An Student Self-grading System

CS297 Report

By Chao Liang

Spring 2006

Advisor: Dr. Chris Pollett

Department of Computer Science

San Jose State University

Introduction:

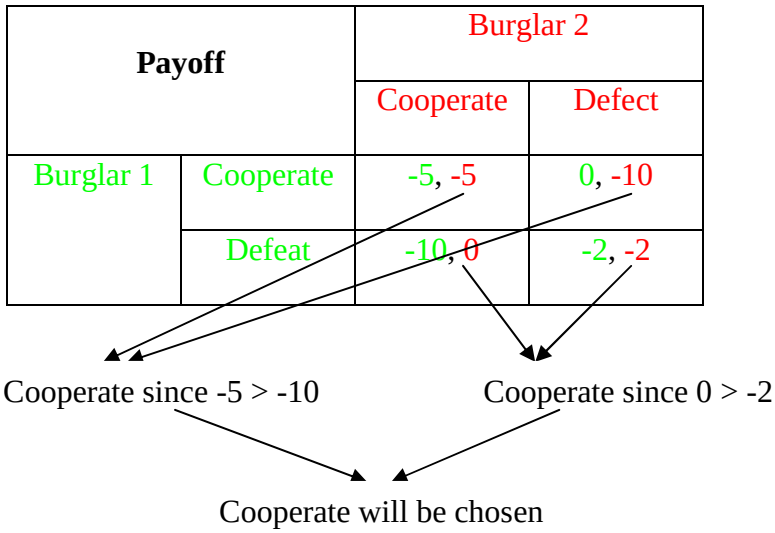
Homework submission in college classes is gradually shifting from paper, in-class hand-in to softcopy, on-line delivery. With heavy professor workloads and the common use of Internet homework submissions, it would be extremely useful to have a grading system in which students earn their grades by grading their peers online. Such a grading system not only allows teachers to focus more on teaching, but also helps students learn from each other. The goal of my project is to develop such a system. Ease of use is of paramount importance to my project, so extensive user testing will be done. Another goal of this project is to exploit ideas from game theory to ensure it is in the best interest of students to grade as accurately as they are capable of. This project will leverage off of Professor Pollett's existing homework system, but modify the homework return system to collect surveys about the materials covered and to collect grading forms. The proposed system will follow the guidelines of the Web Accessibility Initiative to ensure its usability. In addition, the system will also include a Wiki system for students to develop a canonically good homework solution. The system will allow students to view their grades distribution as well.

The following sections include an overview of game theory, three deliverables I have done in CS297, an algorithm for generating a total order of student paper, and my plans for CS298. The first section analyzes some basic concepts of game theory. The second section is a description of Deliverable 1, which is a script rewritten in PHP to retrieve book information from a MySQL database. The third section introduces Deliverable 2 that implements an Ajax progress bar to display uploaded size when files are uploading. An online voting system of Deliverable 3 in the fourth section is coded in such a way to

allow voters to vote according to Byzantine protocol. The fifth section is devoted to an algorithm to come up with a total order of all student papers from five-element partial order from each student. And the last section is my agenda for CS298, which concentrates on implementing student self-grading system by exploiting game theory.

Game Theory Overview:

Game theory studies strategies or actions that players choose to maximize their returns. It was founded by mathematician John von Neumann in 1928. On each stage of a game, each player could play a strategy in his/her own right, but the outcome depends not only on his/her own action but also on others' strategies. Therefore, even though each player tries to choose a strategy that provides the best payoff for him/her, the combinations of all players' strategies may not give each player the best payoff. The classic example of game theory: Prisoner's Dilemma illustrates the situation described above. Burglar 2 would choose to cooperate when burglar 1 cooperates. Burglar 2 would still choose to cooperate when burglar 1 defects. Therefore, burglar 2 would choose cooperate no matter what burglar 1 plays. It is a symmetric game, so burglar 1 would do the same as burglar 2 and choose to cooperate. Since both burglars cooperate, they would both serve in prison for five years, but they could be better off if they both choose defect.



Prisoner's Dilemma also introduces the concept of Nash equilibrium, which is the strategy set for all players such that no player can improve their payoff by switching to other strategies. {Cooperate, cooperate} is a Nash Equilibrium in this example, but it is not the most efficient strategy. If there is some communication among the game participants, it is possible for a player to choose a strategy that is not in Nash Equilibrium set, but yield a better payoff. However, there are some situations that no Nash Equilibrium exists. But if players do not choose a pure strategy but choose their actions over some probabilities, we may still deduce a Nash Equilibrium. In this case, it is called mixed Nash Equilibrium. Mixed Nash Equilibrium is an instance of mixed strategy. A mixed strategy α_i for player **i** is a probability distribution over his/her set of available actions. Each player will have n-tuple mixed strategy if there are n actions available for each player. Such n action vector is defined as $(\alpha_i^1, \alpha_i^2, \alpha_i^3, \ldots, \alpha_i^n)$, such that $\alpha_i^k \geq 0$, and $\sum \alpha_i^k$ (k=1..n) = 1. Prisoner's Dilemma is a two-person game, but in reality, it is often the case that a game involves more than two players. It is possible to transform a multi-player (more than 2) game to a 2-person game by singling out one player and forming a coalition from the rest of the players. In this way, we can represent such a game with a two-person game matrix. It is evident that student self-grading system can be viewed as a multi-player game. Each player (student) would choose a strategy (grading behavior) to maximize his/her payoff (reward for honesty grading behavior). The goal of my project is to design such a grading system that will only reward students who grade others' homework earnestly, but not those who give out effortless grades.

Deliverable 1: Rewrite query.jsp – a script manipulate query results, to PHP.

The goal of this deliverable is to integrate PHP, Apache and MySQL. The integration of PHP and Apache is done by modifying the configuration file of Apache – "httpd.conf" to load PHP as one of its module. To integrate PHP and MySQL, MySQL extension need to be enabled in PHP's configuration file named "php.ini". PHP provides a complete set of functions to talk to MySQL, and accessing MySQL database is through function calls. Following is the code snippets that demonstrate how to connect, query, and modify a MySQL database in PHP.

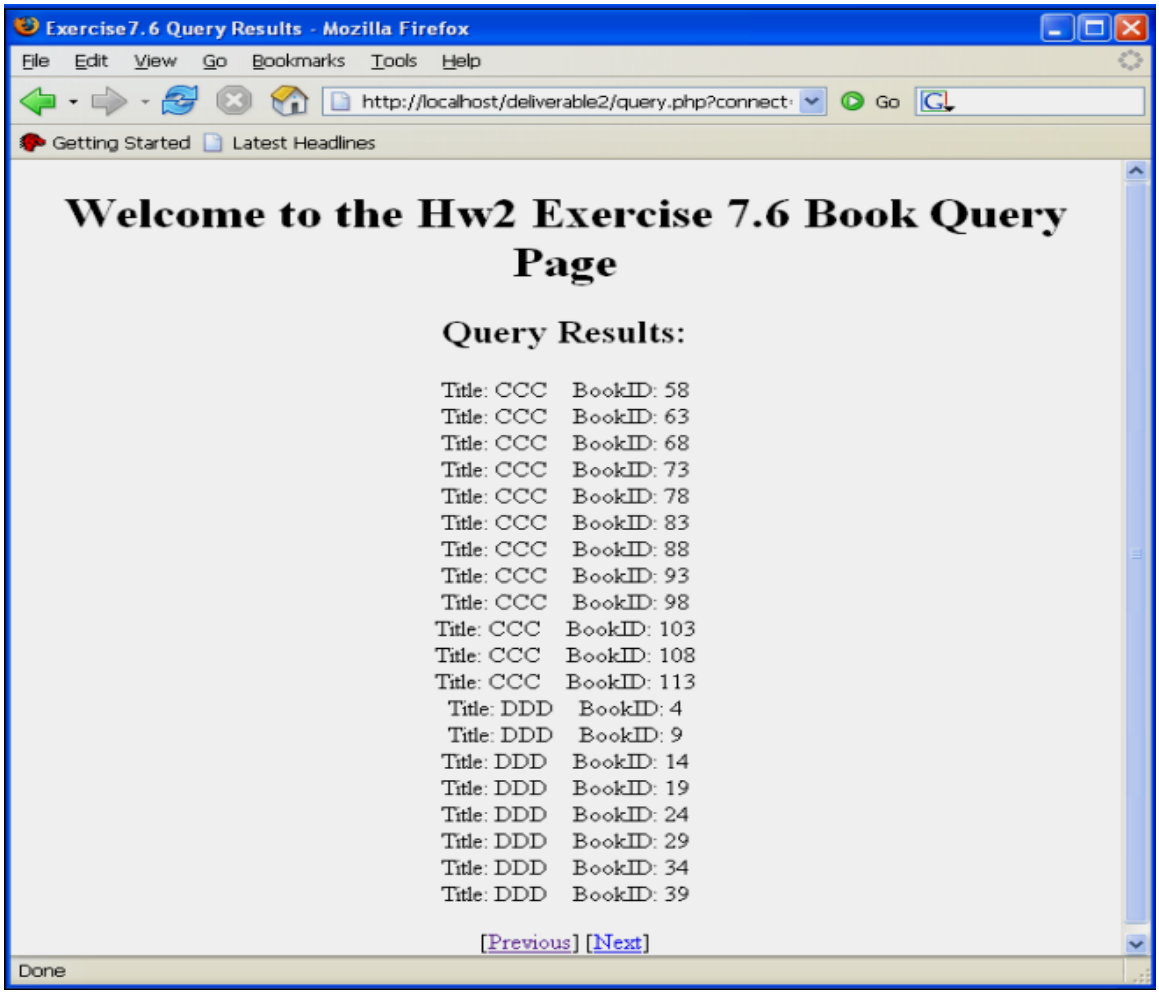
```
{
    $conn = mysql_connect($_REQUEST['connect'],$_REQUEST['login'],    $_REQUEST['password']);
    $query= "INSERT INTO  Book VALUES( 1, 'AAA', 'AAA Author',5.999)";
    mysql_select_db($_REQUEST['database'], $conn);
    $sql = "SELECT title, bookid FROM Book ORDER BY title, bookid";
    $result = mysql_query($query, $conn) or die(mysql_error());
    $result = mysql_query($sql, $conn) or die(mysql_error());
}
```

The programming syntax of PHP is similar to JSP in that we can embed PHP codes into HTML codes. But instead of using the HttpServlet objects, PHP provides some predefined global arrays that we can use to access server environment and global parameters. The server environment can be accessed through $_SERVER array, while user input parameters can be retrieved through $_REQUEST array. $_SESSION is an array that contains current session variables. Besides using super global, a pre-existed "phpinfo.php" can be used to find out all the information about a PHP setup. The following code snippet in PHP shows how to access the server environment and user input parameters.

```
if(isset($_REQUEST['set']) && $_REQUEST['set'] > 0)
 {
        echo "[<a href=\"http://" . $_SERVER['SERVER_NAME'] . $_SERVER['PHP_SELF'] . "?";
        echo ("connect=" . $_REQUEST['connect'] . "&");
        echo ("database=" . $_REQUEST['database'] . "&");
        echo ("login=" . $_REQUEST['login'] . "&");

        echo ("password=" . $_REQUEST['password'] . "&");
```

```
        echo ("set=" . ($_REQUEST['set'] - 1) . "&");
        for($i = 0; $i < $_REQUEST['set']; $i++)
        {
          $temp1 = "t" . $i;
          $temp2 = "b" . $i;
          echo ("t" . "$i=" . $_REQUEST[$temp1] . "&");
          echo ("b" . "$i=" . $_REQUEST[$temp2] . "&");
        }
        echo ("\" >Previous</a>) ");
}
```

This deliverable taught me how to configure a working server environment with PHP, MySQL and Apache. It also gave me the opportunity to have my first hands-on experience writing PHP codes. Therefore, I have better idea on how to pass information between PHP scripts and how to access MySQL database in a PHP page. Here is one of the screenshots that my “query.php” file outputs.



Deliverable 2: A progress bar in Ajax.

Ajax is an acronym for asynchronous JavaScript and XML. With Ajax, a browser can exchange a small amount of data with the server and continue code execution without waiting for responses. Ajax can be used at the front-end client side when PHP and MySQL can provide their supports at the back-end server side. But Ajax web application model is a little different from traditional web model. Under Ajax application model, JavaScript and HTML pages make call to Ajax engine, and Ajax engine in turn makes a HTTP request to the server. When data comes back, Ajax engine accepts the data, and delivers it to the web browser. In Ajax, HTTP requests are done through XMLHttpRequest objects. Once an XMLHttpRequest object is created, we can formulate and send a request by calling “open” and “send” methods of the object. When the browser gets a response from the server, it calls “onreadystatechange” event handler to process the response. However, the creation of XMLHttpRequest object varies between Internet Explorer (IE) and other browser vendors. The following code snippet illustrates how to make request and handle response through a cross-browser XMLHttpRequest object.

```
function createXMLHttpRequest()
{
        var transfer;
        if (window.ActiveXObject)
        {
           transfer = new ActiveXObject('Msxml2.XMLHTTP');
        }
        else if (window.XMLHttpRequest)
        {
           transfer = new XMLHttpRequest();
        }
        return transfer;
}

function requestInfo()
```

```
{
        var xmlHttpObject;
        xmlHttpObject = createXMLHttpRequest();
        xmlHttpObject.open("get", "progress.php?id=" + counter, true);
        xmlHttpObject.onreadystatechange = function ()
        {
          if (xmlHttpObject.readyState == 4)
          {
                if (xmlHttpObject.status == 200)
                {
                      var response = xmlHttpObject.responseText;
                      …
                }
          }
        }
        xmlHttpObject.send(null);
}
```

In Deliverable 2, I implemented a JavaScript program that makes a request every half seconds to the server to get the size of uploaded portion. Then a progress bar will be updated to reflect the changes on the server side. The changes on the progress bar happen automatically without user clicking on the “refresh” button. This method exploits one of Ajax’s design patterns: Periodic Refresh. Under Periodic Refresh pattern, a request is made to the server for new information while another request is taking place. In my case, the request in JavaScript is polling the server for uploaded size while the other request is uploading files. On the PHP server, files are uploaded to a temporary files, and information about these files are stored in a super global array variable called $_FILE. The following code snippet shows you how to handle multiple files uploading in client and server side respectively.

In HTML:

```
<input type="hidden" id="maxSize" name="MAX_FILE_SIZE" value="60000000" />
<input type="file" name="myFile[]" size="30"/>
<input type="file" name="myFile[]" size="30"/>
<input type="file" name="myFile[]" size="30"/>
```

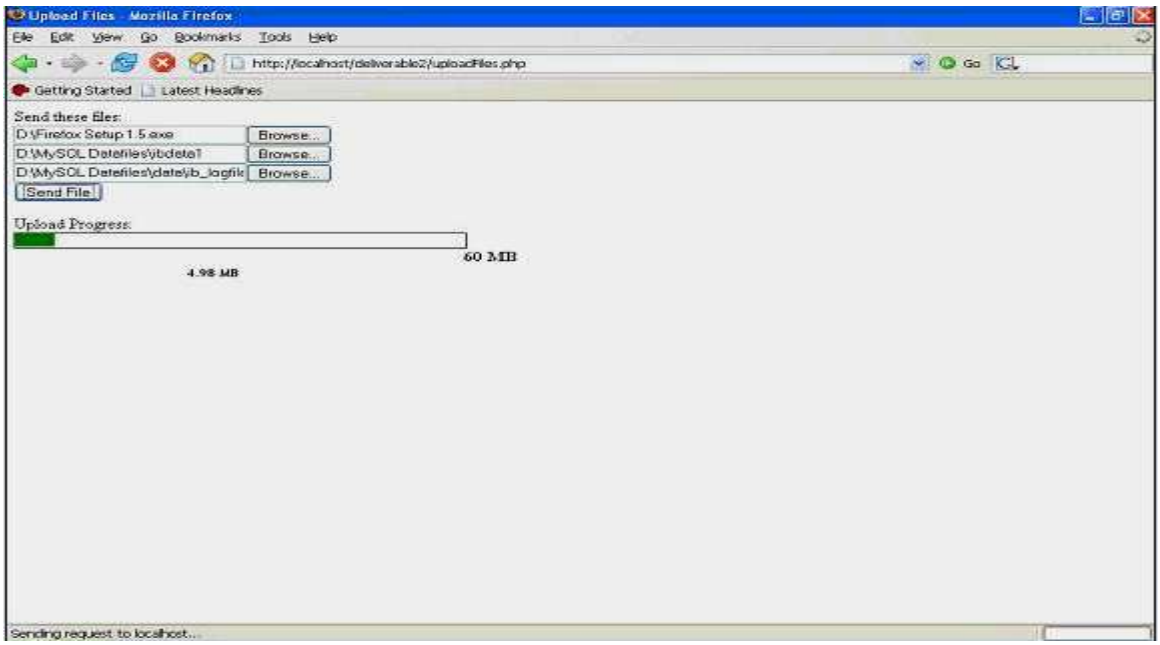
In PHP:

```
is_uploaded_file($_FILES['myFile']['tmp_name'][$key]))
```

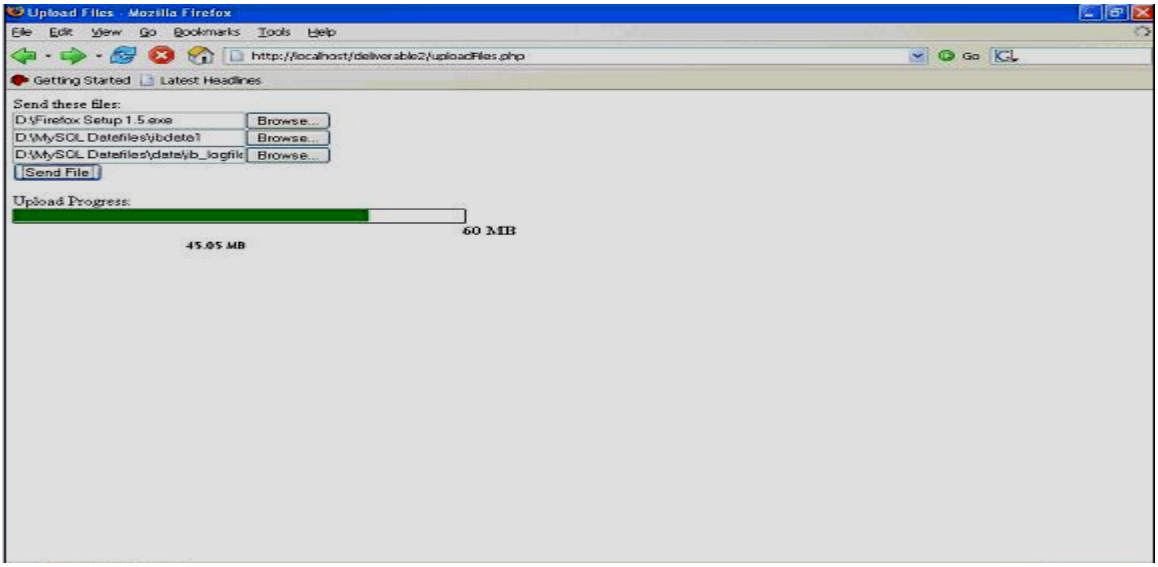
```
{
        if ( $_FILES['myFile']['tmp_name'][$key] != null
          && $_FILES['myFile']['error'][$key] == UPLOAD_ERR_OK)
        {
            if (move_uploaded_file($_FILES['myFile']['tmp_name'][$key], $uploadFile))
            …
        }
}
```

Once the size of uploaded portion comes back to the browser, my program uses Document Object Model (DOM) to reference the progress bar in the HTML page and update it accordingly. Getting the HTML component in JavaScript can be done easily through a function call: “document.getElementById(‘component id ‘). In JavaScript, DOM objects can also be used to traverse and retrieve data from XML document. In my proposed project, student survey results can also be implemented with periodic refresh pattern. When students are filling the form, a JavaScript can poll the server for updated survey results and reflect the results onto the students’ screens. The following screen shots show that a progress bar is asynchronously updated while files are uploading.

Stage 1:



Stage 2:



Deliverable 3: An online voting system

The purpose of this deliverable is to implement an online voting system based on the concept of Byzantine agreement. According to Byzantine protocol, an agreement is expected to be reached after a constant number of rounds, regardless of how voters vote. The central server acts as a manager in such a voting system and opens different voting sessions with a group of voters. PHP has built-in support for sessions. Internally, PHP sessions are implemented by cookies. Each session is assigned a unique ID, and such an ID will not change across different pages. In this voting system, each session ID can be used to identify one single voter. A session ID is generated by calling the function "session_start()." And later, such ID can be accessed by using the "session_id()" function call. The following code snippet shows you how to create, remove and resume sessions in PHP.

In joinAndVote.php:

```
session_start(); // Create a session
…
// Check if the user name and password pair is in the database.
// Redirect back to login page if they are not.
if (mysql_num_rows($resultSet) != 1)
```

```
{
        session_unset();
        session_destroy();  // Remove a session
        header("Location: http://localhost/voter/login.php");
}
```

In voteProcess.php:

```
Session_start();  // Resume a session
…
$id = session_id();  // Access Session ID
```

As we can see, in PHP sharing information across different pages can be done through a session mechanism besides the traditional URL passing method. One way to share data among different sessions is by using a database. In this online system, previous voting information of different voters is stored in “vote” table of a database called “users.” Such information is retrieved from the database to calculate majority and tally values to response every vote. The threshold is generated through a function called “flip()” that implements a coin flip. There is a huge performance penalty for this kind of heavy and frequent database traffic. Here is a code snippet that further demonstrates the ease of manipulating datum of a MySQL database in PHP.

```
public function getReady($round, $groupId)
{
        $sql = "select * from vote where round ='" . $round .  "' and groupNum ='" . $groupId . "'";
        $result = mysql_query($sql) or die(mysql_error());
        $rows = mysql_num_rows($result);
        if ($rows < ElectionManager::$groupSize)
            return false;
        else
            return true;
}


public function retrieveInfo($round, $groupId, $id)
{
         $selectSql = "select * from vote where round >= '" . 0 . "' and groupNum = '" . $groupId . "'";
         $selectResult = mysql_query($selectSql) or die(mysql_error());
         while ($arr = mysql_fetch_array($selectResult))
         {
            if ($arr['vote'] == 1)
                        $numHeads++;
```

```
            else
                    $numTails++;
        }
    …
}
```

The voters cannot move onto the next round unless all voters in the same group have finished casting their votes in the current round. A client will keep polling the server for majority, tally and threshold values every half seconds. When such information is not ready on the server, the response will be a “wait” message. If it is ready, my program will parse the response, update the corresponding fields and stop the timer, so all voters can vote on the next round. The following code snippet illustrates how this is done.

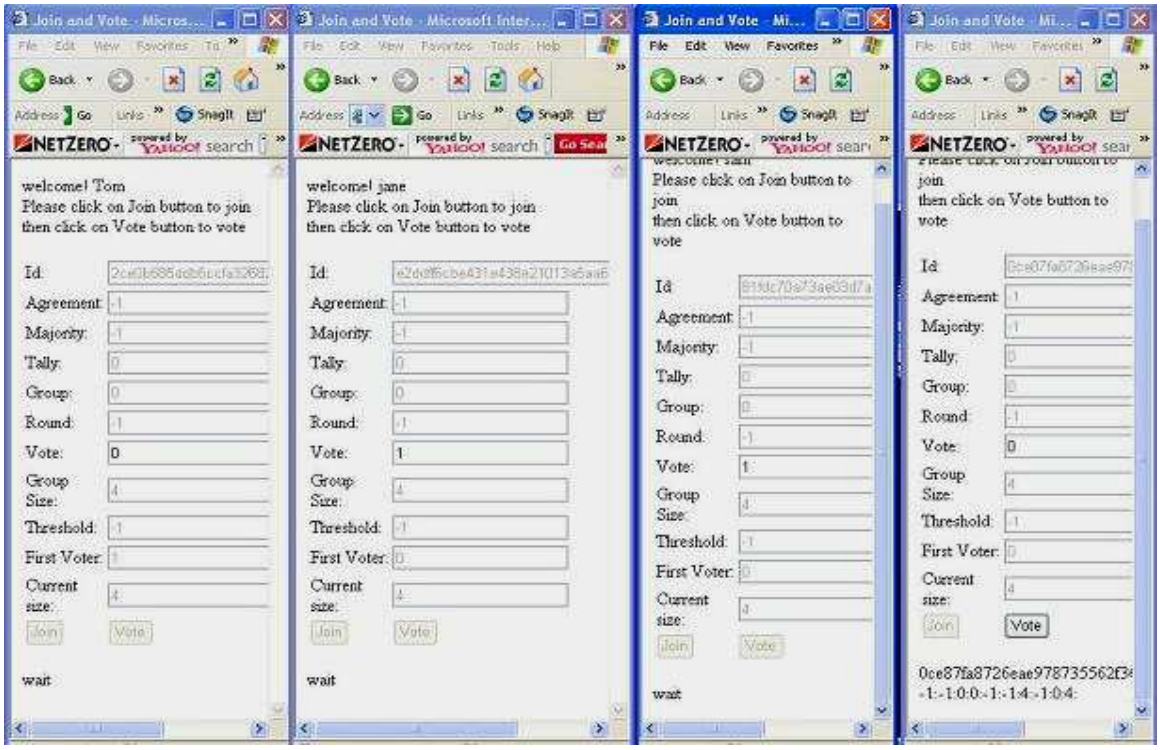
```
Var myInterval;
function voteToServer()
{
        var voteComp = document.getElementById("vote");
        vote = voteComp.value;
        myInterval = window.setInterval("requestInfo()",500);

        // Always return false to mean do not move to other page.
       return false;
}

function requestInfo()
{
        ……
       xmlHttpObject.open("get", "vote.php?vote=" + vote + "&round=" + roundValue +        &dummy=" +
            Math.random(), true);
        ……
        setIntervalOut();
}

function setIntervalOut()
{
        if (myInterval != null)
            clearInterval(myInterval);
}
```

In Internet Explorer (IE), a request is made locally to the cache rather than to the server if IE sees the current URL request is not different from some previous requests’ URL. To avoid such problem, we attach each request with a random number to convince the browser that this is a new request. In this way, each request URL is different with any of

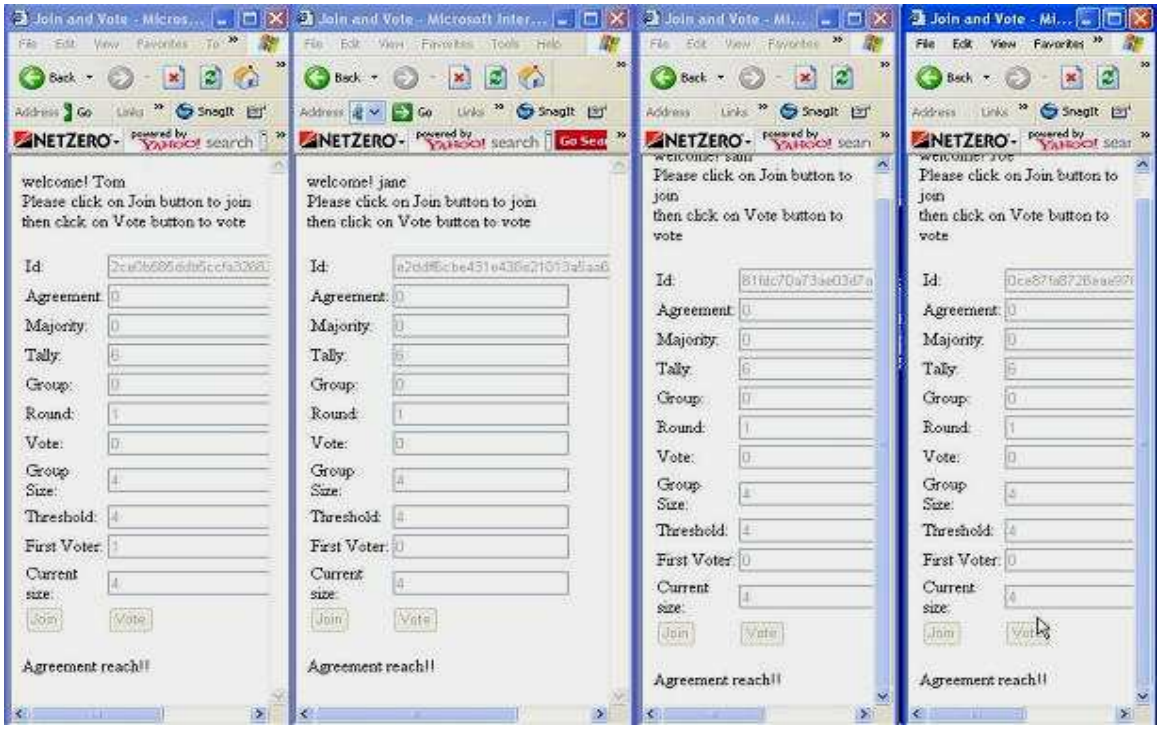
the previous one, and the browser will make each new request to the server, not to the local cache. The purpose of the code below is to show how the desired effect above is achieved.

```
 function requestInfo()
{
        ……
      xmlHttpObject.open("get", "vote.php?vote=" + vote + "&round=" + roundValue + " &dummy =" +
      Math.random(), true);
       ……
}
```

The following screen shot shows that three voters receive a “wait” message because the fourth voter has not voted.



And in this screen shot, an agreement has been reached.



My proposed system is very similar with a voting system. But rather than using Byzantine algorithm, student self-grading system will use permutations to generate a total order. However, implementing such a voting system gives me some ideas on how a similar system should work.

An Algorithm Generating a Total Order from Partial Orders:

In the student self-grading system, each student grades only five randomly chosen papers. We next need to come up with a total order on all student homework based on these partial orders. An algorithm that is a modification of QuickSort algorithm can be used to accomplish this task. First, we initialize an array HW[], so that HW[i]=i. Our eventual goal will be to sort HW so that HW[i] contains the number of the *i* th worst homework. To do this we will call a function VoteSort with actual arguments 0 and n -1. The VoteSort algorithm takes parameters m and n (where m < n, and m and n are subscripts in HW[]). The VoteSort algorithm works as follows: If m = n, VoteSort returns. Otherwise,

VoteSort randomly chooses a pivot position, say i, and swaps the elements in positions i and m, and sets k=m. For each position j between m+1 and n, VoteSort then calculates the number of times, X, that the homework HW[k] is voted worse than the HW[j] by students. It then calculates the number of times, Y, that HW[j] is voted worse than HW[k] by students. If X > Y, VoteSort swaps j and k +1 and then swaps k and k + 1 and sets k=k+1. Next VoteSort recursively calls itself twice: one is on m and k; the other is on k+1 and n. Since VoteSort algorithm is just a slight alteration of QuickSort, like QuickSort, it is an O(n*log(n)) time algorithm. Therefore, a total order can be generated very fast.

Future work:

I will continue research on game theory to analyze the possibility of student grading behaviors and come up with a scheme to ensure the fairness of such self-grading system. In the system, rewards will be given out according to his/her grading effort. At the beginning of CS298, I will focus on analyzing and implementing VoteSort algorithm to generate a total order of all student homework. There might be some other existing total orders that can represent all the partial orders from students, but the total order generated from VoteSort algorithm possesses an important property that if a pair (HW[i], HW[j]) (0 <= i < j <= n-1) appears in the generated total order, HW[i] homework is graded better than HW[j] by students. Next, I will utilize game theory concepts to implement a reward algorithm that will assigns rewards points to students according to their grading effort. Later, student surveys and grading forms will be integrated with Dr. Pollett's current online homework submission. In addition, a WiKi system is planed to be added to provide students an online discussion space for homework solution.

Reference:

[Achour06] PHP Manual. Mehdi Achour. Friedhelm Betz. Antony Dovgal. Nuno Lopes. Philip Olson. Georg Richter. Damien Seguy. And Jakub Vrana. Edited by Gabor Hojtsy. 2006.

[Crane06] Ajax in Action. Dave Crane, Eric Pascarello, and Darren James. Manning Publications Co. 2006.

[Ferguson05] Game Theory. Thomas S. Ferguson. Department of Mathematics at University of California at Los Angeles. 2005.

[Glass04] Beginning PHP, Apache, MySQL Web Development. Michael Glass, Yann Le Souarnec, Elizabeth Naramore, Gary Mailer, Jeremy Stolz, and Jason Gerner. Wiley Publishing, Inc. 2004.

[Goodman04] JavaScript Bible 5th Edition. Danny Goodman, michael morrison, and Brendan Eich. Wiley publishing Inc. 2004.

[Kearns02] A Tutorial on Computational Game Theory. Michael Kearns. Computer and Information Science of University of Pennsylvania. 2002.

[McCain99] Strategy and Conflict: An Introductory Sketch of Game Theory. Roger A. McCain. http://william-king.www.drexel.edu/top/eco/game/intro.html. 1999.

[McLauglin06] Head Rush Ajax. Brett McLauglin. O'Reilly Media, Inc. 2006.

[Meloni04] Sams Teach Yourself PHP, MySQL, and Apache. Julie C. Meloni. Sams Publishing. 2004.

[Myerson91] Game Theory: Analysis of Conflict. Roger B. Myerson. Harvard University Press. 1991.

[Zakas06] Professional Ajax. Nicholas C. Zakas, Jeremy McPeak, Joe Fawcett. Wiley Publishing, Inc. 2006.