RSSE: A New Method of Distributing Datasets and Machine Learning Software

Michael Gohde
Vision and Security Technology Lab
University of Colorado at Colorado Springs
Colorado Springs, Colorado
mgohde@uccs.edu

***Abstract*—Among the challenges faced by machine learning researchers today is that of distributing the datasets and algorithms used in their research. This problem arises mostly from the limitations involved in hosting datasets on servers outside of their origin. RSSE (Really Simple Syndication for Experiments) is intended to provide a message-based system with which researchers can share their data and algorithms.**

I. Introduction

RSSE draws on existing standards, namely XML and RSS, to facilitate easier communication and distribution of data among researchers. While RSSE is not an extension to the existing RSS standard[6], it is intended to be similar in general conventions and syntax to RSS. As such, it extends the concept of RSS, which is that of message-based syndication involving a client "reader" application and a message server established by an institution. RSSE is designed so that messages can be written either by researchers themselves or by automated tools.

Due to current copyright and IP law, it is often difficult or impossible for institutions and researchers to directly distribute external datasets used during computation[4]. Some large dataset providers, such as Yahoo, explicitly prohibit the redistribution of the dataset itself, instead allowing the dataset to be distributed in the form of links[5]. Such datasets usually allow users to cache the data locally. By providing a consistent system by which data and software can be distributed using messages containing URLs and checksums, RSSE should enable institutions to easily monitor and expand on the work supplied by other institutions. Furthermore, by passing URLs to datasets rather than the datasets themselves, experiments can be run by other institutions while still respecting the Intellectual Property rights of the dataset's source.

RSSE will work in a similar fashion to RSS (Really Simple Syndication), with some exceptions. As such, a researcher or automated utility would generate an XML file using RSSE tags and serve it over the HyperText Transfer Protocol (HTTP). Such an XML file would contain the project's title, a brief description of the project itself or changes made recently, several URLs, and checksums for the relevant URLs. Each URL should usually refer to a datasets involved during the course of research. One possibility, however, is that some of the URLs could refer to source code or compiled Java classes, which would, in turn, be executed locally to verify the results of the computation. When an XML file containing RSSE data is posted, client programs could proceed to download the file, parse its contents, then perform a predetermined set of actions, such as downloading all of the datasets and source code involved in the remote experiment. For a graphical representation of the data transfers involved, see figure 1. (figure 1)

II. Previous Work

RSSE draws primarily off of the existing RSS standard[6]. Due to its flexible nature and widespread use, RSS has already utilized as a basis for distributing information to research librarians in an organized fashion[2]. A similar system featuring a dedicated message and client-based infastructure was implemented to distribute climactic data, however it did not explicitly use RSS, rather it directly exposed a database to a network[3].

III. Implementation

A. Implementation History

For this project, a reference implementation of the RSSE reader and file generation utility were written. As RSSE will be an open standard, the implementation discussed here exists solely as a reference for other future implementers to follow.

The first stage of the implementation was considering what tags would be acceptable for each RSSE file. These tags are listed in Table 1. The tags were determined after considering the minimum set of data necessary to represent a message. The most important tags involved are the dataset tag, the checksum tag, and the checksumtype tag. The dataset and checksum tags are self explanatory. The checksumtype tag will contain a string value representing the hashing algorithm used to generate the checksum on the server's side.

The second stage was the implementation of the RSSE reader program. This application was implemented before the RSSE message generation program because of the relative ease in manually writing RSSE files as opposed to manually reading RSSE files. Upon starting, a command line specification was drawn up based on all of the tags and operations expected of the program. The command line interface was implemented first, due to the ease in doing so. Command line options are not included here as it should only be necessary for future implementations to utilize the base set of RSSE tags as opposed to implementing full compatibility. This was followed by the implementation of a GUI, which extended the features of the command line interface.

Once the graphical component of the RSSE Reader was complete, work started on the RSSE Generator. Because all of the features of RSSE were fairly stable by this point, the RSSE generator was far easier to implement. Unfortunately, due to time constraints, the generator does not yet have scripting support, however that has become a priority in the near future.

While each program is currently stable enough for general use, there are some UI elements that do need improvement due to their obtuse or erratic behavior. The most obvious of which is the difference in graphical styles between the reader and file generation utility.

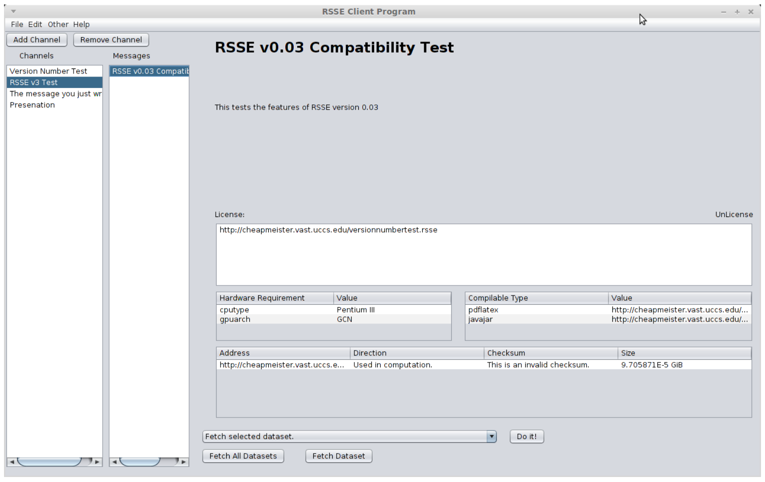
Upon completing the first few revisions of the RSSE reference implementation, the project was demonstrated to a group of machine learning researchers. Based on their suggestions, the RSSE version 0.03 specification was drawn up with several enhancements in the form of four new tags and three new sets of attributes. These new tags should enable both researchers and end users to benefit from increased tailoring to various conditions and systems. The new features are mentioned in separate tables. Upon starting work on this version of the specification, it became very clear that each version of RSSE would in the future cause rendering and generation problems on prior and future versions of the RSSE reference implementation. Because of this, the <rsse> tag now carries an attribute specifying the expected minimum version code necessary to render a given RSSE message. The version encoding scheme is elaborated in Table IV.

B. Reasons For Using The Technologies Used

While doing the project, it became clear that it may be necessary at some point to justify the use of Java and XML as the primary language and data framework of the application, respectively.

Firstly, Java was selected as the primary implementation language due to its feature-set. Java has extensive support for reading and parsing XML files, which proved invaluable for the project as a whole. Furthermore, Java provides easy to use networking and graphical user interface APIs, which contributed to the quick implementation of the project. Finally, the project's code should be easily readable to a wide array of programmers due to the similarities in syntax and usage between Java and other programming languages.

XML was selected mostly because it is very easy for humans and computers alike to read and parse. By using plain-text instead of binary for messages, it allowed the developer to write test messages to pass to the reader before the file generation utility was complete. Furthermore, while it was not a clear focus in the beginning of the project, XML allows for a significant level of complexity and flexibility, which allows the RSSE standard to expand easily in the future. In the future, it is likely that future implementers will write their own messages in order to test their RSSE reader applications.



Version Number Test
RSSE v3 Test
The message you just wr
Presenation

RSSE v0.03 Compatib

RSSE v0.03 Compatibility Test

This tests the features of RSSE version 0.03

License: UnLicense

http://cheapmeister.vast.uccs.edu/versionnumbertest.rsse

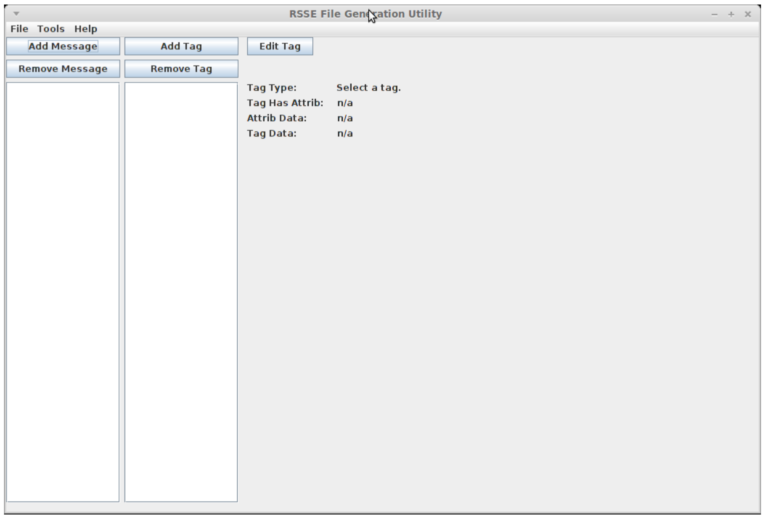
Hardware Requirement	Value
cputype	Pentium III
gpuarch	GCN

Compilable Type	Value
pdflatex	http://cheapmeister.vast.uccs.edu/...
javajar	http://cheapmeister.vast.uccs.edu/...

Address	Direction	Checksum	Size
http://cheapmeister.vast.uccs.e...	Used in computation.	This is an invalid checksum.	9.705871E-5 GiB

Fetch selected dataset.

Fig. 1. The RSSE Reader Program



Tag Type: Select a tag.
Tag Has Attrib: n/a
Attrib Data: n/a
Tag Data: n/a

Fig. 2. The RSSE File Generation Program

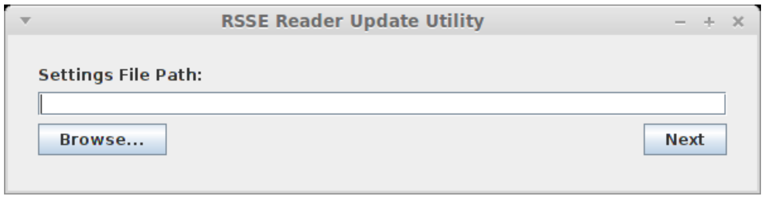


Fig. 3. An early build of the RSSE Updater

C. Interface and Design

While the graphical user interface is in a very early development stage, it is complete enough to be shown here. Please refer to Fig. 1 for an example configuration of the Reader, and Figure 2 for an example configuration of the Generator. In both programs, there are clearly defined lists of values to be modified and modifier buttons either on top of the lists or to their side. This was done to associate the various modifiers with the data involved, which should hopefully lend itself to usability. The menu layout is very sparse, as most of the functions encountered in the program are represented by the buttons present. There are very few dynamic UI elements in order to promote portability to low-power devices and older operating systems. By providing unambiguous functions, the RSSE reference applications should be very easy to learn. Overall, this design aesthetic should prove helpful for the purposes of providing a reference implementation from which other implementations of the RSSE standard can be derived.

Tag	Tag Value
<rsse>	Tag used to denote an RSSE file.
<message>	Tag used to mark the start of a message. This allows for there to be more than one message in each file.
<title>	The title of the project.
<description>	The project's description.
<link>	A website to be visited by the user. Could be used to direct clients to more information.
<dataset direction="in">	Represents a dataset. The direction attribute is used to inform the user as to whether the dataset was used in computation (value "in") or generated as the result of computation (value "out").
<checksumtype>	Represents the type of checksum to generate and check.
<checksum>	Represents an individual checksum. Will be associated in the order of appearance of datasets.

TABLE I
TAGS IMPLEMENTED BY THE RSSE REFERENCE SOFTWARE

Tag	Tag Value
<update url="url">	Tag that could be used to send updates to the reader. Url attribute is used to mark the URL of the update. The tag will contain the
<license>	Tag that could be used to distribute executable code if implemented.
<minspec>	Tag representing the minimum specifications to run software bundled with a message.
<compilable type="type">	Tag that could be used to distribute code with special compilation requirements.

TABLE II
TAGS ADDED IN RSSE VERSION 0.03

IV. CHALLENGES

While writing the checksumming portion of the program, it became very clear that Java's default IO functions were too slow for the task. While it has not yet been implemented, the project will eventually add support for Java's NIO (Non-blocking IO)[1], which should provide a high performance framework for checksumming operations. Another challenge encountered was that of determining how checksums should be transmitted, as it is difficult to parse multiple attributes in each tag. This problem was solved by associating each checksum tag with dataset tags in the order that they appeared, however this is more of a short-term solution that may require the implementation of a complete XML parser within the code. One of the clearest challenges is deciding on which tags should be added to each version of the RSSE specification, as it involves several decisions as to which features would be easiest to implement, as well as which features would be best for end-users. Overall, accepting suggestions from others proved to be very beneficial to the project as a whole.

Attribute	Tag	Description
pdflatex	<compilable>	Allows for the inclusion of LaTeX documents.
makefile	<compilable>	Allows for the distribution of projects utilizing makefiles.
javajar	<compilable>	Allows for the distribution of Java Jar files.
executable	<compilable>	Allows for the direct distribution of executable files. The reference implementation will never allow these files to execute without a prompt.
gpuarch	<minspec>	Allows for researchers to specify different GPU architectures, such as Kepler or GCN.
cpuarch	<minspec>	Allows for researcers to specify a CPU architecture for specific optimizations.
cputype	<minspec>	Allows for researchers to specify a specific type of CPU to be used. Only for heavy optimizations.
corecount	<minspec>	Allows for researchers to specify a minimum number of cores to comfortably run multithreaded software.
minram	<minspec>	Allows for researchers to specify a minimum amount of RAM as a floating point number of gigabytes.
minstorage	<minspec>	Allows for researchers to specify a minimum amount of free hard drive space as a floating point number of gigabytes.
osfamily	<minspec>	Allows for researchers to specity the intended operating system family for their software. This could be used to prevent non-POSIX operating systems from attempting to compile the software included.

TABLE III
ADDITIONAL ATTRIBUTES IMPLEMENTED IN VERSION 0.03

RSSE version	RSSE Tag
v0.01 (Depreciated)	<rsse>
v0.02	<rsse>
v0.03	<rsse version="3">
(Future releases)	<rsse version="(Version number*100)">

TABLE IV
THE RSSE VERSION ENCODING SCHEME.

V. APPLICATIONS

RSSE has the potential to become a very valuable tool for researchers, especially those who wish to use commercial or otherwise difficult to distribute datasets. While RSSE is intended to be used primarily by a machine learning and computer vision audience, it has the potential to be used for scientific research, namely in peer-review. As such, RSSE need not be constrained to just distributing datasets. It has the potential to distribute papers, code, or even precompiled binaries to remote computers for independent verification of results. While it is not the intent of the project, it can be used to assist with distributed computing with only minimal modifications.

Feature or Tag	Information
Automatic Updates	The reference RSSE implementation currently includes a very basic update utility. In the future this utility will be improved and expanded.
Compilation and Execution Support	In its current state, the RSSE reader can only download executable or compilable objects from remote servers. Future revisions will allow for proper compilation and execution of RSSE messages.
System Requirement Checking	The RSSE reader is currently incapable of checking system requirements. In the future, support for this will be added.
Merge Checksum and Dataset	Merging the dataset tag and the checksum tag would streamline the distribution of datasets.
Improving Checksum Performance	Checksumming in the file generation program is unacceptably slow.
Implementing more Minspec Tags	The minspec tag list is currently somewhat incomplete.
Implementing Local Caching	One of the long-term goals of this project is to develop a caching system to avoid several of the problems in distributing datasets.
Implementing the RSSE Executor	RSSE will be functionally divided between the Manager (what the reader is now) and the Executor, which will fetch cached data and try to process it.
Splitting the Reader	The current RSSE Reader application is currently insufficient for caching and the constant update cycle needed by real-world researchers. As such, it will be split into two programs: The RSSE Reader and the RSSE Manager. The RSSE Reader will act as a graphical configuration utility for the RSSE Generator. As such, most of the features of the Reader will be merged into the Manager.
Updating Scripting Support	The RSSe file generator has great potential to be scripted, especially in providing automatic rapid updates to end users.

TABLE V
FEATURES TO BE IMPLEMENTED IN FUTURE RELEASES OF RSSE

Some clear distinctions need to be drawn between the RSSE project and RSS, however. Its focus on academic pursuits should be preserved and remain a primary goal. As such, other applications, such as distributing newsfeeds or non-research related data should be discouraged to avoid feature bloat. Such feature bloat would make implementing additional readers and file generators significantly more difficult than the standard is designed to allow. However, other implementations should be encouraged to deviate somewhat so that additional features can later be brought into the mainstream RSSE specification.

VI. SUMMARY AND CONCLUSIONS

Provided that the RSSE project becomes widely adopted, it will provide a clean, easy to use, and rapid means by which researchers can share data. It has been designed with a clear emphasis on having low barriers to entry. These low barriers to entry should allow RSSE to become a *de facto* standard in research and communication. Given such status, other implementations of the reader and file generation programs would likely be written with more features than could be implemented here. Such implementations can be expected to include features specific to various fields, such as some peer review system for scientific research.

VII. ACKNOWLEDGEMENTS

I would like to thank Dr. Terrance Boult for proposing the original idea for RSSE, as well as the machine learning researchers who demonstrated a need for this software. I would like to thank the NSF for providing funding for the REU research program.

REFERENCES

[1] File i/o (featuring nio.2). http://docs.oracle.com/javase/tutorial/essential/io/fileio.html. Accessed: 2014-07-03.

[2] Alexia D. Estabrook and David L. Rothma. Applications of rss in health sciences libraries. *Medical Reference Services Quarterly*, 26(sup1):51–68, 2007. PMID: 17210549.

[3] Hannes Grobe, Michael Diepenbroek, Nicolas Dittert, Manfred Reinke, and Rainer Sieger. Archiving and distributing earth-science data with the pangaea information system. In DieterKarl Ftterer, Detlef Damaske, Georg Kleinschmidt, Hubert Miller, and Franz Tessensohn, editors, *Antarctica*, pages 403–406. Springer Berlin Heidelberg, 2006.

[4] Gerald Schaefer and Michal Stich. Ucid: an uncompressed color image database, 2003.

[5] David A. Shamma. News: One hundred million creative commons flickr images for research. http://labs.yahoo.com/news/yfcc100m/. Accessed: 2014-07-03.

[6] UserLand Software. Rss 2.0 specification, 2002.