Multi-Parser Architecture for Query Processing

Kui Xu, Fuliang Weng*, Helen Meng**, and Po Chui Luk***

*Intel China Research Center, Intel Corporation, Beijing, China
**Human-Computer Communications Laboratory
The Chinese University of Hong Kong, Hong Kong SAR, China.
{kui.xu, fuliang.weng}@intel.com; {hmmeng, pcluk}@se.cuhk.edu.hk

Abstract

Natural language queries provide a natural means for common people to interact with computers and access to on-line information. Due to the complexity of natural language, the traditional way of using a single grammar for a single language parser leads to an inefficient, fragile, and often very large language processing system. Multi-Parser Architecture (MPA) intends to alleviate these problems, and the modularized MPA also has the advantage of easier portability to new domains and distributed computing. In this paper, we investigate the effect of using different types of parsers on different types of query data in MPA. Three data sets and two types of sub-parsers have been examined. Results show that partitioning grammars leads to superior speed performance for the Earley parser[1] across the three data sets. The GLR parser is faster than the Earley parser for the partitioned grammar, but the GLR parser may have excessive memory usage for the un-partitioned grammars.

1. Introduction

Natural language queries, sometimes as the back-end to speech recognizers, provide a natural means for common people to interact with computers and access to on-line information. Due to the complexity of natural language itself, the grammar that describes the query language can be very complex. The traditional way of using a single grammar leads to an inefficient, fragile, and often very big processing system. These problems become more apparent, with the increasing demand for natural language applications over the Internet.

Various methods that deal with these parsing issues have been studied, e.g., [1][2][7][12]. The proposed Multi-Parser Architecture (MPA) by [5][12][13] intends to alleviate aforesaid problems simultaneously by partitioning a single grammar into multiple sub-grammars and composing sub-parsers for the sub-grammars. Sub-parsers can be combined by parser composition methods such as *cascading* and *predictive pruning* [5]. The MPA is highly modularized with re-usable sub-grammars and this is advantageous for distributed computing and portability to new application domains. Our current work presents several enhancements: (i) a methodology for automatic grammar partitioning; (ii) an improved parser composition method known as *predictive cascading*; and (iii) the use of an Earley parser in addition to the GLR parser used previously. Experiments were conducted with three data sets – a keyword list, a semantic grammar and a syntactic grammar. Results show that grammar partitioning with composition of Earley parsers can speed up processing substantially for both semantic and syntactic grammars when compared to the un-partitioned grammars. Furthermore, while the GLR parser is faster than Earley; the latter utilizes memory more efficiently.

[1] Hereafter we refer to the pre-complied Earley parser simply as the Earley parser.

2. The Multi-Parser Architecture

MPA involves the two main processes of grammar partitioning and parser composition. Grammar partitioning divides a grammar into multiple sub-grammars. Each sub-grammar is used by its corresponding sub-parser, and parsing results across sub-parsers are composed to produce an overall parse output for a natural language query. The interaction among sub-grammars/sub-parsers is achieved by applying a virtual terminal technique. The virtual terminal is essentially a non-terminal, but acts as if it were a terminal. The INPUT set to a sub-grammar is a set of virtual terminals that were previously parsed by other sub-grammars. The OUTPUT set of a sub-grammar is a set of non-terminals that are parsed based on this sub-grammar; and used by other sub-grammars as their INPUT sets. Hence a partition (subset) of production rules can be viewed as a multi-valued function – it takes the virtual terminals in the INPUT set as input, and returns a set of non-terminals in the OUTPUT set as output. Results on manually partitioned grammars were presented previously [5][13]. In this paper, we will present a method for automatic grammar partitioning and related experimental results.

Two methods for parser composition have been presented in [5][13]: composition by cascading and composition by predictive pruning. Cascading is a bottom-up parsing procedure starting from the terminal level to and moves to the SENTENCE level. It begins by converting the input sentence into a lattice called LMG, and invokes parsers at each lattice position. During parsing, newly created virtual terminals *(vt)* are added dynamically to the stack and LMG. The stack stores the terminals and virtual terminals according to the topological order of the LMG. Cascading has the advantage of parser robustness, but is relatively slow due to excessive parser invocation at each lattice (LMG) position. To avoid this, predictive composition is a top-down procedure where the caller sub-parser invokes a callee sub-parser only if the latter satisfies the constraint that the input node must be its left corner. Sub-parsers that do not satisfy the constraint are pruned. While the top-down approach has the advantage of execution efficiency, it may at times be too constrained to be robust. In subsection 2.2, we present an improved parser composition method known as *predictive cascading*.

2.1. Automatic Grammar Partitioning

Our investigation on automatic grammar partitioning involves a syntactic grammar derived from the Penn Treebank ATIS subset (www.ldc.upenn.edu) and is different from the semi-automatically derived semantic grammar used in our previous work [5][13].

The Penn Treebank contains a subset of the ATIS-3 corpus, which is a set of 577 parsed queries including Class A (self-contained) and Class D (dependent on discourse context) queries. Parse trees of these queries are provided, with the tree terminals being part-of-speech (POS) tags. A parse tree example is shown in Figure 1. Our grammar rules are extracted from the parse trees. For the sake of simplicity we ignore the null elements XXX as well as co-indexing in the grammar rules. Hence WHNP-1 is treated the same as WHNP in Figure 2. As we extract the grammar rules from the training parse trees, we also record the frequency of interaction between rules, e.g. Rule 2 in Figure 2 (PP-DIR→#IN NP) has called Rule 1 (NP→NNP) once. Here the INPUT set of Rule 2 is the non-terminal NP and the OUTPUT set is PP-DIR.

Our automatic grammar partitioning procedure begins with the set of finest grammar partitions, i.e. each partition contains exactly one grammar rule. Then we attempt to cluster the sub-grammars to form larger ones based on the frequencies of their interactions. In other words, we want to cluster grammar partitions that have frequent caller/callee interactions into a larger sub-grammar. This procedure references a calling matrix, where the entry at row *i* and column *j* is the frequency of sub-grammar *i* calling sub-grammar *j* (see Figure 3).

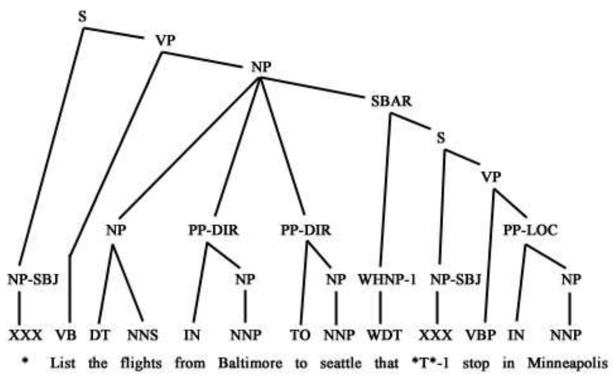
S
VP
NP
SBAR
S
VP
NP
PP-DIR
PP-DIR
PP-LOC
NP-SBJ
NP
NP
WHNP-1
NP-SBJ
NP
XXX VB DT NNS IN NNP TO NNP WDT XXX VBP IN NNP
* List the flights from Baltimore to seattle that *T*-1 stop in Minneapolis

Figure 1: An example parse tree drawn from an ATIS sentence from the Penn Treebank.

There are three steps in the clustering procedure:

1. The initial step first checks for every sub-grammar with an empty input set, then duplicates and merges the sub-grammar for each of its caller sub-grammars. For example, grammar partition 4 in Figure 2 is (WHNP→#WDT). Its INPUT set is empty (since #WDT is a POS terminal), and its OUTPUT set is {WHNP}. For grammar partition 8, the INPUT set is {WHNP S} and the OUTPUT set is {SBAR}. Hence partition 8 is a caller of 4, and we merge them to form partition 11 with INPUT set {S} and OUTPUT set {SBAR}:

Grammar partition 11:
INPUT = {S}
OUTPUT ={SBAR}

SBAR → WHNP S	(from partition 8)
WHNP → #WDT	(from partition 4)

We then update the calling matrix as shown in Figure 4. Columns 4 and 8 are deleted and column 11 is added. The other entries remain unchanged.

0: NP → #DT #NNS
1: NP → #NNP
2: PP-DIR → #IN NP
3: PP-DIR → #TO NP
4: WHNP → #WDT
5: PP-LOC → #IN NP
6: VP → #VBP PP-LOC
7: S →VP
8: SBAR →WHNP S
9: NP →NP PP-DIR PP-DIR SBAR
10: VP → #VB NP

Figure 2: Grammar rules extracted from the parse tree in Figure 1.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	1
8	0	0	0	0	1	0	0	1	0	0	0
9	1	0	1	1	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0	0	1	0

Figure 3: Calling matrix of the grammar in Figure 2.

	0	**1**	**2**	**3**	**5**	**6**	**7**	**9**	**10**	**11**
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	1	0	0	1	0
9	1	0	1	1	0	0	0	0	0	1
10	0	0	0	0	0	0	0	1	0	0
11	0	0	0	0	0	0	1	0	0	0

Figure 4: Updated calling matrix of the grammar in Figure 2.

2. The next step in our clustering procedure is an iterative step. We find the maximum value in the calling matrix, and merge the two sub-grammars into a new grammar. Merging is conditioned upon two thresholds: (i) the

frequency in the matrix must be greater than the minimum count X (set at 3) ; and (ii) the size of the merged grammar must lie below the threshold Y (set at 100) to avoid infinite growth. Grammar size is measured according to Equation (1). *P* is the set of production rules, and *length(x)* is the length of string x.

$$|G| = \sum_{(A \to \alpha) \in P} length(A\alpha) .....Eqn(1)$$

3. The final step in our clustering procedure is to merge two sub-grammars if the OUTPUT set of one contains all the non-terminals in the OUTPUT set of the other.

As such we obtained 46 grammar partitions based on the syntactic rules derived from the 577 Penn Treebank ATIS training parse trees.

2.2. Parser Composition by Predictive Cascading

Predictive cascading is a parser composition method that combines the merits of predictive pruning and cascading. In this process, all edges of the lattice are topologically sorted into a stack, just as simple cascading, but in reversed topological order. The LMG will then be parsed from the last edge to the first one. When one sub-parser parses successfully, an output virtual terminal (a new edge) will be placed on the LMG and added to the stack. This edge will be tested later by a predictive procedure that determines whether or not the edge is a left corner of a sub-grammar (or a callee sub-parser). The left corner of a sub-grammar is the first terminal of a string derived from the sub-grammar, and its corresponding sub-parser can be invoked for further parsing.

3. Experiments

Our experiments include two styles of parsers. The first one is the Earley parser, where the prediction step is pre-compiled. The other is the GLR parser [8][9]. These two parsers have different characteristics. The Earley parser does not encode all the prefix paths in a table, while the GLR parser tries to do it as much as possible. This usually makes GLR faster but bigger when compared with Earley parser.

Three different data sets are used for our experiments. The first one is a Chinese keyword list grammar for key words extraction; the second is a semi-automatically derived ATIS semantic grammar [5]; the third is the fully annotated Penn Treebank ATIS parse trees. All the grammars have two versions -- one is partitioned, and the other is un-partitioned.

Two sets of experiments are conducted. The first one is to compare the speed between the GLR parser and the Earley parser on partitioned grammars. The other is to compare the speeds between the partitioned approach and the un-partitioned approach for different grammar sets. The size of the GLR parsing table for the Penn Treebank ATIS grammar is excessive, so we only use the Earley parser for the effect on speed. Both experiments use Intel Pentium Ⅲ 933MHz processor with 512 Mbytes memory and the Windows 2000 Professional operating system.

We describe the three grammars, their partitioning methods and the experimental setups below.

(i) The Chinese keyword list is obtained from an international news corpus which has 15,426 keywords (rules). For partitioning, all the keywords were first sorted in alphabetical order. Then, the list of keywords is split into 78 sub-grammars with 200 keywords each. The non-terminals are labeled as KEYWORD_i, where *i* is the index of the sub-grammar. Here are some examples of the keyword rules. The testing corpus includes 200 handcrafted Chinese queries. The average sentence length is 10.6 characters. The maximum sentence length is 19 characters.

$\text{KEYWORD}_0 \rightarrow$ 巴 拉 克
$\text{KEYWORD}_0 \rightarrow$ 巴 勒 斯 坦
$\text{KEYWORD}_0 \rightarrow$ 半 岛
$\text{KEYWORD}_0 \rightarrow$ 办 法
$\text{KEYWORD}_0 \rightarrow$ 包 括
$\text{KEYWORD}_0 \rightarrow$ 饱 经 风 霜
$\text{KEYWORD}_0 \rightarrow$ 饱 受

(ii) The ATIS semantic grammar is a set of context-free rules that contain both semantic and syntactic structures. The low level grammar rules are mainly semantic concepts, such as CITY-NAME, CLASS-TYPE, MONTH-NUMBER, etc. They are obtained by a semi-automatic grammar induction algorithm [5]. In this experiment, we partitioned the 1,297 semantic rules into 64 sub-grammars. Examples of the English ATIS-3 rules are given below. For the semantic grammar experiment, the ATIS 1993 Class A data is used. 1564 ATIS training set has been used as the test sentences. Average sentences length is 11.2 words. Max length is 46 words.

S → ASK FLIGHT_NP|...
ASK → show me | list | tell me | give me | ...
FLIGHT_NP → FLIGHT FLIGHT_PP
FLIGHT → flight | flights | flight number | ...
FLIGHT_PP → DEPARTURE | ARRIVAL | ...
DEPARTURE → leaving CITY_NAME | ...
CITY_NAME → phoenix | new york | seattle

(iii) The automatically partitioned syntactic grammar is derived from the hand-bracketed Penn Tree bank ATIS-3 subset, with 577 sentences including class-A and class-D semtemces. In this experiment, we partitioned the 416 rules into 46 sub-grammars. Examples of partitioned syntactic grammar are listed below. For the Penn Treebank ATIS case, most of the ATIS sentences, i.e. 500 out of 577 POS tags sentences, are used as the test sentences. The average sentences length is 6.5 tags, and the max length is 13 tags. The full parse coverage is 99.7% in both partitioned grammar and no partitioned grammar.

S → VP
VP → VB NP NP
NP → DT NNP NNP NNP NNS
NP → RB DT NNP NNP NNS
NP → DT NNP NNP NNS
NP → DT JJS JJ NN NNS
NP → JJ NNS
NP → NNP

Our first set of experiments compare the speed of the Earley parser against that of the GLR parser for partitioned grammars. Results show that the GLR parser runs faster in most cases, especially when the grammar is more complex (see Table 1).

	Earley # of sentences per second	GLR # of sentences per second
Keyword list	40.00	25.0
ATIS semantic grammar	42.30	55.9
ATIS Penn Tree Bank syntactic grammar	1.99	9.8

Table 1: Comparison between the Earley parser and the GLR parser in terms of speed.

The second set of experiments compare the speeds between partitioned and unpartitioned grammars. Results show that the Earley parser runs faster for the partitioned case than for the un-partitioned case (Table 2).

	No partitioning # of sentences per second	Partitioning # of sentences per second
Keyword list	0.016	40.00
ATIS semantic grammar	0.29	42.30
ATIS Penn Tree Bank syntactic grammar	0.22	1.99

Table 2: Comparison of Earley parser's speed for partitioned vs. un-partitioned grammars.

A third experiment was conducted to see the effect of left-corner prediction for Earley parser. The result has shown that using predictive cascading gives 40 times speed up when compared to the previous cascading composition.

4. Comparison and Conclusion

This paper systematically examines the parsing speed based on three different grammar sets (from a simple keyword list to a complicated syntactic grammar), and two different parsers (Earley and GLR). The experimental results show the GLR parser is generally faster than the Earley parser. Earley parsers run faster with partitioned grammars than unpartitioned grammars. The results are consistent for simple grammars and complicated grammars, and for manually partitioned grammars as well as automatically partitioned grammar.

Abney [1] proposed two-level chunking parser, which first converts an input sentence into chunking sequence for lower level processing, then uses an attacher to connect these chunks together for higher level processing. But, no systematic speed result has been reported for the parser. Ruland [7] developed Multi-Parser Multi-Strategy Architecture for noisy input. It uses a full parser first, and then a partial parser when the full parser does not return a result. CMU's Janus parser [14] uses a similar strategy. However, these multiple parsers are used in an ordered fashion, i.e. robust parsing is invoked only after regular parsing fails. Therefore, there is no partitioning in the architecture. Dowding [3] and Moore [6] systematically examined the effect of different left corner constraints on speed. Our results have shown a more significant improvement when using grammar partitioning. We attribute the difference to the block effect, i.e., when a sub-grammar does not meet a left corner constraint, the whole sub-grammar is pruned.

5. References

[1] Abney, S., "Parsing by Chunks", In Principle-Based Parsing: Computation and Psycholinguistics, R. C. Berwick et al.(eds), Kluwer Academic Publishers, 1991.

[2] Amtrup, J. "Parallel Parsing: Different Distribution Schemata for Charts", Proceesings of the 4th International Workshorp on Parsing Technologies, Prague, pp.12-13, Sep 1995.

[3] Dowding, J., Moore, R., Andry, F., and Moran, D., "Interleaving syntax and semantics in an efficient bottom-up parser". In Proc. of the 32 nd Annual Meeting of the Association for Computational Linguistics, pages 110--116, Las Cruces, NM, June27--30 1994.

[4] Korenjak, A., "A Practical Method for Constructing LR(k)", CACM 12, 11, 1969.

[5] Luk, P.C., Meng, H. and Weng, F., "Grammar Partitioning and Parser Composition for Natural Language Understanding". Proceedings of ICSLP, 2000.

[6] Moore, R.C., "Improved Left-corner Chart Parsing for Large Context-free Grammars". Proceedings of 6-th International Workshop on Parsing Technologies, 2000.

[7] Ruland, T., et al "Making the Most of Multiplicity: A Multi-Parser Multi-Strategy Architecture for the Robust Processing of Spoken languages". Proceedings of ICSLP, 1998.

[8] Siu, K.C. and Meng, H. "Semi-automatic acquisition of domain-specific semantic structures". In proc. of Eurospeech, 1999.

[9] Tomita, M. "Efficient Parsing for Natural Language", Kluwer Academic Publishers, MA, 1985.

[10] Tomita, M. "An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition" Proceedings of ICASSP, 1986.

[11] Weng, F.L. "Handling Syntactic Extra-Grammaticality". Proceedings of 3rd International Workshop on Parsing Technologies, 1993.

[12] Weng, F.L. and Stolcke, A. "Partitioning Grammar and Composing Parsers". Proceedings of 4-th International Workshop on Parsing Technologies, 1995.

[13] Weng, F.L., Meng, H. and Luk, P.C., "Parsing a Lattice with Multiple Grammars". Proceedings of 6-th International Workshop on Parsing Technologies, 2000.

[14] Woszczyna, M., Aoki-Waibel, N., Buo, F., D., Coccaro, N., Horiguchi, K., Kemp, T., Lavie, A., McNair, A., Polzin, T., Rogina, I., Rose, C.P., Schultz, T., Suhm, B., Tomita, M., and Waibel, A., "JANUS-93: Towards Spontaneous Speech Translation". In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'94), 1994.