Parallel ILP System for Super Linear Speed Up

Hiroyuki Nishiyama and Hayato Ohwada

Faculty of Sci. and Tech. Tokyo University of Science†,
2641 Yamazaki, Noda-shi,
CHIBA, 278-8510, Japan
hiroyuki@rs.noda.tus.ac.jp, ohwada@rs.tus.ac.jp,

Abstract. In our study, we improve our Parallel Inductive Logic Programming (ILP) System to enable super linear speedup. This improvement redesigns several features of our ILP learning system and our parallel mechanism. The redesigned ILP learning system searches and gathers all rules that have the same evaluation in the learning. The redesigned parallel mechanism adds a communication protocol for sharing the evaluation of found rules. We can thus realize super linear speedup.

1 Introduction

Inductive Logic Programming (ILP) is a great supervised learning tool. However, learning for big problems usually needs a very long time. In addition, if we want to generate good rules, we must do continuous learning while changing parameters of ILP learning. It is thus necessary to shorten ILP learning time. To solve this problem, various studies have been conducted in an effort to speed up ILP by parallel methods [1–3, 6, 7, 9]. Although the problems were partially solved, the process was not sufficiently sped up based on the number of processors provided, and the quality of the generated rules was not optimal, resulting in difficulty in its use as a practical tool. Skillicorn and Wang [8] succeeded in super linear speedup, but there were only 4 or 6 CPUs, and it performed poorly when the used dataset was large.

In our study, we improved our Parallel ILP System [6, 7] to enable super linear speedup. This improvement redesigned several features of our ILP learning system [5] and our parallel mechanism [6]. The redesigned ILP learning system searches and gathers all rules that have the same evaluation in the learning. The redesigned parallel mechanism adds a communication protocol for sharing the evaluation of found rules. To estimate the speedup, we used dairy cattle data (hormones, feed, activity, etc.) to determine successful conditions for artificial insemination [4]. When we used 30 CPUs to solve a middle-size problem (33,333sec for 1 CPU) and a large problem (147,992sec for 1 CPU), we achieved a speedup of 31.66 times (1,053sec) and 48.10 times (3,077sec). Using our parallel ILP system, we thus succeeded in super linear speedup and demonstrated it to be more useful for large problems. In addition, we obtained the same rules perfectly using 1 CPU and 30 CPUs.

2 Improved ILP System

An ILP system finds a hypothesis from a bound hypothesis space. The ideal hypothesis covers as many positive examples as possible and as few negative examples as possible. Let $p(h)$ and $n(h)$ be the numbers of positive and negative examples covered by hypothesis h. The number of literals in h is denoted by $c(h)$. We express it as where $g(h)$ indicates the generality of h and $f(h)$ indicates the compression.

$$g(h) = p(h) - c(h),$$

$$f(h) = g(h) - n(h)$$

In our ILP system [5], we use compression measure $f(h)$ to evaluate hypothesis h. If an evaluation (a value of $f(h)$) is the best in a bound hypothesis space, hypothesis h is the best hypothesis in the space.

However, some hypotheses have the same evaluation that is the best value in a bound hypothesis space. In this case, the ILP system usually keeps the hypothesis that is found first (hypotheses after the second are dropped). However, the hypotheses after the second are potentially good rules. In addition, if the order of finding hypotheses is changed, the found rules are changed in the same problem.

Our new ILP system searches and gathers all rules that have the same (best) evaluation in the learning. In the search, if our system finds a hypothesis that has the same evaluation that is the best value, the system does not drop the hypothesis but stores it. Our system then finds some hypotheses in a bound hypothesis space. Using this method, the found rules are never changed in the same problem even if the order of finding hypotheses is changed. In addition, we succeeded in shortening the learning time by 10% (using a CUP).

3 Improvement of Parallel ILP System

3.1 Our previous parallel ILP system

Our previous research [6] designed and implemented a parallel-processing system for ILP. Figure 1 presents the system, which consists of a master-module and worker-modules (please refer to previous research [6] for details). With this system, the worker-module itself has an autonomous function (unlike Map-Reduce). When a worker-module has no task (e.g., immediately after start-up or immediately after completion of a task), the worker-module accepts a divided task from another worker-module and starts the task. When the workload (the number of relationships in information that the worker must search for) reaches a fixed quantity, the worker-module requests other worker-modules to process the divided task. After the request for the first task issued by a master-module is implemented for one worker-module, autonomous process distribution is started among worker-modules, and all existing worker-modules are engaged (saturation

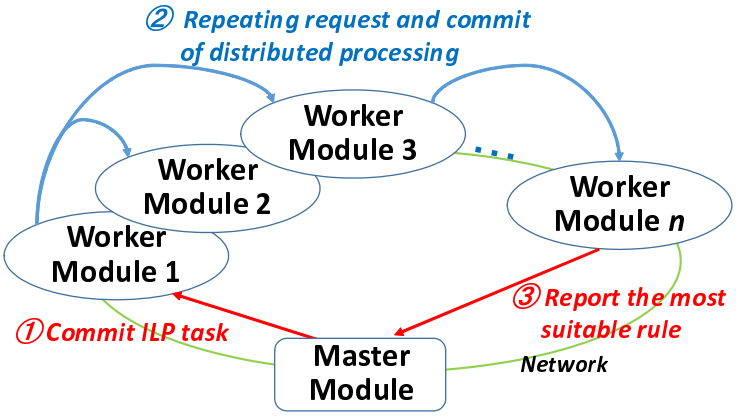

Fig. 1. System configuration of the ILP parallel computation system and flow of distributed processing.

of the task). Because the divided task continues to be repeated among worker-modules, all worker-modules finally complete all processing at approximately the same time, and the master-module receives the processing result (generated rules).

This parallel processing system has the following features.

- 1: The master does not need to consider the division of the process in advance.
- 2: All workers work until the end (no free time) and are finished at the same time.

The first feature indicates that the proposed system does not require the master to perform pre-division processing. The second feature means that the process can be speeded up by increasing the number of computers used.

3.2 Improvement of the Parallel ILP System Communication Protocol

In the previous parallel ILP system, each worker-module has the best evaluation of found hypothesis in its own module. Figure 2 shows one learning situation of our previous parallel ILP system. In this situation, the value of worker-module 3 is 20, but other worker-modules have smaller values. In the branch-and-bound search, the other worker-modules search for useless areas in the bound hypothesis space. This system thus could not speedup sufficiently even if all worker-modules were finished at the same time.

Our new parallel ILP system added a new communication protocol for sharing the best evaluation of found hypotheses. Figure 3 shows one learning situation of our new parallel ILP system. In this situation, worker-module 3 finds

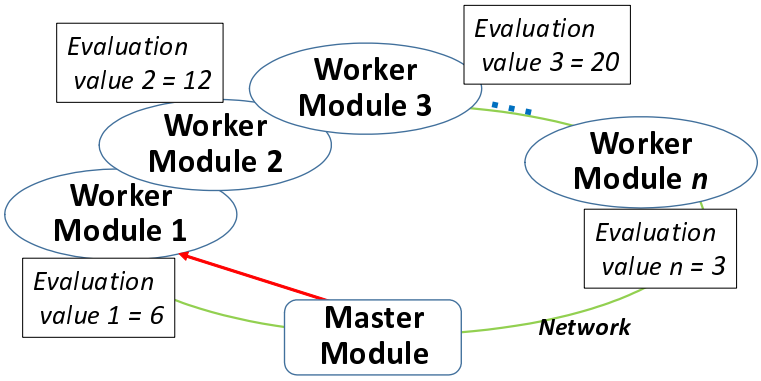

Fig. 2. One learning situation of our previous parallel ILP system. Each worker-module has the best evaluation individually.

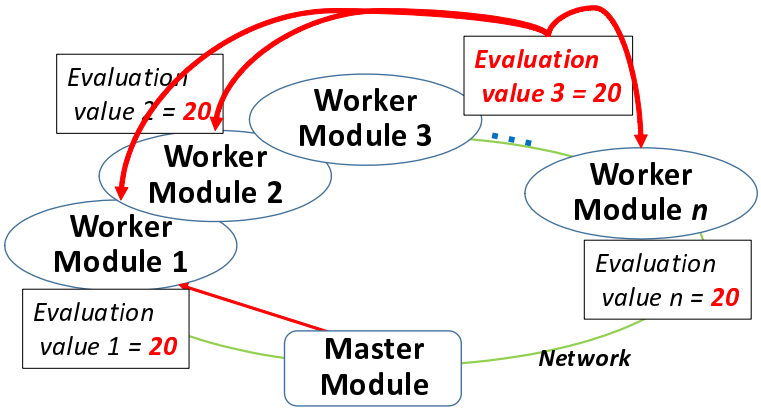

Fig. 3. One learning situation of our new parallel ILP system. Each worker-module shares the best evaluation. All worker-modules thus have the same value.

a hypothesis and the evaluation is 20, the best evaluation of the module and other modules. Worker-module 3 then sends the value to all modules for sharing the best value. When a worker-module receives a value that is better than its own value, the worker updates its own value to the better value and drops its own found hypotheses. This means that the new ILP system can avoid searching for useless areas in the bound hypothesis space. The new system can therefore speedup sufficiently by increasing the number of computers used.

4 Experiment and Results

We used 30 computers (CPU Core i7 5820K 6core/12thread 3.3GH z 64GBRAM) in an experiment to measure speedup. To estimate the speedup, we used data (hormone, feed, activity, etc.) of dairy cattle to determine successful conditions for artificial insemination [4]. In these problems from the data, we use a middle-size problem (33,333sec by 1 CPU) and a large problem (147,992sec by 1 CPU) in the experiment. We successfully speeded up 31.66 times (1,053sec) and 48.10 times (3,077sec) using 30 CPUs. Our original purpose was linear speedup. However, we succeeded in super linear speedup and demonstrated it to be more useful in large problems using our parallel ILP system. This shows all worker-modules can divide a learning task and work ideally without searching for useless area in the bound hypothesis space. In addition, we succeeded in getting the same rules perfectly by using 1 CPU and 30 CPUs. This shows the found rules are not changed in the same problem even if the order of finding hypotheses is changed.

5 Conclusions

In our study, we improved our Parallel ILP System to enable super linear speedup. This improvement redesigned several features of our ILP learning system and our parallel mechanism. The redesigned ILP learning system searches and gathers all rules that have the same evaluation in the learning. The redesigned parallel mechanism added a communication protocol for sharing the evaluations of found rules. To estimate the speedup, we used data of dairy cattle indicating successful conditions for artificial insemination. Finally, we succeeded in super linear speedup and demonstrated that it was more useful in large problems. In addition, we succeeded in getting the same rules perfectly by using 1 CPU and 30 CPUs. Now we are designing an intelligent grid search system using this parallel ILP to acquire more good rules for dairy-cattle problems.

Acknowledgments

This research was supported by grants from the Project of the Bio-oriented Technology Research Advancement Institution, NARO (the special scheme project on advanced research and development for next-generation technology).

References

1. Andreas Fidjeland, Wayne Luk and Stephen Muggleton: Customisable Multi-Processor Acceleration of Inductive Logic Programming, *Latest Advances in Inductive Logic Programming*, pp. 123-141, 2014.
2. Nuno A. Fonseca, Fernando M. A. Silva and Rui Camacho: Strategies to Parallelize ILP Systems, *ILP 2005*, pp. 136-153, 2005.
3. Nikos Katzouris, Alexander Artikis and Georgios Paliouras: Distributed Online Learning of Event Definitions, *Computing Research Repository (CoRR)*, abs/1705.02175, 2017.
4. Atsushi Matsumoto, Chikara Kubota and Hayato Ohwada: Extracting Rules for Successful Conditions for Artificial Insemination in Dairy Cattle Using Inductive Logic Programming, Proc. of the 9th International Conference on Machine Learning and Computing, pp. 6-10, 2017.
5. Fumio Mizoguchi and Hayato Ohwada: Constrained Relative Least General Generalization for Inducing Constraint Logic Programs, *New Generation Computing 13*, pp. 335-368, 1995.
6. Hiroyuki Nishiyama and Hayato Ohwada: Yet Another Parallel Hypothesis Search for Inverse Entailment, *CEUR Workshop Proc. of the Late Breaking Papers of the 25th International Conference on ILP Vol. 1636*, pp. 86-94, 2016.
7. Hayato Ohwada, Hiroyuki Nishiyama and Fumio Mizoguchi: Concurrent Execution of Optimal Hypothesis Search for Inverse Entailment, *Inductive Logic Programming, Lecture Notes in Computer Science Vol. 1866*, pp. 165-173, 2000.
8. David B. Skillicorn and Yu Wang: Parallel and Sequential Algorithms for Data Mining Using Inductive Logic, *Knowledge and Information Systems, Vol. 3, No. 4*, pp. 405-421, 2001.
9. Ashwin Srinivasan: Parallel ILP for distributed-memory architectures, *Machine Learning, Vol. 74, Issue 3*, pp 257-279, 2009.