**Approach**

ESMZone’s appeal is partially due to its highly interactive dialog and scoring functionality. When a member interacts with other members their score changes and their standings are updated in real time. Their score reflects not only their own activity but the ratings of their activity by other members. The system must, therefore, must be very dynamic and smart about how scores are recorded and updated to efficiently maintain real-time results.

An inefficient approach to providing a real-time score would be to sum up all a user’s points and ratings and recalculate there score whenever it is displayed. Since the factors of the score are always changing this would insure that the score is accurate. This can become a potential bottle neck if many users are both updating and calculating score using the same records at the same time. One alternative is to calculation periodically (ex. every 5 minutes) but this would remove the excitement of real-time interaction.

The approach taken to both maintain real-time results and limit performance and concurrency issues is to store the score for quick retrieval and apply incremental changes as new events and ratings occur. This approach significantly reduces the number of queries needed to update and display the standings. However, there still remains the potential for concurrent access to the same scores when several of a user updates postings and ratings for the same items at the same time. The solution for this is to use locking (synchronization) of limited key records to prevent the loss of information. If the locking ends up having significant performance implications or results in deadlock situations, a queuing approach may need to be taken so that score updates can be applied as soon as possible without slowing down the system. In order to validate these approaches, high volume load testing will be done.

**Algorithm**

There are two major categories for scoring in the system: Dialogs and Users.

Dialogs

Dialogs consist of member questions with potentially multiple member answers to these questions. Each question and its respective answers can be rated by any member on a scale of 1 to 5. The rating of a question or answer is the average of all the member ratings for that item. For example if a question has two ratings, 3 and 5, then the question’s rating would be 4.

A dialog has also has a score by virtue of a point value given to the question and answers and a multiplier based on rating. For example, a question is worth 100 points and an answer is worth 10. A question with 2 answers is, therefore, worth at a minimum 120 points. The score also factors in the rating of each item. So an answer rated 5 has a score of 50 points compared to one rated 3 which is worth only 30 points. Similarly, a question rated 5 is worth 500 points. The questions score is the combination of the stand alone question score and all the answer scores. In the above example, of a question and two answers, assuming the question is rated 3 and the answers are rated 3 and 4 respectively, the total question’s score is 300+30+40=370.

Users

Unlike questions and answers which have fixed point values, users derive points for their interaction with the site. They get points for subscribing, points for asking and answering questions, points for rating and even points for login. These points are accumulated in real-time. The user also gets a rating. Their rating is the average of all ratings other members have given to any of their questions/posts or answers. Their total score is the product of their points and their rating. For example if a user has 500 points and a rating of 3, their score will be 1500.

**Real-time Calculation**

The three main challenges of real-time calculation are:

1. Maintenance: Not having to put score related code through the business and control logic to accrue points for events. The more code impacted by the scoring logic, the more work to update and change the scheme.
2. Flexibility: Making the system dynamically able to handle changes in point values for events and easy to add new events.
3. Calculation Propagation: Allowing adds and updates of point events to propagate to related users and posts without having to recalculate all the data. For example, rating an answer will impact the question score as well as the users who authored the answer.

The solution:

1. Maintenance: Scores are entirely handled by the business objects and tie into CRUD events.
   1. Points are added when a business object is created or updated (ex. question, answer, comment, rating, network connection, subscription etc.) Just before these new business objects are stored, a points object is crated to reflect the event and accrue point to the creator. Similarly when a rating is done, that business objects handles the update to related scores.
   2. No controller code is used except to handle scoring for events that have no related business objects. (For example, login, there is no login object, rather login is a change of state in the system and not persisted in the database. Even in this case, rather than use the controller to records the points, it would be reasonable to create a login event log business object and tie the creating of point to that object.)
2. Flexibility: The point value for each event is stored in a separate business object that is loaded and used to determine the value of user activity. When a business object is about to save itself, it looks up the points for this event and creates the appropriate points record.
3. Propagation: When a business event creates points, it also signals the User object to recalculate the users score. Similarly, when a ratings business object saves a rating, it signals the dialog and user objects to recalculate the average rating. If a question is rated, the average rating and score for that question is updated as well as the average rating and score for the author of that question. A slightly more complex example is when an answer is rated. This impacts the answer, the related question and the author of the answer. The system will propagate the changes to rating and score to all the related objects.

Incremental Algorithm

In order to be able to incrementally apply ratings to a user or post, a special incremental calculation is used. This method also requires the permanent storage of partial calculation values. The algorithm is as follows:

Definition: Average Rating = (sum or ratings) / (count of ratings)

1. Store count of ratings in addition to average rating to allow incremental recalculation
2. An new rating can be applied without summing all the prior ratings by using the following technique:
   1. Get the current sum of ratings: current average rating \* count of ratings
   2. Get the new sum of ratings: current sum + new rating
   3. Get the new count of ratings: current count + 1
   4. New average = new sum / new count
   5. Note: above the current count of ratings must be stored along with the average rating to perform this optimization.
3. A change to a rating is done in a similar fashion except that the in addition to the current count, the previous rating must be stored temporarily until the calculation is completed.
   1. Get the current sum of ratings: current average rating \* count of ratings
   2. Get the new sum of ratings: current sum – old rating + new rating
   3. Get the new count or ratings: current count (note: a change does no increase the number of ratings)
   4. New average = new sum / new count