Computing NFA Intersections in Map-Reduce

Gösta Grahne
Concordia University
Montreal, Canada, H3G 1M8
grahne@cs.concordia.ca

Shahab Harrafi
Concordia University
Montreal, Canada, H3G 1M8
s_harraf@encs.concordia.ca

Ali Moallemi
Concordia University
Montreal, Canada, H3G 1M8
moa_ali@encs.concordia.ca

Adrian Onet
Concordia University
Montreal, Canada, H3G 1M8
adrian_onet@yahoo.com

1. INTRODUCTION

Nondeterministic Finite-state Automata (NFA) are simple, yet powerful devices that model a plethora of computationally oriented phenomena. One of the advantages of NFA's is that they are closed under several operations, such as concatenation, intersection, difference, and homomorphic images. This makes NFA's ideally suited for a modular approach, for instance in the context of protocol design and web service composition. A simple, but illustrative example of an e-commerce application designed from components can be found in Chapter 2 in [5]. The salient operation here is the intersection of several finite state automata.

Problems relating to NFA's have been widely studied in the literature. One of the main issues for the NFA intersection problem is that the size of the output NFA is the product of the size of all input NFA's. There is not much hope for improvement, since testing for emptiness of the intersection of a set languages represented by NFA's is known to be PSPACE-complete [8]. The most commonly used algorithm for computing the intersection NFA is to use the *Cartesian construct* for product automata. If there are m input NFA's each having n states, the product NFA will have n^m states. It therefore would be important to come up with good distributed algorithms for the problem.

Google introduced map-reduce as a parallel programming model [4] that can work over large clusters of commodity computers. Map-reduce provides a high-level framework for designing and implementing such parallelism. A growing number of papers deal with map-reduce algorithms for various problems, for instance related to graphs [12, 9, 3, 11], and related to relational joins [2, 6, 7].

In this paper we investigate the problem of implementing the Cartesian construct in map-reduce. We follow the optimization approach of Afrati et al. [1] and analyze the *replication rate* required for computing the product NFA. The replication rate corresponds intuitively to the total amount of communication between the processes in the cluster. We first derive a lower bound for the replication rate in the product computation. We then propose three algorithms for the product computation and analyze their behaviors, thereby obtaining upper bounds for the replication rate. Our study shows that in the case where the size of the alphabet for the NFA's is large and we have a large number of reducers available, an algorithm that distributes the transitions of the input NFA's based on their alphabet symbol achieves an optimal replication rate. For the cases where the alphabet size is smaller than the number of available reducers, a distribution based on both the alphabet symbol and states of the transitions works best. These conclusions are also supported by our experimental results.

The rest of this paper is organized as follows: Section 2 gives the necessary technical definitions, and in Section 3 we derive the lower bound for the replication rate. Section 4 presents and analyzes three concrete algorithms for the problem, and Section 5 describes the experimental results. Conclusions are drawn in the last section.

2. PRELIMINARIES

In this section we introduce the basic technical preliminaries and definitions. We assume familiarity with the map-reduce model (see e.g. [10]).

A *Nondeterministic Finite-state Automaton (NFA)* is a 5-tuple $A = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is a finite set of alphabet symbols, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states. By Σ^* we denote the set of all finite strings over Σ. Let $w = c_1 c_2 \dots c_n$ where $c_i \in \Sigma$ be a string in Σ^*. An *accepting computation path* of w in A is a sequence $(s, c_1, q_1)(q_1, c_2, q_2) \dots (q_{n-1}, c_n, f)$ of elements of δ, where s is the start state and $f \in F$. The *language* accepted by A, denoted $L(A)$, is the set of all strings in Σ^* for which there exists an accepting computation path in A. A language L is regular if and only if there exists an NFA A such that $L(A) = L$.

It is well known that regular languages are closed under intersection. In particular, given NFA's $A_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$, an NFA A, such that $L(A) = L(A_1) \cap L(A_2)$ can be computed by the *Cartesian construct* $A = A_1 \otimes A_2$, where

$$A_1 \otimes A_2 = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2),$$

and

$$\delta = \{((p_1, p_2), c, (q_1, q_2)) : (p_1, c, q_1) \in \delta_1, (p_2, c, q_2) \in \delta_2\}.$$

The $\otimes$ operation clearly is associative, and can be generalized to a polyadic operator $A_1 \otimes \cdots \otimes A_m$. The Cartesian construct amends itself easily to the map-reduce framework by having the mappers emit transitions (p_i, c_i, q_i) from each NFA A_i, and the reducers output a transition $((p_1, \ldots, p_m), c, (q_1, \ldots, q_m))$ upon receiving inputs (p_i, c_i, q_i), where $c = c_1 = \cdots = c_m$. The crucial question is how to distribute the transitions (p_i, c_i, q_i) over the reducers. This is discussed in Section 4.

3. LOWER BOUND ON THE REPLICATION RATE

Recall that each mapper emits *key-value* pairs (K, V), where K determines the reducer that the pair is sent to. Each reducer receives and aggregates *key-value* lists of the form $(K, V_1, \ldots V_q)$, where the (K, V_i) pairs are emitted by the mappers. The largest list associated with one key is called the *reducer size*, and we will denote it by q. A small q-value ensures that the reducer can perform the aggregation in main memory, and also enables more parallelism. On the other hand, more parallelism usually increases the *replication rate*, which is the average number of key-value pairs that mappers create from one input. The replication rate is intended to model the *communication cost*, that is the total amount of information sent from the mappers to the reducers. The trade-off between reducer size q and replication rate r, is usually expressed through a function f, such that $r = f(q)$. The first task in designing a good map-reduce algorithm for a problem is to determine the function f, which gives us a lower bound of the replication rate r.

To start, we derive a tight upper bound, denoted $g(q)$, on the number of outputs that can be produced by a reducer of size q. We suppose that NFA A_i has $|\delta_i|/k$ transitions for each of the k alphabet symbols. To generate a transition for A, the reducer needs m transitions, one from each NFA A_i. The intersection NFA A has $\frac{|\delta_1| \times \cdots \times |\delta_m|}{k^m}$ transitions, for each alphabet symbol $c \in \Sigma$. As there are k alphabet symbols, the total number of transitions will be $k \times \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^m} = \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^{m-1}}$.

It is known that the product of the elements in a partition with a fixed summation is maximum when the blocks of the partition have equal size. We therefore assume that input data is evenly distributed, so each reducer receives q/m transitions from each NFA A_i. The proceeding gives us the following upper bound on the output of one reducer.

LEMMA 1. *In computing $A = A_1 \otimes \cdots \otimes A_m$ a reducer of size q can cover no more than $g(q) = (q/m)^m$ outputs.*

Using Lemma 1, and the total number of transitions in A, we can get a lower bound on the replication rate as a function of q. As shown in [1] the lower bound is given by the expression

$$\frac{q \times |O|}{g(q) \times |I|},$$

where $|I|$ is the size of input, and $|O|$ is the size of the output. The input size will be the sum of the size of the transition relation of all input NFA's, that is $|I| = |\delta_1| + \cdots + |\delta_m|$. As we saw above, the size of the output in terms of the number of transitions will be $|O| = \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^{m-1}}$. This gives us the lower bound on replication rate for our problem as follows

PROPOSITION 1. *The replication rate r for the Cartesian construct $A = A_1 \otimes \cdots \otimes A_m$ is*

$$r \geq \frac{q \times \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^{m-1}}}{(q/m)^m \times (|\delta_1| + \cdots + |\delta_m|)}.$$

4. ALGORITHMS FOR THE CARTESIAN CONSTRUCT

In this section we propose and analyze three different algorithms for computing $A = A_1 \otimes \cdots \otimes A_m$. Our algorithms compute A in one map-reduce round, as opposed to an $m-1$ round cascade $(\ldots(A_1 \otimes A_2) \otimes \ldots) \otimes A_m$. Since the Cartesian construct shares features with the multiway join problem, and the latter has been shown to work more efficiently when done in one round, as opposed to a cascade [2, 6], we only consider the one-round version in this paper.

We note that the main difference between the NFA intersection and the multiway join problem is that in the latter the only possibility for distributing the tuples is based on the value(s) of the join attribute(s) (corresponding to the alphabet symbols in Σ), whereas the NFA intersection problem we can also distribute the tuples of the transition relation based on the states they involve.

4.1 Mapping based on states

Suppose we have n^m reducers, where n is the maximum number of transitions in any of the input NFA's. In our first algorithm the mappers produce keys of the form $(i_1, i_2, \ldots, i_m)$. Let h be a hash-function with range $\{1, \ldots, n\}$. A transition (p_i, c_i, q_i) from NFA A_i is mapped as key-value pairs $(K, (p_i, c_i, q_i))$, where

$$K = (i_1, \ldots, i_{i-1}, h(p_i), i_{i+1}, \ldots, i_m).$$

for each $i_j \in \{1, \ldots, n\}$. In other words, each transition is sent to n^{m-1} reducers.

In this method, the input and output sizes remain unchanged. However, the function $g(q)$ will be affected by presence of transitions with different alphabet symbols inside a single reducer. This gives us a new upper bound on the number of outputs each reducer can produce, namely $g(q) = k\,(q/mk)^m$. We thus have

PROPOSITION 2. *The replication rate r in the state-based mapping scheme is*

$$r \leq \frac{q \times |\delta_1| \times \cdots \times |\delta_m|}{(q/m)^m \times (|\delta_1| + \cdots + |\delta_m|)}.$$

If n is the maximum number of transitions in any of the input NFA's, the upper bound on the replication rate becomes $r \leq (\frac{nm}{q})^{m-1}$.

By comparing propositions 1 and 2, we observe that the upper bound for the replication rate obtained by mapping based on states exceeds the theoretical lower bound by a factor of k^{m-1}. We conclude that the state-based mapping approach is best suited for situations where the alphabet size is small, e.g., when the alphabet is binary.

4.2 Mapping based on alphabet symbols

In our second algorithm, we have one reducer for each of the alphabet symbols. Thus, the number of reducers is equal to the alphabet size k. The mappers will send each transition (p, c, q) to the reducer corresponding the alphabet symbol c. More precisely, from transition (p_i, c, q_i) of NFA A_i the mapper will generate the key-value pair $(h(c), (p_i, c, q_i))$. Here h is a hash function with range $\{1, \ldots, k\}$. Thus each reducer will output transition $((p_1, \ldots, p_m), c, (q_1, \ldots, q_m))$, having received inputs (p_i, c, q_i) for $i = 1, \ldots, m$.

The total number of transitions sent to all reducers is $\sum_{i=1}^{m} |\delta_i|$ which we approximate by mn, assuming that each A_i has at most n transitions. The replication rate is 1, since every transition is mapped to exactly one reducer. This algorithm works well when the alphabet size k is large and the number of reducers is equal to the number of alphabet symbols. In summary:

PROPOSITION 3. *The replication rate in the alphabet-symbol based mapping scheme is 1, assuming that the number of reducers and alphabet symbols are the same.*

Obviously a replication rate of 1 is optimal. This matches the lower bound of Proposition 1, when observing that each reducer has to process $(nm)/k$ inputs, assuming that the alphabet symbols are uniformly distributed. Substituting $q = (nm)/k$ in the lower bound $(\frac{nm}{kq})^{m-1}$ of Proposition 1, gives $r \geq 1$.

4.3 Mapping based on both states and alphabet symbols

On one hand, if we map the transitions only based on the alphabet symbols, the algorithm does not allow for much parallelism if the alphabet Σ is small. On the other hand, as we have observed, if the transitions are mapped based on states only, the replication rate, and consequently the communication cost, will be sharply increased k^{m-1} times. We therefore consider a hybrid algorithm that maps transitions based on a combination of alphabet symbols and states. In the hybrid method we have a function h_s that hashes states into b_s buckets, and a function h_a that hashes the alphabet symbols into b_a buckets. A transition (p_i, c_i, q_i) from A_i is mapped to reducers $(i_1, \ldots, i_{i-1}, h_s(p_i), i_{i+1}, \ldots, i_m, h_a(c_i))$, for each $i_j \in \{1, \ldots, b_s\}$, and the total number of reducers will be $b_s^{m-1} \cdot b_a$.

To compute the replication rate in this method, we note the input and output sizes $|I|$ and $|O|$ remain unchanged. However, the function $g(q)$ will be affected by presence of transitions with different alphabet symbols inside a single reducer. We will now have $g(q) = \ell(q/m\ell)^m$, where ℓ is the average number of alphabet symbols received by a reducer, or equivalently, $\ell = k/b_a$. From this we can derive the replication rate.

PROPOSITION 4. *The replication rate r in the hybrid mapping scheme is*

$$r \leq \frac{q \times \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^{m-1}}}{(q/m)^m \times (|\delta_1| + \cdots + |\delta_m|)} \times \ell^{m-1}.$$

Assuming that the maximum number of transitions in any of the input NFA's is n, we get $r \leq \left(\frac{nm\ell}{qk}\right)^{m-1}$.

Note that if $b_a = 1$ then $\ell = k$ and there is no hashing on alphabet symbols, and as it can be seen, the replication rate will be equal to the replication rate of the first mapping schema. On the other hand, if $b_a = k$, that is if we hash fully on alphabet symbols, then $\ell = 1$ and as it can be seen, the replication rate will be equal to the replication rate of the second mapping schema.

5. EXPERIMENTS

We conducted some experiments to validate the analysis of the previous section. We computed $A_1 \otimes A_2 \otimes A_3$, and varied the size of the NFA's and number of alphabet symbols. Our experiments were run on Hadoop on a 2-node, personal computer, cluster (8 cores per node running at 3.0 GHz and 24GB memory in total). The number of reducers in the experiments was set to 128. The desktops were running Scientific Linux operating system with kernel version 6.0. The NFA's were generated as labelled random graphs, along the lines of [13]. The total number of transitions were determined by the

transition density, that is, the ratio between the number of transitions and the number of states. In the data shown we used a transition density of 2.0.

In the experiments we compared the execution time obtained by hashing the input data based on states (Method I) and on both states and alphabet symbols (Method II).

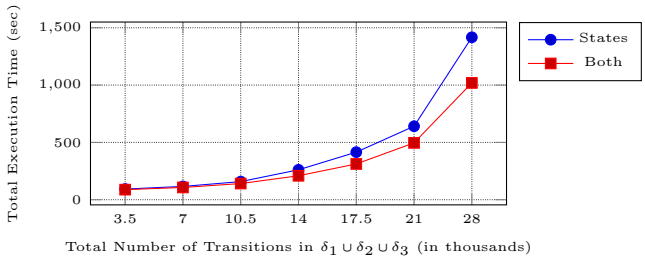

Figure 1: Processing times of two methods for the alphabet size $k = 16$

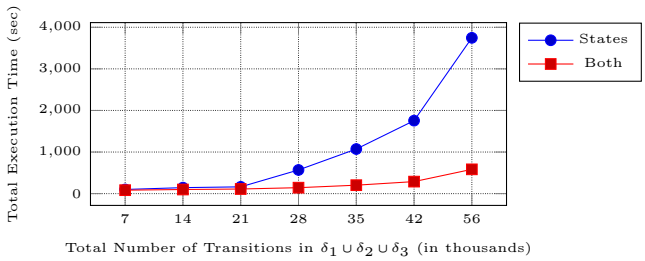

Figure 2: Processing times of two methods for the alphabet size $k = 64$

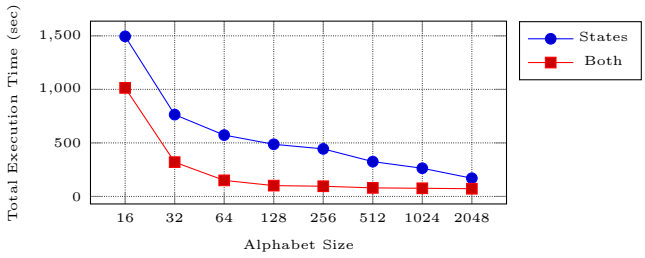

Figure 3: Processing times of two methods where total number of transitions are $28,000$

In Figure 1, we see the execution time for different data sizes with the alphabet size $k = 16$. Figure 2 shows the comparison of Method I and Method II, while the alphabet size $k = 64$. As expected, Method II is clearly more efficient. Figure 3 represents execution time of the two methods for various alphabet sizes when $|\delta_1| + |\delta_2| + |\delta_3| = 28,000$, The figure shows that as the size of alphabet increases, the execution time of both algorithms get closer to each other. This is due to the fact that once the the size of the alphabet exceeds the number of reducers (128), in Method II each reducer has to deal with several alphabet symbols, thus slowing down the computation inside the reducers.

6. CONCLUSIONS

In this paper we proposed and studied methods for computing a product automaton using Map-reduce. Our analysis and experimental results show that carefully optimizing the amount of inter-processor communication indeed pays off in improved processing time.

In future work we will investigate reducing the number of states in the product automaton, either by eliminating all or part of the useless states or by and determinizing and minimizing the automaton.

7. REFERENCES

[1] F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.

[2] F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. Knowl. Data Eng.*, 23(9):1282–1298, 2011.

[3] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in map-reduce. In *19th WWW 2010*, pages 231–240. ACM, 2010.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[5] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson/Addison Wesley, 2007.

[6] B. Kimmett, A. Thomo, and S. Venkatesh. Three-way joins on mapreduce: An experimental study. In *IISA 2014*, pages 227–232, 2014.

[7] I. K. Koumarelas, A. Naskos, and A. Gounaris. Binary theta-joins using mapreduce: Efficiency analysis and improvements. In *BeyondMR 2014*, pages 6–9, 2014.

[8] D. Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266, 1977.

[9] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA 2011*, pages 85–94. ACM, 2011.

[10] J. Leskovec, A. Rajaraman, and J. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[11] G. D. F. Morales, A. Gionis, and M. Sozio. Social content matching in mapreduce. *PVLDB*, 4(7):460–469, 2011.

[12] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *20th WWW 2011*, pages 607–614. ACM, 2011.

[13] D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *LPAR 2005*, pages 396–411, 2005.