Universally Accessible Distributed Computing on Public, Heterogeneous Networks,

Applied to the Search for Mersenne Primes

New Mexico

Supercomputing Challenge

Final Report

April 1, 2009

Team 105

Monte del Sol Charter School and Capshaw Middle School

Team Members:

Max Bond

Harsha Dodda

Teachers:

Natalie Martino

Project Mentor:

Nick Bennett

-

Abstract:

Since the 1970s, distributed computing over the Internet, the practice of dividing various tasks to multiple computers on a public network, has been possible. For the past 20 years, technologies such as “Web 2.0” (JavaScript, Ajax, Dynamic HTML, as well as Java applets), have been used to improve the look and feel of web sites. However, this same technology can be used to create powerful distributed computing networks, harnessing the nearly endless supply of desktop machines to solve computationally difficult problems while they browse web pages.

Introduction:

Distributed computing is the practice of dividing computational tasks among resources on a computer network. This has long been used to solve extremely difficult problems or process complex data, such as the Search for ExtraTerrestrial Life at Home (SETI@Home), the Great Internet Search for Mersenne Primes (GIMPS), distributed.net, and the of cracking two of the last three Enigma Machine encrypted messages, which was accomplished by a distributed computing network in February and March of 2006[1]. There are two bare essentials: networking, to communicate, and persistence, to store results and keep state.

Networking must be provided by physical links between computers, as well as software drivers and protocols to communicate over them. Persistence must be a software-based solution, with one or more processes running on a accessible computer at all times. In our project, networking is provided by the Internet and the TCP/IP protocol suite. Persistence is provided by both a database, and the control server.

Distributed computing began in the 1960's and 70's, when the Internet began[2]. In fact, it was one of the key reasons TCP/IP and the Internet was created; to link the disparate and expensive military computer systems, to get more use out of them[2]. As networking technology advanced, distributed computing became easier, more reliable, and offered greater rewards. Some of the early pioneers of distributed computing were, in fact, network worms[2]. The Creeper worm would use unused CPU cycles, to find its next target, and attack. Later, programmers at Xerox's Palo Alto Research Center would, in 1973, use the same idea to render graphics, by having a virus infect almost 100 computers on their local area network[2].

Its harder to implement distributed computing over a public network, as we are doing, for all the reasons that make any action on the Internet more difficult. For a start, you don't know how long a resource will be able to function. Since we don't own or operate the remote host, we aren't at all responsible for it uptime or downtime. We must implement systems for a resource to verify its connection is still active, by posting an update.

Since we also can't control the configuration of their routers and gateways, we must use protocols and methods that are almost always allowed by networking hardware.

While these pose major challenges that aren't faced in conventional distributed computing, there are generally more advantages than disadvantages; nearly endless computing power available, extremely low cost, and ease of use, for instance.

On a conventional distributed computing network, you don't have to worry about giving any one machine too much work. On our volunteer network, we've had to limit the jobs to something manageable.

Another challenge is firewall and gateway settings. We don't know what kinds of connections the networking hardware will allow. We can predict, however, that almost all of them will allow HTTP connections. Thats why we have our resources interact with the control server, which, in turn, communicates with the database.

Approach:

There is no single way to implement a distributed computing network. For our project, we decided to stay in the following guidelines:

- We will use the Python and Java programming languages
- Our network should be ordinary desktop computers, communicating over the Internet
- Nodes on our network must be able to communicate
- Our network must be universal, or almost universal
- Must be able to store results persistently
- Our network must be able to solve a computational problem

We selected the Python programming language, for its extensive libraries, easy-to-read syntax, and because it is easy to learn and program in. As we are novice programmers, we needed these benefits to complete our projects on time.

We wanted to use desktop machines, because there are an almost unlimited number of them, and they are getting faster and faster every year[3]. We could, given enough time, effort, and skill, get a network with equivalent power to a supercomputer, for free.

Of course, desktop machines can't guarantee uptime, so we've implemented a system whereby, if a host doesn't update the control server with its status at regular intervals, a job will expire. When a job expires, the 'job key', which is a randomly generated 50-digit string given to assigned to each job, is redacted, so that even if the host does attempt to report back, they will be ignored.

For a method of two-way communication with the server, that wouldn't be blocked by most networking hardware, we decided to us a Common Gateway Interface (CGI). To communicate between the resource and the CGI, we found that eXtensible Markup Language Remote Procedure Call (XML-RPC) was one of the more efficient ways.

XML is a way of formatting data independent of computer language, as well as processing data, creating web pages, and general use scripting. Remote Procedure Calls[4] are requests to a remote host, to execute a function. Resources use XML-RPC to report data from tests, as well as request more work.

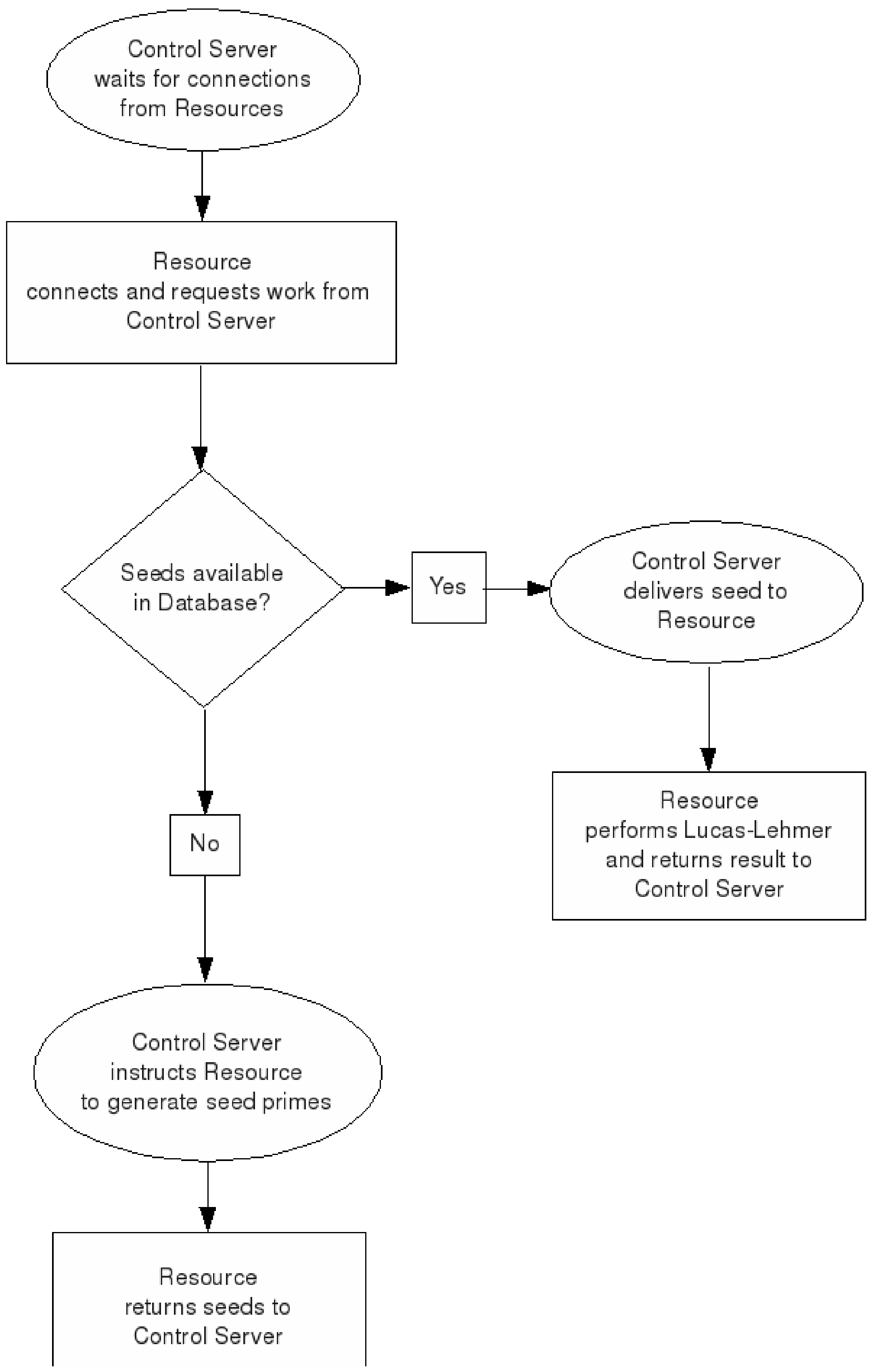
For universality, we wanted to use Java, specifically, a Java applet. This way, it could run the necessary calculations to solve our problem, while the machine's owner surveys the page's content. An applet is a tiny application, which runs in a 'sandbox environment' inside of a browser[5]. This environment helps protect the user from malicious code. Java and browsers are extremely universal, and are most likely on a computer when its shipped by the original equipment manufacturer.

Compatibility is also an issue on heterogeneous networks such as ours. Different operation systems have different standards, and execute programs differently. We work around this by using cross-platform software, such as Java applets, and by moving anything that isn't totally universal (ie, the Python code) up to the control server, and behind a CGI. This allowed it to be accessed by anyone, regardless of operating system. Other distributed computing projects, such as GIMPS and SETI@Home, use low-level languages such as C or Assembly Language, which can be compiled on several different architectures and operating systems to achieve compatibility. However, this doesn't always achieve perfect compatibility, and some of these systems haven't yet been compiled for all operating systems.

Our database, a key part of our persistence, is a MySQL database. It will be isolated from the resources; they will have to ask the control server to update the database for them.

For our problem, we chose to search for Mersenne primes, because it was a topic of interest that was and is quite suitable for distribution. Mersenne primes are prime numbers of the Mersenne sequence, which is determined by 2^n-1, where $n = \dot{\in} \{1, 2, 3...\}$. Mersenne primes are used in random number generation, public-private architecture and rotor encryption, hash mapping (i.e. databases) and various low-level storage functions in computer science[6]. Large Mersenne primes are easier to find than ordinary large primes, because, we can use known primes as seed numbers[6], and because we can more easily determine their primality with Lucas-Lehmer residues.

Based on these guidelines, our network follows this logic:


Control Server waits for connections from Resources
Resource connects and requests work from Control Server
Seeds available in Database?
Yes
Control Server delivers seed to Resource
Resource performs Lucas-Lehmer and returns result to Control Server
No
Control Server instructs Resource to generate seed primes
Resource returns seeds to Control Server

There are three basic parts to our network; resources, control server, and the database.

The resources are any computers that volunteer to be a part of our network. They run the calculations we give them, until they navigate away from the web page.

The control server handles interactions between a resource and the database. When a resource requests work, it retrieves an available task from the server, and interprets the information. It will either ask the resource to test whether or not a number is a Mersenne prime, or, if there are no more possible candidates, it will ask the client to generate more. It will assign the job a job key, and give this to both the database, and the recourse.

From that point, until the completion of the job, there is one requests a resource can make; an update. This update 'renews' the connection, so it can't be expired for half an hour, and adds the reported results to the database.

Finally, there is the database. The database stores all of the information from tests, so that the control server can use it to distributed jobs.

Model:

Mersenne numbers are generated by the equation 2^n-1.

They have certain properties that make calculating Mersenne primes, easier to calculate than normal primes.

For instance; a Mersenne prime must be generated with a prime number as its seed, i.e., 2^p-1, where 'p' is prime[6]. The reason, is because any composite seed can be easily expressed as a polynomial.

If we're using a composite ('c') seed, 2^c-1, we can reduce that to $2^{mn}-1$, where 'mn' are the prime factors of 'c'. By definition, since this is a composite number, these must be greater than one. Reducing that with polynomials, we can see that $2^{mn}-1 = (2^n-1) + (1 + 2^n + 2^{2n} + 2^{3n} + \dots 2^{(m-1)a})$. Since m & n are greater than 1, the equation must be a product of numbers greater than one. And the product of two integers greater than one cannot be prime.

This means we only have to test prime-numbered seeds. We'll generate the primes with the Segmented Sieve of Eratosthenes. The Sieve of Eratosthenes is a very simple prime finder. It works of the principal that every composite number, is an amalgamation of its prime factors. So, if you take an integer greater than 1, and generate all the possible composites it can make, up to an arbitrary upper limit, you can eliminate all of those numbers as primes.

Based on these assumptions, Eratosthenes designed a simple, but effective algorithm for finding all the primes in a given range[7]:

- Write out all the integers in the desired range, from 1, to the upper limit, in a grid..
- Circle two; since there are no integers behind besides one, it is a prime.
- Starting with two, got first to the number's square, and cross it out; it is not prime. Any numbers below the square have been eliminated by smaller primes. Then go to the number's next multiple, and cross it out; it is not prime. Skip numbers that have been crossed out; their multiples have already been eliminated.
- Repeat the previous step until the square root of the upper limit; any number greater than that, squared, it out of the range of this test.
- Circle any remaining numbers; they are prime.

One optimization, which is key to our project, is the segmented sieve[7]. In a segmented sieve, you have a range you want to segment, in which the lower limit is not necessarily 1. You then have a smaller range of all the positive integers less than or equal to the square root of your first range's upper limit. You run a normal sieve against the second range, and use the primes you've generated to eliminate composites from the second range. This is a much more effective method, since you can never reach the end of your range; you can extend it infinitely, and process the primes in between segments.

The next property that makes Mersenne primes more easily computed than others, is their Lucas-Lehmer residues, and the Lucas-Lehmer Test for Mersenne Numbers. These are derived by the equation *r = ((r * r) – 2) MOD* M_p, where *s* is the sequence defined in the Lucas-Lehmer Test, *p* is the seed, and M_p is $2^p - 1$.

This residue makes up the most crucial component of the Lucas-Lehmer Test for Mersenne Numbers. This is a specialized primality test that works only for Mersenne numbers, and is faster than other many other primality tests. The equation is:

$$s_i = \begin{cases} 4 & \text{if } i = 0; \\ s_{i-1}^2 - 2 & \text{otherwise.} \end{cases}$$

(Courtesy Wikipedia)

This test takes the residue, of the residue, of the residue, et cetera, *p – 2* times. If the residue is zero, than the Mersenne number is prime. Otherwise, its composite.

In psuedocode, this might look like[6]:

```
 prime = odd_prime # The test must receive an arbitrary odd prime
residue = 4 # Initial residue
Mersenne = $2^{prime} - 1$ # Mersenne equation
**repeat** p – 2 times:
  residue = ((residue × residue) – 2) mod M # Residue equation
**if** residue == 0
          **return** "PRIME"
     else
          **return** "COMPOSITE"
```

With slight a modification of the psuedocode, this test is also resumable. This makes it easy to distribute. One computer can calculate 1000 repetitions of the test without too much trouble, and then report to the server the current value of residue, and the iteration of the test.

The final property that makes Mersenne numbers easier to compute is the ease of compression. Primes can be hard to distribute, because they have to be transported in full. With the exception of 2, all primes are odd, meaning that putting them into Scientific Notation would actually make them larger. (The decimal point takes up memory, too.) There are very few patterns, so compression algorithms such as .tar and .zip do little good. Mersenne numbers, however, while just as large, can be derived from their seed with one very simple calculation; $2^p - 1$. Therefore, you only need to send the seed to generate the prime.

For instance, sending the seed of the largest known Mersenne Prime, "43,112,609", down the wire to a resource isn't terribly expensive – it takes exactly one byte[8]. But sending the prime it generates, would take about 1 and a half megabytes[8]. This entire document doesn't come close to that!

Implementation:

Our implementation has three basic parts; the database, the control server, and the resource applet.

The first part, the database, uses the open-source MySQL. This database is a reliable, effective database, that is absolutely free. The database accepts commands in the form of the SQL language.

After that, we have the control server. We've implemented this as a Python CGI, that can communicate with the resource. The CGI has four public functions that are available to the resources, through XML-RPC; `request_work(), post_sieved_primes(), post_ll_result(),` and `post_ll_status()`. `request_work()` fetches data from the database, and determines which job to pass to a resource. It decides based on whether or not there is an available seed to test; if there is, it will run a Lucas-Lehmer test. If there is not, the resource will be requested to use a sieve to generate more.

If the resource is given a Lucas-Lehmer job, the resource will use the `post_ll_result()` and `post_ll_status()` methods. The former, updates the server with a completed Lucas-Lehmer test, which in turn puts the values into the database. The ladder updates the server with the current status of the test, as well as functioning as a 'keep alive'; it sets the expire time of the job back half an hour, to verify that the job is still running.

The segmented jobs, being rather faster than Lucas-Lehmer, only need one method, `post_sieved_primes()`. This tells the server which numbers in the segment were prime, in the form of a bitset. Meaning, it marks a place holders representative of the numbers in the segment True, if they are prime, and False, if they are composite.

The server converts the bitset into actual numbers, and inserts these into the database.

The resource is written in Java, and not Python, so it can run in browsers. It is compiled into an applet, and inserted into web pages.

After retrieving its task from the server, the applet uses its implementation of the Lucas Lehmer algorithm, and segmented sieve, to retrieve values to return to the server.

Results:

We were able to start testing on March 31st. We had planned on launching it long before then, but were held back by problems with the Java applet. The original plan, in which the applet was made in Jython, failed because of an error that we couldn't resolve. We finished our first sieve at 10:34:38 AM, and our first Lucas-Lehmer test at 10:36:51 AM.

Out off all the computers we tested, we only managed to get compatibility with about half of them. However, this could be because we compiled it with Java 6, which wasn't yet supported on all browsers. If we had used Java 5, we probably would have gotten much better results.

After only a few hours of running, we managed to find more than half of all known primes, using mainly only one computer. With a second computer, we could more quickly iterate through the primes list, conducting two tests at once.

The applet demonstrated impressive speed, and error handling. By manipulation the database during testing, we were able to test whether the applet would survive conditions that would raise an exception. Even when we modified or deleted the job record, the applet was able to effectively handle the errors, and continue working after only a short delay.

We got up to Mersenne 26, on April First at 12:09:56 AM. As of this writing, and the test is still running, at http://mds.g-r-c.com/

Mersenne 26 was originally discovered February 9th, 1979, by Landon Noll.

Conclusions:

Distributed computing over the Internet, taking advantage of the plentiful desktop machines, is an expensive operation, in terms of labor. For a conventional system, you would only need to produce one application, and it could be in the language of your choosing. But, due to security concerns, and a need for uniformity, projects such as ours need to use languages like Java, which can run in the browser.

However, when you compare that to the massive cost of purchasing and maintaining the hardware needed for high performance computing, the cost is minimal. Its simply a matter of proper deployment; a distributed supercomputer of this fashion, embedded in something truly ubiquitous, like banner ads, could most certainly calculate at rates in the teraflops and, given global deployment and extremely efficient implementation, perhaps even come close to a pedaflop. Dollar-wise, the cost of running this distributed network, compared the cost of running a physical system of comparable power, would be astonishing. The only limit on the power of the system, would be how much power you could extract from the average customer, without grinding their computer to a halt.

A system such as this, unfortunately, would be unlikely, however. The public, hetrogenous nature would mean all the calculations would be public knowledge; so the only types of calculations that could be performed, would be public projects, and not proprietary projects that would provide incentive for companies to build and maintain these networks.

A slightly more plausible model, would be a 'viral marketing' one. Say, for instance, a well-established distributed network, such as GIMPS, made a JavaScript version of their client. They could embed it in their own public web page, for a start, but they could also convince users to do the same by providing some service; say, some sort of dynamic real-time processing of the data into an 100-500 pixel graphic – about the size of a signature banner on bulletin boards. GIMPS could release that to their users, who, in turn, insert it into their signatures. Then, every time someone visits one of the bulletin boards, they perform some very small calculation, such as the next five iterations of the Lucas-Lehmer test, and then return that to GIMPS. As long as theres some method of ensuring that no one ends up running forty scripts at once, if they see a forum with forty posts containing the banner, it could be a workable method.

Going Further:

Firstly, there are some unimplemented optimizations on our client. It could, for example, use multi-threading much more effectively than it currently does. If the client were on a machine with multiple processors, the threading libraries in Java would be able to run separate threads on each. This means, a task could conceivably do twice the work, running two tests at once.

Next, our system is not truly universal. There are some comparability issues with Macintosh, for unknown reasons.

Do to the difficult nature of the tests, it uses an extremely high percentage of CPU, sometimes using a whole core. This poses a problem on non-multiprocessor systems.

Finally, if we had more time, we would gather more data, and further distribute the client. It may be interesting to insert it into actual web pages, and see what kinds of results we get.

APPENDIX:

References:

[1] "Unkown Author." M4 Project. M4. 31 Mar. 2009 <http://www.bytereef.org/m4_project.html>.

[2] "Unknown Author(s)." Distributed Computing History. 31 Mar. 2009 <http://cse.stanford.edu/class/sophomore-college/projects-01/distributed-computing/html/body_history.html>.

[3] Blankenhorn, Dana. The Blankenhorn Effect How to Put Moore's Law to Work for You. Grand Rapids: Trafford, 2006.

[4] "Remote procedure call." Wikipedia, The Free Encycopedia. Wikimedia Foundation. 31 Mar. 2009 <http://en.wikipedia.org/wiki/Remote_procedure_call>.

[5] Oaks, Scott. Java Security. 2nd ed. O'Rielly.

[6] "Mersenne prime." Wikipedia, the free encyclopedia. The Wikimedia Foundation. 31 Mar. 2009 <http://en.wikipedia.org/wiki/Mersenne_numbers>.

[7] "Sieve of Eratosthenes." Wikipedia, the free encyclopedia. Wikimedia Foundation. 31 Mar. 2009 <http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes>.

[8] Bit and Byte Conversion. Bee Sky Consulting. 01 Apr. 2009 <http://www.beesky.com/newsite/bit_byte.htm>.

A Note on Project Resources:

While all the algorithms, code, and other items were, in one form or another, implemented by the students, some of the final products were produced or revised by the project mentor and other adults involved with the projects.

Specifically:

- The final code for the resource was implemented by Nick Bennett, and is used with his permission.
- The final control server was slightly modified from the original, student-produced program by Nick Bennett, in order to insure compatibility with the resource.
- The final version of the network flow chart was created by Robert Bond, based on the original chart and the full source code. It is used with his permission.

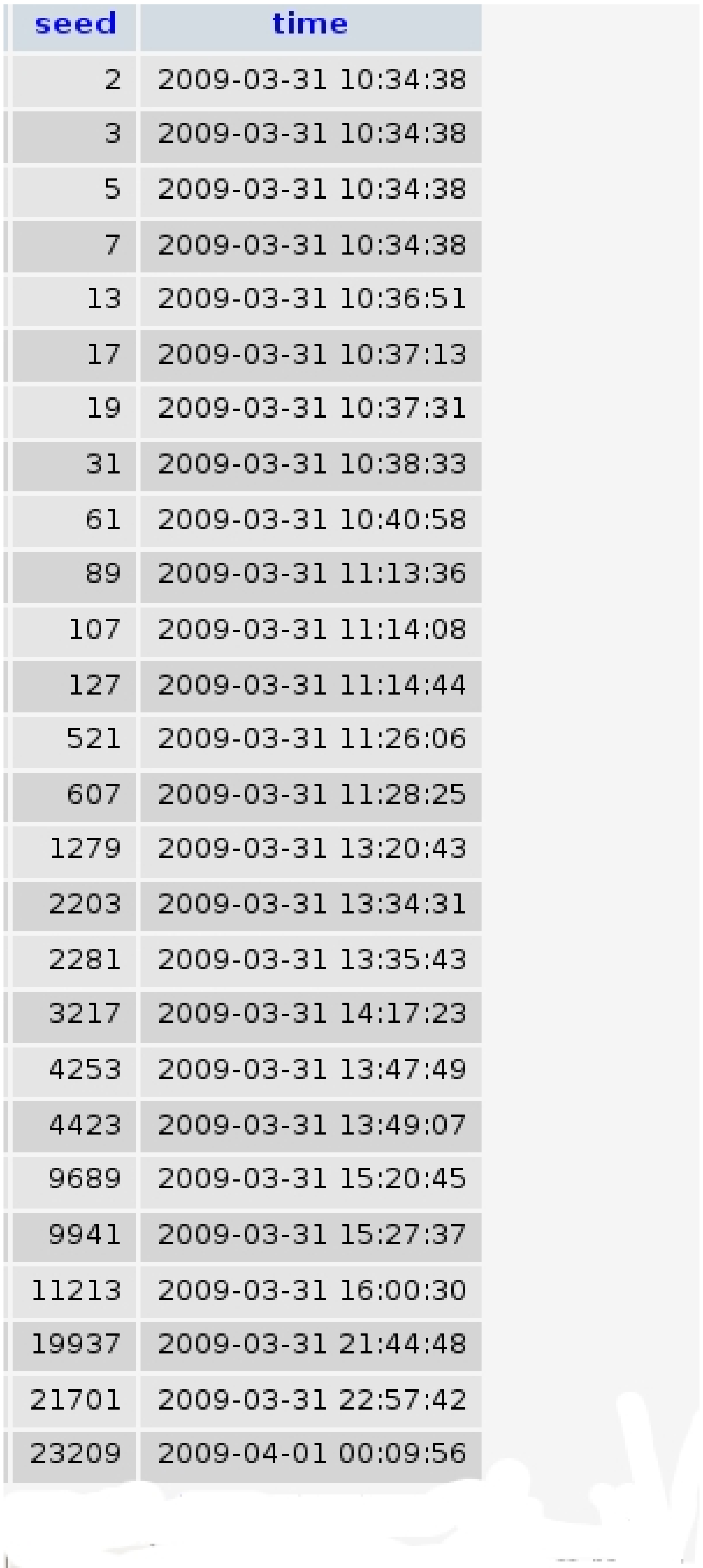
Acknowledgements

We'd like to acknowledge the following people, who've helped our project immensely

- Nick Bennett, for serving as a project mentor, and going far above and beyond the call of duty
- Robert Bond, for his immense assistance in the form of advice, time, and help with the flow chart
- Natalie Martino, our teacher sponsor, who made this project possible
- The Supercomputer Challenge & GUTS Program
- Last, but not least, all the people who participated in the tests!

Primes Found:

This is a record of all the primes we have found, as of 12:09:56 AM, April 1:



seed	time
2	2009-03-31 10:34:38
3	2009-03-31 10:34:38
5	2009-03-31 10:34:38
7	2009-03-31 10:34:38
13	2009-03-31 10:36:51
17	2009-03-31 10:37:13
19	2009-03-31 10:37:31
31	2009-03-31 10:38:33
61	2009-03-31 10:40:58
89	2009-03-31 11:13:36
107	2009-03-31 11:14:08
127	2009-03-31 11:14:44
521	2009-03-31 11:26:06
607	2009-03-31 11:28:25
1279	2009-03-31 13:20:43
2203	2009-03-31 13:34:31
2281	2009-03-31 13:35:43
3217	2009-03-31 14:17:23
4253	2009-03-31 13:47:49
4423	2009-03-31 13:49:07
9689	2009-03-31 15:20:45
9941	2009-03-31 15:27:37
11213	2009-03-31 16:00:30
19937	2009-03-31 21:44:48
21701	2009-03-31 22:57:42
23209	2009-04-01 00:09:56

Source Code:

The following is the full source code for the control server, with only sensitive information omitted:

```
#!/usr/bin/python2.4

import sys
import MySQLdb
from SimpleXMLRPCServer import CGIXMLRPCRequestHandler
from xmlrpclib import Binary
import cgitb
import bstring

cgitb.enable()

host = *****
usr = *****
passwd = *****
dbase = “mds_dist_com”
cflag = *****

key_expiry = 1800 # 30 minutes
segment_size = 1000

def next_ll_job():
    """Get next Lucas-Lehmer task, if one exists."""
    conn = MySQLdb.connect(host, usr, passwd, dbase, client_flag=cflag)
    cursor = conn.cursor()
    cursor.execute("call assign_ll_task(%d)" % key_expiry)
    result = cursor.fetchone()
    cursor.close()
    conn.close()
    return result


def next_sieve_job():
    """Get next Sieve task, if one exists."""
    conn = MySQLdb.connect(host, usr, passwd, dbase, client_flag=cflag)
    cursor = conn.cursor()
    cursor.execute("call assign_sieve_task(%d, %d)" % (key_expiry, segment_size))
    result = cursor.fetchone()
    cursor.close()
    conn.close()
    return result


def request_work():
    """Checks for work on the server."""
    result = None
    try:
        task = next_ll_job()
        if (task is not None):
            seed = task[0]
            iterations = task[1]
            key = task[3]
            if (iterations == 0):
                result = ("ll", (seed, iterations, key))
            else:
                residue = Binary(task[2])
                result = ("ll", (seed, iterations, residue, key))
        else:
            task = next_sieve_job()
            if (task is not None):
                start = task[0]
                end = task[1]
                key = task[2]
                result = ("sieve", (start, end, key))
    except:
        result = str(sys.exc_info()[1])
    return result

def post_sieved_primes(lower, upper, bitset_bytes, job_key):
    """Update the database with the results from performing a segmented sieve."""
    bitset = bstring.bebStr2Long(bitset_bytes.data)
    conn = MySQLdb.connect(host, usr, passwd, dbase, client_flag=cflag)
    cursor = conn.cursor()
    shift = 0
    while (bitset > 0):
        if (bitset & 1):
            try:
                cursor.execute("call add_seed(%d)" % (lower + shift))
            except:
                pass
        bitset >>= 1
        shift += 1
    cursor.execute("call preconfirm_mersenne_primes()")
    cursor.execute("call close_sieve_task(%d, %d, '%s')" % (lower, upper, job_key))
    result = cursor.fetchone()
    cursor.close()
    if ((result is not None) and (result[0] != 0)):
        conn.commit()
    else:
        conn.rollback()
    conn.close()
    return True

def post_ll_result(seed, is_prime, job_key):
    """Update the database with the result of the Lucas-Lehmer primality test."""
    conn = MySQLdb.connect(host, usr, passwd, dbase, client_flag=cflag)
    cursor = conn.cursor()
    cursor.execute("call confirm_mersenne_prime(%d, %d, '%s')" % (seed, is_prime, job_key))
    cursor.close()
    conn.close()
    return True

def post_ll_status(seed, iterations, residue, job_key):
    """Update the database with the in-process status of the Lucas-Lehmer primality test."""
    conn = MySQLdb.connect(host, usr, passwd, dbase, client_flag=cflag)
    cursor = conn.cursor()
    cursor.execute("call update_ll_status(%d, %d, '%s', '%s')" % (seed, iterations,
MySQLdb.escape_string(residue.data), job_key))
    cursor.close()
    conn.commit()
    conn.close()
    return True

handler = CGIXMLRPCRequestHandler()
handler.register_function(request_work)
handler.register_function(post_sieved_primes)
handler.register_function(post_ll_result)
handler.register_function(post_ll_status)

handler.handle_request()
```

The following is the full source code for the resource applet:

```
package org.projectguts.gutsxl.distrib;

import java.math.BigInteger;
import java.net.URL;
import java.util.BitSet;
import java.util.Date;
import org.apache.xmlrpc.XmlRpcException;
import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;
import org.apache.xmlrpc.client.XmlRpcClientException;

public class Applet extends javax.swing.JApplet
        implements Sieve.Listener, LucasLehmer.Listener {

    private static final String SLEEP_TASK = "Idle";
    private static final String COMMUNICATION_TASK_PATTERN =
            "Communicating with server %s";
    private static final String LUCAS_LEHMER_TASK_PATTERN =
            "Testing M(%d) for primality";
    private static final String SIEVE_TASK_PATTERN =
            "Sieving for primes in {%d...%d}";
    private static final String LUCAS_LEHMER_STATUS_PATTERN =
            "Testing M(%d): s(%d) = %s\n";
    private static final String CONDENSED_RESIDUE_PATTERN =
            "%s...%s (%d digits)";
    private static final String LUCAS_LEHMER_RESULT_PATTERN =
            "M(%d) is %s\n";
    private static final String SIEVE_RESULT_PATTERN =
            "Found %d primes in {%d...%d}: %s\n";

    private static final String SERVICE_URL =
            "http://mds.g-r-c.com/distcomp2.cgi";

    private static final String GET_WORK_METHOD = "request_work";
    private static final String LUCAS_LEHMER_RESULT_METHOD = "post_ll_result";
    private static final String LUCAS_LEHMER_STATUS_METHOD = "post_ll_status";
    private static final String SIEVE_RESULT_METHOD = "post_sieved_primes";

    private static final String LUCAS_LEHMER_TASK_IDENTIFIER = "ll";
    private static final String SIEVE_TASK_IDENTIFIER = "sieve";

    // TODO - Should these be read from applet parameters in the HTML page?
    private static final long POLL_INTERVAL = 15000;
    private static final long UPDATE_THRESHOLD = 60000;

    private XmlRpcClient client;
    private boolean running;
    private String jobKey;
    private long lastUpdate;

    private Thread nextTask;
    private Sieve sieve;
    private LucasLehmer lucasLehmer;
    private Thread update;

    @Override
    public void init() {
        try {
            XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
            java.awt.EventQueue.invokeAndWait(new Runnable() {
               @Override
               public void run() {
                  initComponents();
               }
            });
            config.setServerURL(new URL(SERVICE_URL));
            client = new XmlRpcClient();
            client.setConfig(config);
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
```

```
}

@Override
public void start() {
    super.start();
    running = true;
    next();
}

private void next() {
    nextTask = new Thread() {
        @Override
        public void run() {
            boolean assigned = false;
            while (running && !assigned) {
                try {
                    // TODO - Verify reqwork service method takes no params.
                    Object[] params = new Object[] {};
                    Object response;
                    taskField.setText(SLEEP_TASK);
                    Thread.sleep(POLL_INTERVAL);
                    taskField.setText(String.format(
                            COMMUNICATION_TASK_PATTERN, SERVICE_URL));
                    response = client.execute(GET_WORK_METHOD, params);
                    assigned = dispatch(response);
                }
                catch (InterruptedException e) {
                    // Thread has been awakened - probably because applet is
                    // being stopped. No need to take corrective action.
                }
                catch (XmlRpcClientException e) {
                    // Communication with server failed.
                    e.printStackTrace();
                    // TODO - Determine and implement appropriate action,
                    // if different from a simple retry.
                }
                catch (XmlRpcException e) {
                    // An error occurred on the server.
                    e.printStackTrace();
                    // TODO - Determine and implement appropriate action,
                    // if different from a simple retry.
                }
                catch (Exception e) {
                    // Some other error occurred; dump to the Java console.
                    e.printStackTrace();
                    // TODO - Determine and implement appropriate action,
                    // if different from a simple retry.
                }
            }
        }
    };
    nextTask.start();
}

private boolean dispatch(Object taskInfo) {
    boolean assigned = false;
    try {
        Object[] details = (Object[]) taskInfo;
        String task = (String) details[0];
        if (task.equalsIgnoreCase(LUCAS_LEHMER_TASK_IDENTIFIER)) {
            Object[] params = (Object[]) details[1];
            int seed = (Integer) params[0];
            int initialIterations = (Integer) params[1];
            BigInteger initialResidue = (initialIterations != 0) ?
                new BigInteger((byte[]) params[2]) : null;
            jobKey = (initialIterations != 0) ?
```

```
            (String) params[3] : (String) params[2];
        lucasLehmer = (initialIterations == 0) ? new LucasLehmer(seed) :
            new LucasLehmer(seed, initialIterations, initialResidue);
        lucasLehmer.addListener(this);
        lastUpdate = new Date().getTime();
        lucasLehmer.start();
        taskField.setText(String.format(LUCAS_LEHMER_TASK_PATTERN, seed));
        assigned = true;
    }
    else if (task.equalsIgnoreCase(SIEVE_TASK_IDENTIFIER)) {
        Object[] params = (Object[]) details[1];
        int lowerBound = (Integer) params[0];
        int upperBound = (Integer) params[1];
        jobKey = (String) params[2];
        sieve = new Sieve(lowerBound, upperBound);
        sieve.addListener(this);
        sieve.start();
        taskField.setText(
            String.format(SIEVE_TASK_PATTERN, lowerBound, upperBound));
        assigned = true;
    }
    else {
        // Whatever other task was sent to us, we don't know how to do
        // it, so we'll just ignore it.
    }
}
catch (ClassCastException e) {
    // We didn't get an array of objects, or the first item wasn't a
    // String, etc. In any event, all we can do is ignore the task.
}
catch (Exception e) {
    // Something else bad happened.
}
```

```
    return assigned;
}

@Override
public void stop() {
    super.stop();
    running = false;
    if ((nextTask != null) && nextTask.isAlive()) {
        nextTask.interrupt();
    }
    if ((sieve != null) && sieve.isAlive()) {
        sieve.interrupt();
    }
    if ((lucasLehmer != null) && lucasLehmer.isAlive()) {
        lucasLehmer.interrupt();
    }
    if ((update != null) && update.isAlive()) {
        update.interrupt();
    }
}

@Override
public void notifyState(final int seed, final int iterations,
        final BigInteger residue) {
    if ((iterations < seed - 2)
            && ((new Date().getTime() - lastUpdate) > UPDATE_THRESHOLD)) {
        if ((update != null) && update.isAlive()) {
            update.interrupt();
        }
        update = new Thread() {
            @Override
            public void run() {
                Object[] params = new Object[]
        {seed, iterations, residue.toByteArray(), jobKey};
String residueDisplay = "";
if (residue.bitLength() > 50) {
   int digits = 0;
   residueDisplay = residue.toString();
   digits = residueDisplay.length();
   residueDisplay = String.format(
         CONDENSED_RESIDUE_PATTERN,
         residueDisplay.substring(0, 6),
         residueDisplay.substring(digits - 6),
         digits);
}
else {
   residueDisplay = residue.toString();
}
historyField.append(
      String.format(LUCAS_LEHMER_STATUS_PATTERN,
      seed, iterations, residueDisplay));
try {
   client.execute(LUCAS_LEHMER_STATUS_METHOD, params);
   lastUpdate = new Date().getTime();
}
catch (XmlRpcClientException e) {
   // Connection with the server failed.
   e.printStackTrace();
   // TODO - Determine and implement appropriate action,
   // if different from a simple retry.
}
catch (XmlRpcException e) {
   // Server processing error occured.
   e.printStackTrace();
   // TODO - Determine and implement appropriate action,
   // if different from a simple retry.
```

```
                }
                catch (Exception e) {
                    // Some other error occurred.
                    e.printStackTrace();
                    // TODO - Determine and implement appropriate action,
                    // if different from a simple retry.
                }
            }
        };
        update.start();
    }
}

@Override
public void notifyPrime(int seed, boolean isPrime) {
    Object[] params = new Object[] {seed, isPrime, jobKey};
    boolean retry = true;
    historyField.append(String.format(LUCAS_LEHMER_RESULT_PATTERN, seed,
            isPrime ? "prime" : "composite"));
    while (retry) { // TODO - Should there be a maximum # of retries?
        try {
            Object result =
                    client.execute(LUCAS_LEHMER_RESULT_METHOD, params);
            // TODO - Should result be checked?
            retry = false;
        }
        catch (XmlRpcClientException e) {
            // Connection with the server failed.
            e.printStackTrace();
        }
        catch (XmlRpcException e) {
            // Server processing error occured.
            e.printStackTrace();
            // TODO - Determine and implement appropriate action.
        }
        catch (Exception e) {
            // Some other error occurred.
            e.printStackTrace();
            // TODO - Determine and implement appropriate action.
        }
    }
    next();
}

@Override
public void notifyPrimes(int lower, int upper, BitSet primes) {
    Object[] params = new Object[]
            {lower, upper, Utility.bitSetToByteArray(primes), jobKey};
    boolean retry = true;
    historyField.append(String.format(SIEVE_RESULT_PATTERN,
            primes.cardinality(), lower, upper,
            Utility.bitSetToString(primes, lower)));
    while (retry) { // TODO - Should there be a maximum # of retries?
        try {
            Object result =
                    client.execute(SIEVE_RESULT_METHOD, params);
            // TODO - Should result be checked?
                retry = false;
        }
        catch (XmlRpcClientException e) {
            // Connection with the server failed.
            e.printStackTrace();
        }
        catch (XmlRpcException e) {
            // Server processing error occured.
            e.printStackTrace();
            // TODO - Determine and implement appropriate action.
        }
        catch (Exception e) {
            // Some other error occurred.
            e.printStackTrace();
            // TODO - Determine and implement appropriate action.
        }
    }
    next();
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    taskLabel = new javax.swing.JLabel();
    historyLabel = new javax.swing.JLabel();
    jScrollPane1 = new javax.swing.JScrollPane();
    historyField = new javax.swing.JTextArea();
    taskField = new javax.swing.JTextField();

    taskLabel.setText("Current Task");

    historyLabel.setText("History");

    historyField.setColumns(20);
    historyField.setEditable(false);
    historyField.setRows(5);
    historyField.setFocusable(false);
    jScrollPane1.setViewportView(historyField);

    taskField.setEditable(false);
    taskField.setText("(none)");
```

```
        taskField.setFocusable(false);

        org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jScrollPane1, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 380,
Short.MAX_VALUE)
                    .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING, false)
                        .add(org.jdesktop.layout.GroupLayout.LEADING, layout.createSequentialGroup()
                            .add(taskLabel)
                            .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                            .add(taskField, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 314,
Short.MAX_VALUE))
                        .add(org.jdesktop.layout.GroupLayout.LEADING, historyLabel)))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(taskLabel)
                    .add(taskField, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
                .add(historyLabel)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jScrollPane1, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 202,
Short.MAX_VALUE)
```

```
                .addContainerGap())
        );
    }// </editor-fold>


    // Variables declaration - do not modify
    private javax.swing.JTextArea historyField;
    private javax.swing.JLabel historyLabel;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField taskField;
    private javax.swing.JLabel taskLabel;
    // End of variables declaration

}
```