Parallel Numerical Integration on a P2P/Grid Computing Environment

Jun Ni\textsuperscript{1,2}, Tao He\textsuperscript{1,2}, Joe Demaria\textsuperscript{3}, Shaowen Wang\textsuperscript{1}, and Boyd Knosp\textsuperscript{1}

\textsuperscript{1}Academic Technologies-Research Services, ITS
\textsuperscript{2}Department of Computer Science
\textsuperscript{3}Kirkwood Community College, Iowa City, IA

The University of Iowa, Iowa City, IA 52242
\{jun-ni, tao-he, shaowen-wang, boyd-knosp\}@uiowa.edu

\textsuperscript{3}Kirkwood Community College, Iowa City, IA
jdemari@kirkwood.edu

Abstract

This paper presents a preliminary study of using MCMS-powered P2P technology to conduct parallel integrations which has been widely used in applied mathematics, sciences and engineering. In this study, we designed a system which allow migrate numerical integrations in a P2P/Grid based environment. The system’s client responses for a job scheduling, domain decomposition, and a data communications, while the system’ servers response for computations. The detailed proposed parallelism, system configuration, implementation and programming are presented. This study serves as the initiative of using distributed computing technology for solving large-scale scientific computing problems.

1. Introduction

Numerical integration is one of the major numerical methods for scientists and engineers [1-2]. It can be used to numerically solve many mathematical problems in science and engineering, such as in the calculation of values of surface and volume in finite element method (FEM) [3], in the computation of ordinary differential equations (ODEs) or partial differential equations (PDEs) etc al. There are many resources of numerical integration publicly available [4-7]. Due to the its characteristics of integration that an integration can be divided into several sub-integration and accumulate their results together, the numerical integration is perfect to be decompose and parallelized for implementation on a parallel computing system [8-9], since the communication between mater node and working (computational nodes) is very small. The integration can be decomposed to number of sub integrations by passing the two sub bounded values to each computational node, while the final results of each sub-integration are sent back to master node for final summation. Therefore, the numerical integration is often selected as the perfect example for parallelization. The parallelism can be implemented using various parallel computing techniques such as message passing, especially the available parallel utility library such as Message Passing Interface (MPI) [9].

With the rapid development of networking, scientists and researchers begin to use high-end communication technologies to deal with intensive computation in sciences and engineering. The increased globularity of IT technologies encourages people to develop virtually-organized, large-scale, heterogeneous distributed systems called Grid computing. Such system integrate large amount of computational resources to be used for deal with single computational job. It sounds very promising. However, the major discouragement is communication lost which defined as the wasted time during data transfer, and idle time due to data synchronization and other disadvantages. All the shortcomings finally impacts the computational efficiency and cause extra efforts in careful considerations and designs of algorithms for each applications. Therefore, the network impacts on computational performance should be studied before a computation task is migrated to a Grid computing system [10-12].
In this paper, we report how we can implement numerical integration on P2P-based system for conduct parallel integration on the Internet; enhance to study the network influences on performance of parallel computations in a Grid computing system.

Section 2 provides basic model of numerical decomposition integration on P2P/Grid-based system. Section 3 briefly describes the P2P system based on multi-client/multi-server (MCMS) technology. Section 4 presents the performance in terms of speedup vs. the number of processors and efficiency of high performance computing, as well as the network impacts. Followed by, a conclusion and future work will be given.

2. P2P/Grid-based Parallel Integration Algorithm

Newton-Cote numerical algorithm for an integration of a function can be described as

\[ I_j = \int_{a_j}^{b_j} f(x)dx = \sum_{i=1}^{n} w_i f(x_i) + R_n \]

The variables \( a \) and \( b \) are the low and upper limits. \( R_n \) stands for the remainder associated with the numerical integration. The function \( f(x) \) is an integrating function and variable \( w_i \) (i=1,2,3,...n) is a weight coefficient, which depends on the repeated number and equal-spaced discretization of sample points \( (x_i) \). If sample points can be selected as 1, 2, or 3, which respectively represent rectangular, trapezoid, and Simpson’s rules of Newton-Cote numerical integration. Based on the characteristics of integration, the whole integration can calculated as the summation of sub-integrations before applying Newton Cole method. Therefore, one can decompose whole region \( [a, b] \) into multiple sub-regions \( [a_j, b_j] \) \( (j=1,2,3,...N) \), and each sub-region’s integration is subject to compute on individual computational node (hereby it is performed on computational server in P2P/Grid system). After each server node calculates the sub-integration and it returns the sub-integration result back to master (hereby it is called client node in P2P/Grid system) for summation. The whole parallel algorithm can be expressed as

\[ I = \sum_{j=1}^{N} I_j \]

\[ I_j = \int_{a_j}^{b_j} f(x)dx = \sum_{i=1}^{n} w_i f(x_i) + R_n \]

\[ \Delta x_j = \frac{b_j - a_j}{n_j}, x_j = a_j + \Delta x_j \]

where the \( j \) is the index of sub task \( (j=1,2,3,...N) \) and \( N \) stands for number of tasks, which can be taken as number of computational nodes or server nodes. Once \( a, b \) and \( n_{global} \) are given, \( a_j, b_j, \) and \( n_{local} \), are can be calculated and named as the local (sub-regional) low and upper limits, and repeated number for each sub area during server’s numerical computation. Since \( N \) is the total number of sub-regions, it can be selected as the number of servers participated in computation. The total number of repeated number can be calculated as \( n_j = n_{global}/N \), which should be used for sequential computation.

\[ a_j = a + \frac{b - a}{N} \cdot j \]

\[ b_j = a_j + \frac{b - a}{N} \]

For each sub-integral, the client transfers three values \( (a_j, b_j, \) and \( n_j) \) through the Internet to servers. After each server calculates its assigned sub-integral, it returns the value to client. The client accumulates the sub-values from servers and comes up the total value of integration. The result is also compared with sequential computing performed on client along.

3. P2P-based System Development and Configuration

The P2P system was developed based on the distributed client-server model. However, most of the client-server model being used is multi-client and single server [14]. In other words, clients are subject to initiate computational jobs being sent to server. A single server responds the client requisition and conduct intensive computations, and then send the computational results back to a client or multi clients. In traditional P2P model, many geographically distributed systems (PCs or workstations) are voluntarily participating P2P computation as a peer node. The client can submit their pre-prepared
computational jobs over and accomplish their computing tasks. This model has been used in SITE project. However, the client and server system is not symmetric. In other words, each peer node can only contribute their computing resources. They do not have control to initiate and submit jobs. In our test-bed, we developed a symmetric multi-client and multi-server model. The model allows integrate client and server software modules together for achieving symmetric systems. Once the system starts, it can serves as a server for operating intensive computation, and/or a client for submit computational jobs performed on other systems. Such system can be used to conduct parallel computing on Internet. The symmetry of the P2P-based MCMS system is illustrated as

![Figure 1. MCMS system for parallel computing](image)

We developed software called HPCBox. The software is programmed in Java due to the consideration of platform independency. The software were installed on Window-based PCs, LINUX based workstations, or OSX-based Macs. In our test-bed of P2P-envrinbment, we selected 16 heterogeneous legacy computers to participate the performance study. The software first scans available nodes from the pool of participated computers. Since each participated node installed and operates the software, the client node can screen and select nodes through either static or dynamic TCP/IP addresses. For example, following table shows the selected computational nodes as server computational nodes. The client system is a VAIO 550, Sony laptop with 2.4 GHz, 1024MB memory, and 60 GB HD.

<table>
<thead>
<tr>
<th>TCP/IP</th>
<th>OS</th>
<th>CPU (MHz)</th>
<th>Memory (MB)</th>
<th>HD (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>218.255.163.92</td>
<td>MacOSX</td>
<td>350</td>
<td>128</td>
<td>12</td>
</tr>
<tr>
<td>218.255.163.105</td>
<td>MacOSX</td>
<td>350</td>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>218.255.163.106</td>
<td>MacOSX</td>
<td>350</td>
<td>128</td>
<td>12</td>
</tr>
<tr>
<td>218.255.163.219</td>
<td>MacOSX</td>
<td>350</td>
<td>128</td>
<td>12</td>
</tr>
<tr>
<td>218.255.163.227</td>
<td>MacOSX</td>
<td>350</td>
<td>128</td>
<td>12</td>
</tr>
<tr>
<td>218.255.163.228</td>
<td>MacOSX</td>
<td>350</td>
<td>256</td>
<td>6</td>
</tr>
<tr>
<td>218.255.163.237</td>
<td>MacOSX</td>
<td>350</td>
<td>128</td>
<td>12</td>
</tr>
<tr>
<td>218.255.163.16</td>
<td>MacOSX</td>
<td>350</td>
<td>128</td>
<td>12</td>
</tr>
</tbody>
</table>

The client scans, evaluates, and selects computational nodes from available pool. It also responses for decompose and send sub tasks to server nodes for computation. The system can also be used as a cluster or workstations, in which we used a 100-Bits/s Switch (3Com Super stack 3300x24 ports) to connect several legacy PCs.

The architecture of Multi-client and multi-server (MCMS) can be implemented using socket in Java. Each server contains sockets for receiving information from client, while the client contains scanning, selecting, evaluating computational property, and job decomposition, and job distributing, as well as job performance measurement. Each server node has a serverSocket which accepts socket communication from client and sockets on all computational nodes.

![Figure 2. The architecture of MCMS-powered P2P system for parallel integration](image)

The core Java component in client portion is listed in the following.

```java
... public class Client extends JFrame 
    implements ActionListener{ 
    ... 
    private int port=8000; 
    private int numProcesses=8; 
    private DataOutputStream osToServer[]=new 
    DataOutputStream(numProcesses); 
    private DataInputStream isFromServer[]=new 
    DataInputStream(numProcesses); 
```

Table 1. Result of selected computational nodes
private String ipAddress[] = new String[numProcesses];

public static void main(String[] args){
    new Client();
}

public Client(){
    for (int i=0; i<numProcesses; i++)
        ipAddress[i] = selectIP.selectedIP[i];
    try {
        Socket connectToServer[] = new Socket[numProcesses];
        for (int i=0; i<connectToServer.length; i++)
            connectToServer[i] = new Socket(ipAddress[i], port);
        // Create input streams to receive data from the servers
        for (int i=0; i<isFromServer.length; i++)
            isFromServer[i] = new DataInputStream(connectToServer[i].getInputStream());
        for (int i=0; i<osToServer.length; i++)
            osToServer[i] = new DataOutputStream(connectToServer[i].getOutputStream());
    } catch (IOException ex) {
        jta.append(ex.toString() + "problem found\n");
    }
}

public void actionPerformed(ActionEvent event){
    String actionCommand = event.getActionCommand();
    if (event.getSource() instanceof JButton) {
        if ("Compute" .equals(actionCommand) ) {
            try {
                global_a = Double.parseDouble(jtfA.getText().trim());
                global_b = Double.parseDouble(jtfB.getText().trim());
                local_n = Long.parseLong(jtfN.getText().trim());
                global_h = (global_b - global_a) / numProcesses;
                global_n = numProcesses * local_n;
                jta.append("Parallel computations on servers\n");
                time1 = System.currentTimeMillis();
                for (int i=0; i<isFromServer.length; i++)
                    local_a = global_a + i * global_h;
                    local_b = local_a + global_h;
                    osToServer[i].writeDouble(local_a);
                    osToServer[i].writeDouble(local_b);
                    osToServer[i].writeLong(local_n);
                    osToServer[i].flush();
                // Get sub-area from each server
                area = 0.0;
                for (int i=0; i<isFromServer.length; i++)
                    area += isFromServer[i].readDouble();
                time2 = System.currentTimeMillis();
                dtime = time2 - time1;
                jta.append("global low limit: " + global_a + "\n");
                jta.append("global upper limit: " + global_b + "\n");
                jta.append("Accumulated area calculated at different servers is " + area + "\n");
                jta.append("Total computation time: " + dtime + "\n");
                jta.append("Sequential computation on single client node\n");
                time1 = System.currentTimeMillis();
                area = integration(global_a, global_b, global_n, 1);
                time2 = System.currentTimeMillis();
                dtime = time2 - time1;
                jta.append("global low limit: " + global_a + "\n");
                jta.append("global upper limit: " + global_b + "\n");
                jta.append("Accumulated area calculated the client node is " + area + "\n");
                jta.append("Total computation time: " + dtime + "\n");
            } catch (IOException ex) {System.err.println(ex);}
        }
    }
}

public double integration(double a, double b, long n, int nncp){
    double local_area = 0;
    double h, br, xs;
    double[] weight = new double[5];
    newcot(weight, nncp);
    h = (b - a) / n;
    if (nncp >= 2) h = h / (nncp - 1);
    local_area = 0.0;
    for (int i = 0; i < n; i++)
        br = a + i * h;
        for (int j = 0; j < nncp; j++)
            xs = br + j * h;
            local_area += weight[j] * h * function(xs);
        return local_area;
}

static double function(double x){...}

public void newcot(double[] weight, int nnc){
    switch (nnc) {
        case 1: weight[0] = 1.0; break;
        case 2: weight[0] = 0.5; weight[1] = 0.5; break;
        case 3: weight[0] = 0.33333333333333;
                weight[1] = 1.33333333333333;
                weight[2] = 0.33333333333333; break;
        case 4: weight[0] = 0.375; weight[1] = 1.125;
        case 5: weight[0] = 0.31111111111111;
                weight[1] = 1.42222222222222;
                weight[2] = 0.53333333333333; break;
        default: break;
    }
}

The core Java component to establish server computational function is listed in the following...
public static void main(String[] args) {
    new MultiThreadServer();
}

public MultiThreadServer() {
    
    try {
        ServerSocket serverSocket = new ServerSocket(8000);
jta.append("MultiThreadServer started at " + new Date() + '\n');
        int clientNo = 1;
        while (true) {
            Socket connectToClient = serverSocket.accept();
jta.append("Starting thread for client " + clientNo + " at " + new Date() + '\n');
            InetAddress clientInetAddress = connectToClient.getInetAddress();
jta.append("Client " + clientNo + "'s host name is " + clientInetAddress.getHostName() + '\n');
            InetAddress clientInetAddress = connectToClient.getInetAddress();
jta.append("Client " + clientNo + "'s IP Address is " + clientInetAddress.getHostAddress() + '\n');
            // Create a new thread for the connection
            jta.append("HandleAClient thread - new HandleAClient(connectToClient); thread.start();
            clientNo++;
        }
    } catch (IOException ex) {System.err.println(ex);}
}

class HandleAClient extends Thread {
    private Socket connectToClient; // A connected socket
    public HandleAClient(Socket socket) {
        connectToClient = socket;
    }

    public void run() {
        long time1, time2, dttime;
        try {
            DataInputStream isFromClient = new DataInputStream(
                    connectToClient.getInputStream());
            DataOutputStream osToClient = new DataOutputStream(
                    connectToClient.getOutputStream());
            while (true) {
                time1 = System.currentTimeMillis(); double local_a = isFromClient.readDouble();
                double local_b = isFromClient.readDouble();
                long local_n = isFromClient.readLong();
                double area = integration(local_a, local_b, local_n, 1); osToClient.writeDouble(area);
                time2 = System.currentTimeMillis();
                dttime = time2 - time1;
                jta.append("Local_a received from client: " + local_a + '\n');
                jta.append("Local_b received from client: " + local_b + '\n');
                jta.append("Local_n received from client: " + local_n + '\n');
                jta.append("Sub-area is calculated as: " + area + '\n');
                jta.append("Time of computation is:" + dttime + '\n');
            }
        } catch (IOException e) {System.err.println(e);}
    }
}

public double integration(double a, double b, long n, int nnc) {
    Same as the client part}
static double function(double x) {
    Same as the client part}

public void newcot(double[] weight, int nnc) {
    Same as the client part}

3. Result of Performance of Parallel Integration

The computational results are displayed in the following. For more less intensive integrations with a low value of n (number of sub integrations), or example, when n has a value range from $10^4$, $10^5$, and $10^6$, the speedup is relative low. As the n increases, the performance increases and the corresponding speedup approaches to the linear speedup shown in Figure 3.

![Figure 3. Speedup vs. number of computational nodes with low n value during day time](image)

Figure 4 shows the result of performance when n takes a high value. When n has value of $10^8$, the performance of calculating its integration reaches to the linear speedup obeyed by the Amdahl’s law. When n takes a low value, say $10^6$, the performance is very low and the behavior is different in each task of computation. The performance is also affected by the time when the job is submitted. During the day time, the selected computers are more enthusiastic in operating other jobs and the network is very busy. The is why when n takes the same value ($10^7$) the speedup during day time is about 4 shown in Figure 3, while the
speedup during night time is about 7 shown in Figure 4.

![Performance of MCMS Distributed Computing](image)

Figure 4. The likelihood linear performance during night time

The corresponding efficiencies verses different n values can be found in Figure 5. The performance with n=100 millions has very high efficiency, as the number of computational nodes increases. If the n value decreases, the efficiency decreased dramatically.

![Efficiency vs. Number of Computational Nodes](image)

Figure 5. Efficiency vs. the number of computational nodes

In general, the proposed system can be used either on Internet with TCP/IP or on a cluster of workstations with local networking by a network switch. On the dedicated local network, and on a cluster, the performance is strongly depends on the network bandwidth and communications, since the network impacts the data communications, although parallel integration has very less data communications and synchronization. The lost of network communication is relatively small. Figure 6 demonstrates the difference between TCP/IP based system configuration and local network based communications. They all appear in a linear pattern and close to linear speedup. The higher network bandwidth, the better network performance one may have.

![Influence of Network](image)

Figure 6. Network impact on performance of parallel integration

4. Conclusion and Future Work

The P2P-based MCMS system can be considered as a feasible distributed system for parallel numerical integration. Its feasibility and scalability are very promising to many scientific fields, such as in large scale computer aided design in mechanical engineering, image reconstruction in medical processing of radiology etc. The scalability indicates that more peer servers, the better the performance, especially for intensive integrations on each server node. There are several important variables which should be observed during computation on the system. They are mainly (1) the ratio of individual communication time between each client and server and the server computation time for each task, (2) individual communication data transfer rate between client and server (dependent upon the simultaneous bandwidth on network), and (3) the ratio of global communication time vs. global computation time. A large physical domain can be decomposed into many small sub-domains. Each sub task for sub-domain is subject for server’s computation. The speedup is dependant upon the computational intensity. The more intensive computation on each server, the better performance it has. If a computation is less intensive or more communication between servers and client, it gains less benefit. Network strongly affects the final performance. Each computational node takes different period of time. Each global job takes various time to
accomplish; enhanced the total job time varies. The result is modestly frustrated due to the networking effects and the loadings of server nodes. Based on the study, we understood that network bandwidth is an important issue. Different time of job schedule is have different results in terms of total accomplished execution time. Statistical analysis of multi variables is need to evaluation the dynamic behaviors of networking to the Internet-based computing. The reliability of network is an important subject to study. In the future, we continue to study the impacts due to local network system and DNS-based network, as well as Internet. The statistical analysis of property behaviors such as FLOPS of each computing node, dynamic bandwidth, data communication times, load balance for a heterogeneous systems, etc will be studied.

Acknowledgements

The project is financially supported by internal funds at Academic Technologies-Research Services, Information Technology Services of the University of Iowa.

References