Data Universe™

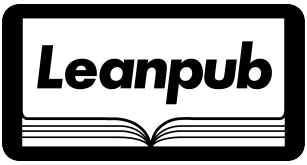
Distributed Storage and Resource Sharing for the 21st Century

Brian McMillin

Data Universe™

Distributed Storage and Resource Sharing for the 21st Century

This version was published on 2012-06-04



This is a Leanpub book, for sale at:

http://leanpub.com/DataUniverse

Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing. To learn more about Lean Publishing, go to: http://leanpub.com/manifesto

To learn more about Leanpub, go to: http://leanpub.com

Tweet This Book!

Please help Brian McMillin by spreading the word about this book on Twitter!

The suggested hashtag for this book is #DataUniverse.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#DataUniverse

Also By Brian McMillin

Replicator Technology

Contents

Data Universe™ Specifications

Brian McMillin

Abstract

This paper describes a distributed data exchange system for the Internet. The techniques allow for a totally decentralized exchange of arbitrary data among participating computer systems. Techniques for maintaining data integrity and anonymity in the presence of arbitrarily unreliable connections and storage media are discussed. All data structures required for a working implementation are described.

The distributed search mechanism described here represents a subset of a more generalized distributed computing capability. Potential extensions would use these data transport and directory search features to form the basis of a global supercomputing network. Such a network would use shared resources for applications ranging from automated data backup to massively parallel computation.

Introduction

The Data Universe™ is a peer-to-peer file exchange system. Its implementation is totally decentralized and anonymous.

Directory structures and bulk data are stored in anonymous, variable-length blocks on one or more host computer systems. File data is divided into multiple blocks, each of which may reside on one or more host computers. Distributed queries allow network-wide searches for files and their constituent blocks. All network transfers take the form of blocks pushed from host to host. Fault tolerance is inherent in the design in that no host or data link is required to be reliable. Redundancy of directory and data storage, distributed processing of search functions and autonomous movement and replication of all data provide network robustness.

All data within the Universe tends to replicate and "popular" data tends to replicate faster due to the action of Queries which tend to duplicate that data. Once introduced into the Universe, it is difficult to remove or censor a piece of information. Queries are non-deterministic, so there is no guarantee that all copies of a particular data block could ever be found. The anonymous nature of data blocks themselves means that individual hosts never need be aware of the actual content of the Blocks in their repository. File Description blocks may be updated with new annotations, typically reflecting user's experiences with a particular file. These newly annotated Blocks are added to the Universe but do not supplant the original descriptions.

Data Blocks tend to migrate to multiple Hosts, each of which automatically make those blocks available to Queries and access by other Hosts. This (conceptually) connectionless data movement

has several advantages over traditional file transfer methods. Data moving between computers with different speeds of physical media do not waste resources. Transfers to a fast computer may be coming from multiple slower computers. Transfers to a slow computer do not tie up connection slots on a fast machine. "Hot spots" in the network tend to be eliminated since Hosts that receive popular data Blocks automatically share them, thus acting as a distributed resource to relieve congestion on the first Hosts for the Block.

Security and Intellectual Property

It is recognized that the Data Universe will be used for the distribution of all types of data. The distribution and ready availability of data from anywhere in the world is one of the fundamental tenets of the entire Internet. Certain jurisdictions and organizations attempt to exercise arbitrary dominion over what they consider proprietary, copyrighted or unlawful sequences of data bits. The Data Universe architecture is designed to protect storage and transport providers from arbitrary legal or regulatory action. All data files in the Universe are separated into at least two parts: a File Description (FD) and the User Data (UD) Block(s). The Description and Data need not reside on the same host, but both are required to re-create the original data file. This insulates the Host computer and its administrator from any claims that it contained proscribed data.

A series of policies enforced on the Host can provide any desired degree of protection from such claims.

1. At the lowest level, the host may ignore all semantic issues and simply participate in the Universe.

2. A host may adopt a policy to never simultaneously have resident in its repository all of the Blocks required to re-createa particular file.

3. A host may choose to allow File Descriptions (FD) or User Data (UD) but not both in its repository.

4. When adding files to the Universe, a host may choose to break even small files (less than the maximum single Block length of 65,500 bytes) into multiple data Blocks.

5. When adding files to the Universe, a host may choose to encrypt the User Data (UD) Blocks. The key information required to decrypt the data would be stored independently within the Universe. This adds a third component (in addition to the FD and UDs) that must be present at a single machine to extract the original data.

6. When adding files to the Universe, a host may break the original file into multiple Blocks in Stripes instead of blocks of consecutive data. Arbitrarily elaborate extensions could allow redundant encoding (for example, Reed-Solomon) to enable reconstruction of files with missing blocks.

Limitations of Prior Systems

Previous peer-to-peer systems suffer from several limitations that are not present in the Data Universe architecture. These limitations leave the users and operators of previous networks vulnerable to serious claims in many legal jurisdictions.

1. The data and descriptions of shared files are kept together on user's systems. This is a very traditional approach that does nothing to mitigate complete exposure of a user's intent in the case where the physical hardware (hard drive) is compromised to an adversary.

2. The user must publish a list of file descriptions that are available from his computer before they can be accessed by other peers. This eliminates any chance of actual anonymity on the part of the user.

3. A list of computer names (IP addresses) and their shared file descriptions are propagated to and stored in other systems on the network. This ensures that the scope of the user's interests are easily available to an adversary.

4. File searches routinely return a list of the users sharing a specific file. This allows an adversary to target groups of individuals with similar interests.

5. Files must be specifically shared by each user. This leads to network bottlenecks where there is a single source of a popular file, and a tendency for users to download files

without providing compensatory upload capability. This also represents a legal vulnerability in that it makes clear the user's intent to redistribute selected files to others.

6. No revision or annotation of file descriptions is possible without the assistance of the user sharing the file. Actual data users do not have the ability to correct errors (intentional or unintentional) in the file descriptions. This allows spammers or other zealots to waste user's time and network bandwidth with malformed or malicious entries.

7. There is no optimization of resources. Bandwidth and disk space are allocated in an uncoordinated manner by individual users.

Glossary

Block - A block is a variable-length sequence of binary bytes. Blocks may contain from 5 to 65,505 bytes. Blocks come in five types based on their content: User Data (UD), File Description (FD), Directory List (DL), Host List (HL), and

Query (QU). The first 5 bytes are reserved for a block signature. The maximum length is selected to ensure that a Block ID and the Block itself can always be transported in a single IP Frame.

Block ID - A Block ID is a sequence of from 28 to 30 printable characters. The Block ID is the printable representation of a L-Hash descriptor of the data contained within the block. Even a zero-length block will have a five character signature so there will always be at least 28 characters in a Block ID.

Block Signature - Five characters at the beginning of each data block that identify the type of data in the block. The first character is literal '#', the second pair identify the block type (either 'UD', 'FD', 'DL', 'HL', or 'QU') and the third pair identify the compression algorithm (either 'RD', or 'LZ'). The '#' is chosen to delimit the end of the variable length Block ID prepended in the data communication channel.

DateTime - standard printable ASCII form of the creation date and time of a file. Used to allow recreation of more complete directory entries for files extracted from the Universe. The value contains up to 14 decimal digits in the format: yyyymmddhhnnss. If the leading digit is a "2" it and any subsequent zeroes are suppressed making this a variable-length form which will use only 11 digits during this decade.

File ID - A File ID is a sequence of at least 27 printable characters.

The File ID is the printable representation of a LHash descriptor of the data within the file. File IDs are used to consolidate different File Names and/or File Descriptions that describe the same content. File IDs are also used to ensure the integrity of reconstructed multi-Block files.

Host Computer - A Host is a computer that participates in the Data Universe by running the Universe kernel application. A host also makes available resources that include CPU time, Storage space, a TCP/IP socket and a certain amount of Bandwidth to the Internet.

Host ID - A Host ID in the current implementation is a URL containing an IP address and port number in printable ASCII text. See RFC 2732 for formats of literal IP addresses.

L-Hash - L-Hash is the algorithm used to create printable Block IDs and File IDs. The printable form uses 64 ASCII characters from the set `['0'..'9','A'..'Z','a'..'z','$','%']` to represent 6-bit values. Leading ASCII zero characters are suppressed to create the variable-length printable form. The recommended algorithm is L-SHA1. This implies that Block IDs and File IDs will be 27 or more printable characters in length.

L-MD5 - A modification of the RFC1321 MD5 message digest algorithm in which the input data length in bits is prepended onto the 128-bit message digest value.

L-SHA1 - A modification of the RFC3174 SHA1 Secure Hash algorithm in which the input data length in bits is prepended onto the 160-bit message digest value.

Query - A type of data Block that contains instructions for searching the Repository of one or more Hosts and returning the results. The Data Universe Idle Process scans the Repository

looking for Query Blocks. As they are found, they are processed and disposed of either by (1) returning results, (2) forwarding to another Host, or (3) discarding. A list of recent Queries prevents the same Query from running more than once on a given Host.

Repository - The storage area on a Host computer that contains data Blocks. A configuration parameter allows the administrator of each Host to limit the size of the Repository. Simple implementations may store each block in a separate disk file using the signature and L-Hash as the file name. (Windows implementations may be restricted by their inability to differentiate upper- and lower-case filenames.)

Slicing - A general term that means breaking up an arbitrarily large data file into a set of one or more User Data (UD) Blocks. Typically, the first step is to run a compression algorithm. The results are then divided into Blocks that do not exceed 65,500 bytes. The size of the blocks and whether they contain consecutive data (or are broken into stripes) are arbitrary decisions made at the time the file is entered into the Universe. Additional blocks may be created to implement error correction logic. These may be simple parity-based blocks or they may incorporate Reed-Solomon Forward Error Correction coding. The goal is to allow complete and accurate reassembly of the original file, even in the absence of all of the data blocks. The reconstruction instructions (including the list of Block IDs, slicing/striping/ECC, and decompression algorithm) are included in the File Description (FD) Block.

Timestamp - standard printable ASCII form for time-of-day values used in File Descriptions, Directory Lists, Host Lists and Queries. The value contains exactly 12 decimal digits representing Universal Time in the format: yymmddhhnnss. Since

the timestamp is predominately used for expiration times and sorting, simple string comparisons will suffice in most instances. Differs from a file's DateTime which is variable length and based on local time.

Implementation

The Data Universe is implemented in a compact, easy-to-distribute form.

`UniverseKernel.exe` implements the data storage interchange functions.

`Universe.ini` is the configuration file that specifies resource allocations for the Kernel.

`Universe.exe` is the user interface that allows files to be shared in the Universe and searches to find files.

`Repository\` is the directory that contains data Blocks stored as individual files with Block IDs as names.

`AddFiles\` is the directory which contains files to add to the Universe.

`ExtractedFiles\` is the directory which contains files retrieved from the Universe.

The Data Universe creates a directory which houses the Repository of data Blocks. These files are named with their Block IDs and use the standard operating system file system for disk management. Any necessary file or disk maintenance may be done with existing tools. None are provided with the Data Universe. Note that the implementation under Windows uses a hexadecimal version of BlockIDs for naming files, since upper- and lower-case is indistinguishable in the file system.

Directories are provided for files to be added to or extracted from the Universe. This provides isolation of files for security and anti-virus quarantine. The speed of add and extract operations is non-deterministic, so autonomous operation is expected.

The Data Universe may be removed from a Host by simply deleting all associated files. Data Universe Configuration File

The operation of the Universe is controlled by a simple configuration file, Universe.ini. A sample configuration file is listed below:

```
[Data Universe]
HostID=192.168.2.10:1234
CPU=10
Interval=5
Inbound=500KB
Outbound=100KB
RAM=15MB
Disk=1000MB
HostFD=Yes
HostUD=Yes
```

Getting Started

When joining the Data Universe, the administrator of a particular host sets some simple policies relating to the resources that he wishes to contribute to the Universe. He chooses an amount of disk space, a TCP/IP socket, a CPU usage limit and inbound and outbound network bandwidth limits. From this point on, the operation of the Universe is autonomous. Files may be added to or extracted from the Universe via a user interface whose operation is essentially independent of the Universe itself.

The initial distribution of the Data Universe software contains a seed version of a Host List (HL) Block. The new Host adds itself to the list. In the absence of anything else to do, the Universe Idle Process periodically pushes its new version of a Host List to the other Hosts already on the list. Eventually, one of these transfers should find a live Host. This host will subsequently return an updated Host List block. The new host has now joined the Data Universe.

An identical process occurs when a Host restarts, except that the initial Host List (HL) Block that it uses is the most recent one stored before shutdown. File Introduction

A user interface is provided to allow files to be introduced into the Data Universe. The file name, a text description, and optional parameters such as compression, slicing and error correction control the process. The resulting File Description and User Data blocks are placed in the local repository. During the normal operation of the network these blocks will be replicated and distributed to multiple hosts. File Extraction

Extracting a file from the Data Universe involves two primary

steps: Selection and Retrieval. Selection is the process of choosing a particular to retrieve. Retrieval involves locating the separate pieces of a file and reassembling them on the user's computer.

File selection begins with the construction of a suitable Query to the network. The query describes the desired file by name or text description. Wildcard pattern matching is used to allow partial or incomplete information to yield relevant results. Over a period of time hosts in the network will return File Description blocks that match the Query parameters. The user examines the text descriptions and chooses appropriate entries for the Retrieval process. The selected File Descriptions contain block lists required to access the data of the actual files. Additional Queries are then made to find and return the required User Data blocks to reconstruct the file. At this point, a degree of sophistication can be incorporated into the Queries to allow optimization of the network bandwidth and processing resources. Initial Queries may return Directory List (DL) records instead of User Data. From these Directory Lists, specific Queries may be formulated to allow parallel transfer of blocks from multiple hosts. This also eliminates the redundant transfers that would result from simple Queries requesting widely available blocks.

When the necessary blocks have been received by the local host the file is reconstructed and its integrity verified. Error correction algorithms may be applied to repair files with missing blocks, or in the (very rare) instance of duplicate LHash block signatures. Data Communication

Several options exist for the transfer of data blocks from one host to another. The conceptually preferred method pushes blocks as connectionless datagrams from an originator to a recipient.

This eliminates the time-consuming handshake required for a TCP/IP connection that will generally be used for only a single block. Although connectionless transmission is preferred, there are many situations (involving firewalls, for example) in which it would fail. Therefore, a second choice involves connections using HTTP. Data blocks may be transferred bidirectionally as a POST and response. The Data Universe transport protocol may be implemented as a script for an Apache web server.

The size of data blocks is chosen to ensure that the total size of a block on the data channel is less than or equal to 65,535 bytes. This is the upper limit of a TCP/IP V4 data packet. The actual contents of a Block on a channel are illustrated as follows:

	Block Length	Hash	Signature	Data
Min Length	8 bits	160 bits	40 bits - 5 bytes	0 bits
Max Length	20 bits	160 bits	40 bits - 5 bytes	65,500 bytes
	28 - 30 bytes (6-bit encoding)		5 bytes	0 - 65,500 bytes

Data Formats

All data transfer and storage in the Universe is based on the use of named, variable-length blocks of data. All Data Transfers are in the form of a connectionless datagram sent from one host to another. The datagram contains the Block ID and the Block of data. The recipient verifies that the Block ID matches the Block and stores the Block in its local repository.

Each Block contains signature bytes on the beginning that identify it as one of the five basic types of Blocks.

UD	User Data	The body of files, usually compressed
FD	File Description	The File Name(s), text description(s) and list of UD blocks that are the data
DL	Directory List	List of Block IDs in the repository of a particular host
HL	Host List	List of Host Addresses and their anticipated longevity in the Universe
QU	Query	Block containing a text description of a search to be performed on the repository

UD - User Data Block

User Data may be stored in the Universe in raw, uncompressed form. The data is broken into segments of up to 65,500 bytes. The signature characters "#UDRD" are the first five characters of each block, followed by the segment of data. User Data may be compressed using a LZH algorithm prior to entry into the Universe. The resulting compressed data are broken into blocks of up to 65,500 bytes. The signature characters "#UDLZ" are the first five characters of each block, followed by the segment of compressed data. Other compression algorithms may be used in the future.

FD - File Description Block

A File Description (FD) Block associates a File ID with one or more file names, zero or more file description text strings, and a list of one or more User Data (UD) Blocks that contain the actual file data. The FD Block is formatted in an XMLlike manner for ease of searching and parsing.

Some files may be broken into an extremely large number of UD Blocks. This may cause the list of Block IDs to exceed the capacity of a single FD Block. Multiple FD Blocks may be cross referenced by including an indirect reference to an UD Block in the list of UD Block IDs. An indirect reference is a Block ID with an "@" on the front. This UD block is interpreted as a sub-list of UD Block IDs to be inserted into the list.

Each File Name and File Description within the FD block has an associated timestamp. Typically, names and descriptive text are sorted into reverse chronological order within a FD, with older information falling off the end. Names and text descriptions that share the same time stamp may be consolidated into one sub-record. A File Description block is a block of ASCII text formatted as follows:

```
#FDRD<ID=FileID>
<bl="BlockID,BlockID,BlockID">
<name="autoexec.bat",text="Everyone should have this file",
date=30721095432, ts=030721123456 />
<text="Solves all your DOS problems", ts=030721123457 />
<text="Danger! Reformats hard disk", signed="Guardian",
pgp="asdf", ts=030722010015 />
</bl>
```

Disk Files are uniquely identified by their File IDs which are based solely on the file content. This unique content is associated with one or more Block ID lists. The list of Block IDs allow the file to be reconstructed from scattered pieces. The integrity of the result is verified by comparing against the File ID.

File Names (Windows, etc. File Names), zero or more Text Descriptions, and an ordered list of one or more Block IDs.

As part of its normal operation, a Host will typically scan its repository for FD Blocks with duplicate File IDs. The Blocks with the most recent timestamps may be arbitrarily retained, or File Name and Text Descriptions may be merged to create new, more appropriate FD Blocks. Note that it is possible for the same data file to be entered into the Universe many times, possibly using different Slicing or Compression strategies. This means that the Block IDs in the Block list would not necessarily be the same.

DL - Directory List Block

A Directory List (DL) Block contains a Host ID, an expiration timestamp and a list of Block IDs that are available on the host. The Block ID list need not be exhaustive and is chosen in a more-or-less arbitrary manner by the host. Directory List Blocks are created periodically during the Data Universe Idle process and pushed to arbitrary Hosts. In addition, Directory List Blocks may be created by the operation of Queries. A Query may specify that the return value be a Directory List indicating which of a set of Block IDs are present on the target system. This is normally used in anticipation of requesting those data Blocks from one of several Hosts.

```
#DLRD<Host=HostID, expire=030721123456>
BlockID, BlockID
BlockID, BlockID, BlockID
```

HL - Host List Block

A Host List (HL) Block contains a list of Host IDS and their anticipated longevity timestamps. Host Lists are composed and

distributed by each Host as part of their Idle processing. Host IDs are contained in Directory List (DL) and Host List (HL) records received by each Host. These are combined and consolidated into new lists which are periodically pushed to other Hosts. Only the largest timestamp associated with a given Host ID is retained. The Host's own ID is included in any Host List (HL) Block pushed.

In general, contents of a Host List (HL) Block are prepared during the Idle phase of a Host's operation. Host List (HL) Blocks are also prepared as a response to a Query in which Directory Lists are scanned for a particular Block ID. Hosts known to possess the required Block ID(s) are included in the response Host List (HL) Block. The longevity timestamp is a time in the future after which the Host ID will be deemed to have expired. The Host computes its own longevity timestamp from the median duration of the most recent five times the Universe Kernel ran.

The longevity of all other Hosts is the latest timestamp that has been seen for that Host.

The format of a Host List (HL) Block is printable ASCII text as follows:

```
#HLRD
<Host=HostID, expire=030721123456>
<Host=HostID, expire=030721123456>
<Host=HostID, expire=030721123456>
```

QU - Query Block

A Query (QU) Block is a block of printable text in an XML-like format. It contains parameters that are used by a Host to

search other Blocks contained in its Repository. Query Blocks are processed asynchronously on each Host, as time permits. Successfully finding a desired Block results in a response being sent back to the originator of the Query. Responses are simply data Blocks pushed back to the originating Host. Queries which fail may be replicated intact to other Hosts where they will also be processed. After processing (or expiration) Query Blocks are eliminated from each Host's repository.

The format of a Query (QU) Block is printable ASCII text as follows:

```
#QURD
<ReplyTo=HostID, expire=030721123456>
<Discard=030721123456>
<BlockInterval=secs>
<Search=UD/FD/DL>
<Reply=UD/FD/DL/HL>
<ReplyMax=nnn>
<FanOut=nn>
<Query=expression>
<Phrase=expression>
```

The `<ReplyTo=HostID,expire=longevity>` parameter indicates the Host that originated the Query. It is the Host ID to which any successful responses will be sent. The longevity is included to allow Hosts that receive the Query to update their Host List (HL) records so that direct communication with the Query originator will be possible. This parameter is required.

The `<Discard=timestamp>` parameter indicates a time after which the Query will be discarded by all Hosts. No further responses

will be sent after this time. In general, Hosts will also discard Queries with Expiration times too far into the future. This helps prevent (hypothetical) malicious or mal-formed Queries from overwhelming the ReplyTo Host. This parameter is required.

The `<BlockInterval=secs>` parameter indicates the number of seconds that will elapse between Block transmissions resulting from this Query. If the query was successful, it may return many result blocks. This interval specifies how fast these blocks will be sent to the ReplyTo HostID. If the query fails, the Query Block itself may be sent to other Hosts to process. This interval specifies how fast these replica Query Blocks are to be sent. If not specified, the Host determines based on its configuration parameters.

The `<Search=UD/FD/DL/HL>` parameter specifies the nature of the Blocks to be searched by the Query. One or more of the four options will actually be included in the parameter. This parameter is required.

The `<Reply=UD/FD/DL/HL>` parameter specifies the nature of the Blocks to be returned by the Query. One or more of the four options will actually be included in the parameter. This parameter is required.

Not all combinations of Search= and Reply= parameters are valid. The table indicates the results to be expected from each possible pair. The actual implementation may execute several of the ten meaningful operations based on multiple values for Search= or Reply=. This allows, for example, the return of FD and UD Blocks pertaining to a particular file with a single Query.

Search the User Data Block IDs (not contents)

Search	Reply	Results

UD	UD	Return the UD Blocks themselves
UD	FD	
UD	DL	Return a DL Block with only the matching UD Block IDs
UD	HL	Return a HL Block with only the current Host listed.

Search the contents of the File Description Blocks

Search	**Reply**	**Results**
FD	UD	Return the UD Blocks listed in matching FDs
FD	FD	Return any FD Blocks with specific contents.
FD	DL	Return a DL Block with only the matching FD Block IDs
FD	HL	Return a HL Block with only the current Host listed.

Search the contents of the Directory List Blocks

Search	**Reply**	**Results**
DL	UD	
DL	FD	
DL	DL	Return any DL Blocks with specific contents.
DL	HL	Return a HL Block with only the current Host listed.

Search the contents of the Host List Blocks

Search	**Reply**	**Results**
HL	UD	
HL	FD	
HL	DL	
HL	HL	Return any HL Blocks with the specific contents.

Using `Reply=HL` is generally reserved for a preliminary Query which could (potentially) result in a flood of responses. The compact, single Host response minimizes bandwidth requirements and allows the ReplyTo Host to choose the strategy for additional queries.

The <ReplyMax=nnn> parameter indicates the maximum number of Blocks that will be sent by each individual responder to the Host ID specified in the ReplyTo= parameter. This allows a limit to be placed on traffic that will be transmitted as a result of a particular Query. If not specified, one response Block will be allowed.

The <FanOut=nn> parameter specifies the number of Hosts that an unsuccessful Query will be replicated to. In general, Queries that succeed return data to the ReplyTo=HostID. Queries that fail are sent to nn randomly selected other Hosts in an attempt to generate some success. If not specified, queries are not replicated and are simply discarded with no response. A FanOut value of one will cause the Query Block to move randomly from one Host to another until (1) it succeeds and sends a response to the ReplyTo HostID, (2) it expires and is discarded, or (3) it returns to a Host that has already processed it and is discarded.

The <Query=expression> parameter specifies the actual boolean expression used to search the repository. Operands in the expression are given names in <Phrase=ID:expression> parameters. One Query=expression parameter is required in a Query Block. The evaluation of the expression yields a True or False value which ultimately determines the success or failure of the entire Query.

The <Phrase=ID:expression> parameter specifies the named operands and literal comparisons to be used in the repository search. A separate Phrase=ID:expression parameter is required to define each different operand used in the Query=expression.

Search Expressions

Search expressions are composed in a partially pre-parsed within Query Blocks. The syntax of complex searches is thus moved to the user interface application and is not resident in the Data Universe Kernel. This is an attempt to provide functionality similar to regular expressions but without the computational overhead. Examples of Search Expressions follow:

Examples are TBD

Future Extensions

A simple ASCII text method of describing simple directory searches was outlined above. Extensions to this concept would allow any computation to be requested via this standard Query Block mechanism. Input data for the computations is available from User Data (UD) blocks in the Universe. Complex computations could be described in User Data (UD) Blocks containing Java applets, or (with much more danger to Host integrity) actual executable programs.

Computational results would be returned to the requesting Host as new data Blocks just as the results of a directory search are in the reference implementation.

Automated data backup could easily be implemented with a variant of the user interface utility. Files to be backed up would be introduced into the Data Universe and tagged with appropriate File Descriptions. Redundant User Data blocks would be automatically detected and would not add to the traffic

associated with the network. Backups could be a continuous, ongoing part of system operation with minimal overhead. Backups from multiple computers (a corporate LAN, for example) would be highly efficient due to the massive duplication of operating system and application files.

Safeguards

Protection from pathological behavior

Pathological behavior is typified by repeated inappropriate behavior in a portion of the network. Examples include flooding a repository with files or data blocks, cascading queries, etc. The general method of protecting the network from these threats involves throttling and timeouts.

Protection from malicious users or hosts

All hosts are controlled by parameters that set the rate at which traffic will be generated and the amount of time that will be allowed to elapse before a request is discarded. Malicious users or hosts may be able to cause adverse effects, but their scope and duration should be limited. The preponderance of hosts operating in a restricted mode should leave these mis-behaved systems with no more influence than with any other networking technology.

Protection from censorship

Censorship can be viewed as an attempt to erase or obliterate data. The network architecture makes it virtually impossible to erase data once it has been introduced to multiple hosts. The data tends to take on a life of its own and to be replicated unpredictably. Obliterating the data would involve corrupting individual data blocks, or the File Description block. It is computationally infeasible to prepare bogus data blocks with the

same L-Hash signature. The ability to cryptographically sign the File Description and Block List provides protection against censors who might try to substitute corrupt data in place of the original. Censors are thus reduced to simply malicious or pathological users.

Protection from host failures

Host failures are a simple case of a loss of connectivity. The decentralized, redundant operation of the network anticipates such failures as a routine part of its activity.

Protection from communication failures

Communication failures take two forms: loss of connectivity and data corruption. The decentralized architecture and the autonomous redundancy of data storage minimizes the effects of any loss of connectivity to any particular host or group of hosts. Data corruption is prevented by multiple layers of robust verification (using L-Hash) and error correction on files.

Vulnerabilities and Countermeasures

It is understood that any distributed networking technology such as this will be exposed to a number of intentional and accidental threats. Since it is expected that the software will run on home and business computers without a knowledgeable administrative staff every effort must be made to design a system with inherent immunity to malicious or pathological behavior by one or more

nodes in the network. By looking at each of the five block types it is possible to examine the effects of introducing mal-formed or malicious data into the network.

Threat Description, Mitigation and Anticipated Effect

UD-1 Introducing high volumes of gibberish data blocks

Inherently limited by outbound bandwidth of malicious host

Irritating waste of storage.

No major network effect.

HL-1 Introducing high volumes of Host List blocks with invalid Host Ids

Automatic expiration based on Host Longevity

Diversion of Query and Propagation bandwidth.

Minor degradation.

FD-1 File Description does not match file contents

Users can add descriptions based on their experience with the content

Users may need to craft better Query Blocks to eliminate these SPAM results.

Irritating waste of resources

No major network effect.

FD-2 Introducing high volumes of File Description blocks with bad Block Lists

Users can add descriptions based on their experience with the content

Users may need to craft better Query Blocks to eliminate these SPAM results.

Only affects files returned by popular Queries.

Irritating waste of resources

No major network effect.

FD-3 Introducing high volumes of File Description blocks which rebuild gibberish or malicious (viral) files.

Users can add descriptions based on their experience with the content

Users may need to craft better Query Blocks to eliminate these SPAM results.

Only affects files returned by popular Queries.

Irritating waste of resources

No major network effect.

DL-1 Introducing high volumes of bogus Directory List blocks

DL blocks expire based on Host longevity

Queries may return invalid host information for some User Data requests.

No major network effect

QU-1 Introducing high volumes of failing Queries with high FanOut= values

Hosts will limit Discard= timeout.

Queries will run only once on each host

Hosts will limit FanOut=

Query cascade.

High bandwidth utilization.

Low processing impact.

QU-2 Introducing high volumes of succeeding Queries with bogus ReplyTo= hosts

Hosts will limit Discard= timeout.

Hosts will limit ReplyMax=

Hosts will limit BlockInterval=

Queries will run only once on each host

HTTP implementations will stop after first failure to a host; host will be marked as expired.

Distributed-Denial-of-Service to affected host.

High network impact on targeted host

Conceptual Background for Modeling the Data Universe

Brian McMillin

Modeling the performance of the Data Universe requires a certain background in the capacities of the storage and communication resources to be used. This discussion breaks each aspect of the Data Universe down into simple elements that build progressively toward the ultimate goal. With luck and patience, the performance of each level of sophistication can be modeled and some idea of the performance of the overall concept can be achieved.

The Data Universe is a conceptual subset distributed across the current Internet. Communication links exist between Host computers and join the elements within the Data Universe. Data also flows between the Data Universe and the Outside World, by which I mean the rest of the Internet.

Conceptual Steps for Building the Data Universe

1. Define a Host - Storage, Bandwidth and Processing capabilities
2. Each Host has a unique address H, perhaps consisting of IP:Port

3. The repository of a given host H may store at most SH blocks.
4. Each Host can attempt to send data blocks to any other host, but there is no guarantee of success
5. Each Host is interconnected with bandwidth capable of sending and receiving RH blocks per second.
6. Host processors are capable of some level of processing based on the content of their repositories.
7. Holographic Diffusion
8. Normal (gaseous) Diffusion moves particles randomly to "nearby" positions in space
9. Holographic Diffusion, for want of a better term, copies data blocks randomly to connected positions in a physical address space
10. At a rate RH, each host will select a random block from SH and attempt to send it to another host address.
11. Fundamental Communication Features

```
1. Random Blocks sent to random Hosts

2. One-way datagrams

3. Point-to-point: No broadcast or multicast needed

  1. On the other hand...couple it with USENET storage and broadcast
NTP
```

```
4. Latency Independent

5. Connectionless

6. No Confirmation
```

1. A data block DN, received at host H will overwrite a randomly selected block in SH.
2. A particular data block DN will occur at most one time in SH.
3. Assume communication between Hosts is a connected graph, like the Internet.

```
1. Explore the ramifications of Connected Subnets (like computers beh
rewalls)

2. Extend addressability to cover such situations

3. Explore the tradeoffs involved in making the communication two-way
```

1. Tends (very slowly) toward static equilibrium
2. Not very efficient
3. Dynamically Add and Remove Hosts
4. Adding and Removing Hosts corresponds to Computers coming online or going offline or changing IPs
5. Htotal is now a function of time

6. New Hosts per second given by MHtotal / Mt
7. Dynamically Add and Remove Data Blocks
8. Adding block DN to a host is equivalent to receiving the block from another Host or the Outside World
9. Removing block DN from a host is what happens when a block is overwritten by a different received block
10. Removing all copies of block DN from all Hosts is what happens when a block expires

```
1. Expiration simply means the block will be preferentially overwritt
received blocks
```

1. Btotal is now a function of time
2. New Data Blocks per second given by MBtotal / Mt
3. Consider the ramifications of the same block DN being added at multiple hosts (i.e. popular .MP3 files).
4. Directed Replication - Make the Holographic Diffusion process more efficient
5. Add the concept of a Host List Block which can propagate from Host to Host like any other Block
6. Hosts periodically build new Host List Blocks based on knowledge of successful communication and the contents of other Host List blocks that may have been received.
7. Gives the transmissions a vastly better chance of succeeding

8. Requires Hosts to examine the content of the data Blocks
9. Implies some form of time synchronization and expiration of obsolete data
10. Group Related Data Blocks - Create the traditional concept of a File.
11. Allows meaningful data to be exchanged between the Data Universe and the Outside World
12. Add the concept of a File Description Block which can propagate like any other Block
13. Describes the file by name, date, author, contents, etc.
14. Lists the additional data Blocks required to reassemble the original data.
15. Include a signature for the data in the entire file, so that

```
1. Successful reconstruction of the entire file can be verified.

2. Multiple copies of the same file with different names or descripti
n be grouped.

3. Different files (versions?) with the same name or description can
t separate.

4. Annotations referring to a specific file can be added in the form
plemental File Descriptions
```

1. Here we should discuss the ramifications of the choice of Block size

2. Initial suggestion is 65500 bytes, leaving room for the Block ID to be stored with the Block
3. Fits in a single IPv4 packet (probably). Does this really matter?
4. Implies a chaining or "include" requirement for File Description Blocks
5. 8192 byte file system "clusters" are way too small. 20th century design constraints in the 21st century.
6. Queries - Provide a method to select particular data to be extracted from the Data Universe
7. Allow searching File Description Blocks on a host.
8. Cause a host to search for the required data Blocks (by ID) to rebuild a file in the Outside World.
9. May be unsuccessful if the necessary Blocks are not present in the Host's repository.
10. Queries are non-deterministic and must be designed to expire.
11. Query Propagation - Provide a method to get the data Blocks I want onto my specific Host
12. Query Blocks have four states: Pending, Successful, Unsuccessful, and Expired.
13. Successful Queries generate one or more result Blocks directed back to a Host specified by the originator.

14. Unsuccessful Queries propagate intact to one or more Hosts (like ripples in a pond) to be tried again.
15. Queries for “popular” data are more likely to be successful because more copies of the target exist.
16. Query Optimization - Make it efficient enough to be useful
    1. Add the concept of a Directory List Block.
    2. List of Block Ids known to reside on a particular Host (at some point in time, maybe not now…)
    3. Variant has data for multiple Hosts, which could give a choice of Hosts for the same Block.
    4. Add the concept of an Index Block.
    5. List of File Description Block Ids known to contain a particular keyword or feature
    6. Expand it to include keywords in file contents, not just descriptions
    7. Consider the effects of a Fan Out parameter and Queries returning multiple Blocks

```
 1. Fan Out > 1 propagates unsuccessful Queries exponentially un
pire

 2. Successful Queries may return multiple Blocks to the designa
ating” Host

 3. Explore ways to prevent pathological or malicious behavior.

 4. Consider the effects of Universe Spiders
```

```
1. Web crawlers adapted to life in the Data Universe.

2. Automated systems that rebuild data files and generate inc
content.

3. Dredge up rarely accessed data and keep multiple copies in

4. Use error recovery mechanisms to recreate blocks that are
available
```

17. Intelligent Insertion - Data transport from the Outside World
18. Automate useful Data, File Description, and Index Block creation from data in the Outside World.
19. Extract metadata from image files
20. Extract individual files from archives (.ZIP or .TAR) and add both the files and the complete archive.
21. Fault Recovery - Make it robust enough to be useful
22. Implement Error Correction Algorithms.
23. Recover missing Blocks.
24. Correct the (vanishingly small) chance of duplicate Block IDs
25. Security - Answer privacy concerns
26. Encrypt the data as it is added to prevent eavesdropping
27. Add cryptographic signatures to ensure authenticity

28. Manage Cryptographic Keys
29. No Key Revocation

```
1. You can say that documents signed with this key after a certain ti
 not valid

2. If the encryption key is compromised an adversary could still forg
ments

3. If the decryption key is compromised *ALL* documents using it becc
nerable

4. This is not a new or unique problem, just important since the Data
rse is, by design, an archive of all such documents.
```

1. Computational Queries - Massively Parallel Distributed Computing
2. Perhaps Javascript in Query Blocks
3. Collect input data from the Data Universe, process it, and return Results to the Universe.
4. Effects of possible pathological behavior
5. Using the Block “include” feature would allow large code libraries, etc.

```
1. Would never suffer from the “wrong version of the .DLL” problem

2. Ensures the code always runs as the author wrote it. Good or Bad.
```

Parameters Required for Modeling

Data Block contents are denoted by DN, N 0 { 0 .. 2128-1 } where N is the Block ID. This is for the MD5 version. N is divided into two distinct subsets: "defined" and "undefined". The "defined" subset is the set for which DN is known. Total Actual Unique Data Blocks Btotal in the Data Universe is a the number of elements in the "defined" subset of N. Physical Host addresses H 0 { 0 .. 248-1 } (assuming naming using IPv4 address space of the form IP:Port). H is divided into two distinct subsets: "implemented" and "not implemented" depending on whether a host with host address H exists and can communicate with the network.

Total number of real Hosts Htotal is a the number of elements in the implemented subset of H.

Storage per Host SH = Number of Blocks that can be stored on Host H. Hosts that are "not implemented" have zero space.

Total Physical Storage in the Data Universe is Stotal = 3 SH.

Diffusion Rate RH is the number of transmissions per second attempted by Host H. Hosts may have different rates based on available bandwidth.

Suggested Models

Start with an even distribution of Btotal / Htotal Blocks stored on each Host.

Compute the storage utilization on each Host after blocks diffuse for time T.

Prove that the equilibrium number of Block copies is Stotal / Btotal, in the limit as time T Y 4 What if the initial condition includes a distribution with more than Btotal / Htotal Blocks stored on each Host. In this case, "popular" data Blocks will already exist on multiple hosts. Note the restriction that a particular Block DN can never be duplicated on the same host.

With each different level of sophistication, design models that allow answers to the following questions to be derived.

1. How many copies of a specific data block DN will exist in the Data Universe?
2. How many Hosts will be involved in a Query for a particular data block DN? Typical and worst case.
3. What is the optimum timeout for Query Blocks?
4. From any Host in the Universe, how long should it take to rebuild a file of a particular size?
5. Is there an optimum bandwidth for a given repository size? Is there any correlation at all?
6. What are the effects of changing the rate at which new data is introduced from the Outside World?
7. What are the effects of changing the rate at which physical storage is added to the Universe?
8. How much network traffic will be devoted to Query processing and how much to permanent data "diffusion"?
9. How much storage will be devoted to permanent data, and how much to blocks that will expire?

10. How much storage will be devoted to File Descriptions and how much to content Blocks?

Consider the effects of a technological adversary capable of fabricating gibberish data Blocks with duplicate hash Ids. This is considered computationally infeasible now, but Quantum Computing or other "outside the box" developments might make understanding the ramifications important. Is the minimum 128-bit Block ID appropriate?