Set reconciliation using rational polynomials

Ivo Kubjas

May 18, 2013

1 Introduction

In various distributed systems, it may happen that parties need to find a union of their data sets. This may happen in database synchronisation, shared file systems and during the use of gossip protocols.

Generic approaches have been using interactive protocols where they also share common records. This approach leads to communication complexity being dependant of the size of sets and is clearly inefficient if the sets are large but the differences are small.

The problem can be defined as follows: given two hosts A and B, having sets S_A and S_B accordingly, how can we find the union $S = S_A \cup S_B$ with minimal communication complexity. That is, how to find union with minimal number of interactions and minimal amount of information transmitted. This problem is called **set reconciliation** problem.

The problem has been studied in [3], [1], [7] and in other papers. To overcome the problem with transmitting information about shared records, researchers have used Reed-Solomon codes, colouring graphs and representing sets as characteristic polynomials. The latter method was also studied in [6], which is also the main article this report is based on.

2 Using polynomials in set reconciliation

Consider we have host A and host B with sets of length b bitstrings S_A and S_B respectively. Then the elements can be considered to be elements of $\mathbb{F}_q$. In this paper we look at the case where q is a prime. Due to Bertrand's postulate, for every $b \in \mathbb{N}$ there is a prime p such that $2^b \leq p \leq 2^{b+1}$. So, we can represent arbitrary length b bitstring as length $b+1$ bitstring in the finite field $\mathbb{F}_q$.

Let $\Delta_A = S_A \setminus S_B$ and $\Delta_B = S_B \setminus S_A$ be the differences between hosts sets. Denote $m_A = |\Delta_A|$ and $m_B = |\Delta_B|$ as the size of the differences. The total difference between hosts is then $m = m_A + m_B$.

If set S is $S = \{x_1, x_2, \ldots, x_n\}$, then we can define characteristic polynomial $\chi_S(Z)$ of set S as following:

$$\chi_S(Z) := (Z - x_1)(Z - x_2) \cdots (Z - x_n)$$

We see that the roots of $\chi_S(Z)$ correspond to the elements of S. So, if we are able to factor $\chi_S(Z)$ then we have found all elements of a host.

Unfortunately, because $\chi_S(Z)$ contains all the information for retrieving S, then transmitting it is not more efficient than transmitting S as a whole. But we can consider rational polynomial

$$\frac{\chi_{S_A}(Z)}{\chi_{S_B}(Z)} = \frac{\chi_{S_A \cap S_B}(Z) \cdot \chi_{\Delta_A}(Z)}{\chi_{S_A \cap S_B}(Z) \cdot \chi_{\Delta_B}(Z)} = \frac{\chi_{\Delta_A}(Z)}{\chi_{\Delta_B}(Z)}$$

This can be done, because $S_A = (S_A \cap S_B) \cup \Delta_A$ and for disjoint sets S and S' we have $\chi_{S \cup S'}(Z) = \chi_S(Z) \cdot \chi_{S'}(Z)$.

Now, if we have a rational polynomial over the whole sets, we can find a rational polynomial over the difference sets and thus we can recover the missing elements.

Generally, calculating the rational polynomial over all elements of the sets is not efficient in means of communication. So, how can we find out the ratio of characteristic polynomials? This can be achieved using interpolation.

Instead of transmitting whole characteristic polynomial, hosts evaluate them at some evaluation points and transmit those values. Now the hosts can divide the results at given points and using these values, difference rational polynomial can be interpolated and polynomials $\chi_{\Delta_A}(Z)$ and $\chi_{\Delta_B}(Z)$ could be recovered. Factoring the received polynomials, both hosts can recover missing elements from the set.

3 Evaluation and interpolation of polynomials

If the the total number of differences is $m_A + m_B = m$, then we denote by $\overline{m}$ an upper bound $m \leq \overline{m}$ of differences of sets to be reconciled. Later we show how to adapt the protocol to case where $\overline{m}$ is not known.

Now, for the protocol we evaluate host polynomials at $\overline{m}$ points. If sets S_A and S_B are large, then it is possible to do these evaluations on updates - on addition the former value is multiplied by $(Z - x)$ and on deletion the former value is divided by $(Z - x)$ where x is a element to add or remove from the set. Because of addition and multiplication, the cost of insertion and deletion is $2\overline{m}$ operations.

If one of the evaluation points falls into either set, then the characteristic polynomial will evaluate to 0 and the protocol is halted. To overcome this problem, new evaluation point should be selected. This could be done by choosing new evaluation point randomly and sending it during the protocol

or using some pseudo-random number generator. If the field $\mathbb{F}_q$ can not hold $2^b + \overline{m}$ elements, then a new q should be chosen such that $2^b + \overline{m} \leq q$. It is now possible to choose $\overline{m}$ evaluation points such that all evaluations are non-trivial.

Interpolating a rational function depends on evaluation points and values. Specifically, if we have polynomials $P(Z) = \sum_i p_i Z^i$ and $Q(Z) = \sum_i q_i Z^i$ with degrees bounded with d_1 and d_2 respectively, then it is necessary to have $d_1 + d_2 + 1$ pairs $(k_i, f_i) \in \mathbb{F}^2$, which we denote as V, to find a unique rational polynomial f such that $f(k_i) = f_i$ [9]. The pairs in V establish linear constraints on f:

$$k_i^{d_1} + p_{d_1-1}k_i^{d_1-1} + \ldots + p_0 = f_i \cdot (k_i^{d_2} + q_{d_2-1}k_i^{d_2-1} + \ldots + q_0)$$

Because bounds on the degrees of polynomials are not known, we can find them through the set sizes. Take $\delta = m_A - m_B$ and $\overline{m}$ is already known. Then:

$$m_A \leq \left\lfloor \frac{\overline{m} + \delta}{2} \right\rfloor =: \overline{m}_A$$
$$m_B \leq \left\lfloor \frac{\overline{m} - \delta}{2} \right\rfloor =: \overline{m}_B$$

Now, because numerator and denominator polynomials are monic in rational polynomial being recovered and if δ and $\overline{m}$ have same parity, then $\overline{m}_A + \overline{m}_B = \overline{m}$ and $\overline{m}$ evaluation points are enough to interpolate f. Proposition 2.2.1.4 from [8] gives sufficient conditions on uniqueness.

Theorem 1. *Let V be a support set with $\overline{m}$ elements over a field $\mathbb{F}$. Assume there exists two monic rational functions f and g that satisfy V, and that the numerator and denominator of f (respectively g) have degrees summing to at most $\overline{m}$. If the difference in degrees between numerator and denominator of f is the same as for g, then f and g are equivalent.*

Now we have all the tools for recovering a unique rational polynomial.

4 Computational example of set reconciliation

There was given an example in [6], to show the workings and steps more clearly.

Consider sets $S_A = \{1, 2, 9, 12, 33\}$ and $S_B = \{1, 2, 9, 10, 12, 28\}$ stored at hosts A and B respectively. The elements are represented as 6-bit integers over $\mathbb{F}_{97}$. Let the evaluation points be $-1, -2, -3, -4, -5$.

As the description was partly ambiguous, the author of this report has written small helper functions to interpolate the functions. Helper functions

are available at [4]. The usage of the module is shown in appendix A. We see that the protocol works and it is easy to implement.

The author still has not solved the problem with finding roots over finite field $\mathbb{F}_q$, but this is briefly covered in [6] and is still being worked on. Currently exhaustive search is used in finding roots.

5 Computational and communicational complexity

This algorithm can be summarised as a protocol:

POLY_RECON

---

1. Hosts A and B evaluate $\chi_{S_A}(Z)$ and $\chi_{S_B}(Z)$ at $\overline{m}$ evaluation points E.
2. The evaluation values $\chi_{S_A}(E)$, $\chi_{S_B}(E)$ and sizes of the sets $|S_A|$, $|S_B|$ are sent to other party.
3. Hosts combine $\chi_{S_A}(E)$ and $\chi_{S_B}(E)$ to get $\chi_{S_A}(E)/\chi_{S_B}(E)$ and interpolate the result to get $\chi_{\Delta_A}(Z)$ and $\chi_{\Delta_B}(Z)$.
4. $\chi_{\Delta_A}(Z)$ and $\chi_{\Delta_B}(Z)$ are factored to get elements of Δ_A and Δ_B.

We will denote subset of POLY_RECON, where evaluations are only sent from host B to A and A returns only Δ_A to B, as POLY_RECON($A \leftarrow B$). For consistency and clarity, we will denote protocol POLY_RECON as POLY_RECON($A \leftrightarrow B$).

If elements of S are represented as bitstrings of length $b+1$, then during POLY_RECON($A \leftarrow B$) we see that for host B it is necessary to transmit

$$(b+1)\overline{m} + b = (\overline{m}+1)(b+1) - 1$$

bits of information. Recall that host B also had to send the size of its set, which can be represented with b bits.

If host A has recovered Δ_A and Δ_B, it can just send the missing elements to other host B. Then it has to send $m_A b \leq \overline{m}b$ bits of information to host B. This leads to total communication complexity of

$$(\overline{m}+1)(b+1) - 1 + m_A b \leq 2(\overline{m}+1)b + \overline{m} - b$$

5.1 Bound on communication complexity

In the paper, it was shown that mb is close to best achievable communication complexity for any set reconciliation protocol.

Let $N = |S_A \cap S_B|$ be the number of elements in common. Then host A has to recover m_B elements from total of $2^b - N - m_A$ elements. Symmetrically host B has to recover m_A elements from $2^b - N - m_B$ elements. We denote $\hat{C}_\infty$ as the number of bits that are needed for set reconciliation. The bound is

$$\hat{C}_\infty \geq \log_2 \left[\binom{2^b - N - m_A}{m_B} \cdot \binom{2^b - N - m_B}{m_A} \right] \geq \log_2 \left[\binom{2^b - N - m}{m} \right]$$

If $2^b \geq 2(N + m)$, then the lower bound becomes $(b - 1 - \log_2 m) \cdot m \approx mb - m \log_2 m$. So

$$\hat{C}_\infty \geq mb - m \log_2 m,$$

thus

$$\frac{\hat{C}_\infty}{mb} \geq 1 - \frac{\log_2 m}{b}$$

If sets are sparse then $\log_2 m \ll b$ and the bound $\hat{C}_\infty$ is arbitrarily close to mb.

5.2 Computational complexity

Evaluation of characteristic polynomials of sets S_A and S_B takes $2|S|\overline{m}$ operations. As previously stated, it could be reduced to $2\overline{m}$ operations per insertion and deletion of element in host set.

Solving system of linear equations with Gaussian elimination method has a complexity of $\mathcal{O}(\overline{m}^3)$. There are better algorithms for solving system of linear equations and so the number of operations could be reduced.

There is additional computation during the factorisation of interpolated polynomials. In [2], Kaltofeen and Shoup achieved $\mathcal{O}(\overline{m}^{1.815} \log q)$ as computational complexity of factoring polynomial of degree $\overline{m}$ over the field $\mathbb{F}_q$. In the underlying paper [6], a simpler approach for finding roots of a polynomial in a finite field was shown. The computational complexity in this case is $\mathcal{O}(\overline{m}^3 \log q)$.

6 Protocol with three and more parties

If there are more than two parties who need to reconciliate their sets, using POLY_RECON($A \leftrightarrow B$) pairwise is not optimal, as this leads to transmission of redundant information. We propose the following modification to the protocol.

Let there be three hosts A, B, C with sets S_A, S_B and S_C accordingly. For hosts A and B, we denote $\Delta_{AB} = S_A \setminus S_B$ and $\Delta_{BA} = S_B \setminus S_A$. Furthermore, we denote hosts A and B pairwise union as $S_{AB} = (S_A \cap S_B) \cup \Delta_{AB} \cup \Delta_{BA}$. Pairwise unions can be simplified to $S_{AB} = (S_A \cap S_B) \cup \Delta_{AB} \cup \Delta_{BA} = S_B \cup \Delta_{AB}$ and $S_{BC} = S_B \cup \Delta_{CB}$. If S_{AB} and S_{BC} are known then $S = S_{AB} \cup S_{BC}$. Now host A and host C do not have $S \setminus S_A$ and $S \setminus S_C$ accordingly. But $S \setminus S_A = S \setminus (S_{AB} \setminus \Delta_{BA}) = (S \setminus S_{AB}) \cup \Delta_{BA}$ and $S \setminus S_C = (S \setminus S_{BC}) \cup \Delta_{BC}$ and thus can be recovered from sets known to host B.

This discussion can be summarised as a protocol for reconciliating sets between three parties:

POLY_RECON3

---

1. Hosts A and C evaluate $\chi_{S_A}(Z)$ and $\chi_{S_C}(Z)$ at $\overline{m}$ evaluation points E.
2. The evaluation values $\chi_{S_A}(E)$, $\chi_{S_C}(E)$ and sizes of the sets $|S_A|$, $|S_C|$ are sent to host B.
3. Host B recovers Δ_{AB}, Δ_{BA}, Δ_{CB} and Δ_{BC}.
4. Host B recovers S_{AB} and S_{BC}.
5. Host B recovers S.
6. Host B sends $S \setminus S_A$ to host A and $S \setminus S_C$ to host C.
7. Hosts A and C recover S.

In this protocol, hosts A and B send values of $\overline{m}$ evaluation points and sizes of their sets. Because host B has all the knowledge about differences, it can send to hosts A and C only elements missing from S_A and S_C. It sends elements from $\Delta_{BA} \cup \Delta_{CA}$ and $\Delta_{BC} \cup \Delta_{AC}$ to hosts A and C respectively. In total we have to transmit $|\Delta_{AC} \cup \Delta_{BC}| + |\Delta_{CA} \cup \Delta_{BA}|$ elements. By looking at disjoint sets, we can simplify:

$$
\begin{aligned}
|\Delta_{AC} \cup \Delta_{BC}| + |\Delta_{CA} \cup \Delta_{BA}| = \\
&= |\Delta_{BA}| + |\Delta_{BC}| + |\Delta_{AC} \setminus \Delta_{BC}| + |\Delta_{CA} \setminus \Delta_{BA}| \\
&= |\Delta_{BA}| + |S_A \setminus (S_B \cup S_C)| + |\Delta_{BC}| + |S_C \setminus (S_A \cup S_B)| \\
&= |\Delta_{BA}| + |\Delta_{AB}| - |\Delta_{AB} \cap \Delta_{CB}| + \\
&+ |\Delta_{BC}| + |\Delta_{CB}| - |\Delta_{AB} \cap \Delta_{CB}| \\
&= 2(\overline{m} - |\Delta_{AB} \cap \Delta_{CB}|)
\end{aligned}
$$

Because no more than $2\overline{m}$ elements are sent in this stage, then total number of transmitted bits is bounded above by

$$2(b+1)\overline{m} + 2b + 2b\overline{m} = 2((2b+1)\overline{m} + b)$$

During the protocol, sets are evaluated 3 times and interpolation is done 2 times.

In POLY_RECON3, host B acted as a proxy between hosts A and C. Similar construction, where one host acts as a master, could also be used to perform set reconciliation between more hosts, but this leads to uneven distribution of computational cost while having balanced use of communication.

It has not been considered if requirement would be to have even computational cost across hosts. Intuitively in this case the amount of communication would increase, as after reconciliation of two host sets upper bounds of differing elements increase and more evaluation values should be sent.

7 Reconciliation without known bound on m

We have discussed case when bound $\overline{m}$ on m is known. In practise this might not always be the case.

The question is how to test if $\overline{m}$ is sufficiently large. The option would be to somehow test the whole sets. For example, by computing hashes of sets and comparing the results. This approach assumes that the sets have been reconciliated and so leads to additional computation on both hosts and additional communication as the hash has to be exchanged.

Another option would be to test the equivalence of $g(Z)$, which is rational polynomial returned after interpolation and $f(Z) = \chi_{\Delta_A}(Z)/\chi_{\Delta_B}(Z)$. Equivalence can be tested by evaluating both polynomials at random points and see if the values agree.

Let D be the upper bound on degrees of $g(Z)$ and $f(Z)$. From theorem 1 we know that D evaluations are enough for polynomials equivalence. In a worst case scenario, we can choose $D-1$ evaluation points such that different polynomials $g(Z)$ and $f(Z)$ agree on these points. So, the probability that test succeeds for different polynomials is $\rho = (D-1)/|E|$, where $E \subset \mathbb{F}_q$ is the set where evaluation points are chosen.

Choosing q such that sets $S_A \subset \mathbb{F}_q$ and $S_B \subset \mathbb{F}_q$ are sparse and taking D as $D = |S_A| + |S_B|$ and $E \approx |\mathbb{F}_q|$, then ρ is small and repeating test k times (using different evaluation points), the probability for false positive is $\rho^k \xrightarrow{k} 0$.

Using this approach, evaluations can be sent one at a time. Then $g(Z)$ should be recalculated only then if evaluations and $g(Z)$ do not agree. After k sequential confirming tests it can be considered that $g(Z) = f(Z)$. The probability that polynomials differ after k succeeded tests is bounded above

by $m\rho^k$. If we want to achieve failure tolerance of $1-\epsilon$, then k should be chosen

$$k \geq \log_\rho(\epsilon/m) > \log_\rho(\epsilon/|S_A| + |S_B|)$$

Total number of transmitted bits is at most

$$(b+2)(m+k)+b$$

and if the host has recovered differences of sets then it has to send back $m_A b$ bits of information.

Computational complexity is bounded by recalculation of $g(Z)$, which is done m times. As Gaussian elimination is $\mathcal{O}(m^3)$, then total complexity is $\mathcal{O}(m^4)$.

If the goal is to optimize computational complexity, then another approach should be used. Instead of sending one evaluation at a time, the number of evaluations could be increased by a factor of c after each round. In this case, the total number of evaluations sent is $(c^n-1)/(c-1)$[1] for some $n > 0$ and total number of rounds is $n = \lceil \log_c[(m+k)(c-1)+1] \rceil$. In the worst case only one value is left to last block. Then extra $(c-1)(m+k-1)$ values are sent.

In this case, maximum of

$$2b+1+(b+1)c(m+k-1)+\lceil \log_c[(m+k)(c-1)+1] \rceil$$

bits are transmitted. As before, receiving host has to return $m_A b$ bits of information to finish the reconciliation.

For fixed c, as total number of rounds is proportional to $\log_c(m+k)$, computational complexity is reduced to $\mathcal{O}((m+k)^3 \log_c(m+k))$.

As a further note, it could be studied if both latter protocols could be used to perform reconciliation between many hosts. Suppose there is a Hamiltonian cycle in the graph of hosts and all hosts have pseudo-random generator seeded with same value. For simplicity we can assume that the range of pseudo-random generator does not intersect with any of the hosts set. We give a definition to one step - we call it a stage. In a stage, at each iteration, each host sends evaluation value of their set's characteristic polynomial at evaluation point received from the generator to its neighbouring hosts.

Upon receiving values from its neighbouring hosts, each host performs the algorithm to interpolate the difference polynomial if the values disagree. After some threshold each host can conclude if their set has been reconciliated with its neighbouring hosts, find roots of the difference polynomial and add them to its set. Having added new elements to their set, each host can

[1] In the original paper it was given that number of evaluations sent is $c^n - 1$ but this is not true for $c > 2$. This mistake propagated to consequential corollaries about total number of rounds, numbers of bits sent *etc.*

send the differences with one neighbour to the second neighbour. If this is done, then all hosts start a new stage as before. The algorithm halts when all hosts have received information about all sets.

8 Conclusion

This report studied article [6]. We saw, how to perform set reconciliation between two hosts using rational polynomials. Furthermore, trivial implementation was given and it was shown that the protocol works.

Bounds on transmitted bits and computational complexity in different use-cases were shown. If bound on missing number of elements is not known, then computational complexity increases to quadratic or cubic logarithmic in a number of the missing elements. If the number of elements added and removed to hosts sets is large and the amount of these elements is unpredictable, then this protocol might not be feasible to use.

As an example of a practical implementation of this protocol is CONFLUX[5], which is a distributed database synchronization library still in development.

References

[1] K.A.S. Abdel-Ghaffar and A. El Abbadi. An optimal strategy for comparing file copies. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):87–93, 1994.

[2] E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Mathematics of Computation of the American Mathematical Society*, 67(223):1179–1197, 1998.

[3] M.G. Karpovsky, L.B. Levitin, and A. Trachtenberg. Data verification and reconciliation with generalized error-control codes. *Information Theory, IEEE Transactions on*, 49(7):1788–1793, 2003.

[4] I. Kubjas. Polynomial reconciliation solver. `https://github.com/ivokub/polyrecon`.

[5] C. Marshall. CONFLUX - distributed database synchronization library. `https://github.com/cmars/conflux`.

[6] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. 2004.

[7] D. Starobinski, A. Trachtenberg, and S. Agarwal. Efficient pda synchronization. *Mobile Computing, IEEE Transactions on*, 2(1):40–51, 2003.

[8] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, New York, 2nd edition, 1993.

[9] R.E. Zippel. *Effective Polynomial Computation*. Kluwer Academic Press, Boston, 1993.

A Example in real life

```
import solve_recon

# Define sets and evaluation points, base and upper bound
set1 = [1, 2, 9, 12, 33]
set2 = [1, 2, 9, 10, 12, 28]
evpoints = [-1, -2, -3, -4, -5]
base = 97
m = 5

# Evaluates sets at given points
ev1 = solve_recon.evaluate(set1, evpoints, base)
ev2 = solve_recon.evaluate(set2, evpoints, base)
print ev1 # [mpz(58), mpz(19), mpz(89), mpz(77), mpz(4)]
print ev2 # [mpz(15), mpz(54), mpz(68), mpz(77), mpz(50)]

# Divides evaluation values
intvalues = solve_recon.divide(ev1, ev2, base)
print intvalues # [mpz(75), mpz(74), mpz(17), mpz(1), mpz(35)]

# Calculate polynomial bounds
d1, d2 = solve_recon.poly_bounds(set1, set2, m)
print d1, d2 # 2 3

# Create system we are going to solve
constraints = solve_recon.create_equations(evpoints,
              intvalues, d1, d2, base)
print constraints
# [[96 1 22 75 22 21]
#  [95 1 92 51 23 83]
#  [94 1 41 51 80 17]
#  [93 1 81  4 96 17]
#  [92 1 95 78 62 62]]

# Solve it
solved = solve_recon.solve(constraints, base)
print solved
# [[1 0 0 0 53 64]
#  [0 1 0 0 94  0]
#  [0 0 1 0 53 59]
#  [0 0 0 1 23 86]
#  [0 0 0 0  0  0]]

```

```
43 # See which values are independent
44 indep, dep = solve_recon.indep_solutions(solved, base)
45
46 # Output polynomial coefficients ...
47 coef_n, coef_d = solve_recon.rat_poly_sing(solved,
48                  indep, d1, d2, base, coeff = True)
49 print coef_n # [(0, mpz(3)), (1, mpz(11))]
50              # == x^2 + 11 x + 3
51 print coef_d # [(0, mpz(1)), (1, mpz(63)), (2, mpz(6))]
52              # == x^3 + 6 x^2 + 63 x + 1
53
54 # ...or the polynomials themselves
55 fn_n, fn_d = solve_recon.rat_poly_sing(solved,
56              indep, d1, d2, base, coeff = False)
57
58 # Test, if the polynomials are correct
59 sol1, sol2 = solve_recon.test(fn_n, fn_d, base)
60 print sol1  # [33]
61 print sol2  # [10, 28]
```