Robust Multi-Agent Decision Making in Faulty Environment

(Preliminary Version)

Amotz Bar-Noy
Stanford University

Danny Dolev
IBM Almaden
Research Center

Dragutin Petković
IBM Almaden
Research Center

Abstract

Due to the trends toward cheaper and more powerful processors and communications, we are witnessing the increase in distributed processing in many computer applications. Pattern recognition and decision making will most likely follow this trend, especially in manufacturing (large number of inspection devices, autonomous robots etc.) and in military (arrays of sensors, satellites etc). In this paper we are dealing with the problem of achieving common decision (consensus) among multi-agents (processors) making their independent decisions in the presence of faults. For example, how to achieve global control (i.e. shut down) of array of robots in manufacturing, assuming all robots can make their own decisions, are connected to each other, but failures in robots and/or communication lines can occur?

The main contributions of the paper are: (a) novel algorithm for achieving consensus among multi-agents in a faulty environment, and (b) overview of the state of the art in this field that has been studied in the Computer Science community. The algorithms we present are deterministic and can be easily implemented in hardware. They guarantee reaching consensus while sending single bit messages. The algorithms can be pipelined for better total throughput.

1 Introduction

With the proliferation of computer technology and the trends toward better, low-priced communication, and smaller, more powerful and cheaper processors, we are witnessing increase in distributed processing in many computer applications. Pattern recognition and decision making will most likely follow the same trend. One way of exploiting this new technology is in using parallel processing in solving pattern recognition problems. Another is the use of many independent systems that use pattern recognition and decision making (like inspection devices, monitors, robots etc.) that are connected in a large distributed systems. While the above solutions offer many advantages like modularity, superfluity, accessibility, etc., special care has to be devoted to the algorithms to ensure efficient and proper performance. The questions of fault tolerance, agreeing upon common data, and coordination of activities naturally arise in these applications.

In order to motivate the reader, let us analyze the following hypothetical case: We have independent agents R_i that are capable in making independent decisions (example of robots in manufacturing floor, military warning systems etc.). There are also control centers C_j that are connected to R_i and among themselves. A subset of the system must agree on common decision for the system to function properly (all robots to shut off, warning to be issued etc.). How can we ensure that the consensus is reached even in the presence of failures in some R_i, C_j, and/or noisy communications. This problem has been studied in the Computer Science community and is known as Distributed Consensus, or Byzantine Agreement. The importance of it resides in overcoming the uncertainty that faults introduce. This can be achieved if all the reliable members of the system agree upon the content of messages being sent (or the state of elements) in the system, especially of those corresponding to the faulty parts of the system, even where the faulty parts cannot be uniquely identified. Typical faults can be failure to send or relay proper information to some of the agents, or failure to receive information being sent. It may include a crash of elements, etc.

In this paper we will give a brief overview of the current state of the art in Distributed Consensus and also present new approach to the consensus algorithms [BD]. We will concentrate on a simple model and a binary decision value in order to exhibit the algorithms. These algorithms are derived having technological implementation in mind, i.e. they can easily be built and incorporated in existing systems. The implementation algorithm requires the participating elements (to which we refer as *agents* or *processors*) to occasionally send single bit messages. All previous algorithms, excluding those in [BD], required sending messages of large size. The proposed family of algorithms in our paper are deterministic, require sending single bit messages, and can be implemented in hardware. Several instances of the algorithm can be pipelined, one following the other in an on-line fashion.

In the literature various models and several variation on the basic consensus problem appear. The basic and simple case studied in this paper can be served as a guideline to both the difficulties and capabilities of such an approach. The algorithms presented here are applicable in organizing complex pattern recognition systems that will soon be emerging in industry and military applications.

CH2614-6/88/0000/0966$01.00 © 1988 IEEE

The paper is organized as follows: Section 2 contains the basic definitions and a short description of some of the primary known results. In Section 3 we present a description of our basic algorithm. Its generalizations and possible implementations are given in Section 4. Concluding remarks are included in Section 5.

2 The Consensus Problem

The consensus problem is the key property in coordinating multi-agent system, though in many cases this is not the sole goal of the system. Studying this agreement enables us to isolate the main difficulties in coordinating a system in a faulty environment.

Given a distributed system of n agents, $p_1, \ldots, p_n$, where each one has an initial binary value v_i ($v_i \in \{0,1\}$); let t be the upper bound on the number of agents that might become *faulty* during the algorithm. We are looking for an algorithm that at its completion the following two conditions hold:

- *validity* – All the nonfaulty (*correct*) agents agree on the same binary value.
- *consistency* – If the initial value was the same for all the correct agents, then this value will be the final *decision* value.

This problem is called the *Consensus Problem*. It is a variant of the known Byzantine General Problem [PSL], in which the agents are required to agree on a binary value of one of them called the general. One can easily generalize the above problem to one in which we are concerned with the input values of only a subset of the agents. All these problems were asked in various models. In this paper we concentrate on the very basic and simple model:

- The system is completely synchronized.
- The communication is via a complete and reliable network, i.e., each agent can send a message to every other agent and messages arrive unaltered and on time.
- The agents are deterministic, and no randomized operations are allowed.
- The faulty agents can be malicious. They might collude in order to prevent reaching the agreement.

Note that an actual system may differ from the above suggested model. Agents are not malicious, but overcoming this kind of faulty behavior implies overcoming any possible fault. Most of the basic lower bounds still hold for even a much simpler type of fault. For simplicity communication errors are ignored, though one can associate faulty communication lines with faulty agents that are adjacent to these lines.

The synchronization of the system is described by *rounds*. In each round an agent performs three operations. It sends messages, receives messages, and performs internal computation. In the last round of the algorithm, after the internal computation is completed, the agent must *decide*.

Three complexity measures determine the efficiency of a solution:

1. The ratio between n and t.
2. The maximal number of rounds required in the worst case, denoted by r.
3. The maximum size in bits of a single message, denoted by m.

Traditionally researchers use the total number of messages (or bits), being sent during the algorithm, instead of the third parameter. We believe that in order to explore the right trade-off among the parameters, the size of each message is a better measure.

To demonstrate the saddle issues in reaching agreement or consensus consider the following example. Agents C_1, C_2, and C_3 have to communicate their individual binary values to every agent. They either send 1 or nothing. Thus, not sending means having a 0. All agents know beforehand when they are scheduled to communicate. One can think in terms of control centers issuing a possible "shut-off" to all robots. All control centers and robots should reach the same final decision. Consider the scenario in which C_1 has a 1, and succeeds to send it to everybody, C_3 has a 0 and do not send anything, but C_2 has a 1 and due to a failure it can send it to exactly half of the agents. In this scenario half of the agents will decide 1 and half will decide 0. Moreover, having additional round in which everybody sends its majority to everybody else will not help, as long as C_2 will be able to communicate with exactly half of the agents.

The above example clarifies why a simple majority voting will not enable us to reach consensus when faults exist. Reaching consensus requires processes to communicate in several rounds, notifying each other on values being received. The main obstacle resides in the uncertainty when the number of votes is almost equal. The next idea one will come with is using thresholds, but again similar scenarios can be applied around those thresholds. In later sections we will show how our algorithm overcomes the problems reflected by the above example.

2.1 Known Results

In this subsection we list the main known results on the consensus and byzantine agreements.

There are three known lower bounds for the complexity measures listed in the previous section,

1. $n \geq 3t + 1$ [PSL].
2. $r \geq t + 1$ [DS].
3. $m \geq 1$ (obvious).

Other lower bounds relates the total number of messages to n and t [DR], but are left out of this paper. The known upper bounds optimize two of the parameters but "pay" in the third one:

1. The *Full Information* algorithm uses $n \geq 3t + 1$ agents and $r = t + 1$ rounds, but there are messages of size $m = O(n^t) = O(t^t)$ [PSL,BDDS].
2. The *Linear Size Message* algorithm, where $m = O(n) = O(t)$ (almost optimal), uses minimal number $n \geq 3t + 1$, of agents, but takes $r = 2t$ rounds [BDDS].
3. The *Square* algorithm with $m \leq n$ and $r = t + 1$, but $n \geq 2t^2 - 3t + 2$ [DRS].

Recently three families of algorithms were presented [Co,BDDS,BD]. Every family matches one of the lower bounds and gives a trade-off between the other two.

2.2 Our Results

The basic version of our algorithm requires $n = (2t + 1)(t + 1)$ agents, but only $r = t + 1$ rounds and uses only 1-bit messages.

Thus, when the number of agents is large enough we can improve the maximum message size by a factor of n. An interesting property of this algorithm is that each agent sends a value only once. We call this algorithm the *Beep Once* algorithm (to be presented in Section 3).

From this algorithm we derive the following family of algorithms (to be presented in Section 4), parameterized by k:

The Beep algorithms: For every k, $1 \leq k \leq \infty$, there exists an algorithm with single bit messages. (k determines the trade-off between the number of rounds and the ratio of correct to faulty agents). The number of agents is,

$$n = 4t^{\frac{(k-1)}{k}} + t^{\frac{(k-1)}{k}} \leq 5t^{1+\frac{1}{k}},$$

and the number of rounds is,

$$r = \sum_{i=1}^{k} t^{\frac{1}{k}} \leq t + 2t^{1-\frac{1}{k}}.$$

Asymptotically, the result is,

$$\forall_{1 \leq k \leq \infty} \forall_{0 < \epsilon \leq 1} \exists T \; s.t. (t \geq T \wedge n \geq 5t^{1+\frac{1}{k}}) \Longrightarrow r \leq (1+\epsilon)t.$$

Observe that almost every algorithm can be simulated by a single-bit algorithm by using many more rounds and encoding messages by bits and time. Our algorithms use the same number of rounds and still require sending only single bits.

3 The Beep Once algorithm

In this algorithm the number of agents is $n = (2t+1)(t+1)$. It requires $r = t+1$ rounds and uses only 1-bit messages.

The algorithm: Partition the n agents into $t+1$ disjoint sets, each of cardinality $2t+1$. Denote these sets by $S_1, S_2, \ldots, S_{t+1}$. There are $t+1$ rounds in the algorithm, in round number i only agents from set S_i send messages. In the algorithm, whenever a message that is supposed to be sent does not arrive, the receiver assumes it has received a default value, 0.

- $i = 1$: Every agent in S_1 sends its initial value to every agent in S_2.
- $1 < i < t+1$: Each agent $p \in S_i$ receives $2t+1$ bits. Agent p sends the majority value of all these $2t+1$ values to all the agents in S_{i+1}.
- $i = t+1$: The agents in S_{t+1} send the majority value of the $2t+1$ values they have received from S_t to all the n agents.
- **Decision:** The decision value for every agent is the majority value of the $2t+1$ bits it receives from the agents in S_{t+1}.

Proof of correctness: In each set S_i, there are $2t+1$ agents, at least $t+1$ of them are correct. If these $t+1$ agents send to S_{i+1} (or to everybody in case $i = t+1$) the same value, this will be the majority value of all the correct agents in the set S_{i+1} (or the decision value).

In such a case, the correct agents in S_{i+1} also forward the same value. This process continues until decision is made on this value. In particular, the consistency condition holds, because all the correct agents have the same initial value and this will be the decision value.

There are $t+1$ disjoint sets and at most t faulty agents, which implies the existence of a set S_i that does not contain any fault. Therefore, all the correct agents from S_{i+1} (or everybody when $i = t+1$) received the same $2t+1$ bits and evaluate the same majority value. By the previous argument this will be the decision value and the validity condition holds. ∎

Let us return now to the example of Section 2. Recall that agents C_1, C_2, C_3 have to communicate their individual binary values to all the agents. In case of a single fault, as in that example, it is enough to choose a set of three additional agents, say R_1, R_2, R_3, such that the three C_i's will send their values to the three R_j's. Each one of them will take a majority value and send its majority to every other agent. Every agent will take the majority of the values it receives from the R_j's as the decision value. One can easily see that no matter whether the fault occurs among the C_i's or among the R_j's, all agents do reach a consensus. Thus, by asking a subset to take a majority we prevent C_2 fooling us again and again, and thus overcoming the problems we had in the example of Section 2.

4 The Beep Family of algorithms

This section presents a generalization of the Beep Once algorithm and very simple hardware implementation of it. An explicit algorithm for every k, is given in the first subsection. In the second subsection a pseudo pascal program is presented based on the algorithms of the first subsection. The complete hardware implementation can be easily derived from that presentation. In the third subsection an implementation of the Beep Once algorithm is given. This implementation is simpler, but do require the ratio of $O(t^2)$ between correct to faulty, which is the limitation of the Beep Once algorithm.

4.1 The description of the algorithm

In order to present the algorithm a virtual tree is used. The tree is recursively described, but the algorithm itself will be explicit. Each node in the tree represents a set of agents. Assume that $n = 4t^{(k+1)/k} + t^{(k-1)/k}$ and denote the tree by $T(n,t,k)$. For simplicity assume that $t = s^k$ for some integer s.

- In case $k = 1$ partition the n agents into $t+1$ disjoint sets of almost equal cardinality (the cardinalities of every two sets differ at most by 1). The root of the tree is the set of all n agents and the children are these $t+1$ sets.
- In case $k > 1$ the agents are partitioned into $t^{1/k}$ disjoint sets of equal cardinality. Define n' and t' according to the following expression,

$$n' = 4t + t^{\frac{k-2}{k}} = 4(t^{\frac{k-1}{k}})^{\frac{k}{k-1}} + (t^{\frac{k-1}{k}})^{\frac{k-2}{k-1}} =$$
$$= 4t'^{\frac{(k-1)-1}{k-1}} + t'^{\frac{(k-1)-1}{k-1}}.$$

  The root of this tree is the set of all n agents and each child of the root is a $T(n',t',k-1)$ tree associated with one of these disjoint sets. The above formula proves that the relations between n', t' and $k-1$ are preserved. In Figure 1, an example of such a tree for $k = 3$ is given.

Unfolding the recursion one can see that for each agent there is a corresponding path in the tree. The agent belongs to every set that is represented by the nodes of its corresponding path. The set represented by the root consists of all the agents.

The tree is assumed to be ordered from left to right. The tree is post order numbered and from now on we refer to each node in the

tree by its post order number. The root of the tree is numbered by 0.

For a given tree, the following definitions and notations are used in specifying the rules of the algorithm:

- *Left*: The set of all the nodes in the tree that do not have a left sibling.
- *Right*: The set of all the nodes that do not have a right sibling.
- *Start*: The nodes that neither them nor any one of their ancestors have a left sibling (the left edge of the tree).
- $Next(x)$: A function from the nodes of the tree into the nodes of the tree.
  - $x = 0$ (the root): $Next(0)$ = the left most children of the root.
  - $x \notin Right$: $Next(x)$ = the right sibling of x.
  - $x \in Right$: $Next(x)$ = the parent of x.
- $Prev(x)$: A function from the nodes of the tree into the nodes of the tree.
  - $x = 0$ (the root): Undefined.
  - $x \in Left$: $Prev(x)$ = the left sibling of x.
  - $x \in Start$: $Prev(x) = 0$, the root.
  - $x \in Left \bigwedge x \notin Start$: $Prev(x) = Prev(y)$ where y is the parent of x. That is the left sibling of the closest ancestor that has a left sibling.

The algorithm: The number of rounds, denoted by r, is the same as the number of nodes in the tree. In each round i, $0 \leq i \leq r$, the agents follow the following rules:

- *Who send messages?* The agents that belong to the set represented by node number i.
- *Whom to send messages to?* To the agents in the set represented by node number $Next(i)$.
- *What to send?*
  1. If $i = 0$, send your initial value.
  2. If node i is a leaf, send the majority of the values that you got from $Prev(i)$.
  3. Otherwise, node i is neither the root nor a leaf. The set represented by $Prev(i)$ consists of $m = 4t^{j/k} + t^{(j-2)/k}$ nodes, for some j that is a function of the level of node i in the tree.
     (a) In case the majority value from $Prev(i)$ was supported by at least $m - t^{j/k}$ agents, send this value.
     (b) Otherwise, send the majority of the values you received from your right most child.

Decision: The majority of the values you received from the right most child of the root.

Note that every agent in node i receives messages from all agents in $Prev(i)$, because either $i = Next(Prev(i))$ or i is a predecessor of $Next(Prev(i))$. Thus the algorithm is well defined.

The proof of correctness is a variation over the one in [BD] and will be omitted from this Technical Report.

4.2 The implementation

The following properties of the algorithm facilitate the construction of a hardware implementation:

1. All the messages are one bit. Therefore, the communication patterns are very simple.
2. The internal computation is very elementary. Each agent needs only two kinds of gates. A *majority* gate with at most n inputs and one output. A *vast majority* gate with two output lines: the first one is the majority value and the second one is 1 if this value is supported by the threshold determined by the algorithm. Denote this gates by MAJ and $VMAJ$, respectively.
3. The agents that everyone interacts with (send or receive messages) are determined only by the round number. Therefore, it is possible to simplify communication by fixing the communication patterns beforehand. The agent uses the command $SEND(x, S)$, which means sending the value x to the agents in the set S. The following notations are used:
   - $S(p, i)$: The set of agents to which agent p sends messages in round i.
   - $R(p, i)$: The set of agents from which agent p receives messages in round i.
   - $N(p)$: The set of round numbers in which agent p sends messages.
   - $M(p)$: The set of round numbers in which agent p uses a MAJ gate.
   - $VM(p)$: The set of round numbers in which agent p uses a $VMAJ$ gate.
   - $l(p)$: The round number in which agent p is in a leaf in the tree.
4. The internal memory is very small. Each agent needs a LIFO queue (last in first out) with k cells each one consists of two bits, (a possible outcome of a $VMAJ$ gate) and it knows how to handle the queue by the $PUSH$ and POP commands. In addition each one has three registers denoted by $x, y, z,$, the initial binary value denoted by *init*, and the decision value denoted by *decision*.

The code for an agent p:

```
SEND(init, S(p,0));
for i := 1 to r - 1 do
begin
    if i ∈ M(p) then x := MAJ(R(p,i));
    if i ∈ VM(p) then PUSH(VMAJ(R(p,i)));
    if i ∈ N(p) then if i = l(p)
                     then SEND(x, S(p,i))
                     else begin
                              POP(⟨y,z⟩);
                              if z = 1
                              then SEND(y, S(p,i))
                              else SEND(x, S(p,i))
                          end
end;
decision := x;
```

Again, we omit the proof. Although the code uses the **if** command in the implementation this command is embedded in the hardware by using the common clock pulses (recall that these input and output commands depend only on the round number).

4.3 The implementation of the Beep Once Algorithm

The code for the Beep once algorithm is much simpler. Throughout the algorithm each agent p sends messages in at most one round, denoted by $round(p)$. Recall that the agents are partitioned into the $t+1$ disjoint sets: $S_1, S_2, \ldots, S_{t+1}$ and denote by P the set of all agents.

Code for an agent p:

If $round(p) = 1$ then $SEND(init, S_2)$;
For $i := 2$ to t do
    If $round(p) = i$ then $SEND(MAJ(S_{i-1}), Si + 1)$;
If $round(p) = t + 1$ then $SEND(MAJ(S_i), P)$;
$Decision := MAJ(S_{t+1})$;

Observe that in the above code only the MAJ gate is used, and no additional internal memory is required. Pipelining this algorithm is even simpler, because each agent participates in the algorithm only once.

5 Conclusion

In this paper we gave a brief overview of current state of the art in robust multi-agent decision making in a faulty environment (distributed consensus); and also presented a family of novel algorithms to achieve consensus that are deterministic, simple, require single bit messages, and can be implemented in hardware. Although these types of algorithms were studied in the CS community, we feel that they did not receive enough attention by other communities and especially by the designers of complex pattern recognition or decision systems.

As the computer technology improves, we will see more and more distributed systems, where a number of independent and intelligent agents are compounded into a system. Faults inevitably occur in such complex systems. In order for such systems to work properly the issues of reaching common decision (consensus) in the presence of faults have to be addressed. The work presented in this paper offers a structural solution to this problem.

Our approach is illustrated on a hypothetical example of a number of autonomous robots in manufacturing that all have to "agree" on a common decision determined by the values of some central controllers. Consequently, our algorithms are easily applicable to other fields like military for example (case of an array of autonomous sensor-weapon platforms etc.). We hope that this paper will offer valuable guidelines to designers of such complex, distributed pattern recognition systems.

Acknowledgment:

The authors would like to thank Ray Strong for his part in developing the Beep Once Algorithm.

References

[BD] A. Bar-Noy, and D. Dolev, "Families of Consensus Algorithms," *IBM Research Report*, 1987.

[BDDS] A. Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong, "Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement," *proceedings, the 6th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Aug. 87, pp. 42-51.

[Co] B. A. Coan, "A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols," *Proceedings, the 5th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Aug. 1986.

[DR] D. Dolev, and R. Reischuk, "Bounds on Information Exchange for Byzantine Agreement," *Journal of the ACM*, Vol. 32, pp. 191-204, 1985.

[DRS] D. Dolev, R. Reischuk, and H. R. Strong, "Early Stopping in Byzantine Agreement," *IBM Research Report RJ5406 (55357)*, 1986.

[DS] D. Dolev, and H. R. Strong, "Authenticated Algorithms for Byzantine Agreement," *Siam Journal on Computing*, Vol. 12, pp. 656-666, 1983.

[PSL] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *JACM*, Vol. 27, 1980.