CAS CS 537: Randomized Algorithms, Fall 2013 **Student Lecture 15**

Randomized Byzantine Agreement

December 10, 2013 Lecture by: Sachin Vasant

1 Introduction

Distributed computing often involves a set of n processes arriving at a consensus/agreement on some bit v, in a de-centralized fashion. The problem is particularly interesting in the presence of Byzantine failures, *i.e.,* an adversary may adaptively corrupt upto t among these n processes and cause them to behave arbitrarily in the consensus protocol. To elaborate, these processes may either completely avoid participation or send differing values to different correct processes. In short, these faulty processes aim to delay the achievement of consensus in the system as far as possible. In this lecture we shall make a few limiting assumptions on these faulty processes. These shall be introduced as and when required.

In this lecture we introduce randomized Byzantine consensus protocols, where the processes are completely asynchronous. This was shown to be impossible in the deterministic setting. Note that a trivial byzantine agreement protocol simply agrees to 0 irrespective of its current state. To avoid such trivial protocols, we require two properties of any randomized byzantine agreement protocol (same as in [3]):

Validity If all correct processes start with the same value v, then they all decide v.

Agreement All correct processes decide on the same value.

Furthermore, a protocol is *t-resilient* if despite the presence of t faulty processes, the protocol is yet valid and can reach an agreement.

Outline: In section 2 we describe a simple asynchronous consensus protocol in the presence of Byzantine faults. The protocol discussed achieves correctness and validity always (probability 1) when fewer than one-fifth of the processes can be corrupted. We elaborate on the proof of the same in section 2. The given protocol always terminates, but has a constant expected running time only when $\sqrt{n}$ of the n processes are corrupted. For larger number of corrupted processes, the expected running time is exponential in n.

In section 3, we introduce a protocol that reduces the expected running time to $O(\log n)$ (for n processes), even when at most a fifth of the processes are corrupted. The protocol introduces the idea of compound processes, which are nothing but a subset of the existing processes that will together behave as one coherent virutal process. The above protocol requires $O(n^2)$ such compound processes, where each compound process has $O(\log n)$ of the actual processes. In section 3.3, we show that such compound processes can be selected uniformly at random. The probability that the protocol fails then is at most $O(2^{-n})$.

2 Simple Randomized Protocol

As a first step, we describe a simple randomized Byzantine Agreement protocol, from [2]. The protocol is fully asynchronous, *i.e.,* different processes are executed at different speeds.

We assume that there are n processes $\{p_1, p_2 \dots p_n\}$, where at most t can become faulty. Once a process p_i decides on a value, then it cannot revert the decision. Now the protocol at any process p_i having initial value x_i is as follows (same as in [2])

Step 0 Set round number $r := 0$.

Step 1 Send $(1, r, x_i)$ to all processes.

Step 2 Wait until messages of type $(1, r, *)$ are received from $n - t$ distinct processes. If more than $(n + t)/2$ messages have the same value v, then send $(2, r, v, D)$ to all processes, else send $(2, r, \perp)$ to all processes.

Step 3 Wait until messages of type $(2, r, *)$ are received from $n - t$ distinct processes.

1. If at least $(n + t)/2$ of these contain v, then decide v.
2. If at least $t + 1$ messages contains v, set $x_i := v$.
3. Else, $x_i \overset{R}{\leftarrow} \{0, 1\}$, *i.e.,* the binary value for x_i is set to 0 or 1, each with probability 1/2.

Step 4 Increment r and go to step 1.

Theorem 1. *[3] Let $n > 5t$. For any initial assignment of values the protocol, the protocol is a* t-resilient *consensus protocol.*

The above theorem is argued based on the arguments from [1] (Section 5.1). The difference is that the argument there is for the model where a faulty processor doesn't deviate from the protocol, but can halt mid-way. Their arguments have been adapted to the Byzantine case here.

Proof. We only show a sketch of the proof. Here we show, why the protocol reaches agreement. The case for validity is left to the reader as an exercise.

Firstly, there are no two valid messages $(2, r, v, D)$, $(2, r, v', D)$ such that $v \neq v'$. To see why, first we note that for any correct process to send $(2, r, v, D)$ it must receive more than $(n+t)/2$ messages of the form $(1, r, v)$. Similarly, some correct process saw more than $(n + t)/2$ of the form $(1, r, v')$. Therefore, more than $n + t$ distinct messages of the form $(1, r, *)$ were sent. Therefore, there must have been at least one correct process that sent out two distinct messages $(1, r, v)$ and $(1, r, v')$. Our protocol clearly doesn't allow that.

Next, if some process decides on v in round r, it sees at least $(n + t)/2$ messages of the form $(2, r, v, D)$. Therefore, every other correct process sees at least $((n + t)/2) - t = (n - t)/2$ such messages. Since, $n > 5t$, we see that $(n - t)/2 > t + 1$. Therefore, every correct process (including those that haven't decided v) sets v as its value for the next round.

Since $n > 5t$, all correct processes decide on v in the next round($r + 1$). Note that to send message $(2, r+1, v, D)$ all processes must find at least $(n+t)/2$ messages of the form $(1, r+1, v)$ among the first $n-t$ messages it receives. Accounting for the fact that it may yet receive messages from the faulty processes, we need that $(n - t) - t > (n + t)/2$ (all other correct processes will send $(1, r + 1, v)$ for the reasons described above). Since $n > 5t$, we have $(n - t) - t > (n + t)/2$. Similar argument, will hold for all correct processes receiving at least $(n + t)/2$ messages of the form $(2, r, v, D)$. Thus, all process that haven't decided on v in round r, will decide on v by round $r + 1$. □

Note that there is an exponentially small, albeit non-zero probability that the coin flips of the required amount of processes agree. Therefore, the algorithm will definitely terminate, but requires an exponential expected running time. But, when $t = O(\sqrt{n})$, the expected running time of the protocol is constant. The proof of the same has been omitted. We refer interested readers to [4] (section 4.2). Though the protocol discussed here is different from the protocol in [4], the efficiency analysis required for the above protocol is similar to theirs. A concise idea on efficiency analysis is :

Model the system as a Markov chain. The system is in state i, if i of the correct processes have value 1. The accepting states thus, are 0,1,... $(n - 3t)/2$ and $(n + t)/2 + 1, \ldots n - t$. In case of $t = \sqrt{n}$, the number of accepting states overwhelm the number of non-accepting states. Thus, the probability of transitioning to an accepting state is constant (and so is the expected running time).

3 Reducing the Expected Running Time

Note that the above protocol has a constant expected running time if $t = O(\sqrt{n})$. But restricting any protocol to such a small number of faults may not always be practical.

Bracha tried restricting the faults to be always $O(\sqrt{n})$ by using the given n processes to create m "compound processes", where $m = O(n^2)$. Every compound process P_i is nothing but a s sized subset $\{p_i^1, p_i^2 \ldots p_i^s\}$ of the n processes, where $s = O(\log n)$. A compound process P_i is correct if at least $2s/3$ of the constituent processes are correct and faulty otherwise.

3.1 Compound Process

Note that a compound process is just a virtual entity. Thus, its variables and operations must be implemented at the level of its individual constituents. Variables and functions at the level of the compound level are represented in all caps and its corresponding implementation in the individual processes in small. For example VAR_P is a variable of the compound process P and $var_{P,i}$ is its copy in process p_i.

All processes hold their initial value of the compound process as $var_{P,i}$. Every correct process p_i has to communicate to every other process p_j this variable. This is done using the already known deterministic byzantine general protocols. Thus, every process p_j runs $Byzantine(var_{P,i})$ on all other i. VAR_P is arrived at using a consensus protocol

$$Consensus(VAR_P) = \underset{1 \leq k \leq s}{\text{majority}}(Byzantine(var_{P,k}))$$

Thus, arriving at a consensus on VAR_P, $var_{P,i}$ across all correct processes p_i is consistent. Note that such a protocol runs in time $t+1$, which is $O(\log n)$. Note that for correctness, the above protocols require $t < N/3$, which in this case $s/3$. Thus, if there are fewer $2s/3$ correct processes the compound process is deemed faulty.

Further, note that the basic randomized protocol required a coin toss by individual processes. But in our case, we require the compound process to implement such a coin toss (called *COINTOSS*). There are known protocols, that allow for such group coin-tosses assuming cryptography, where the same bit is generated uniformly at random by every correct participating process. We refer the readers to [5] for an example. Again, such protocols run in time $t+1$ and tolerate upto $t = N/2$ faults.

3.2 Protocol

Before implementing the protocol, we implement individual functionalities. Consider two compound processes $P = \{p_1, p_2 \ldots p_s\}$ and $Q = \{q_1, q_2 \ldots q_s\}$. The following protocol is verbatim from [3].

INITIALIZATION for all $1 \leq i \leq s$, $value_{P,i} := value_{p_i}$; $value_{P,i} := Consensus(VALUE_P)$.

SEND P *SEND*s m to Q such that for all $1 \leq i, j \leq s$, p_i sends (P, m) to q_j.

RECEIVE P *RECEIVE*s m from Q such that for every process p_i, if p_i receives more than $2s/3$ messages of the form (Q, m) from distinct subprocesses of Q, then $\text{msg}(Q)_{P,i} := m$, else it is undefined. Finally, every process p_i receives $Consensus(\text{msg}(Q)_{P,i})$.

Now, we use the above implementations to describe the byzantine agreement protocol for N compound processes off which T are faulty. For some compound process P every entails:

1. *SEND* $VALUE_P$ to all processes.
2. If *RECEIVE* more than $(N+T)/2$ messages with the same value v, then $VALUE_P := v$ and *SEND* $VALUE_P$ to all the processes.
3. (a) If *RECEIVE* more than $2T$ messages with value v, then $DECISION_P := v$.
   (b) If *RECEIVE* more than T messages with value v, then $VALUE_P := v$.
   (c) Otherwise, $VALUE_P :=$ *COINTOSS*.
4. Go to step 1 (of the next round).

The above protocol is correct because every correct subprocess mimics the compound process (based on the correctness of the consensus and coin tossing protocols). Thus, every correct compound process behaves like a correct atomic process. The above protocol for atomic processes is nothing but a generalization of the basic protocol given earlier. For further details on the proof, we refer the readers to [3].

3.3 Choosing m and s

The following lemma is taken verbatim from [3]. The proof sketch here is merely a shorter description of the proof from [3].

Lemma 1. *Given n processes such that t of them are faulty and $t/n = p$, for any $\epsilon > 0$, there is a collection of $m = O(n^2)$ subsets of size $s = c \log n$ such that no more than $O(n)$ of them contain more than $(p+\epsilon)c \log n$ faulty processes for any choice of faulty processes.*

Proof Sketch. An (s, m) configuration is a list of m sequences of s processes. Let us list all such configurations. The list L then contains n^{ms} configurations. Thus, to represent every configuration we require $ms \log n$ bits.

Suppose there is an assignment A of t faults, such that all (s, m) configurations have more than $\sqrt{m}$ faults. Then let B be the sequence of $\sqrt{m}$ faulty processes such that each of them contains more than $(p + \epsilon).s$ members of A. Let C be a list of $m - \sqrt{m}$ correct processes. Let D be a method of merging B and C. Note, that A, B, C, D can then be used to uniquely identify any (s, m) configuration.

A can be represented in n bits (a single vector, with the bit i set to 1 if i is faulty, 0 otherwise). L^A is the list of all sequences that have more than $s.p$ members from A. We see that

$$|L^A| = \sum_{s(p+\epsilon) \leq k \leq s} \binom{s}{k} t^k (n-t)^{s-k} < n^s.e^{-\epsilon^2 s}$$

Therefore to represent B we require $\sqrt{m}. \log |L^A| \leq \sqrt{m}.(s \log n - \epsilon^2 s \log e)$ bits. To represent C we require $(m - \sqrt{m})s \log n$ bits. To merge C and B we need a list $i_1, \ldots, i_{\sqrt{m}}$. The $j^t h$ element in B is inserted after the i_jth sequence in C. Thus D requires $\sqrt{m} \log m$ bits.

Therefore, total bits required in this representation is $n + \sqrt{m}.(s \log n - \epsilon^2 s \log e) + (m - \sqrt{m})s \log n + \sqrt{m} \log m$ bits. Comparing with the canonical representation, the canonical representation requires $\sqrt{m}(s\epsilon^2 \log e - \log m) - n$ more bits. Suppose $m = n^2$ and $s = (3 + 2 \log n)/(\epsilon^2. \log e)$. Substituting, we get that totally the canonical representation requires n bits more than the above representation. Clearly, this is impossible, since the canonical representation is the most compact bitwise representation. □

Since, all faulty processes can be represented with n bits fewer than all possible compound processes, only 2^{-n} fraction of all compound processes are faulty. Therefore any randomly selected (s, m) configuration is a faulty configuration with probability 2^{-n}.

References

[1] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.

[2] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In Probert et al. [6], pages 27–30.

[3] Gabriel Bracha. An O(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.

[4] Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In Probert et al. [6], pages 12–26.

[5] Andrei Z. Broder and Danny Dolev. Flipping coins in many pockets (byzantine agreement on uniformly random values). In *FOCS*, pages 157–170. IEEE Computer Society, 1984.

[6] Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors. *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983.* ACM, 1983.