SLINC: A FRAMEWORK FOR VOLUNTEER COMPUTING

James Baldassari
Raytheon
Network Centric Systems
Tewksbury, MA
USA
James_D_Baldassari@Raytheon.com

David Finkel
Worcester Polytechnic Institute
Computer Science Department
Worcester, MA
USA
dfinkel@cs.wpi.edu

David Toth
Worcester Polytechnic Institute
Computer Science Department
Worcester, MA
USA
toth@cs.wpi.edu

ABSTRACT
Volunteer computing is an innovative approach to high performance computing that relies on volunteers who donate their personal computers' unused resources to a computationally intensive research project. Prominent volunteer computing projects include SETI@home, Folding@Home, and The Great Internet Mersenne Prime Search (GIMPS). Many volunteer computing projects are built upon a volunteer computing framework that abstracts functionality that is common to all volunteer computing projects, such as network communications, database access, and project management. These volunteer computing frameworks tend to be complex, limiting, and difficult to use. We have designed and implemented a new volunteer computing framework called the Simple Light-weight Infrastructure for Network Computing (SLINC) that addresses the disadvantages we identified with existing frameworks. SLINC is a flexible and extensible volunteer computing framework that will enable researchers to more easily build volunteer computing projects.

KEY WORDS
Volunteer computing and distributed computing.

1. Introduction

Volunteer computing is a form of high performance computing (HPC) in which volunteers from around the world donate a portion of their computers' resources to a computationally intensive research project. Researchers who do not have access to supercomputing facilities can use volunteer computing to increase their ability to process compute-intensive data at little to no extra cost.

Although volunteer computing is a powerful tool for research, not all research projects are compatible with the volunteer computing model. To take advantage of the computing resources volunteer computing provides, the program that is used to analyze research data, called the *science application* [1], must be parallelizable. A parallel program can be converted into a client-server system in which the clients perform the computations and the server coordinates the clients. The server's primary responsibilities are to partition the input data into smaller *work units* for the clients to compute and to store the results returned by the clients for further analysis.

The primary goal of our research was to create a new volunteer computing framework that simplified the process of creating volunteer computing projects. To accomplish this goal, we first designed a modular, object-oriented architecture for the framework. We named this framework the Simple Light-weight Infrastructure for Network Computing (SLINC). During the design process we tried to make this architecture scalable by allowing the separate modules to be run on different physical computers. After designing the architecture we researched tools and third-party libraries that would decrease our development time by providing certain functionality that was critical to the framework.

The framework we developed is simple, flexible, and scalable. It can be used with science applications written in many different programming languages, so project developers can use whichever language they are most comfortable with to develop volunteer computing projects using SLINC. SLINC supports a wide range of operating systems and architectures, as well as several different databases, allowing for greater flexibility in deployment. To make SLINC easier to use, we created step-by-step guides for developing volunteer computing projects, tools and GUI utilities to expedite and simplify the project creation process, an example volunteer computing project implementation to use as a reference, and a template project that can be easily modified to create a complete volunteer computing project.

2. Related Work

2.1 Berkeley Open Infrastructure for Network Computing (BOINC)

BOINC is a framework for volunteer computing that is comprised of both server and client components [1]. The BOINC client itself uses very few system resources; it only acts as an interface between the BOINC server and the science application running on the

volunteers' computers. The BOINC client requests work units from the BOINC server on behalf of the science application and returns the results computed by the science application to the BOINC server. The BOINC client and the science application communicate through the BOINC Application Programming Interface (API), which is written in C++.

The BOINC architecture is highly modular and scalable. If the project server becomes inundated with client requests, additional servers can be added to the project with daemons running on all of the servers, each handling only a fraction of the total incoming requests. With a sufficient number of project servers, the only bottleneck in the system is the MySQL server [2].

2.2 Limitations and Disadvantages of BOINC

BOINC is a powerful and robust system for volunteer computing, but it has some significant limitations. The BOINC server can only be executed on GNU/Linux-based operating systems, which requires researchers to have experience with Linux system administration in order to create a new BOINC project.

BOINC is a composite of several distinct components. Researchers creating BOINC projects must learn the BOINC programming API and be proficient in Linux system administration, MySQL relational database administration, the Extensible Markup Language (XML), and the C++ programming language. In addition, there is limited documentation and very few tools to facilitate the creation of new projects, resulting in a long, manual process.

The BOINC system is well suited to large scale volunteer computing projects, but the limitations and complexities of BOINC can be prohibitive factors for researchers interested in creating small to medium size volunteer computing projects or those interested in volunteer computing research.

3. Goals

3.1 Create an Easy-to -Use Volunteer Computing Framework

The primary goal of this research was to create a framework to support research through volunteer computing that addressed the usability shortcomings we identified with frameworks like BOINC. Reflecting our commitment to usability, we named our framework the Simple Light-weight Infrastructure for Network Computing (SLINC). In order for SLINC to be an improvement over existing frameworks, it had to be easier to use and more flexible. SLINC had to simplify the process of creating a volunteer computing project and also to decrease the amount of time necessary to do so. Additionally, the framework's performance had to scale reasonably well. SLINC was designed to be both modular and extensible so that future developers and researchers would be able to easily modify and extend the functionality of the system. This modularity also improved the scalability of the framework.

One of our goals for the implementation of SLINC was to create a cross-platform server and client that were compatible with the most common operating systems. The cross-platform property of SLINC contributed to our goals of flexibility and ease of use. The implementation of the framework used open standards wherever possible to increase interoperability and facilitate the addition of new features to the system. The concept of scalability was extended from the architecture through the implementation using a combination of design patterns, features of the programming language, and third party libraries based on open standards.

3.2 Create a Volunteer Computing Project for Testing

Once the implementation had achieved sufficient functionality, our next goal was to create a simple volunteer computing project. This example project allowed us to test the system in several different ways. We were able to test the basic functionality of the framework components, as well as the client-server communication. We also tested SLINC's ability to scale when multiple clients were contacting the server simultaneously. The example volunteer computing project allowed us to compare SLINC to BOINC in terms of the complexity of creating a project. Going through the steps of creating a project with SLINC also helped us to create better documentation about the process for researchers who will use the framework to create their own volunteer computing projects.

3.3 Create Documentation and Tools

Our last goal was to make SLINC as easy to use as possible. We accomplished this goal mainly by creating thorough documentation, both in the code itself and in the form of user manuals, implementation guides, and interface specifications. We also developed tools to automate as much of the project creation and maintenance process as possible. One of the most important tools we created was a graphical utility that guides users through the process of configuring a SLINC project.

4. Architecture Overview

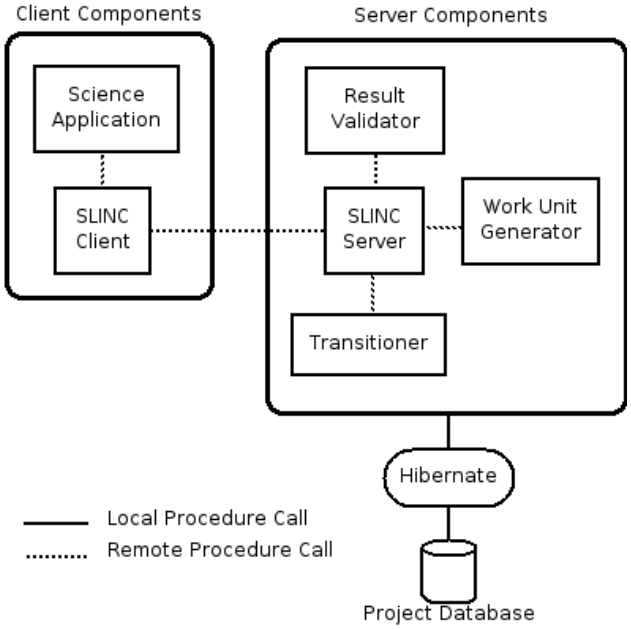

Figure 1: SLINC Architecture

SLINC consists of several distinct components, similar to the way the BOINC framework was designed. Also like BOINC, some of these components are provided by SLINC, but others need to be implemented by the creators of a volunteer computing project. The components are classified into server-side and client-side components. All components that are separate processes communicate via XML-RPC [3], an open specification for XML-based remote procedure calls. The server-side components are designed to be run on machines maintained by the project administrators, and the client-side components are meant to run on the computers of people who choose to contribute to the project, referred to as volunteers. Since most components communicate via XML-RPC, they can be located on different physical computers.

The server-side components are responsible for maintaining the project database, partitioning input data into work units, distributing work units to clients, and processing and validating results for each work unit. There is a single, central server application for each project that all clients connect to. The client-side components request work units from the project server, compute the result for each work unit, and return the result to the server. There is exactly one client that runs on each volunteer's computer and communicates with the central project server.

5. Design Decisions

The programming language for the framework had to be portable and execute on different operating systems and architectures without modification. We also needed to use a language that had a variety of third-party tools and libraries available to decrease implementation time. For these reasons we decided to implement SLINC in Java. The other programming language decision we had to make was which languages researchers should be able to use when creating volunteer computing projects with SLINC. We decided that, at a minimum, SLINC should support C++ and Java due to their ubiquity and common use in volunteer computing. If possible, other popular languages would be supported.

For inter-process communication (IPC) we decided to use XML-RPC, which uses HTTP as its transport protocol and XML as its encoding method. There are implementations of XML-RPC for many languages, including C, C++, Java, Perl, PHP, Python, Scheme, Ruby, ASP, and others. Due to the number of popular programming languages that have XML-RPC implementations, the use of XML-RPC interfaces makes SLINC quite flexible. Researchers who develop applications with SLINC have the freedom to choose the programming language that they are most familiar with.

BOINC uses a MySQL database to store project data. However, external databases add complexity to project configuration and maintenance. We wanted SLINC to be easy to use for small projects, powerful enough to support large projects, and flexible enough to be used in many different environments. To achieve this goal we utilized the open-source Hibernate [4] library. Hibernate is a Java library that serves as an abstraction layer for database access. It provides the ability to persist objects to a number of different relational database management systems (RDBMSs). Through Hibernate SLINC supports several different types of RDBMSs to accommodate the needs of different users and projects. Small projects can use an embedded Java database called HSQLDB [5], which saves data to a plaintext file on disk. Using HSQLDB is easier because the project developers would not have to install and configure a separate RDBMS, and small projects probably do not need the performance of a standalone database. Larger projects have the option to use a normal RDBMS. SLINC supports some of the most popular databases, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

6. Testing

We used regression testing and functional testing to find faults during the development of the framework. After we completed the implementation of the framework we conducted performance tests to determine whether there existed any serious performance problems. In addition to quantitative testing, we used two forms of qualitative testing to assess the utility of our framework. One method was a comparison of the steps required to create a volunteer computing project using BOINC to the steps required to create the same project using SLINC. The other qualitative assessment was an interview with another researcher in the field of volunteer computing about SLINC and how it could be improved.

6.1 Performance Testing

SLINC must be scalable if it is to be a useful tool for volunteer computing. Most of the functional tests were conducted with only one or two clients. We decided to test the scalability of the system by comparing the amount of time it took to complete a set number of work units with different numbers of clients. We also used the performance tests to determine whether the choice of programming language had any effect on the overall performance of the project server. One of the most significant advantages of SLINC is that project-specific components can be developed in many different languages, so we wanted to identify any significant performance discrepancies between languages. We chose to compare science applications written in C++ and Java in our performance test. The results of these tests appear in Section 7.1.

6.1.1 Methodology

We used the example project to test the performance of the framework. The example work unit generator partitions the set of positive integers into work units containing 1,000,000 consecutive integers. The example science application searches that range of integers for prime numbers using a simple $O(\sqrt{n})$ algorithm. We decided to conduct the performance test using both the Java and C++ versions of the example science application. Since the science application was so simple we did not expect the choice of programming language to have a significant impact on its performance. Therefore, any appreciable difference in performance would probably indicate that one or more components of SLINC behaved differently depending on the programming language used to implement the project-specific components. Programming language independence was critical to of our goal of creating a flexible framework, so it was important to verify that different languages would work equally well when used with SLINC.

We began each test by purging the HSQLDB project database, starting the project server, and executing the example work unit generator. The example work unit generator would then create 64 work units and send them to the project server. After the initial 64 work units were created, we shut down the work unit generator so that it would not generate more work units. Once the work units had been generated, we started one or more clients simultaneously. We measured the time to complete all work units by examining the server log files. The time to completion was computed by comparing the time the first work unit was requested to the time the last result was returned.

We used several different configurations for the performance test. We varied the number of clients in the system using 1, 2, 4, 8, 16, and 32 computers with one client per computer. We tested each configuration twice, once using the Java science application and once using the C++ science application. The C++ application was compiled with g++ with no optimizations, and the Java application was compiled with version 1.5 of Sun's JDK.

Each test was conducted three times, and the average time for each test was used in our analysis. It is important to note that this performance test was incomplete. Volunteer computing projects may have hundreds or thousands of users, but our tests used a maximum of only 32 clients. The purpose of this test was not to determine how the framework would react to extreme load, but rather to verify that the framework would behave as expected when multiple clients were used and to determine whether the amount of work completed would scale efficiently as the number of clients increased.

6.1.2 Test Environment

To carry out the performance test we constructed a 32-node Linux cluster in WPI's distributed computing lab. Of those 32, 23 had 350MHz Pentium II processors, and the remaining 9 had 400MHz Celeron processors. All of the machines had 128MB of RAM. The head node was a dual processor Athlon MP 1.2GHz machine with 1GB of RAM. We chose to install the OSCAR [6] cluster management software due to its ease of installation and compatibility with our older hardware.

Due to the heterogeneous composition of our cluster in terms of processor architecture, we tried to minimize the effects of any performance discrepancy during our testing. The tests using 1, 2, 4, 8, and 16 nodes were all executed on the Pentium II machines. Only the last test, using all 32 nodes, included a mixture of both Pentium II 350MHz and Celeron 400MHz machines.

6.2 Usability Testing

The purpose of the usability testing was to determine whether we had succeeded in creating a volunteer computing framework that was easy to use. We performed this testing by comparing SLINC to BOINC and by asking another volunteer computing researcher to evaluate our framework.

6.2.1 Comparison with BOINC

Using documentation available on the BOINC website, we made a qualitative comparison of SLINC to BOINC. We compared the two frameworks using three different measures: the number of steps involved in creating a project, system and project requirements, and major features. The purpose of the comparison was to determine whether the process of making a simple volunteer computing project was easier using SLINC or using the BOINC. Using this comparison we analyzed the distinguishing features of our framework to determine

whether they contributed to our primary objective of creating a volunteer computing framework that was easy to use. The comparison with BOINC can be found in Section 7.2.1.

6.2.2 Interview with a Volunteer Computing Researcher

In addition to the performance testing and qualitative comparison to BOINC, we were interested in the opinions of other researchers in the field of volunteer computing. One of the authors, David Toth, who had not been involved in the software development, agreed to help us test the usability of SLINC by using it to create a simple volunteer computing project himself. Using the SLINC documentation and the latest release of SLINC, David modified our examples to create a new volunteer computing project. We asked for his opinion about the current state of SLINC and for suggestions for ways in which we might improve it. We used David's responses in our analysis, which can be found in Section 7.2.2.

7. Analysis of Results

7.1 Performance Analysis

Figure 1 shows the average number of seconds required to complete all 64 work units using the C++ and Java versions of the science application.

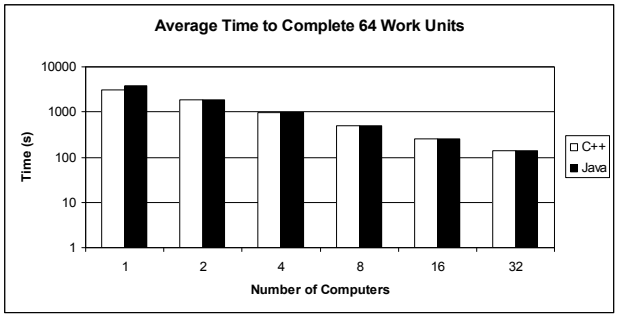

Figure 2: Average Time to Complete 64 Work Units

Our results indicated that both our C++ and Java example projects scaled well up to 32 clients. The results also showed that the C++ and Java versions of the example project performed within one percent of each other in most cases, so there did not appear to be any major performance discrepancies between the two languages. The results of the performance test suggest that we were successful in our goal of creating a scalable framework, although additional testing with larger numbers of clients would be necessary to verify this claim with greater certainty.

7.2 Usability Analysis

One of the problems we identified with the BOINC framework was that the process of creating a volunteer computing project with BOINC was difficult, especially for those who were not familiar with Linux, MySQL, Apache, and C++. Furthermore, the process of installing BOINC and developing even a simple project involved many steps and required a high level of proficiency in Linux administration and C++. We used two methods to qualitatively measure the usability of SLINC: comparing SLINC to BOINC and having a colleague conduct a peer review of SLINC.

7.2.1 Comparison with BOINC

We compared SLINC to BOINC in three ways. We compared the process of creating the simplest possible project with SLINC to the process required to do the same with the BOINC framework. We also compared the requirements of our framework to the requirements of the BOINC framework. Lastly, we compared the major features of the two frameworks.

Due to its dependency on other software packages, such as MySQL and Apache, there are several steps involved in the installation of BOINC. Installing and configuring these other software packages requires proficiency in Linux system administration. By contrast, SLINC is only dependent on the Java Virtual Machine (JVM), and there is no installation necessary to use SLINC. Everything required to make simple projects with SLINC is included in each release of the framework.

It is necessary to develop most of the same components with SLINC as with BOINC in order to create a simple project. These components include the science application and the work unit generator. There is one additional component, called the *assimilator*, which needs to be developed for a BOINC project but not for a SLINC project.

The method each framework uses to communicate with the project-specific components is very different. BOINC uses an API written in C++ with FORTRAN wrappers, so components developed for the BOINC framework must be written in C++ or FORTRAN. SLINC's interface is based on XML-RPC, so components can be implemented in any language for which there is an XML-RPC library. Developers are more likely to be familiar with XML-RPC because it is an open, general purpose web standard. The BOINC API, although it is open, is not a general API; it can only be used with BOINC projects.

Perhaps the most significant difference between SLINC and BOINC is the project configuration process. With SLINC, projects can be configured using the Project Builder tool. The Project Builder is a graphical tool that is modeled after the ubiquitous installation wizards that most people are accustomed to using for installing and configuring software. After a user has completed the Project Builder, no further configuration is necessary. The Project Builder generates the project configuration files automatically. BOINC has many configuration steps, each of which requires editing an XML file or executing a script.

Our comparison of the requirements of BOINC and SLINC revealed that SLINC is more flexible than BOINC. Since our entire framework is Java-based, both the server and client can be used on any platform supported by Java. Another limitation of BOINC is the choice of languages in which the various server-side and client-side components can be developed. BOINC requires project developers to use C++ or FORTRAN. SLINC has a degree of language-independence due to its use of XML-RPC. Components for SLINC can be developed in any language for which there is an XML-RPC library. SLINC's use of Hibernate allows flexibility in the type of database a project can use. BOINC requires that every project use a MySQL database, but SLINC provides the ability to use the embedded HSQLDB database, MySQL, PostgreSQL, Oracle, or Microsoft SQL Server. The databases that are supported by SLINC can easily be extended to include any database that is compatible with Hibernate.

One of the major advantages of SLINC over BOINC is the complete example project and templates provided with the project. Future project developers will be able to learn about how to create projects using SLINC by examining an example of a simple, but completely functional, project. The component templates are intended to simplify and shorten the development process for projects written in the C++ or Java languages.

7.2.2 Analysis of Interview

We asked David Toth how we could improve the usability of SLINC. His first suggestion was that we develop a simple help system in the Project Builder tool so that users would be able to view basic information about each step in the project configuration without the need to search through the separate documentation. He also suggested ways in which we could improve the documentation. Another important usability issue that David identified was the requirement that project developers learn XML-RPC in order to write the project components that would interact with SLINC. His observation was that the XML-RPC code for a given component would not necessarily need to change; only the logic specific to each project would need to change. David's suggestion was to separate the implementation of the XML-RPC code from the project-specific logic by creating templates for each component. Each project could simply add its own logic to certain classes in these templates without the need to implement the XML-RPC code. We developed templates for C++ and Java based on the established use of those programming languages in volunteer computing.

8. Conclusions

Our primary goal in this research was to design and implement a volunteer computing framework that addressed the usability issues we identified with similar frameworks like BOINC. Our focus was on ease of use, but it was also important to create a framework whose core functionality was equivalent to existing frameworks and whose performance could scale efficiently as volunteers joined the project.

SLINC has all of the core features necessary to create a volunteer computing project, as well as features designed to make the framework easier to use and more flexible. Our results showed that SLINC scaled efficiently as tested.

Although we have succeeded in creating an easy-to-use, fully functional volunteer computing framework, there are several features that could be added or further developed to provide additional functionality. Volunteer computing projects can have hundreds or thousands of active volunteers, but we were only able to test SLINC with 32 clients. We would like to be able to analyze the performance of the framework with a large number of clients. SLINC also does not use encryption because security was not one of our primary goals. However, the security of SLINC should be improved in the future.

SLINC development is ongoing in order to provide advanced functionality and improve usability and security. The latest release of SLINC as well as complete source code for SLINC can be downloaded from its SourceForge site: http://slinc.sourceforge.net. SLINC is free software and is released under the GNU General Public License (GPL).

References

[1] Anderson, David P. "BOINC: A System for Public-Resource Computing and Storage," *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, (November 8, 2004, Pittsburgh, USA).

[2] Anderson, David P., Eric Korpela, and Rom Walton. "High-Performance Task Distribution for Volunteer Computing," *Proceedings of the First IEEE International Conference on e-Science and Grid Technologies*, (December 5-8, 2005, Melbourne, Australia).

[3] "XML-RPC Home Page," http://www.xmlrpc.org, UserLand Software, (accessed October 8, 2005).

[4] "Hibernate," http://hibernate.org, JBoss, Inc, (accessed October 22, 2005).

[5] "HSQLDB," http://www.hsqldb.org, The hsqldb Development Group, (accessed January 7, 2005).

[6] "OSCAR: Open Source Cluster Application Resources," http://oscar.openclustergroup.org, Open Cluster Group, (accessed February 3, 2006).