Michigan State University
Computer Science and Engineering
CSE 498: Collaborative Design

Team MATRIX
Event Logging System for the Kora Archive

Chung-Hi Kim
Dustin Manning
Chris Samiadji-Benthin
Jared Wein

Table of Contents

0. Team Introduction

Chung-Hi Kim <kimchun2@msu.edu>, Webmaster / Developer

Dustin Manning <mannin65@msu.edu>, Developer

Chris Samiadji-Benthin <samiadji@msu.edu>, Client Contact / Manager

Jared Wein <weinjare@msu.edu>, Technical Writer / Developer

1. Software License

This software is under the MIT License:

2. Statement of Problem

MATRIX is Michigan State University's Center for Humane Arts, Letters and Social Sciences Online. MATRIX, in its mission statement: “seeks to advance critical understanding and promote access to knowledge through world-class research in humanities technology.” In keeping with this mission, MATRIX is currently developing a digital archiving system with content management capabilities called KORA. This system has some specific requirements it must satisfy in order to be labeled as a digital archive. One of these requirements is an event recording system that also performs data integrity or 'fixity' tests. While this seems like a trivial problem to solve at first, the system should scale to thousands of files and hundreds or thousands of gigabytes of information. The large amount of information that may be entered into this system poses interesting challenges to the system designers.

An accepted solution to aid in ensuring reliability of data is to store meta-data, or data about the data. One form of meta-data is a digital signature such as a 'checksum' of the data. Checksums are small values which can be computed using binary operations to test the integrity of a piece of data such as a file. A desirable property, for a checksum algorithm, is an 'avalanche' property- when a single bit in the data is changed the resulting checksum will be completely different. While checksums can be computed using several different algorithms- MATRIX is inclined to use the Secure Hash Algorithm (SHA) standard developed by the National Security Agency. Since the checksum is generally much smaller than the data itself, it's quite useful to use for comparison because it will be much faster to compare checksums rather than entire files.

The Event Logging system operates as follows - the system takes in data from a database, checksums it, and stores the checksum and some identifying information, such as a filename, in a database. The point of the system is to ensure the data does not degrade over time. To accomplish this the Event Logging system will intermittently checksum each data element and create a log file to note any additions, removals, or errors in the archive.

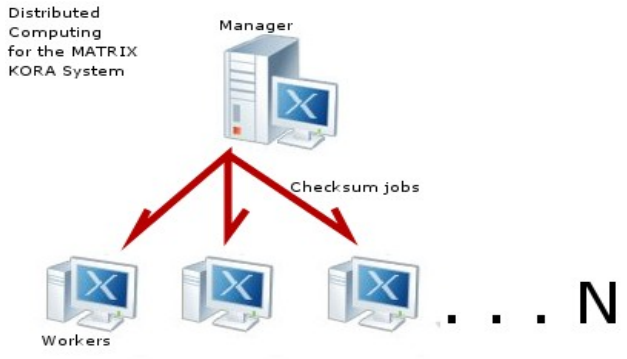
(Author's note: the reader is encouraged to reference the dictionary in section 15 for clarification of technical terms presented in the following)

3. Proposed Solution

3.1 Approach

Since each file can be tested completely independently of any other file, our problem is considered to be “embarrassingly parallel” or very easy to implement as a parallel computation. It's useful to visualize the problem of testing archive integrity, by computing independent checksum on files, as the dividing of labor across multiple agents or 'Workers'. The task of dividing up the labor is performed by a 'Manager' who passes out work to each 'Worker'. In our case this division of labor is performed by passing filenames to each 'Worker' which then retrieve the files over a network drive, compute the checksum, and then return the checksums to the 'Manager'. We decided to utilize the Python programming language to implement this solution as Parallel Python is a tremendously useful library for helping us in abstracting out the client server interaction between the Manager and Worker machines. Parallel Python requires us only to define the checksum function to pass to the Worker machines and provide the data needed to perform the checksum computation.

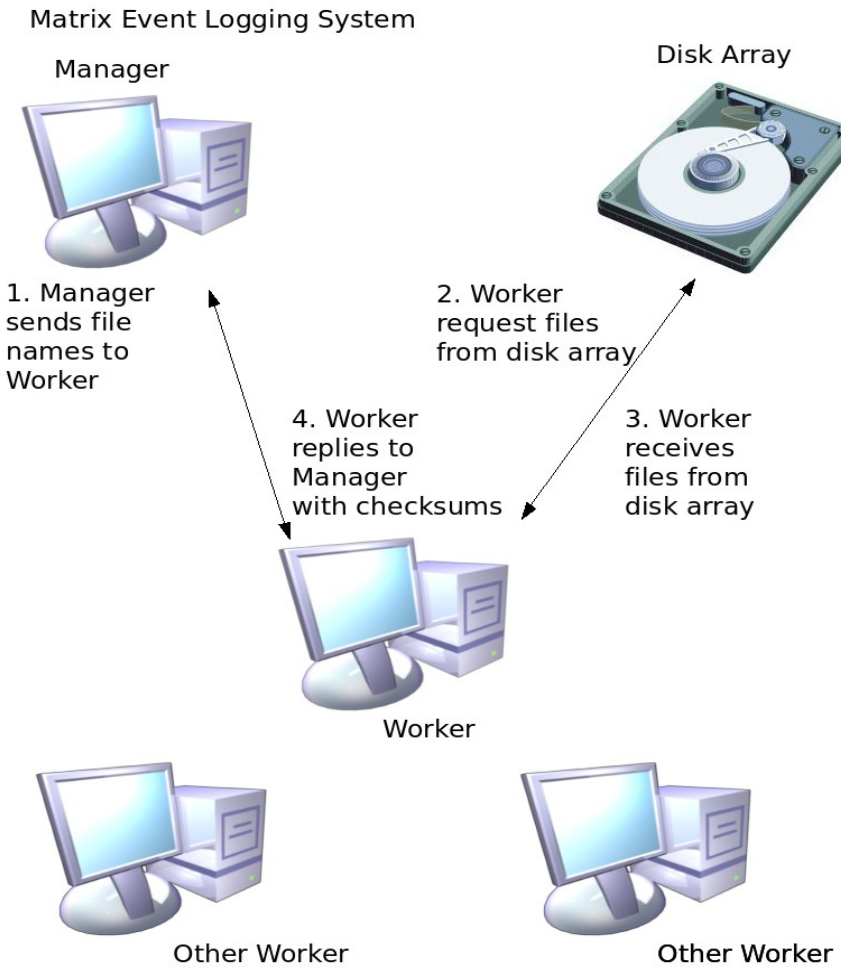
The following diagram idealizes the division of labor across N machines.



To obtain the filename and filepath information, our system must interact with a filepath server which is part of Matrix's Kora Archive. This component will, ideally, be located on the same machine as the Manager and the file information will be accessible via a simple MySQL query from the Manager. The data itself, will be stored in a Disk Array: MATRIX uses a RAID 5 configuration for the Kora Archive which spreads the data across multiple disks. The data will be accessible to the Workers via a network drive, such as SAMBA or NFS.

Our system is designed to handle an arbitrary number of Workers. If there are no remote Worker machines available, the Manager machine can act as Worker. To execute a fixity test on the archive, the administrator must start a server instance on each potential Worker machine and then run the Manager program. Once the Manager has the filepaths of the Kora archive, it can begin distributing filepaths to the Worker machines, collecting the results upon completion.

This arrangement can be idealized by the following diagram.



3.2 Required Features

The system will have these accepted features:

- Be platform-independent
- Support MySQL
- Support remote procedure calls
- API for making remote procedure calls
- Allow different checksum algorithms to be utilized
- Human-readable configuration file in XML
- Command Line User Interface
- Generate text files for log events

4. Use Cases

There are essentially 4 use cases supported by the Event Logging System:

1. Run a single fixity test and generate any 'error' or 'failed to open file' events.
2. Run multiple fixity tests according to a schedule in the configuration file, generating any 'error' or 'failed to open file' events.
3. Add a file: checksum and new file and record it to the database and generate an 'added' or 'failed to open file' log event.
4. Remove a file: remove the file's record from the archive and generate a 'removed' log event.

All these use cases can be performed either via the command line interface or via remote procedure calls from a separate client.

5. User Interface

The user interface features the following simple options:

1. Run a Fixity Test
2. Run the Scheduled Test
3. Add a File
4. Remove a File
5. Listen for XML-RPC Commands

6. Classes Used in Implementation

6.1 Manager

The Manager class is the main component of the project. This object will run fixity checks on administrator command (or on a preset schedule). When the Manager is called to execute a fixity test, the Manager will test all the Worker nodes it detects for availability ignoring those which aren't functional. The Manager will then retrieve file paths from the Kora server for the test. Workers that become unavailable as the fixity test is occurring will no longer receive filenames to checksum. When the fixity test is completed, the Manager will dump all of the events it has recorded during the test into log files. The following log messages are generated:

1. **FAILED TO OPEN**: meaning that a file couldn't be opened during the fixity test.
2. **ERROR**: meaning the checksums don't match between the stored checksum value and the current checksum computed from the file in its present state.
3. **ADDED**: a new file was added to the archive.
4. REMOVED: a file was removed from the archive.

The Worker is actually a function in the Manager class called computeChecksums(filename, workerToFilepath, algorithm). This function basically takes in the filename, a dictionary relating the Worker's IP address to the filepath to the network drive with which it will retrieve files and the type of SHA checksum algorithm to use.

The Manager also provides methods for updating its own local database of checksums in case files have been removed or added by an administrator.

Also included is a scheduling component which will allow for jobs to be performed at administrator configured intervals- for example: every 40 days at 1900.

Email functionality is also implemented on the Manager to provide test completion notification but this functionality depends on the existence of an available SMTP mail server for use.

6.2 ManagerServer

The ManagerServer is used for direct remote procedure calls, it utilizes an XML-RPC server

to listen to client commands over HTTP for adding files, removing files and performing fixity tests.

6.3 LogGenerator

The LogGenerator is used to generate the Event log for the system. The class is relatively trivial as the actual Events are generated in the Manager- essentially its sole purpose it to, on Manager command, take a list of events and save them to text files.

6.4 Parser

This class is responsible for parsing the XML configuration file. Also if the configuration file has been erased, it can generate a new template for the file- the administrator, however, is responsible for correctly configuring the system.

7. Configuration File

Configuration of the Kora Event Logging system is done using an XML file, called 'config.xml', in the following format:

```
<?xml version="1.0" ?>
<config>
	<setup algorithm="sha512" email_host='mail.msu.edu' log_size="5"/>
	<schedule run_job_every_x_days="40" job_start_time="1900"
	check_db_every_x_minutes="5"/>
	<worker IP="35.9.22.153" path="Z:\" port="60000" />
	<worker IP="35.9.22.158" path="/home/administrator/Desktop/matrix/"
	port="70000" />
	<worker IP="35.9.22.111" path="/home/administrator/Desktop/matrix/"
	port="70000" />
	<worker IP="35.9.22.112" path="/home/administrator/Desktop/matrix/"
	port="70000" />
	<worker IP="35.9.22.104" path="/home/administrator/Desktop/matrix/"
	port="70000" />
	<database URL="35.9.22.104" dbname="matrix_pathdb" password="kora5"
	table="paths" username="matrix_dbuser" filename="path" size="filesize"/>
	<notify email="samiadji@msu.edu" />
</config>
```

There are only 5 XML elements in the configuration file:

1. **setup**: this element has attributes for selecting the algorithm, the size of the log files (in kilobytes), and the email host which will allow the Logging System to email administrator when a fixity test is complete.
2. **schedule**: has attributes for selecting the frequency, in days, of fixity test runs, the start time on the day of a fixity test
3. **worker**: has attributes for the IP address of the worker machine, which port the Worker is listening on, and the local path for the network drive
4. **database**: has attributes for the URL of the database, the database name, the user name and password for accessing it, the table name, as well the name of the columns which specify the filename and the file size.
5. **notify**: has an email attribute to specify whom to email upon completion of a fixity test.

8. XML-RPC Background

To provide communication facilities between the Manager and the administrator, via the ManagerServer, we provided a programming-language independent interface to allow remote procedures to be requested for the Manager to execute. To accomplish this, we used XML-RPC.

1. XML-RPC is also platform independent so that even if the server runs on Linux and the client runs on Windows, the two processes can communicate with each other.
2. Also XML-RPC is language independent, this is of note as Kora as implemented in a different programming language than our Event Logging System.
3. XML-RPC uses HTTP as a transport layer and XML as the encoding. We will implement the ManagerServer using the Python SimpleXMLRPCServer library, which abstracts the XML formatting out to remove another layer of complexity.

A sample method call rendered in XML as would be used in XML-RPC follows:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
        <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

A sample method response rendered in XML as would be used in XML-RPC follows:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

9. Database Schema

Our solution depends on the use of two separate databases. The filepath database is a database that exists on the Kora Server and is expected to provide at least the filenames for all the data in the Kora Archive. 'managerdb' is a database that is part of the Manager- it will store the checksum values for each file in the Kora Archive.

9.1 Anticipated Schema for filepath database

The filepath database is not under our control, and thus we can only provide an expected database schema for our solution. However, since our interface to the Kora Archive filepath database is configurable, the particular name of the attributes is unimportant as an administrator can configure them appropriately.

- **Primary Key**: can be anything
- **filesize**: the size of the file
- **filename:** the name of the file

9.2 Database Schema for Manager database

The Manager contains its own database that will store a table with the stored checksums and the filename that goes with them:

- **Primary Key**: <filename>_<algorithm>
- **Algorithm**: the type of algorithm used
- **Filename:** the name of the file
- **Checksum**: the computed checksum
- **Timestamp**: the month, day, year, hour, minute the checksum was first computed

10. Parallel Computing Background

10.1 Amdahl's Law

The key law, with regards to parallel computing, is Amdahl's Law which states that the increase in speedup in using a parallel computing system is directly proportional to the amount of computation which can be performed in parallel on the original system and number of CPUs available for the parallel computation. Specifically:

```
max speedup = 1/((1-P)+P/N)
```

```
where P is the percentage of the computation which can be
  performed in parallel and N is the number of Workers
```

10.2 Some Examples

If there is no speedup using parallel computation then P=0, then:

```
max speedup = 1/((1-0)+0/N) = 1
```

If there is maximum speedup using parallel computation then P=1, then:

```
max speedup = 1/((1-1)+1/N) = 1/(1/N) = N
```

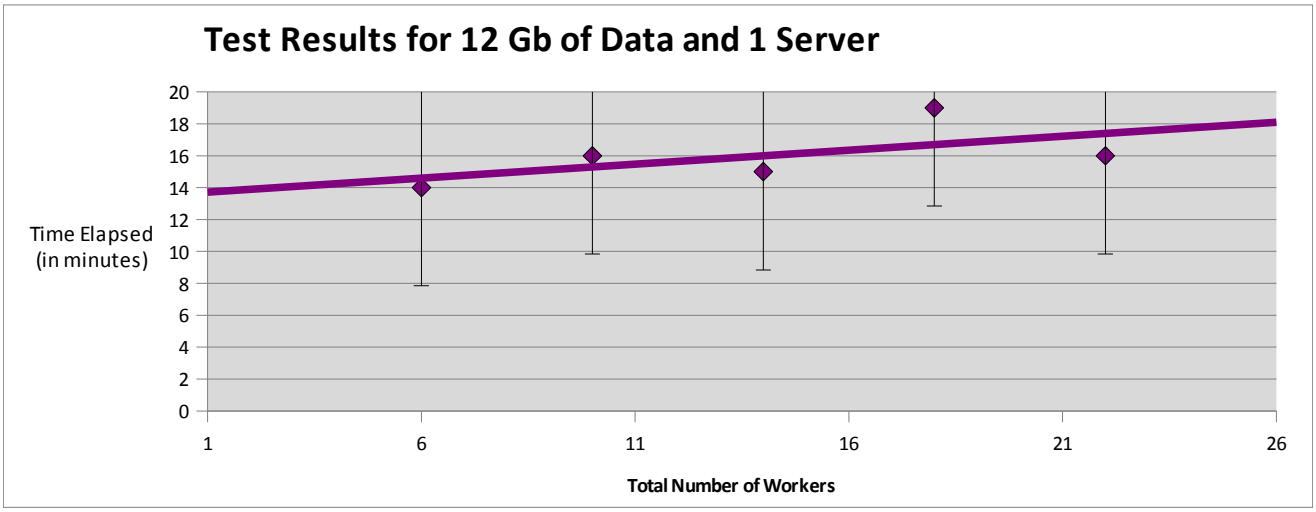
10.3 Other Considerations

A corollary to Amdahl's Law is Amdahl's Rule of Thumb which states that 1 byte of memory and 1 byte/second of input/output (I/O) are required for each instruction/second supported by the computer. In practice this means, that there must be enough bandwidth to support computation and that while the CPU itself may be fast enough, disk I/O or network bandwidth may become a bottleneck and reduce performance. Since our system is design to pull files over network drives high network bandwidth and good data availability is a must.

11. Initial Performance Test Results

11.1 Test 1

For our first test we used 12 Gigabytes of randomly generated data in files sized between 12 Kilobytes to 60 Megabytes. This data was located on a single server. We went to the Business College and used one of their labs to set up the Workers. Our typical network bandwidth was about 15 Megabytes/sec from the server. We used between 3 to 11 machines with two Workers processes per machine or between 6 to 22 workers.

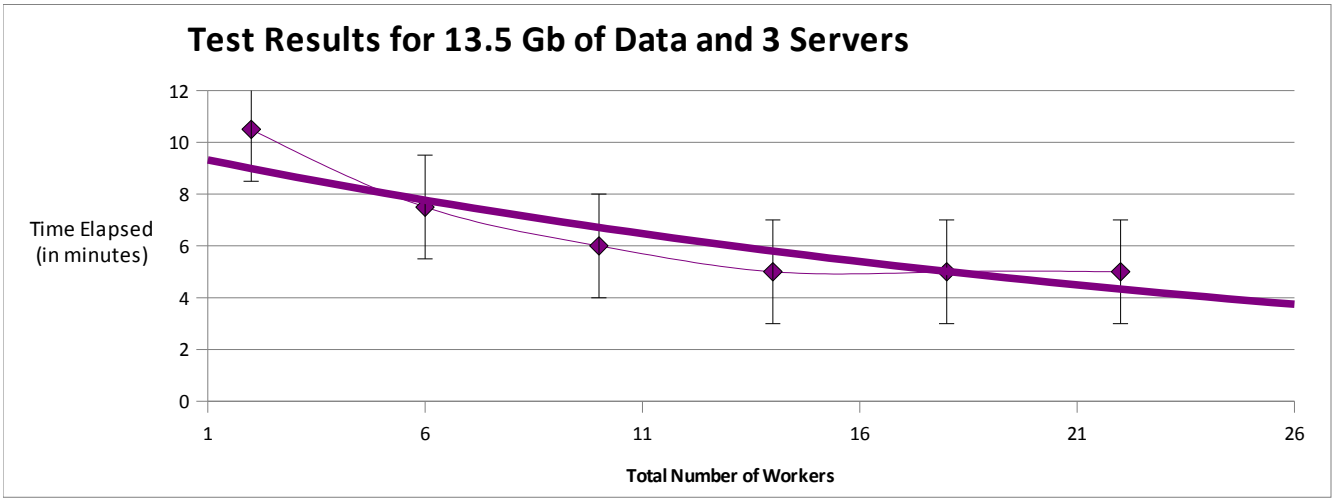


The first test gave us a poor result: the amount of time it took to checksum the data increased slightly when the number of Workers was increased. We attributed this to having Workers all attempting to open files at the same time on the same server and saturating the bandwidth of that server such that the handling of too many multiple request decreased effective bandwidth and thus decreased performance.

11.2 Test 2

For our second test: we used slightly more data and split the data across 3 servers instead

of a single server thus tripling our effective network bandwidth. Again we used the Business College as our test lab with typical bandwidth 15 Megabytes/sec from the servers. We used between 1 to 11 machines with the same 2 Worker processes per machine, i.e. 2-22 Workers.



Our results show that the run time now decreases as the number of Workers increases but only up to a point. There is a Law of Diminishing Returns here- considering Amdahl's Law, if the non-parallel component of the program is significant then after a certain point there will be no substantial increases in performance. Our maximum speed up here is a factor of 2.

To determine roughly how much of our system is parallelized on the second test we work backwards using Amdahl's Law:

Max speed up is:

$2 = 1/((1-P)+P/N)$ with $N=14$ Workers minimum

$2 = 1/((1-P)+P/14)$

$2[(1-P)+P/14] = 1$

$2-2P+2P/14 = 1$

$-1.86P = -1$

$P = 1/1.86 = \sim.54$ thus approximately 54% of our program is parallelized given this particular setup.

12. Hardware Requirements

The Event Logging system requires that the data to be tested must be available either in situ or through network drives for all machines used in the system.

- Ethernet connections
- Server Computer
- Client Computers

13. Software Requirements

13.1 Python and extra libraries

The software requirements for our system are minimal. As a team, we have decided upon developing our solution with a platform-independent language. Python was chosen as a platform-independent language that will scale to our needs, and gives us XML-RPC and hashing as built-in libraries. Due to our use of Python, we require that a Python interpreter be installed on the system. The Python software need to run this Event Logging System includes:

- Python 2.5.1 (www.python.org) , the Python interpreter
- Parallel Python 1.5.3 (www.parallelpython.com) , distributed computing support
- python-adodb (adodb.sourceforge.net) , generic database support
- python-mysqldb (mysql-python.sourceforge.net) , MySQL specific database support

13.2 Database requirements

Our system is designed to interact with a MySQL database which stores the filenames of the data in the archive. Also our Event Logging System maintains an SQLite database in a file- Python provides support for this natively.

13.3 Operating System Requirements

There is no specific operating system requirement, with the exception that there must be a Python-interpreter implemented along with the above software installed for the specific operating system. Python interpreters have been implemented for the following major operating systems:

- Microsoft Windows
- Apple Macintosh
- Linux

14. Project Schedule

Week	Task
Jan 21	-Find right distributed framework, prototyping, presentation slides (Dustin) -XML-RPC documentation (Kim) -Communication With Matt G., coordination of personnel, prototyping (Chris) -Documentation (Jared)
Jan 28	DEADLINE: Presentation with slides and tech-spec (all hands) -Continue prototyping (Dustin) -Decide on distributed framework (Chris, Dustin) -Client meeting (Chris)
Feb 4	-Begin coding alpha: distributed portion (Chris, Dustin) -Begin coding alpha: database interface, schema, file path issues (Kim, Jared) -Client meeting (Chris)
Feb 11	-Continue coding respective portions of alpha (all hands) -Client meeting (Chris)
Feb 18	DEADLINE: alpha ready for demo. Should have distributed portion working in command line and database interface to test data set. -Continue coding for beta, debugging (all hands) -Collect benchmarking data for scalability (all hands) -Client meeting (Chris)
Feb 25	SOFT DEADLINE: beta finished. -Finish up last few beta issues (Chris, Justin) -Begin API documentation for client (Chris, Justin) -Collect benchmarking data for scalability (all hands) -Client meeting (Chris)
Mar 3	-Spring break
Mar 10	DEADLINE: Progress Report & Demo (Chris) -Continue working on API documentation (Chris, Dustin) -Client Meeting (Chris)
Mar 17	DEADLINE: Beta finished & Presentation (all hands) SOFT DEADLINES: API documentation finished -Begin working on project video (All hands) -Client Meeting (Chris)
Mar 24	-Debugging (all hands) -Continue Working on Video (all hands) -Client Meeting (Chris)
Apr 7	-Collect benchmarking data for scalability (all hands) -Continue Working on Video (all hands) -Client Meeting (Chris)
Apr 14	DEADLINE: Final Demo and Presentation (all hands) -Debugging (all hands) -Client Meeting (Chris)
Apr 21	DEADLINE: Demo Video (all hands) DEADLINE: Debugging Finished (all hands) Design Day Demo (all hands)

15. Dictionary

ADOdb – a layer of abstraction for programming languages, such as Python, to allow the use of different database architectures.

Checksum – a bit string generated by performing certain binary operations on files, this is used to test for correctness of data. The checksum is usually much smaller than the original data.

Distributed Computing – using multiple computers to perform a computation.

Fixity Check – comparing two checksums to see if they are different. A fixity check fails if they are different and can show if the data has degraded.

HTTP – HyperText Transport Protocol

MATRIX – Michigan State University's award winning, world class, humanities and technology research center. The mission of MATRIX is to serve as a catalyst for and incubator of the emerging fields and disciplines resulting from the integration of the humanities with information technologies.[1]

MySQL – An open source database which is wide use.[2]

NFS – Network File Share: this is a protocol to allow for sharing of files across a network using a network drive.

OPEN SSL – An open source implementation of the Secure Sockets Library (SSL) and its predecessor the Transport Security Layer (TSL). These protocols were developed to ensure privacy in network communications by using encryption an such.[3]

Platform Independence – The ability to run the software on any major operating system platform, such as Linux, Microsoft Windows, or Mac OS.

Python – an interpreted language used in programming for web pages, mathematics, distributed computing, networking, and other application areas. [4]

RAID – Redundant Array of Inexpensive or Independent Disks, is technology for storing data across hard disks to ensure reliability. There are several versions with different features such as parity checking, mirror copies of files, or having a backup disk.

SAMBA – Shared Memory Block, this is a standard protocol for sharing files across a network through a network drive.

SHA – Secure Hashing algorithm. A set of algorithms used to generate checksums on data. The algorithms are available in 160, 224, 256, 384, and 512 bit flavors.

SMTP – Simple Mail Transfer Protocol, a protocol for sending email over the Internet.

SQL – Structured Query Language, a standard database language used for creating, using, and searching through databases.

SQLite – A lightweight version of the SQL database, designed for simplicity.[5]

URL – Uniform Resource Locater

XML – eXtensible Markup Language, a human and machine readable text format.

XML-RPC - a specification and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. It can be used for remote procedure calling using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.[6]

[1] www.matrix.msu.edu

[2] www.mysql.com

[3] www.openssl.org

[4] www.python.org

[5] www.sqlite.org

[6] www.xmlrpc.com

16. Additional References

- http://docs.python.org/lib/module-SimpleXMLRPCServer.html
- http://en.wikipedia.org/wiki/Amdahl's_law
- http://en.wikipedia.org/wiki/XML-RPC
- http://matrix.msu.edu/
- http://parallelpython.com/
- http://python.org/
- http://theserverside.com/tt/articles/content/DistCompute/figure1.jpg
- http://xmlrpc.com/