for data sharing that would restrict the set of peers that are allowed to propagate data.
Disadvantages
A point of failure: the Lookup Server
Volunteer Computing Extension
Attic can be integrated with BOINC using the Attic-BOINC proxy.

N. Squirrel

It is a decentralized, P2P web cache with a behavior of a centralized web cache system. The key point is to enable web browsers on desktop machines to share their local caches to form an efficient and scalable web cache, without the need for dedicated hardware and the associated administrative cost. Overall, Squirrel is a combination of cooperative web caching and P2P request routing.

Squirrel uses a self-organizated P2P routing substrate Pastry. Pastry is resilient to concurrent node failures and so is Squirrel. The system has to re-fetch a small fraction of cached objects from the origin web server.

Web caching is a different way of decentralized storage application but it still uses similar techniques to provide availability to its system. The main difference of the web cache system is that still there is an opportunity to data recovery from the original server, so that centralized part supports decentralized components in the system. Hence, in such systems there is less attention to the availability side.

Current system Squirrel [5] is a prototype of the volunteer computing application with some centralized support. The main point of the system is a request routing and handling the presence of files inside the system according to clients’ needs.

Advantages	Disadvantages
Latency comparable to the centralized analog. Low-management. Fault resilient .	Web-Cache storage only. System for corporate LAN. Failures result in loss of content.
Volunteer Computing Extension	
Decentralized Web Cache systems could be applicable to the volunteer computing, but centralized control of the system should be implemented as well. In this case also replication and fault-tolerance techniques should be developed further.	

Overall, decentralized web caching in Squirrel, by its nature, is volunteer computing application as users consciously contribute to the system. Hence, their behavior is likely to be more reliable. That’s why, usually, there is no need for supporting more sophisticated replication and availability techniques as the data could be accessed from the original server eventually.

O. Transparent File System (TFS)

A key barrier to the adoption of contributory storage systems is that contributing a large quantity of local storage interferes with the principal user of the machine. To overcome this barrier, the Transparent File System (TFS) [16] is introduced. It provides background tasks with large amounts of unreliable storage without impacting the performance of ordinary file access operations. TFS is a method for contributing disk space in peer-to-peer storage systems.

Advantages
A file system that can contribute 100% of the idle space on a disk while imposing a negligible performance penalty on the local user. Operates by storing files in the free space of the file system so that they are invisible to ordinary files. Normal file allocation *proceeds as if the system were not contributing any space at all*. Ensure the **transparency of contributed data**: the presence of contributed storage should have no measurable effect on the file system, either in performance, or in capacity. **Transparent files:** files that are transparent in the local file system (local file system does not see these files). **Transparent data** or **transparent blocks:** data belonging to such transparent files. Treats transparent blocks as if they were free, overwriting whatever data might be currently stored in them. An ordinary file can be allocated over a transparent file. TFS imposes nearly no overhead on the local user. TFS is specifically designed for contributory applications. The key benefit is that it leaves the allocation for local files intact, avoiding issues of fragmentation
Disadvantages
Sacrifices file persistence. When TFS allocates a block for an ordinary file, it treats free blocks and transparent blocks the same, and thus may overwrite transparent data. Files marked transparent may be overwritten and deleted at any time. Applications using transparent files must ensure the correctness of all file data. Sensitive information stored in ordinary files must be protected from applications trying to read transparent files. The unreliability of files in TFS produces a bandwidth consumption due to the replication that is needed to handle deleted files.
Volunteer Computing Extension
This technique cannot be used for storage in a volunteer system because it requires changes in the local file systems of the volunteers. It means that the all the local file systems, such as Ext3, NTFS and others must be modified

IV. State of the Art

A. Introduction

WHY? In current scientific volunteer computing software infrastructures, such as BOINC and XtremWeb, data is distributed centrally from a project's coordinating nodes or servers. In BOINC, this is achieved through a set of HTTP mirrors, each providing clients with full copies of data input files. Similarly, in XtremWeb, clients are given the URIs of data input files. These centralized systems require projects to not only have the necessary network capacity needed to provide data to all volunteers, but also have data readily available and persistent on their servers at all times to fulfill client requests. Furthermore, the network throughput requirements of serving so many client machines can prove to be an impediment to projects wishing to explore new types of data intensive application scenarios that are currently prohibitive in terms of their large data transfer needs.

Alternatively, a viable approach to such centralized systems is to employ the use of peer-to-peer (P2P) techniques to implement data distribution.

GOALS?

- Offload the central network needs. P2P data storage techniques can be used to introduce a new kind of data distribution system for volunteer computing projects, one that takes advantage of volunteer-side network capabilities.
- Scalability not only needs to take into account the network bandwidth, but also the potential sizes of data with respect to the data and job throughput in a particular VC project, and their distribution over time.
- Security in these systems goes beyond the traditional notion of simply ensuring authentication or file integrity. Due to the volatile and insecure nature of volunteer networks, a product of their open participation policies, there can be reason to enforce limitations on which nodes are allowed to distribute and cache data. By opening the data distribution channels to public participation, security now becomes a larger concern for projects that previously had centrally managed servers. The area of security with respect to VCS can be roughly split into the following: user security and data security. It is important to support both data integrity and reliability, whilst also providing safeguards that can limit a peer nodes' exposure to malicious attacks.
  - *User Security:* any P2P data distribution scheme that is implemented must allow users to opt-out if they do not wish to share their bandwidth or storage capacity.
  - *Data Security:* find security schemes and policies and how to apply them to volunteer networks when selecting and distributing data-sets to peers.

All mentioned above could be presented in the following Figure 4.

HOW? There are many ways this could be implemented, ranging from a BitTorrent-style network, where data is centrally tracked and all participants share relatively equal loads, to KaZaa-like super-peer networks, where select nodes are assigned greater responsibility in the network.

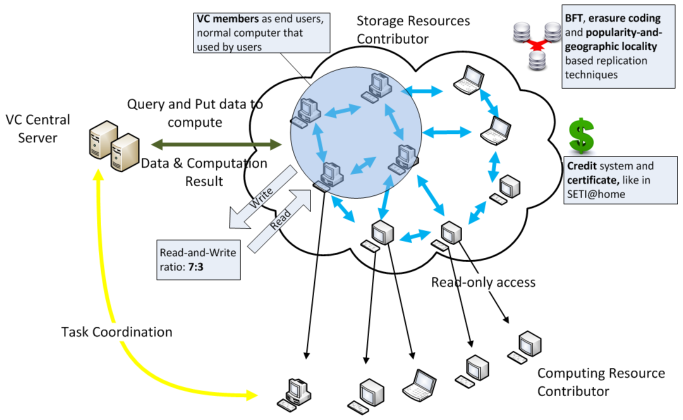

VC members as end users, normal computer that used by users
Storage Resources Contributor
BFT, erasure coding and popularity-and-geographic locality based replication techniques
VC Central Server
Query and Put data to compute
Data & Computation Result
Write
Read
Read-and-Write ratio: 7:3
$
Credit system and certificate, like in SETI@home
Read-only access
Task Coordination
Computing Resource Contributor

Fig. 4. State-of-the-art Decentralized Storage System used by VC

However, applying a traditional P2P network infrastructure to scientific computing, particularly in volunteer computing, can be highly problematic. In such environments, policies and safeguards for scientific data and users' computers become more critical concerns for limiting consumption rather than any technical feasibility.

A tailor-made solution that could take into account the requirements of scientific communities, as opposed to a generic overarching P2P architecture, would have the advantage of facilitating different network topologies and data distribution algorithms, whilst retaining the safety of each participant's computer. Furthermore, each scientific application has different network and data needs, and customized solutions would allow for tailoring the network towards individual requirements, although with the disadvantage of increased development effort, complexity, and code maintenance.

As example there is ADICS [47], a customizable and brokered Peer-to-Peer Architecture for Data-Intensive Cycle Sharing that allows fine-grained provisioning of resources and application of project-based roles to network participants. Specifically, ADICS provides a brokered P2P system that offloads central network needs while limiting client exposure to foreign hosts. The brokered network-overlay introduced in ADICS acts as a buffer, in the form of a select group of trusted data-sharing nodes, between the scheduler and the clients.

B. Characteristics

In designing of the state-of-the-art distributed storage system for VC, it is important to bear in mind that the VC members are the end users of the system. Hence, given the right incentives to both VC members and storage nodes, the following assumptions could be safely made:

- Storage nodes in this system are relatively trust-worthier than those in more hostile systems surveyed in Section III.
- Not only are the storage nodes trustworthy, but also they are more committed. Thus, we expect lower churn rate among the nodes, and the system should not need to take into account the worst-case scenarios mentioned in [14].

By evaluating the successful deployments of other p2p storage systems in our survey, the following characteristics

System	Description	Focus
Farsite	Large scale persistent P2P storage system	SE
Ivy	P2P storage system based on Dhash table logs	ECO, AV
OverNet	P2P DHT-based storage system that used XOR-based metrics for routing	ECO
PAST	Large scale persistent P2P storage system	SC, SE, AV
Pastis	Highly scalable P2P storage system based on trust	SC, ECO
Voldemort	Big, distributed, fault tolerant hashtable	AV
OceanStore	Globally persistent DSS for any-file-anywhere	SC, SE
Glacier	Highly durable DSS for archiving, no remote write	SC, AV, P
Total Recall	P2P storage system. Automatically manage its availability in dynamic changing environment	AV, SC
Cassandra	DSS with no point of failure to store structured data	SC, AV, ECO
Riak	Dynamo-inspired NOSQL storage system	SC, AV
Dynamo	Large scale DSS developed by Amazon	P, AV, ECO
Attic	Secure DSS and BitTorrent-like file swarming techniques to serve data and manage load	SE, AC
Squirrel	Decentralized Web Cache	P, ECO

TABLE I
COMPARISON OF DIFFERENT DECENTRALIZED STORAGE SYSTEM

were defined for state-of-the-art distributed storage system: read/write access, replication and fault-tolerance, and symmetry and availability. A short suggestion of incentives is mentioned, since incentives strongly influence the behavior of any distributed systems.

Read and Write Access:

A need for read/write access for distributed storage system depends entirely on the frequency of file updates. Implementation of file updates and the maintenance of their consistencies among all replications in the distributed system create issues mentioned in [18]. Consideration of updated files as different from their older versions, therefore adding them as new files in the distributed system is mentioned in [11]. This method is more appropriate for distributed system with nodes that do not frequently update data. Also, a distributed storage system can be implemented as an archival storage for computed data from volunteer computing system. Such storage systems designed as archives are mentioned in [14], and [12]. The frequency of file updates from the volunteer computing nodes, such as re-submitting or correction of computational data, should be rare. It is expected to see more of uploading of computational data, rather than correction of one's work. Therefore, the read access vs. write access in our system should have a ratio of about 7:3. Implementations of this type of read/write access is very common in other distributed storage systems, such as [11].

Fault Tolerance & Replication Techniques:

As with aforementioned assumptions, the necessity of extremely high fault tolerance is insignificant in the design of a distributed storage system for volunteer computing. The assumption of extremely high churn tolerance is unnecessary, but a reasonable amount of smart replication techniques and fault tolerance is inevitably important. The replication technique requirements should be low memory and resource usage, and implementation of locality based load balancing. Erasure coding and Byzantine-Fault-Tolerant (BFT) algorithm provide certain amount of fault tolerance, load balancing and lower memory usage, and they have been implemented in [8], [11], [14], and [19]. Also, a replication technique based on popularity, access rate, and file types is used in [5], [10], and [36]. The ideal distributed storage system should use a combination of BFT, erasure coding and popularity-and-locality based replication techniques. To further reduce overhead from data fragment recovery, the storage system could use network coding technique as presented in [32].

Availability and Symmetry:

We define data availability as data localization, an easy location and acquisition of data when necessary. Therefore, data availability and symmetry among the nodes goes hand-in-hand. In the distributed storage system, however, a global symmetry is not necessary, as shown in systems such as [7], [9],[10] and [11]. One way to achieve high availability is by employing geographic locality based replication techniques, such as mentioned in [4]. The majority of the volunteers are located in USA, Germany, UK and Canada [28]. Therefore, geographic locality awareness could improve fast data acquisition as well as global traffic balancing. In addition to geography-based locality, implementation of groups hash tables containing metadata of file replicas is a common method for increasing data acquisition. Finally, a use of BitTorrent like swarming techniques was mentioned in [31] for data serving and load managing. To further improve the availability, data types classification to determine fault tolerance techniques could be employed. One example of this technique is when storing video files, the storage system could use technique from P2P video streaming application.Several independent so-called descriptions will be created from a video file. This method is called Multiple Description Coding (MDC). Each description is a video segment that contributes a certain amount to the video quality and can be decoded independently. The more descriptions are received, the higher is the received quality. Hence, each data file could be divided into n amount of pieces, e.g. odd and even horizontal and vertical frames for the video file, and then erasure coding/network coding could be applied to each of the piece itself.

Incentives:

The use of incentives in the distributed storage system nodes, as well as the volunteers in the volunteer computing system, is an effective way to increase and maintain highly committed members. Incentives based on credit system, in which individuals are granted credits and certificates for their contribution of computing powers, have proven very effective in systems like SETI@home [37]. These type incentives encourage users to compete among each other with their credits, thus contributing more towards the computing system. Similarly, a credit and certificate based on amount of storage shared, and available time in the system would contribute towards a highly available distributed storage system. Another possible incentive technique is a government tax break for both volunteer computing members as well as the storage node members. There are over 1 million volunteer computing nodes in the world [31] and the majority of them are located in four countries – USA, Germany, UK and Canada. Thus, it is likely that the distributed storage nodes would be also located in these countries. A collaboration and support of the responsible government bodies in these countries would be a powerful driving force in increased number of storage nodes and volunteer members.

C. Challenges

Providing the right incentive

As mentioned in Section IV-B, proposed distributed storage system for VC is based on peer-to-peer concept, which involves volunteer computing participants. The volunteer-nature of this model requires appropriate incentive system to attract participants. Our proposal here is to use existing models of incentives, with further testing to prove its effectiveness in volunteer-computing-based storage sharing

Security

Another challenge that may surface is security. The proposed system should ensure that user data is not affected by VC data that stored in participants' system or user security must be ensured [47]. Client system should not provide loophole for hackers to intrude the system. Existing security techniques that are currently used in sharing computational power need to be re-tested and re-evaluated before they are re-used for storage sharing in VC environment.

VC system should also consider the security of the data that are stored in volunteer nodes (data security), especially the integrity of the data. It should be able to enforce and detect the correctness of data stored in volunteer nodes and prevent malicious user to change or remove the data. For example the usage of Byzantine-Fault-Tolerant (BFT) replica in FARSITE, extensive use of erasure coding in Glacier, and MD5 checksum in Attic.

Integration into current VC Environment

Once we define or implement the decentralized storage system, we need to integrate it to an existing VC environment. Challenge in this area includes differences in programming language in implementing the storage system and VC platform (such as BOINC). This challenge is highlighted in [47]. To solve this challenge, introduction of proxy solution to interface both sides may be useful.

We may also need to introduce separate layer between VC participants that contribute computing resources and storage resources. These layers will provide several API as interface hence we will have flexibility in interchanging the implementation of storage layer.

V. CONCLUSIONS

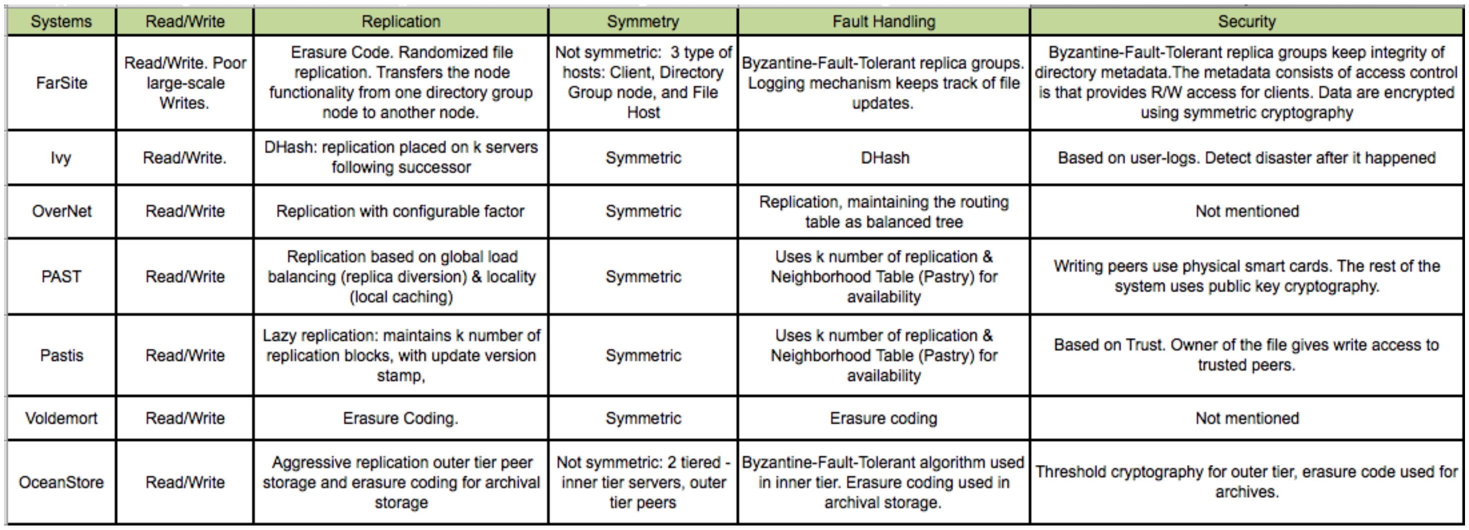
The implementation of decentralized storage system (DSS) in volunteer computing (VC) field may open doors to numerous research ideas for both VC and DSS fields. In this survey, DSS characteristics critical for VC integration are identified. They are availability, scalability, eventual consistency, performance and security. In addition to those, various properties of DSS were discussed and evaluated based on VC suitability. Attic, Dynamo and Riak are the top-three most suitable DSS for VC. Attic is built with VC in mind, and it focuses on availability and security. Dynamo focuses in availability, which is beneficial for VC. It has also been used in production environment by Amazon, which brings significant advantage over the other DSS. Riak is the third most useful DSS for VC. It is inspired by Dynamo, and it focuses on scalability and availability.

The main characteristics of ideal DSS for VS are proposed. They are

- seven-to-three (7:3) read and write ratio
- VC-based membership in storage system
- BFT, erasure coding, and popularity-and-geographic locality based replication techniques
- Credit system and certificate for incentive

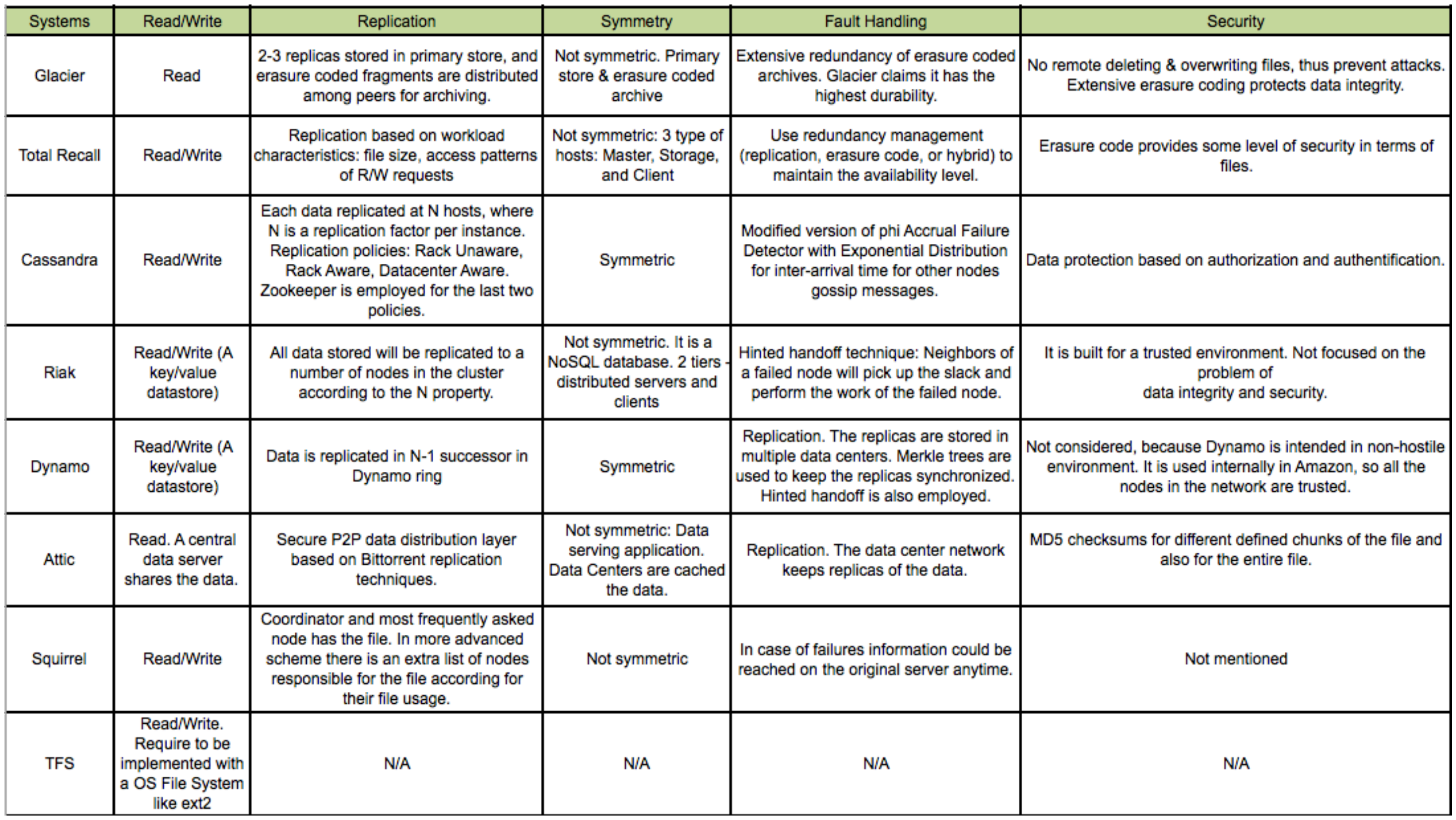
Challenges in designing and implementing the ideal DSS are also discussed.

APPENDIX



Systems	Read/Write	Replication	Symmetry	Fault Handling	Security
FarSite	Read/Write. Poor large-scale Writes.	Erasure Code. Randomized file replication. Transfers the node functionality from one directory group node to another node.	Not symmetric: 3 type of hosts: Client, Directory Group node, and File Host	Byzantine-Fault-Tolerant replica groups. Logging mechanism keeps track of file updates.	Byzantine-Fault-Tolerant replica groups keep integrity of directory metadata.The metadata consists of access control is that provides R/W access for clients. Data are encrypted using symmetric cryptography
Ivy	Read/Write.	DHash: replication placed on k servers following successor	Symmetric	DHash	Based on user-logs. Detect disaster after it happened
OverNet	Read/Write	Replication with configurable factor	Symmetric	Replication, maintaining the routing table as balanced tree	Not mentioned
PAST	Read/Write	Replication based on global load balancing (replica diversion) & locality (local caching)	Symmetric	Uses k number of replication & Neighborhood Table (Pastry) for availability	Writing peers use physical smart cards. The rest of the system uses public key cryptography.
Pastis	Read/Write	Lazy replication: maintains k number of replication blocks, with update version stamp,	Symmetric	Uses k number of replication & Neighborhood Table (Pastry) for availability	Based on Trust. Owner of the file gives write access to trusted peers.
Voldemort	Read/Write	Erasure Coding.	Symmetric	Erasure coding	Not mentioned
OceanStore	Read/Write	Aggressive replication outer tier peer storage and erasure coding for archival storage	Not symmetric: 2 tiered - inner tier servers, outer tier peers	Byzantine-Fault-Tolerant algorithm used in inner tier. Erasure coding used in archival storage.	Threshold cryptography for outer tier, erasure code used for archives.

Fig. 5. Decentralized Storage Systems Features



Systems	Read/Write	Replication	Symmetry	Fault Handling	Security
Glacier	Read	2-3 replicas stored in primary store, and erasure coded fragments are distributed among peers for archiving.	Not symmetric. Primary store & erasure coded archive	Extensive redundancy of erasure coded archives. Glacier claims it has the highest durability.	No remote deleting & overwriting files, thus prevent attacks. Extensive erasure coding protects data integrity.
Total Recall	Read/Write	Replication based on workload characteristics: file size, access patterns of R/W requests	Not symmetric: 3 type of hosts: Master, Storage, and Client	Use redundancy management (replication, erasure code, or hybrid) to maintain the availability level.	Erasure code provides some level of security in terms of files.
Cassandra	Read/Write	Each data replicated at N hosts, where N is a replication factor per instance. Replication policies: Rack Unaware, Rack Aware, Datacenter Aware. Zookeeper is employed for the last two policies.	Symmetric	Modified version of phi Accrual Failure Detector with Exponential Distribution for inter-arrival time for other nodes gossip messages.	Data protection based on authorization and authentification.
Riak	Read/Write (A key/value datastore)	All data stored will be replicated to a number of nodes in the cluster according to the N property.	Not symmetric. It is a NoSQL database. 2 tiers - distributed servers and clients	Hinted handoff technique: Neighbors of a failed node will pick up the slack and perform the work of the failed node.	It is built for a trusted environment. Not focused on the problem of data integrity and security.
Dynamo	Read/Write (A key/value datastore)	Data is replicated in N-1 successor in Dynamo ring	Symmetric	Replication. The replicas are stored in multiple data centers. Merkle trees are used to keep the replicas synchronized. Hinted handoff is also employed.	Not considered, because Dynamo is intended in non-hostile environment. It is used internally in Amazon, so all the nodes in the network are trusted.
Attic	Read. A central data server shares the data.	Secure P2P data distribution layer based on Bittorrent replication techniques.	Not symmetric: Data serving application. Data Centers are cached the data.	Replication. The data center network keeps replicas of the data.	MD5 checksums for different defined chunks of the file and also for the entire file.
Squirrel	Read/Write	Coordinator and most frequently asked node has the file. In more advanced scheme there is an extra list of nodes responsible for the file according for their file usage.	Not symmetric	In case of failures information could be reached on the original server anytime.	Not mentioned
TFS	Read/Write. Require to be implemented with a OS File System like ext2	N/A	N/A	N/A	N/A

Fig. 6. Decentralized Storage Systems Features

References

[1] J.-M. Busca, F. Picconi, and P. Sens, *Euro-Par 2005 Parallel Processing*, vol. 3648, Springer Berlin / Heidelberg, 2005, p. 644.

[2] *Distributiveness.* [Online]. Available: http://blog.proskurnia.in.ua/. [Accessed: 27-Mar-2012].

[3] A. Verma, S. Venkataraman, M. Caesar, and R. H. Campbell, *Handbook of Data Intensive Computing,* Springer New York, 2011, pp. 109–127.

[4] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, *Ivy: a read/write peer-to-peer file system,* SIGOPS Oper. Syst. Rev., vol. 36, no. SI, pp. 31–44, Dec. 2002.

[5] S. Iyer, A. Rowstron, and P. Druschel, *Squirrel: a decentralized peer-to-peer web cache,* in Proceedings of the twenty-first annual symposium on Principles of distributed computing, New York, NY, USA, 2002, pp. 213–222.

[6] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, *A survey of peer-to-peer storage techniques for distributed file systems,* in International Conference on Information Technology: Coding and Computing, 2005. ITCC 2005, 2005, vol. 2, pp. 205–213 Vol. 2.

[7] *OceanStore. Prototype:Pond.* [Online]. Available: http://static.usenix.org/events/fast03/tech/rhea/rhea_html/.

[8] *Voldemort, Project Voldemort. A distributed Database.* [Online]. Available: http://project-voldemort.com/.

[9] *Riak.* [Online]. Available: http://wiki.basho.com/.

[10] R. Bhagwan, K. Tati, Y. C. Cheng, S. Savage, and G. M. Voelker, *Total recall: System support for automated availability management.*

[11] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, *FARSITE: Federated, available, and reliable storage for an incompletely trusted environment,* ACM SIGOPS Operating Systems Review, vol. 36, no. SI, pp. 1–14, 2002.

[12] A. Rowstron and P. Druschel, *Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility,* in Proceedings of the eighteenth ACM symposium on Operating systems principles, New York, NY, USA, 2001, pp. 188–201.

[13] A. Lakshman and P. Malik, *Cassandra,* ACM SIGOPS Operating Systems Review, vol. 44, no. 2, p. 35, Apr. 2010.

[14] A. Haeberlen, A. Mislove, and P. Druschel, *Glacier: highly durable, decentralized storage despite massive correlated failures,* in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2, Berkeley, CA, USA, 2005, pp. 143–158.

[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, *Dynamo: Amazon's highly available key-value store,* 2007, pp. 205–220.

[16] J. Cipar, M. D. Corner, and E. D. Berger, *Contributing storage using the transparent file system,* Trans. Storage, vol. 3, no. 3, Oct. 2007.

[17] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris, *Efficient replica maintenance for distributed storage systems* in Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3, Berkeley, CA, USA, 2006, p. 4–4.

[18] J. Busca, F. Picconi, and P. Sens, *Pastis: A highly-scalable multi-user peer-to-peer file system,* in in Euro-Par 2005, 2005.

[19] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, *Pond: the OceanStore Prototype,* 2003, pp. 1–14.

[20] D. Toth, R. Mayer, and W. Nichols, *Increasing Participation in Volunteer Computing,* in 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011, pp. 1878–1882.

[21] Lei Ni and A. Harwood, *P2P-Tuple: Towards a Robust Volunteer Computing Platform,* in 2009 International Conference on Parallel and Distributed Computing, Applications and Technologies, 2009, pp. 217–223.

[22] D. Lázaro, D. Kondo, and J. M. Marquès, *Long-term availability prediction for groups of volunteer resources,* Journal of Parallel and Distributed Computing, vol. 72, no. 2, pp. 281–296, Feb. 2012.

[23] E. M. Heien, N. Fujimoto, and K. Hagihara, *Computing low latency batches with unreliable workers in volunteer computing environments,* in IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008, 2008, pp. 1–8.

[24] T. Ghafarian-M., H. Deldari, H. Mohhamad, and M. Yaghmaee, *Proximity-Aware Resource Discovery Architecture in Peer-to-Peer Based Volunteer Computing System,* in 2011 IEEE 11th International Conference on Computer and Information Technology (CIT), 2011, pp. 83–90.

[25] F. Costa, L. Silva, and M. Dahlin, *Volunteer Cloud Computing: MapReduce over the Internet,* in 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011, pp. 1855–1862.

[26] D. Bermbach, M. Klems, S. Tai, and M. Menzel, *MetaStorage: A Federated Cloud Storage System to Manage Consistency-Latency Tradeoffs,* 2011, pp. 452–459.

[27] A. L. Beberg and V. S. Pande, *Storage@home: Petascale Distributed Storage,* in Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, 2007, pp. 1–6.

[28] D. P. Anderson and G. Fedak, *The Computational and Storage Potential of Volunteer Computing,* in Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06, 2006, vol. 1, pp. 73–80.

[29] *SAN.* [Online]. Available: http://snia.org/education/storage_networking_primer/san.

[30] J. W. Mickens and B. D. Noble, *Exploiting availability prediction in distributed systems,* in Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3, Berkeley, CA, USA, 2006, p. 6–6.

[31] A. Elwaer, A. Harrison, I. Kelley, and I. Taylor, *Attic: A Case Study for Distributing Data in BOINC Projects,* in Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, 2011, pp. 1863–1870.

[32] M. Martalo and, M. Picone, M. Amoretti, G. Ferrari, and R. Raheli, *Randomized network coding in distributed storage systems with layered overlay,* in Information Theory and Applications Workshop (ITA), 2011, 2011, pp. 1–7.

[33] *An evaluation of distributed datastores using the appscale cloud platform,* pp. 305–312, 2010.

[34] *Erasure coding vs. replication: A quantitative comparison,* pp. 328–337, 2002.

[35] M. Brantner, D. Graf, D. Kossmann, T. Kraska, and D. Florescu, *Building a Database in the Cloud,* ETH Zürich, 2009.

[36] Wikipedia contributors, *Overnet,* Wikipedia, the free encyclopedia. Wikimedia Foundation, Inc., 09-Apr-2012.

[37] SETI@ Home. [Online]: http://setiathome.berkeley.edu/cert_print.php

[38] S. Androutsellis-theotokis, *A Survey of Peer-to-Peer File Sharing Technologies.* 2002.

[39] B. Y. Zhao, J. Kubiatowicz, A. D. Joseph, and others, *Tapestry: An infrastructure for fault-tolerant wide-area location and routing,* 2001.

[40] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, and others, *Oceanstore: An architecture for global-scale persistent storage,* ACM Sigplan Notices, vol. 35, no. 11, pp. 190–201, 2000.

[41] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, *Network coding for distributed storage systems,* Information Theory, IEEE Transactions on, vol. 56, no. 9, pp. 4539–4551, 2010.

[42] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, *Wide-area cooperative storage with CFS,* ACM SIGOPS Operating Systems Review, vol. 35, no. 5, pp. 202–215, 2001.

[43] D. P. Anderson, *BOINC: A system for public-resource computing and storage,* 2004, pp. 4–10.

[44] O. Costa, L. Silva, I. Kelley, I. Taylor, O. Costa, L. Silva, I. Kelley, I. Taylor, and C. Tr, *Peer-To-Peer Techniques for Data Distribution in Desktop Grid Computing Platforms.* 2007.

[45] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Neri, and O. Lodygensky, *Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid,* Future Generation Computer Systems, vol. 21, no. 3, pp. 417–437, Mar. 2005.

[46] D. P. Anderson, *BOINC: A System for Public-Resource Computing and Storage,* in Grid Computing, IEEE/ACM International Workshop on, Los Alamitos, CA, USA, 2004, vol. 0, pp. 4–10.

[47] I. Kelley and I. Taylor, *A peer-to-peer architecture for data-intensive cycle sharing,* in Proceedings of the first international workshop on Network-aware data management, New York, NY, USA, 2011, pp. 65–72.

[48] F. Desprez, *Grids, P2P and Services Computing.* Springer, 2010.