Analysis of an Algorithm for Choice Coordination and Byzantine Agreement

Randomized Algorithms

Martijn Dwars
4156730
M.T.Dwars@student.tudelft.nl

Jeffrey de Lange
4018605
J.G.A.deLange@student.tudelft.nl

ABSTRACT

We analysed two random algorithms for distributed computation: one for Choice Coordination and one for Byzantine Agreement. The failure bounds of both algorithms are evaluated experimentally. For evaluating the Byzantine Agreement algorithm we assume that adversaries send random votes. Our analysis of the Byzantine Agreement problem leads to an improved bound of $t < \frac{n}{6}$ on the number of adversaries t. Finally, we show that no better bound can be achieved with the current algorithm.

1. INTRODUCTION

In this paper we analyse two algorithms for distributed computation: one for the Choice Coordination problem and another for the Byzantine Agreement problem. For the Choice Coordination Problem we consider both the synchronous and asynchronous algorithm. For the Byzantine Agreement problem we only look at a synchronous algorithm.

In section 2 the problems are introduced in more detail. The algorithms we analysed are described in section 3. In section 4 the results are discussed. A more theoretic analysis of the algorithm for Byzantine Agreement can be found in section 5. Finally, section 6 gives a conclusion.

2. PROBLEM AND MODEL

The Choice Coordination problem [6] is to have n identical processors reach consensus on a unique choice out of a collection of m identical options.

In our model there are m read-write registers that are accessible to all processors. The only way for processors to communicate is through these registers. The registers are equipped with a locking mechanism to prevent concurrent access. At the end of the algorithm, exactly one of the registers should contain a special symbol ($\checkmark$). The algorithm we consider is designed for $n = m = 2$, but the analysis generalizes in a rather straightforward manner [4].

It is known that any deterministic algorithm for solving this problem will have a complexity of $\Omega(n^{\frac{1}{3}})$ operations [4]. We analyse a randomized algorithm that will solve the problem using c operations with a probability of success at least $1 - 2^{-\Omega(c)}$.

The Byzantine Agreement problem is a classic problem in distributed computing [3]. It consists of n processors, at most t of which are faulty. We refer to the faulty processors as *adversaries* and the remaining processors as *good processors*. The good processors do not know which processors are faulty. The good processors all start with a binary value called its *vote*. Faulty processors may collude and try to subvert the agreement process. Specifically, they may send arbitrary, inconsistent votes to good processors. When a processor ends the algorithm with a decision, it must satisfy the following properties:

- All good processors should finish with the same decision.
- If all good processors started with the same vote, they finish with a decision equal to that vote.

In the model that we consider all processors have access to an unbiased coin which is tossed each round.

It is known that in this model any deterministic algorithm requires $t+1$ rounds to reach agreement in the worst case [1]. An algorithm that achieves this bound is given by Pease [5]. It is impossible to achieve agreement using a deterministic algorithm in an asynchronous setting [2]. In this paper we analyse the randomized ByzGen algorithm in Motwani [4]. In the analysis the algorithm is shown to work correctly for $t < \frac{n}{8}$ adversaries, though we will prove that $t < \frac{n}{6}$ can also be achieved. The ByzGen algorithm has an expected number of rounds that is a constant. Furthermore, it is able to achieve agreement even in an asynchronous setting.

The two problems are similar in that they both aim to reach an agreement. In the Choice Coordination problem, n processors aim to reach agreement on a unique choice out of m options. In the Byzantine Agreement problem, n processors aim to reach agreement on a boolean value in the presence of t faulty processors. Notice that the Byzantine Agreement problem can be reduced to the Choice Coordination problem. The binary representation of the unique choice of the Choice Coordination problem requires $\log m$ bits. We can let the processors in the Byzantine Agreement problem agree on every bit by repeating the algorithm $\log m$ times, each time establishing the next bit.

3. ALGORITHMS

In this section the algorithms used in our experiments are presented. In algorithm 1 and algorithm 2 the pseudo code implementations of SYNCH-CCP and ASYNCH-CCP, as described in [4], can be found.

In the algorithm for ASYNCH-CCP the variables T_i and t_i respectively are a time stamp in for local variable B_i and for registers C_i.

Since the analysis of for SYNCH-CCP also holds for ASYNCH-CCP we will only run the experiments on SYNCH-CCP.

Algorithm 1: SYNCH-CCP

```
Input: Registers $C_0$ and $C_1$ initialized to 0.
Output: Exactly one of the two registers has the
        value ✓
Step 0: $P_i$ is initially scanning the register $C_i$ and has
its local variable $B_i$ initialized to 0;
Step 1: Read the current register and obtain a bit $R_i$;
Step 2: Select one of three cases:
case $R_i = ✓$
    halt;
case $R_i = 0, B_i = 1$
    Write ✓into the current register and halt;
case otherwise
    Assign an unbiased random bit to $B_i$ and write $B_i$
    to current register;
Step 3: $P_i$ exchanges its current register with $P_{1-i}$ and
returns to Step 1.
```

Algorithm 2: ASYNCH-CCP

```
Input: Registers $C_0$ and $C_1$ initialized to $\langle 0, 0 \rangle$.
Output: Exactly one of the two registers has the
        value ✓
Step 0: $P_i$ is initially scanning a randomly chosen
register. Thereafter, it changes its current register at
the end of each iteration. The local variables $T_i$ and $B_i$
are initialized to zero;
Step 1: $P_i$ obtains a lock on the current register and
reads $\langle t_i, R_i \rangle$;
Step 2: $P_i$ selects on of five cases:
case $R_i = ✓$
    halt;
case $T_i < t_i$
    $T_i \leftarrow t_i$ and $B_i \leftarrow R_i$
case $T_i > t_i$
    Write ✓into the current register and halt;
case $T_i = t_i, R_i = 0, B_i = 1$
    Write ✓into current register and halt;
case otherwise
    $T_i \leftarrow T_i + 1, t_i \leftarrow t_i + 1$, assign random (unbiased)
    bit to $B_i$ and write $\langle t_i, B_i \rangle$ into the current register.
Step 3: $P_i$ releases the lock on its current register,
moves to the other register, and returns to Step 1.
```

The ByzGen algorithm is presented in algorithm 3. We define $L = 5n/8$, $H = 3n/4$, and $G = 7n/8$, as described in Motwani [4]. This choice of L, H, and G guarantees that the algorithm is correct for $t < \frac{n}{8}$.

Throughout this paper we use the terms *voting* and *deciding*. When we say that a processor votes v we refer to the process of adopting v as the new vote. When we say that a processor decides v we refer to the process of setting d_i to v permanently.

During the experiments for Byzantine Agreement we initialize the processors with random votes. We make the simplifying assumption that an adversary sends every processor a random vote equiprobably drawn from $\{0, 1\}$.

Algorithm 3: ByzGen

```
Input: A value $b_i$
Output: A decision $d_i$
vote = $b_i$;
for each round do
    Broadcast vote;
    Receive votes from all other processors;
    $maj \leftarrow$ majority value (0 or 1) among votes
    received including own vote;
    $tally \leftarrow$ number of occurrences of $maj$ among votes
    received;
    if $coin = HEADS$ then threshold $\leftarrow L$;
    else threshold $\leftarrow H$;
    if $tally \geq threshold$ then $vote \leftarrow maj$;
    else $vote \leftarrow 0$;
    if $tally \geq G$ then set $d_i$ to $maj$ permanently;
end
```

4. EXPERIMENTAL RESULTS

In this section the experiments and the results of these experiments are discussed.

4.1 Choice Coordination

Figure 1 shows a boxplot of the number of iterations for 10,000 trials of the SYNCH-CCP algorithm. Figure 2 shows the distribution of this data. The geometric density function for $p = \frac{1}{2}$ is superimposed on the histogram. As expected, the distribution of iterations closely resembles the geometric distribution.

4.2 Byzantine Agreement

For $t < \frac{n}{8}$ the algorithm is guaranteed to operate correctly and have an expected number of rounds to reach agreement that is a constant [4]. In each iteration the probability of needing another iteration is at most $\frac{1}{2}$. Thus the number of iterations can be modeled by a geometric distribution with parameter $p \geq \frac{1}{2}$. To investigate the distribution of the number of iterations we ran 1,000 trials with $n = 40, t = 4$. Choosing $t < \frac{n}{8}$ guarantees that the algorithm works correctly. Additionally, by choosing the maximum allowable number of adversaries $t = \frac{n}{8} - 1$ we focus on a worst case scenario. A histogram of the number of iterations is shown in Figure 3. We see that most trials need only two iterations. The data confirms that the number of iterations can be modeled as a geometric distribution. However, the parameter p seems to be larger than $\frac{1}{2}$. This can be explained

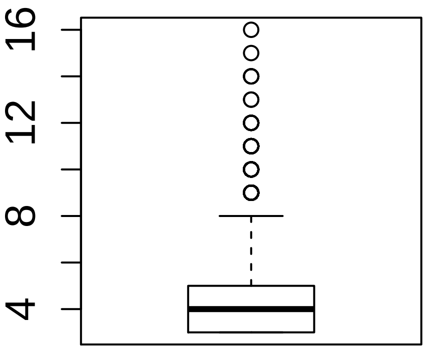

Figure 1: Boxplot number of iterations SYNCH-CCP

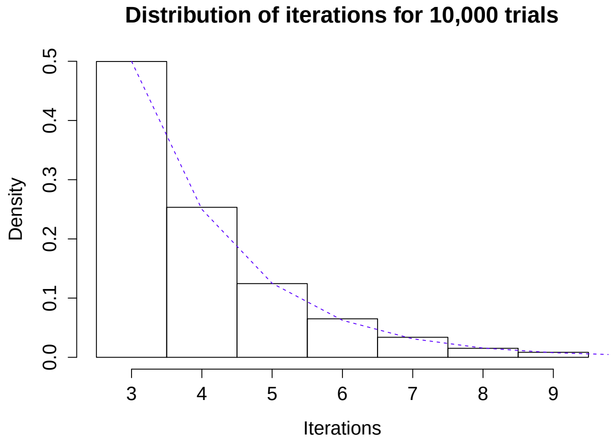

Figure 2: Distribution of iterations in 10,000 trials of the SYNCH-CCP algorithm

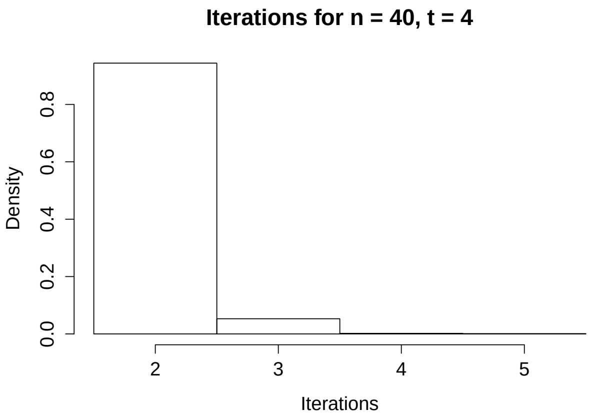

Figure 3: Iterations for $n = 40, t = 4$ in 1,000 trials of the ByzGen algorithm.

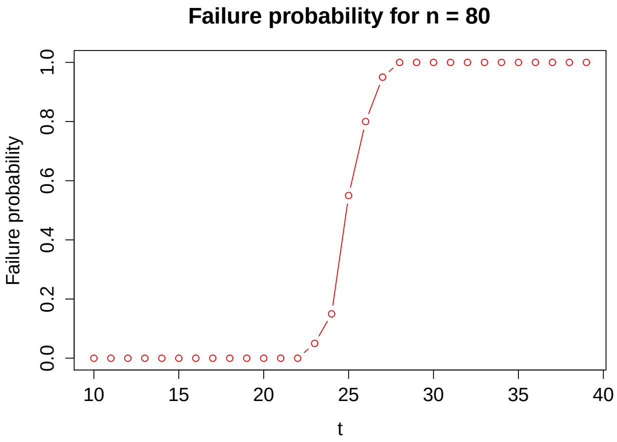

Figure 4: Failure probability for $n = 80$ and increasing t. The failure probability is computed as the fraction of failures in 20 trials.

by our assumption that adversaries sent send random votes, instead of colluding to disrupt the agreement process.

Next we investigated how large the probability of failure is under the assumption that all adversaries send random votes. For $n = 80$ and $t = 10, 11, ..., 40$ we performed 20 trials and computed the probability as the fraction of failures. If a single trial took more than 20 rounds we counted it as a failure as well. The fraction of failures is taken as an estimate of the probability of failure. Figure 4 shows the results. We observe that for $t < \frac{n}{4}$ the algorithm works correctly, after which the failure probability sharply rises and remains 1.

Finally we investigated how good the algorithm performs when $t = \frac{n}{4}$. For this value of t the algorithm it not guaranteed to work correctly. Figure 5 shows the results of this experiment. We used a nonlinear least squares regression to fit the function $C * (1 - exp(k * t))$. This function is superimposed for $C = 8.84, k = -0.06$. Interestingly, as the number

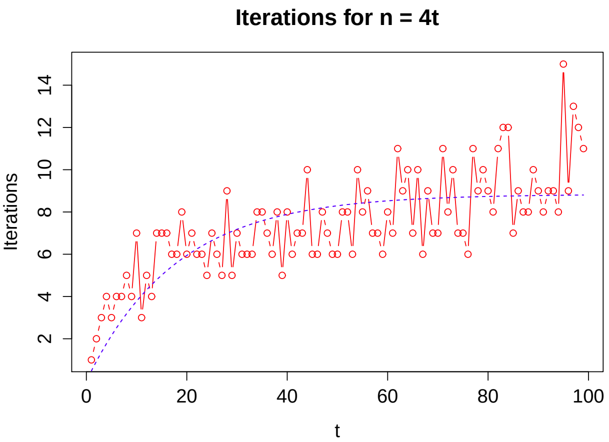

Figure 5: Number of iterations before all good processors make a decision for $n = 4t$ processors, t of which are adversaries.

of adversaries increases (keeping the ratio of adversaries to total processors the same) the number of iterations grows slowly.

5. ANALYSIS

We now turn to a more t[illegible]tic analysis of the ByzGen algorithm.

THEOREM 5.1. *It is not true that all good processors always determine their decision in the same round.*

PROOF. Let there be 16 processors, one of which is an adversary (i.e. $t = 1, n = 16$). In this case, $G = \frac{7n}{8} = 14$. Let two good processors start with initial value 1 and the remaining 13 good processors with initial value 0. In the first round all good processors will receive 0 at least $13 < G$ times. If the adversary sends 0 to some processor p_i and 1 to some processor p_j, then $tally_i = 14 \geq G$ but $tally_j = 13 < G$. Thus, in this round p_i can make a decision whereas p_j cannot. □

Though not all processors make their decision in the same round, we can say something about how many rounds it takes for all processors to make a decision after some processors has made a decision.

THEOREM 5.2. *At most one round after some process determines its decision all other processors will do so as well.*

PROOF. Let r be the first round in which some processor p_i decides on maj. To make a decision, it must be the case that $tally_i \leq G = \frac{7n}{8}$. Since the number of adversaries $t < \frac{n}{8}$, there must have been strictly more than $\frac{6n}{8}$ (i.e. at least $\frac{6n}{8} + 1$) good processors that sent b_i. It follows that these good processors p_j have $tally_j \leq H = \frac{6n}{8} + 1$. No matter the outcome of the coin toss, $tally_j \leq threshold$. As a result, all p_j adopt the majority vote (if they did not already do so). In round $s+1$, all good processors broadcast this majority vote, causing $tally_j \leq G = \frac{7n}{8}$. Thus, all good processors will decide on maj in round $r + 1$. □

The ByzGen algorithm as presented in Motwani [4] does not contain a termination condition. However, from Theorem 5.2 we note that a process only needs to participate for one more round after it has made a decision. Specifically, since a process that has made a decision will never change its vote, it only needs to broadcast its vote.

We now turn to the problem of determining how large a value of t the ByzGen algorithm can tolerate. The analysis in Motwani [4] shows that $t < \frac{n}{8}$. A more careful analysis shows that this bound can be improved while guaranteeing a correct solution and keeping the same exepcted number of iterations.

THEOREM 5.3. *ByzGen tolerates $t < \frac{n}{6}$ faulty processors when $G = \frac{5n}{6}$, $H = \frac{4n}{6}$, and $L = \frac{3n}{6}$.*

PROOF. If all $n - t$ good processors start with the same value v, then all good processors will receive this value $n - t > \frac{5n}{6}$ times. It follows that $maj = v$, $tally > \frac{5n}{6} = G$ and all good processors will decide v.

Now assume some good processors start with 0 and some start with 1. We consider two cases: some good processors compute different values for maj and all good processors compute the same value for maj.

- If some good processors to compute different values for maj, then the number of 1's and 0's sent by the $n - t$ good processors must be close to each other. Let c_0 be the number of 0's and c_1 be the number of 1's sent by the $n-t$ good processors. It must hold that $|c_1 - c_2| \leq t < \frac{n}{6}$. It follows that $tally \leq \frac{n-t}{2} + \frac{t}{2} + t = \frac{n}{2} + t$ for all good processors. With probability $\frac{1}{2}$, $threshold = L$, otherwise $threshold = H$.
  - If $threshold = L = \frac{3n}{6}$, then $tally \geq threshold$ because by definition $tally \geq \frac{n}{2}$. All good processors will vote maj, the value of which may differ among processors. However, because $tally < G$, no good processor will determine a value.
  - If $threshold = H$, then $tally < threshold$, and all good processors vote 0.
- All good processors compute the same value for maj. We say that a threshold is foiled if one process has $tally \geq threshold$ and another has $tally < threshold$. Since $H - L = \frac{n}{6} > t$, at most one threshold can be foiled and the probability of this happening is $\frac{1}{2}$. If this happens, then the number of votes originating from good processors for maj was less then $\frac{4n}{6}$. But then $tally < \frac{5n}{6}$ for all processors and no processor will decide. If the threshold is not foiled, all good processors vote the same.

□

It remains to show that no larger value of t can be tolerated by the algorithm.

THEOREM 5.4. *The ByzGen algorithm tolerates no more than $t < \frac{n}{6}$ adversaries.*

PROOF. We prove this theorem by deducing three bounds that are necessary for the algorithm to operate correctly.

- $n-t \geq G$: Assume to the contrary that $G > n-t$. The t adversaries can always prevent the good processors from making a decision. Specifically, if all $n-t$ good processors agree on a value v, then $tally \geq n-t$. If t adversaries vote $1-v$ the previous inequality becomes an equality and $tally = n-t < G$. As a consequence, no good processor can decide.

- $G - H \geq t$: Assume to the contrary that $G - H < t$. Now assume $G - t$ good processors vote 1 and the remaining $n - t - (G - t) = n - G$ good processors vote 0. Let p_i be one of the good processors voting 1. Let all t adversaries send 1 to p_i and 0 to the other processors. Now p_i has received G votes for 1 and decides on 1. All processors p_j for $j \neq i$ have received $G - t$ votes for 1 and $n - G + t$ for 0. If $maj_j = 0$, then all p_j will vote 0 independent of the coin toss. If $maj_j = 1$, then $tally_j = G - t$. With probability $\frac{1}{2}$ we have $threshold = H$. Since $tally_j = G - t < H$ all p_j will vote 0. Once all p_j vote 0, they will never be able to decide on 1.

- $\frac{n}{2} + t < H$: Assume to the contrary that $\frac{n}{2} + t \geq H$. Let some processors i have $maj_i = 0$ and other processors j have $maj_j = 1$. For this to happen the $n-t$ good processors must send the same vote at most $\frac{n-t}{2} + \frac{t}{2} = \frac{n}{2}$ times. After including the votes of the adversaries, the same vote is sent at most $\frac{n}{2} + t$ times. Now assume $tally_j = \frac{n}{2} + t$. The processors i will vote 0 independent of the coin toss, because $maj_i = 0$. The processors j will vote 1 independent of the toin coss, because $tally_j = \frac{n}{2} + t \geq H$. As a result, all good processors have kept the same vote. If the adversaries keep doing this no processor will be able to decide.

. For combining these bounds we start with $\frac{n}{2} + t < H$, or equivalently, $t < H - \frac{n}{2}$. Now we plug in $G - H \geq t$, or equivalently, $H \leq G - t$. This results in $t < G - t - \frac{n}{2}$. Finally, we plug in the $n-t \geq G$, or equivalently, $G \leq n-t$. This brings us to the final result of $t < n - t - t - \frac{n}{2}$, or equivalently, $t < \frac{n}{6}$. □

6. CONCLUSION

In this paper we have presented an empirical analysis of both the Choice Coordination problem and the Byzantine Agreement problem.

The Choice Coordination experiments yielded the expected results, which was that it had a constant expected run time.

In the empirical analysis of the Byzantine Agreement problem we have shown that the algorithm only needs two iterations when $t < \frac{n}{8}$ holds. We have also shown the number of iterations does not increase very much when $t < \frac{n}{4}$ if the adversaries send arbitrary votes.

Our theoretical analysis of the Byzantine Agreement problem lead to an improved bound of $t < \frac{n}{6}$ on the number of adversaries. Furthermore, we showed that no better bound can be achieved without modifying the algorithm.

7. REFERENCES

[1] M. J. Fischer and N. A. Lynch. A lower bound for the time to assure interactive consistency. *Information processing letters*, 14(4):183–186, 1982.

[2] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[3] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[4] R. Motwani and P. Raghavan. *Randomized Algorithms*, pages 355–361. Cambridge: Cambridge UP, 1995.

[5] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[6] M. O. Rabin. The choice coordination problem. *Acta Informatica*, 17(2):121–134, 1982.