



Distributed computation of π

by

Christian Kroken, Sigbjørn Tvedt

Supervisor: Morten Goodwin Olsen

Project report for distributed systems in spring 2007

based on report template version 3.0 (2006)

Agder University College
Faculty of Engineering and Science

Grimstad, 14 May 2007

Status: Final

Keywords: Pi, Distributed system, Distributed computing

Abstract:

This report contains the result from an attempt of creating a distributed system for computing the digits of Pi. As this project has been given in the course distributed systems, we are keeping our focus on the distributed system instead of the mathematics that is used to compute Pi.

After completing this project, we have created a framework that is able to detect most errors introduced by hardware and transmission errors. It still has some room for improvement, but it is possible to use it to do a distributed computation of pi.

Table of Contents

1	Introduction.....	3
1.1	Report outline.....	3
2	Problem description.....	4
3	Background.....	5
4	Solution.....	6
4.1	Requirements.....	6
4.2	Design Specification.....	7
4.3	Implementation.....	9
4.4	Validation and Testing.....	11
5	Discussion.....	12
5.1	Future work.....	12
6	Conclusion.....	14
	Appendices.....	15
	Appendix 1 Glossary & Abbreviations.....	15
	References.....	15

1 Introduction

This is a project suggested by Morten G. Olsen where we have been given a formula for computing any given digit of π ^[2], and are going to create a distributed system that should be able to compute any number of digits. As the goal of this course is to learn how to create and work with distributed systems, we have decided to use a pre programmed version of a formula^[1] to compute pi, while maintaining focus on the creation of a flexible and robust framework that should be able to cope with most errors encountered.

As we have most experience when it comes to working with C#, we selected C# as our main development language. We also decided to keep the algorithm^[1] as a separate C++ program to ensure as high efficiency as possible while it is easy to swap it with another algorithm if we want to do another distributed computation.

We have decided to implement the client as a screensaver that could run when the host computer is idle. This client should also be able to withstand most attempts of sabotage and random errors. Another goal we were aiming towards was that the server should do as little work as possible. The completed server code only has a few tasks, like checking control numbers, and database access.

1.1 Report outline

2. Problem description

Explains the problem itself and gives a description of the main issues we will have to handle to ensure a good result.

3. Background

Gives a background of the main issues in the project. Explains how these issues have been handled in the past and a brief outline of how we handle the problem.

4. Solution

Shows the requirements for the different parts of our project. Gives a model of the implementation and provides an in-depth explanation of how we solved this. Also shows how we validated and tested the implementation to make sure it worked.

5. Discussion

Discusses our implementation and possible changes we could have made. Also future work that can be done on the project, so it can be improved.

6. Conclusion

A final note on the status of the project and what we achieved. Also has any possible limitations of the implementation.

2 Problem description

We have been given an algorithm for computing any decimal of pi, and we are going to create a system to spread the computation over several ordinary computers in order to compute the digits faster than one large computer.

The fact that we are moving away from single trusted computer, to a network of unknown and thus untrustworthy clients introduces some extra problems. These problems could be hardware faults, transmission errors, clients that return wrong digits on purpose or random errors that occur only once.

The easiest way of validating these results are of course, to run the computation 2 or 3 times on each digit, and compare the results, but it is also highly inefficient. By knowing this, it could be interesting to explore methods for eliminating clients that are trying to “cheat” by reporting wrong digits while not having to compute all digits several times. This could possibly be done by recomputing parts of submitted digits or by recomputing all digits in some of the series.

We also want to create a server code that uses as few resources as possible, as this would give us the advantage of using a cheap server to care for a large amount of clients.

3 Background

The computation of the decimals of Pi has been attempted since ancient times, but during the last 40 years, the increased computing power available have enabled us to compute Pi with several million digits. At first, all formulas demanded that all digits, up to the one we were trying to find, had to be known in order to compute the next digit. This, however, changed during the 90s when researchers that where working with elliptic curves discovered that theories from this field could be used to calculate any digit of Pi^[4]. This discovery enabled the creation of several distributed systems for computing Pi, like the old PiHex^[5] and the ongoing Pi Segment^[6] project, and different adaptations have been made to the formula^[1].

Using a distributed system to calculate this formula adds another issue, making sure that the calculation on each client is carried out without errors. This adds the need to implement some method of determining whether the client has calculated the correct value of Pi or not. One must also make sure that the value has actually reached the server i.e. transmission faults.

Our outline of solving the problem is to:

- Clients calculate Pi and send to a server
- Clients validate results from other clients

4 Solution

4.1 Requirements

The Client

- Compute the value of PI
- Transfer results back to the server
- Create control digits

The Server

- (re)assign ranges of digits that the clients should compute
- Receive computed digits from the clients and store them.
- Make the different clients check the results supplied by other clients
- Low load / network traffic
- Check control digits

4.2 Design Specification

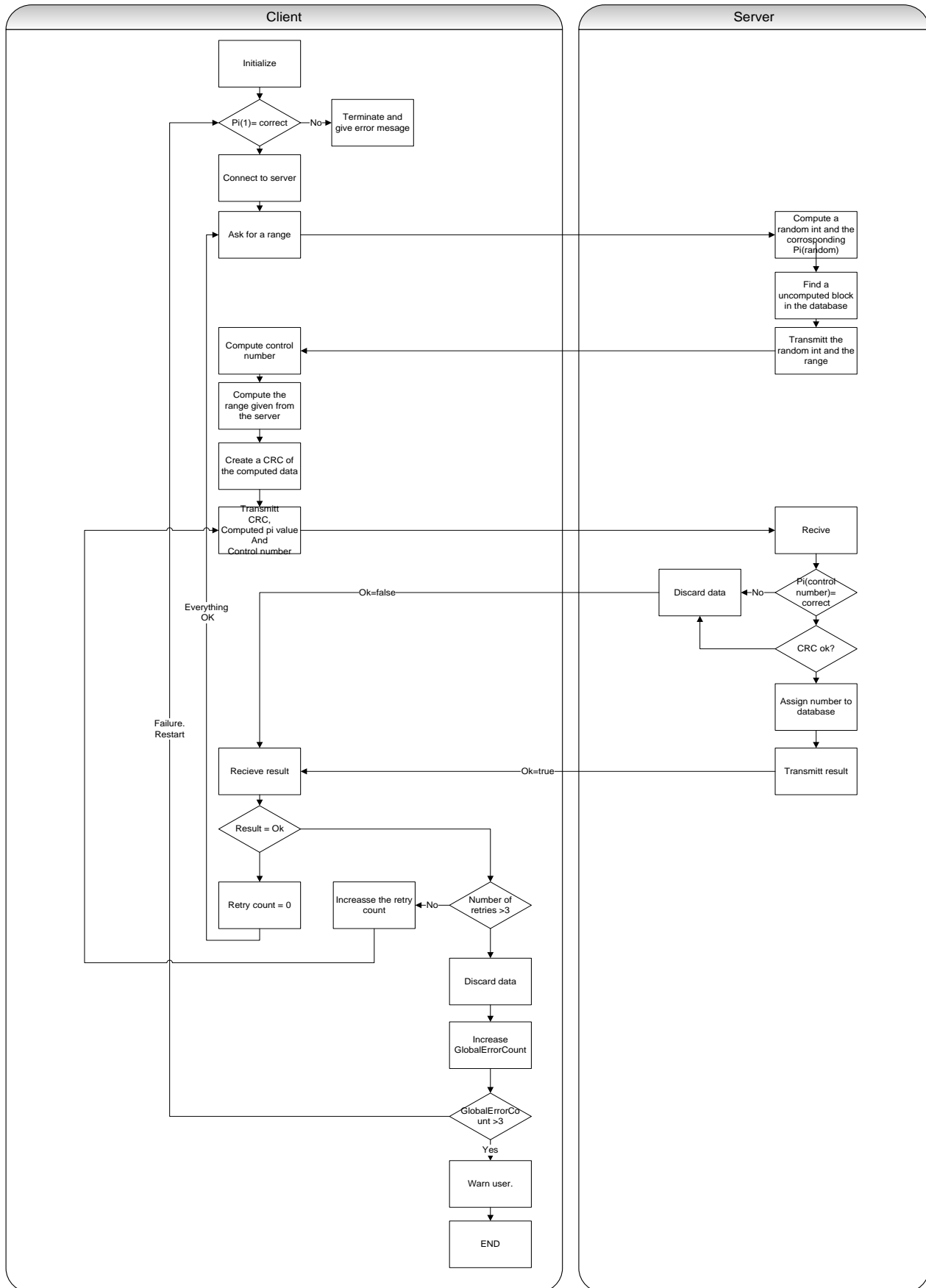
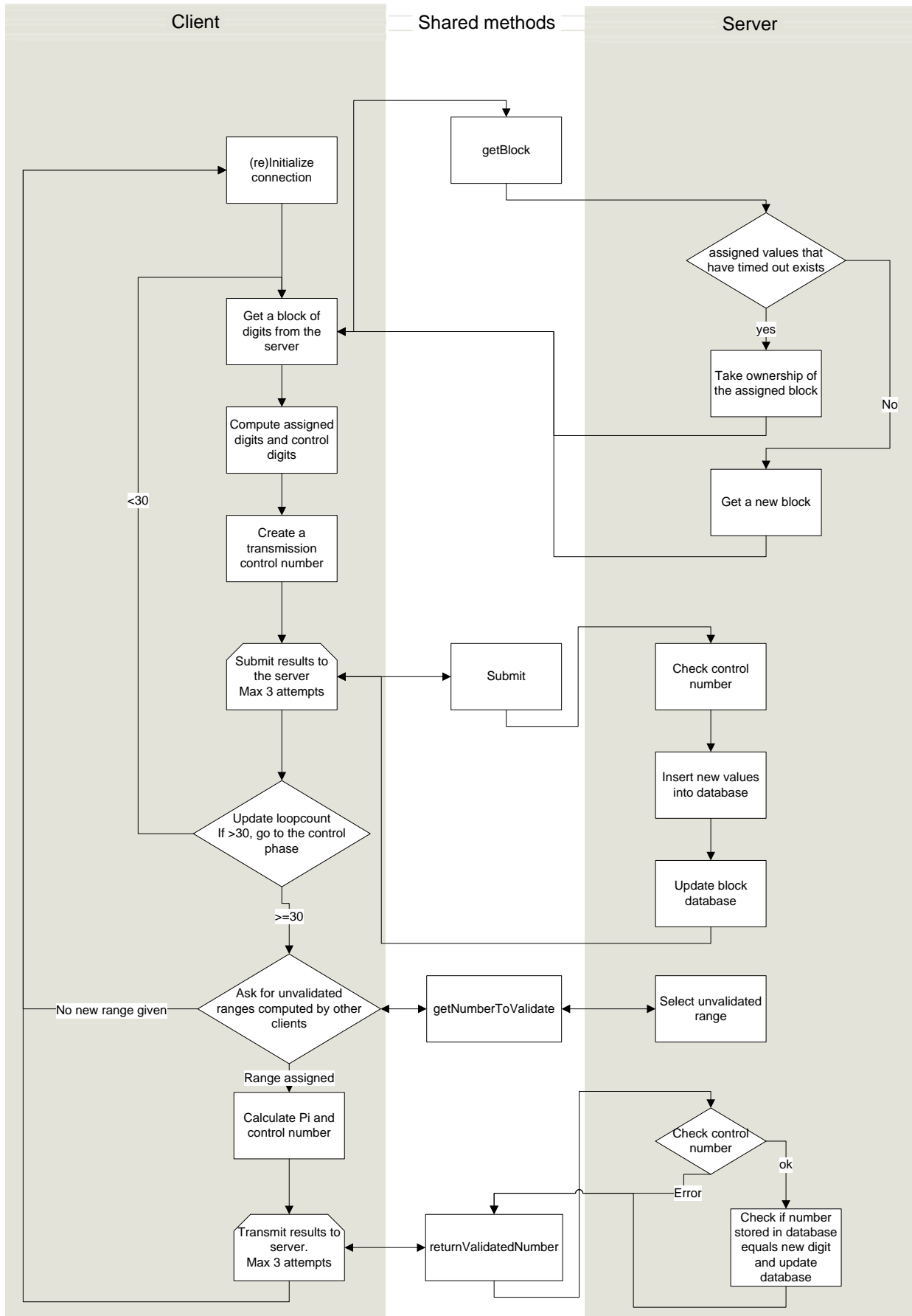


Figure 1: Initial plan for implementing the algorithm

We would like to create an implementation of the system in C#.net using an ms.sql database. The system will be divided into 3 parts; A Client that is responsible for doing most of the work, A server that communicates with the clients and the database, and a database that maintain control over all computed digits as well as what client that is responsible for computing the different values of Pi. The testing of the results will be done manually by comparing the digits stored at the server with digits stored at a repository on the internet^[3].

4.3 Implementation



*Figure 2: Implementation of the system***Client**

The client could be divided into two parts, computing new digits and checking if the already computed digits are correct.

As the primary goal is to calculate as many digits as possible, most of the time will be used to compute new digits.

The first step here is to ask the server for a range that should be computed, in our case 180 digits, and a control range. It then computes the values of the two ranges the server assigned to it, and creates a control number that is used to verify that the values have been transmitted without errors.

After the client has completed the computation of Pi, the control block and control digits, it submits the data to the server.

After the server has accepted the submission, or refused it 3 times, the client has two choices; it usually asks the server for a new digit, and repeats the process, but on a regular interval, it will continue into part two.

Part two is the validation of results from other clients. The client starts by asking the server for a block of digits that another user has computed but none of the other clients have validated yet. If no unverified blocks are found, or all unverified blocks have been computed by the same client, the client reinitializes itself, and restarts. If there are unverified blocks computed by other clients, the client computes the value and a transfer control number. These 3 numbers are then transmitted back to the server

Server

The server maintains a repository containing the computed values of pi and is responsible for maintaining control over which clients that are working on the different digits. It also has to decide if the data submitted by the client should be treated as genuine, or if it should ask the client to attempt a retransmission

Control value

When a client wants to return digits to the server, it has to perform some checks. One of them is a functionality test. This test is to compute 9 digits of pi, selected from the first few digits. These digits are quite fast to work with, and usually only add 0.5 seconds of extra computation time. These values are known to the server, and are used to validate all data that are returned to the server.

Control number

Each time a client submits data to the server, it has to create a control number. This is done to ensure that all communication from the client to the server remains intact.

To create the transmission number, we add all digits in the computed Pi string and all digits in the control value digit by digit (*a*). We then add the position of the first digit the client have computed and the first position of the control value (*b*). Finally we perform the operation (*a*) mod (*b*) to get the control number.

Example algorithm for computing the control number:

```
ControlNumber(pi, cValue, p_0, p_1)
{
    //Loop over all digits in pi
    for (i = 0; i < digits in pi; i++)
    {
        total += pi[i];
    }
    //Loop over all digits in control value
    for (int i = 0; i < digits in cValue; i++)
    {
        total += cValue[i];
    }
}
```

```
    }  
    // total modulus (p_0+ p_1)  
    total %= (p_0 + p_1);  
    return total;  
}
```

4.4 Validation and Testing

The easiest way of testing the results are to start two clients and run them. It is then possible to compare the results and previously computed digits that are available online^[3].

As the system should be able to cope with errors, we have also tried to introduce random errors, and found that it usually is possible to detect and remove these. As some of the digits submitted to the database are checked by other clients, it is also possible to detect clients that often transmit data containing errors, thus making it easier to get a correct result.

5 Discussion

We were able to create a system that works as intended in a controlled environment. We were also able to simulate network errors and faulty client algorithm implementations and successfully detected these errors. Random faults in the algorithm are usually not detected, and are thus accepted by the server. If a client wants to sabotage, the saboteur has to know about the internal messages and checks in the software. If a client were to have an invalid algorithm, all data returned to the server would simply be discarded. The internal checks also secure the system when it comes to clients modifying the tcp packets, as the checksum will be wrong. If a client were to break these checks, the client should be detected by the other clients performing the digit validation. This is since they are probably going to return wrong digits all the time.

When it comes to random bit faults during client processing, we do not have any good solutions for detecting these problems, but it is possible to detect some of these faults when the client performs the checking of the results submitted by the other clients, but as only a few of the submitted digits are checked, such errors will probably not be detected before a full rerun is performed.

When it comes to the number of digits each client is assigned to compute each time, there are advantages for using both large and small numbers.

Large amount of digits:

- Reduced amount of overhead traffic
- Fewer control numbers have to be computed and checked

Small amount of digits:

- New digits are returned to the database quicker
- Fewer digits have to be recomputed if errors occur
- Clients that have unstable connections will be able to return more results
- It will be easier for slow clients to return the results before timeout is reached

On a high speed network, small blocks could be the best solution, as the primary restriction here are the server and if it is able to perform the database operations fast enough. On a network with limited bandwidth, we should use large blocks of digits, thus reducing the percentage of each package that are used for error control.

The validation of digits computed by other clients could be omitted since it introduces extra computation, and does not validate all digits. We still want to keep it since it does validate some of the digits. These digits may be used to identify clients that are trying to destroy the computation by transmitting wrong results. When it is possible to remove these clients, it is possible to recompute all digits calculated by them, thus increasing the chances of the final result being correct.

Another way of detecting errors could be to calculate parts of each series, and hope that the rest of the digits are correct. When comparing this to the method we are using, this seems to be a more efficient way to create a quick check, but it does not remove random errors that may occur as a part of the unchecked part of the digit. When recomputing the whole series to check that all digits are correct, it is also necessary to compute all digits one more time while the solution we have selected eliminates some of the ranges since they have already been computed, thus reducing the time required to perform the validation of all digits.

5.1 Future work

Before this project is ready to be deployed, there is some more work that should be done

- User administration:

We need some kind of user account administration, and also some way of identifying different computers belonging to the same user.

- Checkpoints:
Since the client has been implemented as a screensaver, we have no guarantee for the computation to complete before the user starts working with the computer. We should therefore alter the software so that it creates checkpoints containing the digits that have been computed and store them on a non volatile storage space. By doing this, we may also increase the amount of digits assigned to each client each time it communicates with the server.
- Website/statistics
In order to get people to donate their CPU cycles to us, they need to feel that this is some kind of competition. By creating a way of them to monitor their progress, and compare themselves versus other users, we should get more people to use this client.
- Improved multithreading
The client should be able to detect the number of CPU cores available for use in order to ensure maximum efficiency.
- Improved error detection
It should be possible to improve the error checking so that a few digits of all submissions are checked to verify that at least some part of the string is correct.
- Improved error correction
When an error has been identified, we should automate the process of assigning the digit to a third client. Depending on the result from this client, the software should decide if it is necessary to recompute the digit or not. It should also give a "warning" to the client that returned the wrong value so that it is possible to identify the computers that return wrong results.
- Database
The database should be moved to a better system than a MS sql express file. Tables should also be revised and merged with a user database.
- Server
It could be desirable to upgrade the system so that we have several servers that are responsible for communication between the clients and the server containing the database. This could be done to increase the security of the database, or because the servers are too slow. A scenario could be that we are using 4 servers. One is then responsible for the database, and the three other shares the load of communicating with the clients and computing checksums while relaying valid sql commands to the first server. If we where to run out of computing power, it should then be easy to expand the system with more frontend servers.

6 Conclusion

As this was an implementation problem, we have not touched the mathematics used to compute Pi and instead we have focused on the creation of a client/server.

The client/server we have created works as we expected. It is able to detect communication errors and permanent client errors. The system has been tested with up to 8000 digits in a controlled environment, and we could not find errors that were undetected. It is however still possible to “cheat”, but the user has to know about the internal control messages to be able to do any harm.

The implementation is not yet mature enough for deployment in a public setting as more features should be added to ensure correct operation.

Appendices

Appendix 1 Glossary & Abbreviations

Short definitions of terms used and references to further relevant glossary sources

References

- [1] <http://fabrice.bellard.free.fr/pi/pi.c>
- [2] http://fabrice.bellard.free.fr/pi/pi_n2/pi_n2.html
- [3] <http://www.joyofpi.com/pi.html>
- [4] <http://www.cs.uwaterloo.ca/~alopez-o/math-faq/node38.html>
- [5] <http://oldweb.cecm.sfu.ca/projects/pihex/>
- [6] <http://pisegment.vicp.net/>