A Computer Aided Grading System for Subjective Tests

Yijian Wu
Fudan University, Shanghai, China
wuyijian@fudan.edu.cn

Wenyun Zhao
Fudan University, Shanghai, China
wyzhao@fudan.edu.cn

Xin Peng
Fudan University, Shanghai, China
pengxin@fudan.edu.cn

Yunjiao Xue
Fudan University Shanghai, China
yjxue@fudan.edu.cn

Abstract

Computer aided tests replace traditional written answers on paper sheets with electronic records. For subjective tests, computers are not able to do all grading jobs, due to limited comprehensive ability of computers. Subjective answers need be reviewed by different graders in order to improve justice. We propose a distributed computational model for grading electronic subjective answers. Answer data are divided into numerous independent data cells. Computational resources are automatically allocated by the system to do grading jobs and independently report grading results to the system. Since the grading results may vary slightly or magnificently, we assign computational factors to data cells and computational resources to minimize the variation. In addition, rules are defined to obtain a final result. We successfully apply the proposed model in our grading system in computer aided spoken English tests.

1. Introduction

Grid computing, as a newly-emerging computing model for distributed computing, enables widespread sharing and coordinated use of resources on network [5]. It seeks to harness the power of computational resources in the network to maximize efficiency of computation. Grid computing is now widely under research and adopted in high-performance computing, scientific computing and other computational work in distributed systems [1, 2, 6, 8].

There are now various grid applications and infrastructures including grid middleware, toolkits and architecture supporting grid computing, such as open grid services architecture (OGSA) [9], Globus toolkits [11], TCM-Grid [3] and SETI@home project [12]. However, none of them provide solutions to indefinite problems.

Subjective grading tasks are among indefinite problems, which do not lead to an accurate or definite result and need human interference. In such circumstance, existing distributed technology cannot perfectly meet the computing requirements on uncertainty. A typical task is to grade answers to open-ended questions in subjective tests.

In this paper, we first try to build a mathematical model for indefinite computing work; then we discuss some extra control mechanisms while doing the computation work. The model will be used to build a computer aided grading system for subjective tests. We will explain how the model works in an actual grading task in a spoken English test.

2. Computation with Computational Factors and Privileges

2.1. Basic computation model

The subjective grading system to be described is based on the following basic computation model. It is a simple but quite extensible model.

Definition 1: A ***basic computation*** is a triplet (DI, CU, DO), where DI is reasonable and meaningful input data, CU is computational unit with a certain computation method, and DO is a reasonable output of the computation – a result.

The input data of our computation task are decomposed into numerous basic "elements", which will be computed by computational node in a grid. These "elements" are the basic input data units. Note that not all kinds of decomposition of original input data are valid in our computation model. For the "elements" we have constraints: all "elements" in the decomposition should be computable, i.e. there exists at least one *basic computation* applicable for each "element".

Definition 2: A ***Data Cell*** is a basic input data unit, any part of whom cannot be computed by any computational units in a basic computation.

Now we can regard all input data in our computational task as a composition of Data Cells, i.e. DI is composed of a set of Data Cells.

2.2. Computational Factors and Privilege

In this section we introduce an extension of our basic model. A Data Cell contains not only data, but extended information about the data it contains as well. This information is abstracted as Computational Factors (CF). It is one type of Meta-information [4] of the computational grid.

Computational Factors, or CFs, are common properties of all Data Cells. A Data Cell can be presented as DC(data, F1, F2, … , Fn), where Fi (i=1, 2, ..., n) are values of corresponding CFs.

Moreover, we are able to attach constraints to Computational Nodes (CNs) in the grid according to CFs. A Computational Node is represented as CN({F1}, {F2}, … , {Fn}), where {Fi} (i=1, … ,n) are a set of corresponding CF values. For example, there are 3 CFs (named CFA, CFB, CFC), where CFA has a definition domain {1, 2, 3}, CFB {8, 9}, and CFC {'a', 'b'}. We have a data cell DC(data, 1, 8, 'a'), a computational node CN({1, 2}, {8}, {'a', 'b'}). This means DC has property CFA with value 1, CFB with value 8 and CFC with value 'a'; CN has the privilege to compute those data cells in the set {DC(CFA, CFB, CFC)|CFA in {1, 2} and CFB in {8} and CFC in {'a'}}.

Definition 3: A computational node CN({F1}, {F2}, … {Fn}) has the ***privilege*** of computing a data cell DC(data, F1', F2', …, Fn') iff for all Fi' we have Fi'$\in${Fi} where i = 1, 2, … ,n.

For example, consider DC(data, 1, 8, 'a'), CN({1, 2}, {8}, {'a', 'b'}) and another computational node CN'({2}, {8}, {'a'}). We say CN has a privilege to compute DC while CN' has not.

As all Data Cells have properties in terms of computational factors while computational nodes in the grid have privileges accordingly, our computational grid has n dimensions, where n is the number of CFs associated with all data cells. With these privilege dimensions, we are able to control our computation task dimension by dimension, manually, semi-automatically, even automatically. When the computation begins, each data cell will be dispatched to at least one most "suitable" computational node, where "suitable" can mean relevant or inexpensive or other user-defined features, depending how users inspect the dimensions. In Section 3.2 and 3.3, we will do more discussion.

2.3. Computational Grid with and without Privileges

Here we compares computational grid functioning *with* and *without* privileges. Obviously, privileges come from Computational Factors. Computational Factors are properties used to describe Data Cells' characteristics, but not originally associated with computational nodes. Therefore, if we do not apply computational factors on computational nodes in a computation, no privileges will be applicable any more. In this case, all computational nodes in the grid are treated equally and able to compute all data cells. When we assign CF values on these nodes, we put constraints on them and the nodes become different to each other for our computing task and may take different jobs.

3. Solving Indefinite Problems

3.1. Gathering raw data.

In order to get our computational grid working, we have to provide input data. As we have discussed in Section 2.2, we introduce computational factors to our input data. Therefore when we are preparing our input data, we have to value all the CFs. There are two ways doing this. One is that we assign or define a CF value for each CF in a Data Cell while gathering them. This means when we are gathering data, we know exactly to what scale or granularity our Data Cells are. But it is not applicable when we gather data continuously without noticing what a data cell will be like. Here is the other way: we decompose raw data into data cells later on and assign a value to each CF within a data cell. In this case, data cells are decomposed by a back-end service machine called Data Cell Builder (DCB), right after all raw data are gathered. We will take the latter situation for further discussion.

In either situation, we have data cells with CF values attached before we begin our computation. Simply, in this step we take raw data as input and the output is Data Cells with CF values assigned.

3.2. Assigning CF constraints to computational nodes

For each computational node, we have a rule to assign the node CF constraints. There are still two situations. One is that the nodes are known and accessible before it joins our grid for computing. This means we can evaluate the node and manually or semi-automatically set the constraints. This is just like centralized control where the node in computation is totally predictable. The other situation is that we are not able to predict which computer (or computational resources, in case it appears not to be a computer but other device) will be our node and what abilities it may have. Thus the CF constraints must be dynamically evaluated and assigned. We will take the latter situation for further discussion.

3.3. Starting the computation

When there are data cells prepared and computational nodes ready, the computation may begin. Note that it is unnecessary that all data cells be prepared before our computation begins. In fact, preparing data cells and computing prepared data cells can be performed simultaneously. In our architecture, the computation task is controlled by a front-end machine called Dispatcher. As is discussed in 3.1, data cells are continuously produced by DCB. When a new node joins the computation, the Dispatcher is informed and emits a detecting agent to evaluate what constraints is applicable and fetches the node status. The detecting agent is a mobile agent containing evaluation codes, working on a virtual machine designed for agent execution. [7, 10] The Dispatcher delivers appropriate Data Cells to the node according to its status fetched by the detecting agent. Note that one node may not take only one Data Cell at a time. The Dispatcher may select a set of Data Cells for the computational node according to the node status (incl. self-condition like available memory or CPU load, and constraints defined by CF values).

3.4. Finalizing the computation

When all data cells are computed by computational units, the results of separated data cells should be collected, reviewed and organized to build up a meaningful result of our original computation task. Due to some reasons, such as fault input data, random computation failures, or network errors, not all results are valid. On one hand, these individual results should be organized together for further inspection to meet our original computation purpose; on the other, the integrity of all results should be examined, since conflict results may exist. This is particularly important when solving indefinite problems, which will be discussed in the next section.

4. On indefinite problems

We are going to discuss how indefinite problems can be solved in our computation grid. An indefinite problem means the computation result may be slightly different when it is assigned to different computational nodes in the grid. The difference cannot be precisely foretold or predicted. Each Data Cell may be an indefinite problem. And that is why, in our definition to simple computation in 2.1 (Definition 1), we do not specify a one-to-one mapping between DI and DO. In fact, assume that both computational nodes CUi and CUj (CUi≠CUj) have the privilege to compute DI, then CUi(DI) may be slightly different from CUj(DI); even when CUi computes DI n times repeatedly, the n CUi(DI) may vary. In our model, we assume all CU(DI) follow some statistic rules. In other words, the output is not totally unpredictable, but can be statistically calculated, not precisely calculated, though.

Because of all these uncertainty, the computing should not be only “one pass”. Accurate computing executes only once, and the result to the same problem will be exactly the same if the computation restarts. This is one pass computing: each data cell is computed only once and the computation can be exactly repeated. Dealing with our indefinite problem, every result may change a little, thus we have to follow statistic rules to do “multi-pass” computation. That is to say, one data cell may be delivered several times to diverse computational nodes for computing. The delivery is controlled by a Dispatcher, which consults Dispatch Strategy before selecting data cells for delivery when it is informed that there is computational resource available. Dispatch Strategy is a configurable data structure, storing information for Dispatcher to make decisions, including:

- ♦ Which data cells need computing how many times?
- ♦ When a data cell is re-delivered, is it allowed to send it to the node which has computed it?
- ♦ When different results are obtained, what to do to the results? Keep all? Keep random one? Keep the first one? Or do customized computation? ...

5. A computer aided grading system

In a subjective test, all students’ answers are no absolute right or wrong. They may be partially right or partially wrong, even no right or wrong at all. Unlike multiple-choice questions or fill-in-the-blank questions, most open-ended questions need to be graded by human (the graders). Different graders may give different scores to a certain student’s answer. The grading system decomposes students’ answers into data cells and dispatches the data cells to graders according to predefined rules and privilege settings. After a grader has graded the data cells he/she has to upload the results to Dispatcher and the server will determine how to calculate the final result according to Dispatch Strategy. Chances are that the results to the same answer are too far-away to agree on a final score. The dispatch will then decide to do additional dispatches for further scoring to ensure justice.

5.1. Getting started

To start the grading task, we first gather students' original answers. The answers should be electronically stored. In our experiment, we tried spoken English tests in college. Students' answers are voice records directly recorded, compressed and packed in personal computers. These data can hardly be graded by computers automatically with technologies nowadays because it is difficult for a computer to understand speech content and judge how well the speech conforms to a given topic. The original data are organized and packed student by student. Before we can store these data into our database, we have to divide them into small elements. Every answer to one question is regarded as a data cell, thus raw data decomposed into data cells. Figure 1 shows how data cells are prepared.

A single data cell cannot be directly dispatched to graders. We have to assign computational factors to them. We represent a data cell as the following:

DC(data, F1, F2, F3), where data=answer data to one question, i.e. a student's voice record; F1=test paper's code; F2=question type code; F3=student's ID.

F1 has a definition domain of {1, 2, 3, 4, 5, 6, 7, 8}, as we used eight different test papers in the spoken English test. F2 has a definition domain of {1, 2, 3, 4, 5}, as every paper consists of five types of questions. F3 has a definition domain of all students' IDs.

Meanwhile, all graders are assigned proper privileges. Grader T1, for instance, has the privilege of grading answers to test paper No1 and No2, and Grader T2 is able to grade answers to test paper No7 and No8, then we have the following rules:

T1({1, 2}, All, All), T2({7, 8}, All, All).

This means that for all data cells DC(data, F1, F2, F3), only those satisfying $F1 \in \{1,2\}$ can be dispatched to Grader T1, and those satisfying $F1 \in \{7,8\}$ can be dispatched to Grader T2.

Now we are ready to have the system running. A dispatch server is working between the database and graders. When a grader connects to the server (i.e. logon), the server will automatically select a predefined number of appropriate data cells, pack them up and transfer the package to the grader. When the grader has given each of the data cells a score, the result will be automatically uploaded to the database through the dispatcher. Figure 2 shows how data packages are distributed and results are collected.

The results returned to the database may not be the final results. One data cell can be graded by two different graders, and if the difference of the two results is beyond a predefined threshold the data cell will be sent to a third grader for grading.

When a third result is returned, or the difference between the former two results is within the threshold, the system will calculate the final result, using a predefined method, such as AVERAGE, MAX, MIN, or a user define method. We can also set up rules to eliminate some "unexpected" grades out of three returned results.

When all data cells have their final results, the subjective grading task comes to an end.

5.2. Grader training and spot-checking

As the grading task is a subjective process, the graders should be trained before actual grading task begins. In fact, we designed some sample data cells for grader training. The sample data cells already have their final computation results. Graders have to grade the sample data cells before they start grading normal

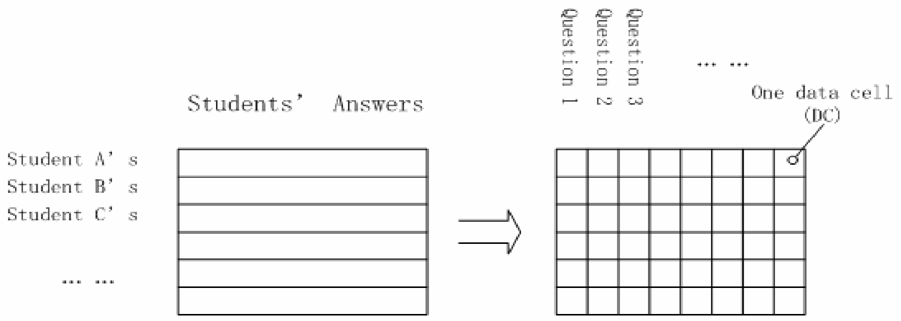

Figure 1. Dividing students' answers into data cells

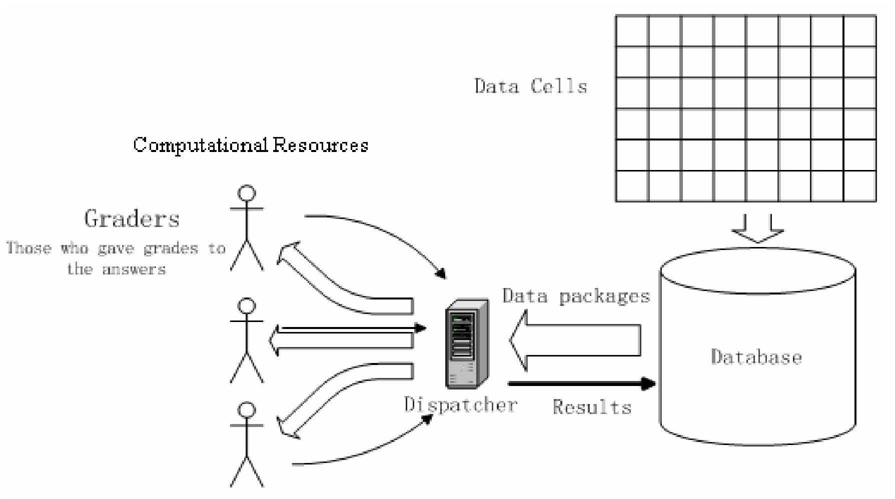

Figure 2. Data dispatching and results returning

data cells. If they gave much higher grades or much lower grades, they are informed to adjust their grading standards. They will not be able to grade normal data cells until they grade sample data cells not too high or too low.

Sample data cells are also used to check graders work during the whole task. It can be predefined how frequently the sample data cells should be sent to a grader when he/she is doing his/her job. This spot-checking mechanism helps task administrators to keep an eye on the whole process, avoiding graders being too strict or too loose for a long time.

5.3. Educational benefits

There are several educational benefits applying the proposed model.

First of all is *flexibility*. All answer data are unpacked and divided into small data cells. These small, basic elements can be dispatched individually or in custom defined groups. The model is quite suitable for organizing a computer aided grading task for a subjective test, such as computer based spoken language test or written test whose answer sheets are electronically scanned for online grading.

Second is *justice*. The proposed model offers a mechanism of independent double checking. All data cells may be delivered to two or more individuals for independent grading. Spot-checking feature enables grading procedure control.

Third is *efficiency*. Data cells can be grouped and packed as needed, which offers an opportunity for graders to focus on answers to the same (type of) question in a comparatively short period of time. This helps increasing grading efficiency because graders need less time reading the questions.

There are still more benefits applying the model. For example, any graders who have passed the grader training are allowed to participate in grading task while the grading task is proceeding. This dynamic feature brings a good *scalability* to the system and to the task. This feature is borrowed from the theory of the Grid [5], different in that each node is actually a human instead of a computer.

6. Conclusion

We have discussed a mechanism with which indefinite problems can be solved. The grading system for subjective tests is an application system based on the model. Computational factors are introduced into the computation (or grading) task, and computational resources (or graders) are bound with privileges. Thus on one hand, we have more manual control method in our distributed computation; on the other, automatic or semi-automatic control can be easily realized in our computation with the mathematical model.

However the model has defects. The basic data cells should be independent to each other. But unfortunately not all computation tasks can be divided into independent basic data cells. If the data cells are relevant to each other, new improvement should be made to our model and its application.

7. Acknowledgements

The work presented in this paper is supported in part by the China 863 Research Program under Contract 2005AA113120; National Natural Science Foundation of China under Contract 60473061;

Shanghai Municipal Science and Technology Commission under Contract 04DZ15022.

8. References

[1] Beiriger, J.I.; Bivens, H.P.; Humphreys, S.L.; Johnson, W.R.; Rhea, R.E, Constructing the ASCI computational grid, In *Proc. 9th IEEE Symp. on High-Performance Distributed Computing,* 2000, pp 193 - 199

[2] Brunett, S., Czajkowski, K., Fitzgerald, S., Foster, I., Johnson, A., Kesselman, C., Leigh, J. and Tuecke, S., Application Experiences with the Globus Toolkit, In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, 1998, pp 81-89

[3] Chen, H.J., Wu, Z.H., Huang, C., Open grid services of traditional Chinese medicine, *IEEE International Conference on Systems, Man and Cybernetics, 2003.* Vol. 5, Oct. 2003. pp 4546 - 4551

[4] Costa, F.M.; Blair, G.S, Integrating meta-information management and reflection in middleware, In *Proc. DOA '00. International Symposium on Distributed Objects and Applications*, Sept. 2000. pp 133 – 143

[5] Foster, I. and Kesselman, C., *the Grid: Blueprint for a New Computing Infrastructure,* Harper Collins, 1999.

[6] Johnston, W.E., Gannon, D. and Nitzberg, B., Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. In *Proc. 8th IEEE Symposium on High Performance Distributed Computing*, 1999

[7] Li, C.L., Li, L.Y., An agent-based approach for grid computing, In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies 2003 (PDCAT'2003.)*, Aug. 2003, pp 608 – 611

[8] Stevens, R., Woodward, P., DeFanti, T. and Catlett, C., From the I-WAY to the National Technology Grid, *Communications of the ACM, 40*(11), pp 50-61, 1997.

[9] Talia, D., The Open Grid Services Architecture: where the grid meets the Web, *Internet Computing, IEEE, Volume: 6, Issue: 6,* Nov.-Dec. 2002, pp 67 – 71

[10] Valetto, G., Kaiser, G. and Kc, G.S., A Mobile Agent Approach to Process-based Dynamic Adaptation, In *Proc/EWSPT 2001, 8th European workshop Software Process Technology*, (*Lecture notes in computer science, vol 2077*), pp 102-116

[11] The Globus project home page: http://www.globus.org

[12] The SETI@home home page: http://setiathome.berkeley.edu