Parallel Processing for Fuzzy Queries in Human Resources Websites

Lien-Fu Lai, Chao-Chin Wu, Ming-Yi Shih, Wen Chiou
Department of Computer Science and Information Engineering
National Changhua University of Education
Changhua City, Taiwan
{lflai, ccwu, myshih}@cc.ncue.edu.tw, a86651234@hotmail.com

ABSTRACT

In this paper, we use Parallel FuzzyCLIPS to parallelize the execution of FQHR websites for two kinds of task partitioning in both grid and cluster environments. First, a new architecture of FQHR websites is proposed to parallelize the execution of fuzzy queries on grid and cluster systems. Second, our approach supports two kinds of task partitioning: data level and module level. The data level partitions the facts and allocates them to multiple processors for parallel execution, while the module level partitions the rules and allocates them to different processors. Third, a performance evaluation model is presented to analyze the proposed approach. Finally, we implement a parallelized FQHR website to test the speedups by experiments and to verify the results of performance analysis.

Keywords: Parallel Processing, Grid Systems, Cluster Systems, Fuzzy Query, FuzzyCLIPS, Human Resource Websites

I. INTRODUCTION

In human resource websites, the query requirements of job seekers and hiring companies often contain imprecision and uncertainty that are difficult for traditional SQL queries to deal with. For example, when a user hopes to find a job which is near Taipei City and pays good salary, he can only make a SQL query like "SELECT * FROM Job WHERE (Location='Taipei City' or Location= 'Taipei County') and Salary≧40000". However, both 'near Taipei City' and 'good salary' are fuzzy terms and cannot be expressed appropriately by merely crisp values. A job which locates in 'Taoyuan County' with salary of 50000 may be acceptable in user's original intention, but it would be excluded by the traditional SQL query. SQL queries fail to deal with the compensation between different conditions. Moreover, traditional database queries cannot effectively differentiate between the retrieved jobs according to the degrees of satisfaction. The results to a query are very often a large amount of data, and the problem of the information overload makes it difficult for users to find really useful information. Hence, it is required to sort results based on the degrees of satisfaction to the retrieved jobs. Computing the degree of satisfaction to a job needs to aggregate all matching degrees on individual conditions (e.g. location, salary, industry type, experience, education etc.). In addition, traditional database queries do not differentiate between conditions according to the degrees of importance. One condition may be more important than another condition for some user (e.g. salary is more important than location in someone's opinion). Both the degree of importance and the degree of matching to every condition should be considered to compute the degree of satisfaction to a job.

We have proposed a fuzzy query mechanism for human resource websites (FQHR) [1] to alleviate the mentioned problems: (1) Users' preferences often contain imprecision and uncertainty. FQHR provides a mechanism to express fuzzy data in human resource websites and to store fuzzy data into conventional database management systems without modifying DBMS models. (2) Traditional SQL queries are based on total matching which is limited in its ability to come to grips with the issues of fuzziness. FQHR provides a mechanism to state fuzzy queries by fuzzy conditions and to differentiate between fuzzy conditions according to their degrees of importance. (3) Traditional SQL queries fail to deal with the compensation between different conditions. FQHR provides a mechanism to aggregate all fuzzy conditions based on their degrees of importance and degrees of matching. The ordering of query results via the mutual compensation of all fuzzy conditions is helpful to alleviate the problem of the information overload.

In FQHR, the fuzzy logic theory [2] is used to develop a fuzzy query mechanism for human resource websites and FuzzyCLIPS [3] is used to offer the capability of fuzzy computation and fuzzy reasoning for matching and aggregating fuzzy conditions. The matching and the aggregating of fuzzy conditions are implemented as FuzzyCLIPS rules. Therefore, the number of fuzzy data needed to compute the degrees is the main factor that affects the response time of making a fuzzy query. A large amount of fuzzy data may increase the execution time substantially. We have proposed a Parallel FuzzyCLIPS programming language [4] that defines new syntax to call the MPICH library [5] by adding external functions into the FuzzyCLIPS inference engine. New defined syntax follows the same style of the FuzzyCLIPS language and is able to execute a FuzzyCLIPS application in parallel on the cluster system. In this paper, we use Parallel FuzzyCLIPS to parallelize the execution of FQHR websites for two kinds of task partitioning in both grid and cluster environments. First, a new architecture of FQHR websites is proposed to parallelize the execution of fuzzy queries on grid and cluster systems. Second, our approach supports two kinds of task partitioning: data level and module level. The data level partitions the facts and allocates them to multiple processors for parallel execution, while the module level partitions the rules and allocates them to different processors. Third, a performance evaluation model is presented to analyze the proposed approach. Finally, we implement a parallelized FQHR website to test the speedups by experiments and to verify the results of performance analysis.

II. FQHR: A FUZZY QUERY MECHANISM FOR HUAMN RESOURCE WEBSITES

FQHR [1] applies the fuzzy logic theory to develop a fuzzy query mechanism for human resource websites. It contains (1) a storing mechanism to represent and store fuzzy data, and (2) a fuzzy query language to make fuzzy queries on fuzzy databases.

A. Storing Fuzzy Data into Databases

FQHR adopts the notions of Galindo's work [6] to classify fuzzy data into three types: discrete fuzzy data, continuous fuzzy data, and crisp data. The discrete fuzzy data is represented by a discrete fuzzy set which consists of a set of discrete data items with their degrees of conformity. The linguistic degrees of conformity (i.e. totally unsatisfactory, unsatisfactory, rather unsatisfactory, moderately satisfactory, rather satisfactory, very satisfactory, and totally satisfactory) are utilized to make it easier and clearer for users to grade degrees. 'Totally unsatisfactory' and 'totally satisfactory' stand for 0 and 1 respectively, while the others grade values between 0 and 1. For example, a fuzzy term 'near Taipei City' can be expressed as a discrete fuzzy set like {(Taipei City, totally satisfactory), (Taipei County, very satisfactory), (Taoyuan County, moderately satisfactory)}. FQHR uses the membership functions in [7] to define the linguistic degree of conformity between a discrete data item and a fuzzy term (see Figure 1).

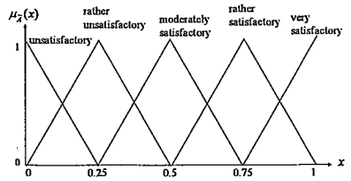

Figure 1. The membership functions for degrees of conformity

A fuzzy number $\tilde{A}$ can be defined by a triplet (a, b, c) and the membership function $\mu_{\tilde{A}}(x)$ is defined as:

$$\mu_{\tilde{A}}(x) = \begin{cases} 0 & ,x < a \\ \frac{x-a}{b-a} & ,a \le x \le b \\ \frac{c-x}{c-b} & ,b \le x \le c \\ 0 & ,x > c \end{cases}$$

Therefore, each linguistic degree of conformity can be mapped to a triangular fuzzy number [8], e.g. 'rather satisfactory' is mapped to (0.5,0.75,1).

The continuous fuzzy data is represented by a continuous fuzzy set which consists of a set of continuous data items with their degrees of conformity. For example, a fuzzy term 'good salary' can be expressed as a continuous fuzzy set like {(50000, totally satisfactory), (45000, very satisfactory), (30000, totally unsatisfactory)}. The degree of conformity can be defuzzified by using mathematical integral to compute the center of the area that is covered by the corresponding triangular fuzzy number [9], e.g. 'very satisfactory' is defuzzified by computing its center of gravity of (0.75, 1, 1) as follows.

$$\frac{\int_{0.75}^{1} x(4x-3)dx}{\int_{0.75}^{1} (4x-3)dx} = 0.92$$

Therefore, the membership function corresponding to the given 'good salary' can be constructed by {(50000,1), (45000,0.92), (30000,0)} (see Figure 2).

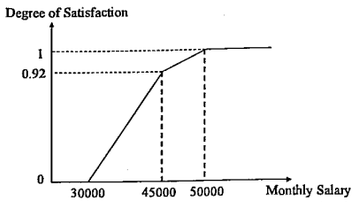

Figure 2. The membership function of 'good salary'

B. Making Fuzzy Queries on Web Databases

A fuzzy query consists of a set of fuzzy conditions. The fuzzy set that defines a fuzzy condition could be discrete, continuous, or crisp. Each fuzzy condition associates with a fuzzy importance to differentiate between fuzzy conditions according to the degrees of importance and uses a fuzzy set to state the degrees of conformity for different attribute values. The linguistic degrees of importance (i.e. don't care, unimportant, rather unimportant, moderately important, rather important, very important, and most important) are utilized to make it easier for users to grade relative importance.

For matching fuzzy conditions with fuzzy data, FQHR adopts the possibility measure in the fuzzy logic theory [10] to compute the degree of matching between a fuzzy condition $\tilde{A}$ and the fuzzy data $\tilde{B}$:

$$\text{Poss}\{\tilde{B} \text{ is } \tilde{A}\} = \sup_{x \in U}[\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))]$$

Where the possibility of $\tilde{B}$ being $\tilde{A}$ is obtained by: for each data item x∈U (universe), we get a minimum of two degrees of conformity $\mu_{\tilde{A}}(x)$ and $\mu_{\tilde{B}}(x)$, and the possibility measure is the maximum of these minimums. Computing the overall degree of satisfaction between a fuzzy query and the fuzzy data needs to aggregate all fuzzy conditions based on their degrees of importance and degrees of matching. The fuzzy weighted average (FWA) [11] is applied to calculate the overall degree of satisfaction using triangular fuzzy numbers. In FQHR, the matching degrees of all fuzzy conditions are indicators (x_i) that rate the overall degree of satisfaction between a fuzzy query and the fuzzy data. The degrees of importance are weights (w_i) that act upon indicators. Therefore, the fuzzy weighted average y can be defined as

$$y = f(x_i,...,x_n,w_1,...,w_n) = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

Where there is n fuzzy conditions, the degree of matching x_i, $1 \le i \le n$ is represented by a crisp value or a triangular fuzzy number, and the degree of importance w_i, $1 \le i \le n$ is represented by a triangular fuzzy number. FQHR adopts the approximate expressions on ⊕ and ⊗ operators for the computation of L-R fuzzy numbers, which is suggested by Dubois and Prade [12]. By applying FWA to calculate fuzzy data's overall degrees of satisfaction to a fuzzy query, the ordering of all fuzzy data is obtained according to their overall degrees of satisfaction.

III. PARALLEL PROCESSING ON FQHR WEBSITES

In FQHR websites, both the matching of fuzzy conditions with fuzzy data and the aggregating of all fuzzy conditions are implemented as FuzzyCLIPS rules. Therefore, the number of fuzzy data needed to compute the degrees is the main factor that affects the response time of making a fuzzy query. A large amount of fuzzy data may increase the execution time substantially. For example, the matching of two continuous fuzzy sets can be accomplished by the fuzzy-intersection function in FuzzyCLIPS as follows.

```
(defrules match-salary-preference
  (query (qid ?qid) (salary-preference ?sp))
  (job (jid ?jid) (salary-offer ?so))
  =>
  (bind ?x (get-fs-value (fuzzy-intersection ?sp ?so)
           (maximum-defuzzify (fuzzy-intersection ?sp ?so))))
  (assert (salary-match (qid ?qid) (jid ?jid) (degree ?x))))
```

Parallel FuzzyCLIPS [4] use the SPMD computational model (Single Program Multiple Data) [13] and the MPI library [5] to develop a parallel FuzzyCLIPS programming language on cluster systems. To improve the response time, we apply Parallel FuzzyCLIPS to construct the architecture of FQHR websites for parallelizing the execution of fuzzy queries in grid and cluster environments (see Figure 7).

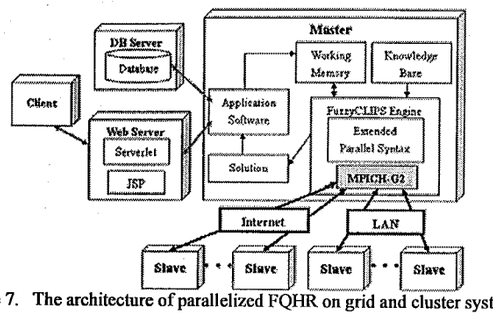

Figure 7. The architecture of parallelized FQHR on grid and cluster systems

In parallelized FQHR, clients can input fuzzy data (resumes or jobs) and make fuzzy queries via a browser. The web server receives requests from clients and controls the system flow. The application software in the master node receives system messages from the web server and accesses the database in the DB server. All retrieved data are translated into FuzzyCLIPS facts that are loaded into the working memory in the master node. The knowledge base in each computing node contains all FuzzyCLIPS rules that implement the matching and the aggregating of fuzzy conditions. We propose to embed the MPICH-G2 library [14] into the FuzzyCLIPS inference engine. MPICH-G2 is able to deal with the parallelism on both grid and cluster systems. New defined syntax mentioned in Section III is implemented to transmit messages between processes in the MPICH-G2 library. The slave node can be any computer in the Internet or LAN. The master node first executes *(makePartition <data_size>)* to allocate facts into slave nodes by load balancing. The master node then executes *(packFact <rank_of_receiver> <a_fact>)* to buffer all allocated facts into the corresponding buckets for slave nodes. The master node executes *(packageSendTo)* in FuzzyCLIPS to call *MPI_Send()* in MPICH-G2 for sending all packages of facts. Each slave node executes *(packageRecvFrom <rank_of_sender>)* to call *MPI_Recv()* for receiving its allocated facts. Accordingly, the FuzzyCLIPS inference engine in each slave node can then use its allocated facts to execute the FuzzyCLIPS rules in parallel. Finally, the results generated by slave nodes are transmitted back to the master node through MPICH-G2 functions. The master node gathers the returned results to form the solution and displays the solution in the web page.

The partitioning of parallelized task content can be varying. Different partitioning approaches may suit to different cases or affect the performance of parallel processing. In FuzzyCLIPS, the partitioning of tasks could be data or rules. Therefore, we propose two kinds of task partitioning approaches for FuzzyCLIPS applications: data level and module level. In data-level task partitioning, the facts will be partitioned and allocated to multiple processors for parallel execution. In module-level task partitioning, the rules are partitioned and allocated to different processors for parallel execution.

A. The Data-Level Parallel Execution

Transmitting a large amount of facts may affect the performance of parallel processing. Our data-level approach transmits the indexes of fact partitions instead of real facts. We define a new template for indexes of fact partitions as follows.

```
(deftemplate (partition (slot rank) (slot from) (slot to)))
```

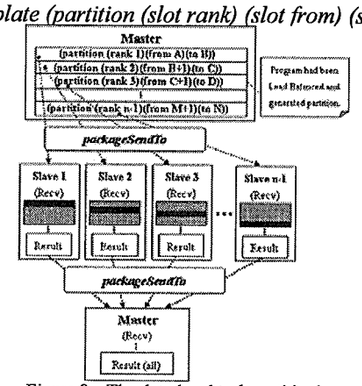

Figure 8. The data-level task partitioning

```
(defrule Package
  (declare (salience 800))
  (MPI (RANK 0))
  ?f<- (partition (rank ?r)(from ?) (to ?))
  =>
  (retract ?f)
  (packFact ?r ?f))
(defrule Send
  (declare (salience 700))
  (MPI (RANK 0))
  =>
  (packageSendTo))
(defrule Receive
  (MPI (RANK ?r))
  =>
  (packageRecvFrom 0))
(defmodule DELETE (import MAIN ?ALL) (export ?ALL))
(defrule DELETE::delete-not-allocated-data
  (MPI (RANK ?r))
  (partition (rank ?r) (from ?f) (to ?t))
  ?fact <- (DATA (ID ?n&:(or (< ?n ?f) (> ?n ?t))))
  =>
  (retract ?fact))
```

Figure 9. The implementation of data-level task partitioning

Each *partition* fact indicates the index range (i.e. [*from*,*to*]) of the allocated facts for the *rank* of a slave node (see Figure 8). The master node doesn't transmit real facts to slave nodes but transmit each partition to the corresponding slave node. That is, the master node only transmits one *partition* fact to each slave node. The transmission time can thus be reduced. Each slave node can execute the *reset* command to assert all facts in the working memory and then delete not-allocated facts according to the index range of the received *partition* fact. The results generated by slave nodes are sent back to the master node. The master node gathers all returned results to form the solution. For implementing the data-level task partitioning, we add a *DELETE* module for each node to delete not-allocated facts according to its received *partition* fact (see Figure 9).

B. The Module-Level Parallel Execution

To avoid unnecessary coupling among parallelized rules, our module-level approach utilizes the *defmodule* construct to partition rules into different modules where each module has its own agenda. Execution can then be controlled by selecting a certain module's agenda for executing rules. Instead of data, the programmer can allocate different modules of rules to different processors by using the *focus* command.

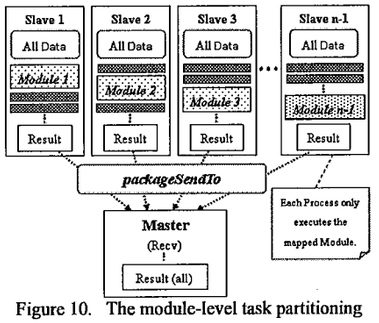

Figure 10. The module-level task partitioning

The example in Figure 10 illustrates how to execute partitioned rules in parallel for a FuzzyCLIPS program. All rules can be partitioned into n modules. Each module is allocated to one computing node. Each computing node can execute the *reset* command to assert all facts in the working memory, but it only *focus* its allocated module for execution. The results generated by slave nodes are sent to the master node. The master node gathers all returned results to form the solution.

```
(deftemplate MPI (slot RANK) (slot PROCS))
(defmodule Module0)
(defmodule Module1)
(defmodule Module2)
(defrule MAIN::MODULE_ASSIGNMENT_Module0
  (MPI (RANK 0))
  =>
  (focus Module0))
(defrule MAIN::MODULE_ASSIGNMENT_Module1
  (MPI (RANK 1))
  =>
  (focus Module1))
(defrule MAIN::MODULE_ASSIGNMENT_Module2
  (MPI (RANK 2))
  =>
  (focus Module2))
```

Figure 11. The implementation of module-level task partitioning

For implementing the module-level task partitioning, we add a module to group rules for each computing nodes (see Figure 11). At the beginning of execution, every node must execute the *MAIN* module. In the *MAIN* module, we define rules for assigning modules to the corresponding nodes based on their ranks. Each node will *focus* its allocated module. Therefore, partitioned rules can be executed in parallel.

IV. PERFORMACE ANALYSIS

A. Performance Analysis for Data-Level Parallel Execution

The data-level method partitions facts and allocates them to multiple processors for parallel execution. Let T_a, T_b, T_c, T_d, and T_e be the time spent in each of five phases, respectively. The analytic costs for all phases are as follows.

1) **The time spent for packing all *partition* facts in the master node.** The total number of processors is denoted as P and the time for packing one fact is denoted as T_{pack}. Each computing node needs a *partition* fact to indicate the index range. The master node needs to send P-1 *partition* fact to P-1 slave nodes, respectively. Therefore, the total packing time T_a for P-1 facts is
$$T_a = (P-1) \times T_{pack}$$
2) **The time spent for sending facts to slave nodes.** The master node calls *packageSendTo* function to send a coresponding packed *partition* fact to each slave node, and slave nodes call *packageRecvFrom* function to receive it. Since the *packageSendTo* function is implemented by the local complete function, the transmissions to different slave nodes are overlapped. The communication startup time is denoted as α and the transmission time for sending one fact is denoted as β. Therefore, the total sending time T_b for P-1 facts is
$$T_b = \alpha + \beta$$
3) **The time spent for deleting facts in computing nodes.** Each node needs to delete not-allocated facts according to its *partition* fact. All nodes execute the deleting processes in parallel. Through load balancing, each node's performance ratio can be obtained by $R_i = S_i / \sum_{j=1}^{P} S_j$. Where R_i is the performance ratio for each processor ranked i ($1 \le i \le P$ and $0 \le R_i \le 1$) and S_i is the execution efficiency for processor i to execute the NAS Parallel Benchmark (NPB) [15]. The total number of facts is denoted as D and the time for deleting one fact is denoted as T_{delete}. Therefore, the total deleting time T_c for P nodes is
$$T_c = \max_{1 \le i \le P}(D \times (1 - R_i) \times T_{delete})$$
4) **The time spent for matching allocated fuzzy data with all fuzzy conditions and aggrating the matching results in computing nodes.** Each node needs to match allocated facts with 7 fuzzy conditions in FQHR. The execution time for processor i to match the size x of fuzzy data with 7 fuzzy conditions is denoted as $TD_{i,x}$. The aggregation for one fact needs to add up 7 matching