

Building Consensus in Context-Aware Systems Using Ben-Or's Algorithm: Some Proposals for Improving the Convergence Speed

Phuong T. Nguyen^(✉)

Research and Development Center, Duy Tan University, 182 Nguyen Van Linh,
Danang, Vietnam
phuong.nguyen@duytan.edu.vn

Abstract. For context-aware systems, it is essential for the constituent components to react flexibly to changes happening in the surrounding environment. In distributed networks, a failure of some nodes might disrupt the whole system. Given the circumstances, it is necessary for the remaining nodes to find consensus on a new organizational structure. In this paper, we propose an approach for improving the convergence speed of a prominent algorithm for finding consensus: Ben-Or's algorithm.

Keywords: Context-aware systems · Ben-Or's algorithm

1 Introduction

Our problem involves searching for consensus in a group of network nodes where the nodes have to self-organize to deal with environmental stimuli occurring at execution time [4]. Fault tolerance, when being confronted with, means that in the event of network crashes network nodes are able to organize themselves to recover to a stable state. This leads to the problem of searching for an eventual agreement among distributed nodes.

Building consensus among distributed network nodes in presence of failure has been identified as a thorny issue in distributed computing. When at least one node fails, it has been shown that there exists no deterministic algorithm for solving consensus using asynchronous message passing [2,3]. In this paper, we present an approach of building consensus in distributed systems as an amendment for an existing randomized consensus algorithm, Ben-Or's algorithm [1]. The experimental results have shown that our proposed algorithm considerably improves the convergence speed of the original approach.

The paper is organized as follows. Section 2 introduces consensus in distributed systems. Section 3 presents a biological background. Section 4 brings in our approach, a honey bee inspired algorithm for attaining consensus. An implementation as well as simulation and the experimental results for the two algorithms are highlighted in Sect. 5. Section 6 draws future work and finally concludes the paper.

2 Literature Review

2.1 The Problem of Finding Consensus

The problem of building consensus in distributed systems is described in detail as follows: A set of $n \geq 2$ nodes $P = \{p_1, p_2, \dots, p_n\}$ has to negotiate using asynchronous message passing and decide on the same value from a common set of inputs. In the scope of this paper, we discuss binary consensus, i.e. the input values $v \in \{0, 1\}$ [1, 3, 9, 12]. The properties of consensus [9]:

- At most f nodes may fail. The nodes that fail are called faulty nodes; the nodes that still work are called non-faulty nodes.
- Each node p broadcasts its proposal as a message to all other nodes.
- A node makes a decision based on the set of messages it received.
- A message never fails once it has been sent.
- A node p can send its report or proposal value v to some nodes, and it may crash before sending the message to the remaining non-faulty nodes. As a result, some nodes have v and some don't.

Three key requirements need to be met by every consensus algorithm [3]:

- Termination: Every node must ultimately decide.
- Agreement: All correct nodes decide on the same value.
- Validity: The chosen value must be the input of at least one of the nodes.

2.2 The FLP Theorem

No matter how simple the definition is, the solution for the problem remains a challenge in distributed computing. In [2] Fischer, Lynch and Paterson prove that with an asynchronous message passing system, there exists no deterministic algorithm for solving the problem described in Sect. 2.1 in presence of failures. The theorem has been named after the authors - the FLP theorem [2].

2.3 Randomized Consensus: Ben-Or's Algorithm

Ben-Or's randomized consensus algorithm is considered as the first one for solving consensus. In his approach, it is assumed that at most $f < n/2$ nodes may fail during execution. The algorithm is described in the pseudo code Algorithm 1 [1]. A proof for the correctness of the algorithm is available in [8].

- Procedure *Report*(k, x) broadcasts a message containing information about the current round k and the proposed value to all other non-faulty nodes.
- Procedure *WaitFor*($k, *$) waits for all incoming messages containing k and a report/proposal value.
- Procedure *Propose*(k, v) broadcasts a proposal value v in round k to all non-faulty nodes.
- Procedure *Decide*(v) makes a decision on the value v .
- Procedure *ChooseRandom*(0, 1) returns either 0 or 1 with equal probability.

Algorithm 1. Ben-Or's Consensus Algorithm

```

1: procedure BENORCONSENSUS( $v_p$ )
2:    $x \leftarrow v_p$ 
3:    $k \leftarrow 0$ 
4:   while true do
5:      $k \leftarrow k + 1$ 
6:     Report( $k, x$ )
7:     WaitFor( $k, *$ ) from  $n-f$  nodes  $\triangleright *$  is either 0 or 1
8:     if there are more than  $n/2$  messages containing  $v$  then
9:       Propose( $k, v$ )
10:    else
11:      Propose( $k, abstention$ )
12:    end if
13:    WaitFor( $k, *$ ) from  $n-f$  nodes
14:    if there are at least  $f+1$  messages containing  $v$  then
15:      Decide( $v$ )
16:    else if there is at least 1 message containing  $v$  then
17:       $x \leftarrow v$ 
18:    else
19:       $x \leftarrow \mathbf{ChooseRandom}(0,1)$ 
20:    end if
21:  end while
22: end procedure

```

Remarks. In Algorithm 1, the *abstention* value in Line 11 indicates that a node p prefers neither 0 nor 1, it abstains. The value is something like a “nuisance” since it does not contribute to a convergence of consensus.

In [8], it has been proven that in a round k if every two nodes propose then they propose the same value. Once a node p decides in Line 15 in round k then all nodes will decide in the next round $k+1$ [8]. But a node decides on a value v only if it receives more than f propose messages containing v in Line 14. In Line 9, a node proposes a value only if it has received more than $n/2$ report messages with the same value v . That means, when the number of messages containing the same value does not exceed $n/2$ then the node cannot propose any value. If so, then a quorum cannot be reached in Line 14. Given the circumstances Ben-Or's algorithm gains a consensus very slowly.

3 Biological Background

3.1 Bio-inspired Computing

The main idea of bio-inspired computing is to emulate activities from natural communities in computer systems to infuse alike features. As a result, they can react to environmental changes and be resilient to perturbations and errors. So far, there have been several computational solutions inspired by colonies in nature, e.g. ant and honey bee optimization algorithms. By looking into the

honey bee colony, we found that honey bees have a mechanism to reach an agreement on choosing a new beehive given that various candidate sites may have been nominated during the decision making process. To our knowledge, the mechanism is notable and can be applied to the problem of consensus among distributed network nodes.

3.2 Bee Communication

Foraging takes place in summer, worker bees spread out to collect and accumulate food for the whole year. Each individual bee may find different food sources, but as the time goes by, the colony tends to head for rich food source where they can get more nectar and pollen. A scout bee, after finding a new food source, returns to the hive and performs a *waggle dance* to notify other honey bees about the food source. With this information, other honey bees are able to know where and how far they must fly in order to get to the food sources.

3.3 Consensus

Swarming occurs when the bee population increases thus causing congestion and difficulties in maintaining good hygiene in the beehive. Before swarming, the queen lays eggs to raise a new queen bee. When the new queen bee arrives, the old queen bee takes a number of workers with her and they fly to a new temporary location, e.g. a tree branch, that is near to their beehive. The swarm stays on the temporary residence for a short time while waiting for a new residence. Although many sites may have been nominated during swarming, eventually all scouts make a unanimous decision, they reach a final agreement for a site, normally the best one. The honey bee colony has a mechanism to build consensus among all scout bees. Scout bees dance for good sites vigorously and lastingly than for sites with inferior quality [5]. It has been shown that the strength of a dance is decreased linearly over the time [6]. When the dance expires, scout bee stops dancing for the site and chooses randomly a new site to follow and dance for. The rate of recruitment for a site is proportionate to the number of waggle dances by bees. Since dances for good sites prolong, a good site attracts more bees. As a result, a good site gains a quorum much easier than sites with inferior quality [7]. The consensus mechanism is summarized as follows:

- Phase 1: Each scout explores a site, evaluates and assigns a quality to the site.
- Phase 2: The scout flies back to the swarm and dances to advertise for the site it has found. The better the site quality, the stronger and livelier the bee dances for it.
- Phase 3: After dancing, the bee returns and explores the site and then backs to the cluster. It dances for the site again, but less stronger. This step is repeated and the strength of the dance is decreased until it reaches 0.
- Phase 4: When the dance expires, the scout selects randomly a dance to follow. It then flies to the site that is being promoted by the dance it follows. The process repeats from Step 1 to Step 4.

- Phase 5: If the number of bees gathering around a site exceeds a certain threshold then a quorum is reached. The scouts fly back to the swarm and signal the whole colony to depart for the chosen site. The search ends.

4 A Honey Bee Inspired Algorithm for Building Consensus in Distributed Systems

4.1 Similarities

We see that Ben-Or’s algorithm may reach a pretty slow convergence given that nodes do not choose the same value in the randomization phase. Regarding the problem of reaching an eventual agreement among nodes in distributed computing, we witness a substantial coincidence between consensus in distributed systems and consensus in the honey bee colony as shown in Table 1.

The fact that the two phenomena have a lot in common and honey bees possess a good mechanism to deal with consensus encourages us to employ the honey bee’s consensus mechanism in building consensus in distributed computing.

4.2 Proposed Amendments

We call $M_p(k)$ is the set of messages received by node p at round k ; $C_p(k, 0)$ and $C_p(k, 1)$ are the cardinality of the set of messages that contain 0 and 1, respectively. When $C_p(k, 0) = C_p(k, 1)$ we say it is a tie for the two sets. As we have seen in Algorithm 1, it is expected that the majority of nodes report the same value in Line 6 and then a quorum is reached in Line 14. However, a node proposes a value only if it receives more than $n/2$ messages containing the same value, otherwise it proposes the value *abstention*. We may “waste” the chance that nodes report the same value in round $k + 1$ given that almost $n/2$ messages containing the same value v are sent in Line 13.

It should be noted that from rounds $k > 1$, the value x reported in Line 6 is the result from either Line 15 or 17 or 19 in round $k - 1$. We propose a

Table 1. Analogies

Honey bee consensus	Distributed system consensus
Scout bees	Nodes
Sites	Values
Each scout bees dances for a site	Each node reports a value
Scouts nominate a site by dancing	Nodes propose a value by broadcasting messages
Scouts select a site if the number of bees gathering around it exceeds a threshold	Nodes decide on a value v if they receive more than f report messages containing v
All bees reach eventual agreement for a site	All nodes eventually decide on the same value

way to *guide* the nodes to opt for the most sensible value v instead of choosing a random value. We consider the case that the majority requirement is not satisfied: $C_p(k, v) < n/2$; but there is a *plurality* of nodes that report v , that means: $C_p(k, 1 - v) < C_p(k, v) < n/2$. The honey bee inspired algorithm for building a consensus is illustrated in the pseudo code Algorithm 2. The main functions are explained as follows:

Algorithm 2. Honey Bee Consensus Algorithm

```

1: procedure HONEYBEECONSENSUS( $v_p$ )
2:    $x \leftarrow v_p$ 
3:    $k \leftarrow 0$ 
4:    $strength \leftarrow 0$ 
5:   while true do
6:      $k \leftarrow k + 1$ 
7:     Report( $k, x$ )
8:     WaitFor( $k, *$ ) from n-f nodes
9:     if there are more than n/2 messages containing  $v$  then
10:      Propose( $k, v$ )
11:    else
12:      if  $strength \leq 0$  and  $C_p(k, 0) \neq C_p(k, 1)$  then           ▷ start to follow  $v$ 
13:         $v \leftarrow \mathbf{Plurality}(0, 1)$ 
14:        Follow( $v$ )
15:         $strength \leftarrow \mathbf{Evaluate}(0, 1)$ 
16:      end if
17:      Propose( $k, abstention$ )
18:    end if
19:    WaitFor( $k, *$ ) from n-f nodes
20:    if there are more than f messages containing  $v$  then
21:      Decide( $v$ )
22:    else if there is at least 1 message containing  $v$  then
23:       $x \leftarrow v$ 
24:    else
25:      if  $strength > 0$  and Follow( $v$ ) then           ▷ currently follow a value
26:         $x \leftarrow v$ 
27:        Decrease( $strength$ )
28:      else
29:         $x \leftarrow \mathbf{ChooseRandom}(0, 1)$ 
30:         $strength \leftarrow 0$ 
31:      end if
32:    end if
33:  end while
34: end procedure

```

- Function $\mathbf{Plurality}(0, 1)$ returns 0 if $C_p(k, 0) > C_p(k, 1)$ and returns 1 if $C_p(k, 0) < C_p(k, 1)$.
- Procedure $\mathbf{Follow}(v)$ indicates that a node prefers a value v .

- Function $Evaluate(0, 1)$ calculates a correlative value between $C_p(k, 0)$ and $C_p(k, 1)$; a proposal is $Evaluate(0, 1) = abs(C_p(k, 0) - C_p(k, 1))$.
- Procedure $Decrease(strength)$ subtracts a specified number from $strength$.

By a node, the internal variable $strength$ is added and it is decreased linearly at every round. This aims to make sure that a node does not insist on a fixed value and as a result, a deadlock can be averted. The variable $strength$ is calculated in the sense that the bigger the difference between the two cardinalities, the larger the value. The value that is being favoured by the plurality of the nodes will be more likely to be followed by others.

The value that is being reported by plurality of nodes can be expressed as a site with a better quality in the metaphor of bee swarming. A node prefers a value v so long as $strength > 0$. If $strength \leq 0$ the node chooses randomly a new value (Line 29). Randomization is also invoked in the worst case when there is no plausible argument of choosing 0 or 1, i.e. $C_p(k, 0) = C_p(k, 1)$. In the given case, deliberate choosing a fixed value, either 0 or 1, would not make sense.

With this approach, the chance that the majority of nodes report the same v in Line 7 should be higher than that in Ben-Or's algorithm. In the best case, if all nodes receive the same set of report messages in round k (Line 8), they all will follow the same value and as a result will report it in Line 7 in round $k + 1$. Afterwards, they all propose the same value in the next step. That leads to a consensus which is impossible in the original Ben-Or's algorithm.

It should be noted that the while loop from Line 5 to Line 33 does not have a stop statement. In a real implementation, it is necessary to specify a point where the loop halts. In [8] the authors propose a solution for the problem. In their approach, after a node decides on a value v it sends a message with the content $(decide, v)$ to all nodes and then stops. Every node that receives the message also decides on the value and sends the message to other nodes and halts.

We acknowledge that the changes made by the honey bee inspired approach might possibly have a side effect on the stability of the original algorithm. The issue needs to be thoroughly studied and should be considered as an open research topic. It is unfortunately beyond the scope of this paper.

5 Evaluation

5.1 A Test Program: Multicast Delivering of Messages

To evaluate the performance of the algorithm inspired by honey bees, we implement a test program in the Java programming language. The program consists of two modules, sending and receiving multicast messages. A group of connected computers equipped with this program can exchange multicast messages to negotiate a common solution.

5.2 Simulation Tools: NS2 and AgentJ

Since a more precise evaluation result can be obtained if performance tests are performed with presence of several nodes, we decided to export the test program to run on a simulation environment. Network Simulator NS2 is a discrete

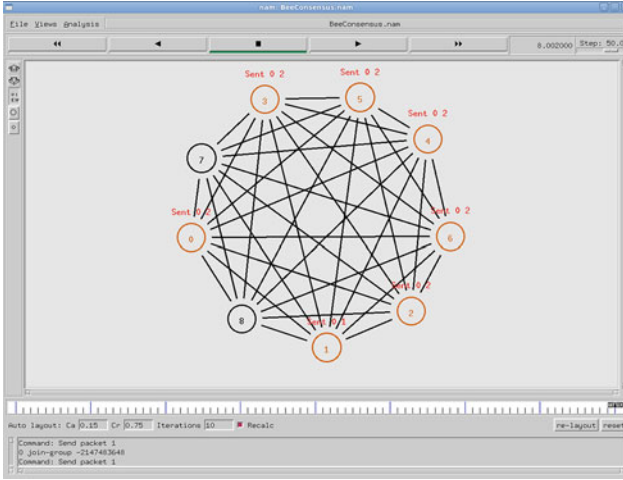


Fig. 1. NAM screenshot for NS2 simulation with 9 network nodes

event simulation framework. It was written in the programming languages C++ and Python and has been used widely for simulating applications running in wired and wireless networks. NS2 supports various routing protocols, e.g. TCP, UDP, multicast. Tcl (Tool Command Language) is used for scripting simulation scenarios in NS2 [11].

AgentJ is a software tool for embedding real applications written in Java in the NS2 simulation environment. AgentJ allows Java source code to execute on NS-2 with minor modification [10]. The use of AgentJ is practical given that there is a need for simulating applications written in Java with a large number of nodes. The combination NS2 and AgentJ provides us with a convenient way to simulate evaluation tests without needing to setup a real network.

Figure 1 depicts an example of multicast network nodes in Network Animator, a circle represents a node and a line between one pair of nodes corresponds to the link between them. In a multicast scenario, each node is connected to all other nodes. For a clear representation, in the figure we depict only a handful of nodes, however, we can increase the number of network nodes to meet our requirement.

5.3 Experimental Results

For deploying some test scenarios, we setup NS2 and AgentJ in Linux Fedora 12. Network crash is simulated by shutting down some nodes randomly during execution given that at most $f < n/2$ nodes may fail. A node fails with a probability of a randomized value ranging from 0 to 1. In the evaluation, the number of nodes is set to the following values $n = 5, 10, 15, 20, 25, 30$, $f \in \{0, \lfloor n/2 \rfloor\}$. We ran the tests with different sets of input values and with

Table 2. Performance comparison

<hr/>													
$n = 5, 10, 15$													
n	5			10				15					
f	—	0	1	2	—	0	3	4	0	4	6	7	
r_{BenOr}	—	1	1	12	—	1	10	20	1	4	23	60	
$r_{HoneyBee}$	—	1	1	2	—	1	2	2	1	2	2	2	
<hr/>													
$n = 20, 25, 30$													
n	20				25				30				
f	0	6	8	9	0	8	10	12	8	10	12	14	
r_{BenOr}	1	17	32	189	1	32	291	1319	29	112	1531	7554	
$r_{HoneyBee}$	1	2	2	2	1	2	3	2	2	4	2	2	
<hr/>													

repetition. For the implementation, Function $Evaluate(0, 1)$ in Line 15 is determined as $Evaluate(0, 1) = abs(C_p(k, 0) - C_p(k, 1))$. Once a node starts to follow a value v , it assigns the difference to the strength value $strength = Evaluate(0, 1)$. Every time it is called, Procedure $Decrease(strength)$ subtracts 1 from $strength$ until it is smaller than 0.

The outcomes of the execution are shown in Table 2. For each category, the first row represents the number of nodes taking part in building consensus n . The second row is the number of nodes that fail during execution f . The third and fourth rows are the corresponding numbers of rounds r that Ben-Or’s algorithm and the honey bee inspired version reach a consensus, respectively.

It can be seen that, the larger the number of nodes is, the more difficult a consensus can be reached. If no node fails, that means $f = 0$ then both approaches can gain a swift consensus. When failure is present, there are differences in performance. By the original Ben-Or’s algorithm, if the number of failed nodes increases, the number of rounds that it gains a consensus increases correspondingly. Especially, when f is nearly approaching $n/2$ a large number of rounds can be seen. Both tables show that the honey bee inspired algorithm has a considerably better computational performance, especially when network nodes fail en masse. Compared to Ben-Or’s algorithm, it can gain a consensus after a small number of rounds of exchanging messages. In addition, its performance is stable towards the number of failed nodes. We witness an improvement in performance when applying the honey inspired mechanism.

6 Conclusion and Future Work

In this paper, we have introduced our approach for solving consensus in distributed systems. In the algorithm we proposed some changes to the original Ben-Or’s randomized consensus algorithm based on the organizing model inspired by the honey bee colony. The experimental results showed that the approach gains an improvement in computational performance in comparison to the original Ben-Or’s algorithm.

For future work, we expect to perform further investigations on the proposed algorithm. We anticipate that the amendments might have side effects on specific circumstances that have not been perceived yet in the scope of this paper. This issue needs to be scrutinized and remains an open topic. In addition, the performance of the algorithm needs to be thoroughly analysed as well as compared with that of other existing algorithms.

References

1. Ben-Or, M.: Another advantage of free choice (extended abstract): completely asynchronous agreement protocols. In: Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing (1983)
2. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. In: Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, pp. 1–7 (1983)
3. Vavala, B., Neves, N., Moniz, H., Verissimo, P.: Randomized consensus in wireless environments: a case where more is better. In: Proceedings of the 2010 Third International Conference on Dependability, pp. 7–12 (2010)
4. Nguyen, P.T., Schau, V., Rossak, W.R.: Towards an adaptive communication model for mobile agents in highly dynamic networks based on swarming behaviour. In: EUMAS 2011 European Workshop on Multi-agent Systems (2011)
5. Seeley, T.D., Kirk Visscher, P.: Choosing a home: how the scouts in a honey bee swarm perceive the completion of their group decision making. *J. Behav. Ecol. Sociobiol.* **54**, 511–520 (2003)
6. Seeley, T.D., Kirk Visscher, P.: Group decision making in nest-site selection by honey bees. *J. Apidologie* **35**, 101–116 (2004)
7. Seeley, T.D., Buhrman, S.C.: Nest-site selection in honey bees: how well do swarms implement the best-of-N decision rule? *J. Behav. Ecol. Sociobiol.* **49**, 101–116 (2001)
8. Aguilera, M.K., Toueg, S.: The correctness proof of Ben-Or’s randomised consensus algorithm. *Distrib. Comput.* **25**, 371–381 (2012)
9. Aspnes, J.: Randomized protocols for asynchronous consensus. *J. Distrib. Comput.* **16**, 165–175 (2003)
10. Taylor, I., Downard, I., Adamson, B., Macker, J.: AgentJ: enabling java NS-2 simulations for large scale distributed multimedia applications. In: Second International Conference on Distributed Frameworks for Multimedia DFMA 2006, Penang, Malaysia, pp. 1–7 (2006)
11. The Network Simulator NS2. <http://www.cs.virginia.edu/~cs757/slidespdf/cs757-ns2-tutorial1.pdf>
12. Aspnes, J.: Fast deterministic consensus in a noisy environment. *J. Algorithms* **45**(1), 16–39 (2002)