# Patterns in a Hopfield Linear Associator as Autocorrelatory Simultaneous Byzantine Agreement

Paule Ecimovic
University of Ljubljana, Department of Philosophy, Aškerčeva 2, Ljubljana, Slovenia
Phone: +386 61 176 9200 int. 386
E-mail: ecimovic@ctklj.ctk.si

**Keywords:** PDP, neural networks, Byzantine Agreement, patterns, attractors, interactive consistency, convergence, fault tolerance, k-resilience, distributed protocols, two-phase commit, Ising spin quantum computers

**Edited by:** Rudi Murn

**Received:** April 16, 1997 **Revised:** December 8, 1997 **Accepted:** February 23, 1998

*In the first part, the Byzantine Agreement problem in parallel distributed processing is formulated for generalized, completely-interconnected networks of interacting processors. An overview of the main cases of this problem are presented in brief. Among standard optimal algorithms for reaching Simultaneous Byzantine Agreement, only the two-phase commit protocol is set out in any detail. In the second part, the process of pattern formation in Hopfield linear associators, realized as single-layer neural networks with Hebbian weight adjustment rules, is discussed. The main result of the paper is then presented, according to which pattern formation in Hopfield linear associators is a solution to a form of Simultaneous Byzantine Agreement. In conclusion, it is argued that such associative memory solutions to interactive consistency problems in generalized transaction processing systems may finally prove viable, despite decades of neglect due to inavailability or prohibitive expense of sufficient processing power for their large-scale implementation.*

## 1 Introduction

The transaction processing problem[1] of achieving and/or maintaining interactive consistency, as typified by the Byzantine Agreement (BA) scheme (Pease *et al.* 1980), (Lamport *et al.* 1982), formulated below, is exemplary among workable approaches to the design and implementation of fault-tolerant distributed protocols in parallel distributed processing (PDP) (transaction) systems. Wide-spread commercial applications, especially in banking, depend critically on guaranteeing a minimally-sufficient reliability of correct execution of distributed protocols in the presence of faults, be they faulty connections or faulty (or ill-synchronized) processors, or combinations of both (Wang 1995: p.420). In particular, achieving interactive consistency among inter-connected and distributed processors in the presence of faults is an important problem to which standard round-optimized solutions exist, such as the Dwork-Moses protocol or the two-phase commit protocol. The time might well have come, however, to (re-)consider the advantages of associative strategies typified by Hopfield's linear associator (LA) with a Hebbian association rule, most commonly realized as a fully-connected, single-layer neural network, where the connection strengths are determined by a form of Hebb's learning rule.[2] Before expanding on the close analogy between BA and associative pattern formation in the Hopfield LA, let us consider the BA scheme, its most prevalent instances, and some round-optimized protocols for reaching various instances of BA.

[1]The author is currently funded by the Slovene Science Foundation, while pursuing a master's degree in logic at the College of Philosophy, at the University of Ljubljana, where he is also engaged in research.

## 2 Byzantine Agreement (BA)

### 2.1 Fundamental Definitions

The fundamental ingredients of problem of reaching Byzantine agreement are the following. We take a processor to be any suitable Turing machine capable of carrying out some effective elementary instruction-set. Theoretically, this can be construed as a universal Tur-

[2]We have deliberately avoided establishing an "understood" transition to neural networks *per se* in order to facilitate realizations of the Hopfield LA scheme in other physical systems, which might not behave "neurally" at all, in any currently-simulated sense. We feel that this gives both neural information processing researchers an opportunity to reappraise the (in)adequacy of their neuron models from an information processing point of view as well as the neuro-biological community a breather from computational neuron-modelling strategies, which have recently been shown to miss a vast array of interaction detail. See: (Koch 1997) for hints.

ing machine. Practically, this can be anything as simple as an "intelligent" bistable switch enhanced with some automated processing logic, e.g., a McCullough and Pitts binary neuron, or something as complex as a human operator sitting at a computer terminal connected to a local area network. Some interconnected network of such processors is required across which to distribute and on which to execute a protocol thus distributed. By *distributed protocol*, we are to understand any non-contradictory set of instructions P which can be divided into n parts $P_i$ such that

$$P = \bigcup_{i=1}^{n} \{P_i\}$$

each part of which is assigned to a given processor in the course of a task designation phase of (pre-)processing of the protocol. Atop this layer, we must have a meta-layer for regulating and monitoring inter-processor communication in such a way as to achieve *interactive consistency*, which means in this context that none of the processors contradicts any action(s) of other processors on their data or on its own data or of itself on its own data. Figuratively speaking, if one were to set a team of people to sweep a room, there would be a supervisor to ensure that no member of the team throws dust on an area just swept by another member or by the member him- or her- self, thereby guarding against "stupid mistakes"(Novak,1993:p.27). We also assume a *global clock* (generator of regular events) relative to which all inter-processor communication in the PDP network is synchronized and measured. A *round of computation* is defined as the time interval (number of regular events) generated by the global clock required for all the parts $P_i$ of protocol P to be executed by the corresponding processors $p_i$ *once*. [3] Informally, a *run* is the entire state transition history of all the states in which all the processors were in the course of all rounds of computation of a given protocol P from some arbitrary starting tick of the global clock to some other arbitrary tick, i.e., in some time interval as measure on the global clock.[4] For practical reasons, we often normalize the clock ticks to coincide with rounds of computation of a given protocol, making the obvious execution uniformity assumptions. By *crash failure* is meant the state of a PDP system in which execution of a protocol from a certain round onward generates results incompatible with any admissible run of the given protocol. [5] The class of parallel-distributed processing (PDP) systems to be considered for the rest of the paper is defined by all PDP systems, which satisfy the following five conditions (Wang 1995: p.420): For natural numbers k and n, such that $n \geq k + 2$:

PDP1 There are n processors, at most k of which are faulty with respect to a given semantics, whether functional, operational, or declarative (elaborated below), without incurring crash-failure of the system in executing a given distributed protocol P

PDP2 the processors can communicate directly with each other through message exchange in a fully-connected network (FCN)

PDP3 the message sender is always identifiable by the receiver

PDP4 an arbitrary set of processors are chosen as sources and their initial value $v_s$ is broadcasted to other processors and to themselves at the start of parallely-distributed execution of P

PDP5 The only faulty components considered are processors.

PDP4 is a significant generalization from the PDP specifications presented in (Wang,1995: p.20), since it allows for more than one processor to carry the initial state which is to be distributed to and agreed upon by the entire PDP network. This is critical for our LA realization, because it admits the case of an "initial configuration", see the next section, being distributed initially across the entire network. PDP1, *mutatis mutandis*, can be taken as a general definition of what it means for some protocol P to be *k-resilient*. In other words, a protocol P is k-resilient if, and only if, for $k \leq n+2$, (at most) k processors can fail and P still executes and terminates correctly. Thus, k-resilient distributed protocols are prime examples of fault-tolerant design. The following are ways in which a processor can be *faulty*. A processor $p_i$ is termed *faulty* if it satisfies either or both of the following conditions:

- it fails to send *appropriate messages (defined either by a monitoring meta-level algorithm or at the object protocol level and just monitored "from above")*

- it fails to send otherwise appropriate messages at a time on the global clock foreseen for it in a given protocol.

A processor which is not faulty is termed *healthy*.

[3]Henceforth, processors will be denoted by lower case subscripted p's, whereas the part of a given protocol P assigned to each by upper case subscripted P's.

[4]Formally, this can be defined in terms of the modal logic $S_5$, where it is possible to define the set of all possible state transitions and quantify over them with possibility and necessity operators. (Halpern 1986)

[5]Crash failure could be defined more "severely" and absolutely as a state of a PDP system after which the system ceases to function. This definition is vague and thus prone to all sorts of error and might better be avoided.

## 2.2 The BA Scheme and its major variants

An n-processor PDP system, satisfying PDP1-5 of which at most k processors are or prove faulty, in the above sense, reach **Byzantine Agreement** if, and only if, (Wang,1995: p.20):

BA1 Every healthy processor computes the common value v which is used to determine the agreement

BA2 If the source, in the sense of PDP4, is healthy, then the common value should be the source's initial value $v_s$.

There are two major variants to the above general scheme, which we call BA. The one which we will consider is called **Simultaneous Byzantine Agreement** and is obtained from the above scheme by adding the following condition:

SBA3 All healthy processors reach agreement in the sense of BA1 & BA2 *in the same round of computation.*

We mention the other major variant in passing, namely, **Eventual Byzantine Agreement**, which is obtained by adding the following requirement to BA1,BA2:

EBA3 Each healthy processor will reach agreement with all other healthy processors, in the sense of BA1 & BA2 *in some round of computation, if the run lasts long enough with respect to the global clock.*

## 2.3 Round-optimized BA Solution Algorithms

The solution algorithms for BA mentioned or presented here impose more stringent restrictions on the number of faulty processors than we imposed in the framework above, in which solutions, let alone optimal solutions, are not guaranteed. In this respect, the above framework can be taken heuristically a means of exploring the problem space, rather than the solution space. In their original treatment, Pease, Shostak, and Lamport see (Pease *et al.* 1980) solved the so-called Byzantine Agreement problem for a fully-connected network. The formulation of the problem axiomatized as BA1 and BA2 above along with its variants are derived from (Pease *et al.* 1980), (Lamport 1982), (Halpern 1986), and (Wang 1995). For a fully-connected system of n units, BA is reachable if, and only if, $n \geq 3k+1$. This formulation leaves malicious units anonymous (unidentified) in the sense of (Novak 1993: p.22). For n=4 and k=1, two "phases" (rounds of computation) suffice to reach BA. In our notation, the situation is as follows, denoting messages sent by processor $p_i$ by $m_i$. See (Novak 1993) for a diagram and re-label accordingly. Here the specification of P and its partitioning is unimportant, since it applies to most all parallelly-distributable protocols, with some solvability and other application-specific restrictions. Processors $p_1$, $p_2$, and $p_3$ send their messages, $m_1$, $m_2$,and $m_3$ to the other three processors respectively, whereas processor $p_4$ sends $m_4$ only to $p_2$ and $p_3$, but not necessarily to $p_1$. Instead, $p_4$ sends $p_1$ some message X. Assuming that for $i \in \{1,2,3\}$, $p_i$ is fault-free and that $p_4$ is faulty in the above sense as well as being "maliciously" faulty, i.e., sending some message X, and claiming it sent the expected message, $m_4$, with the claim not being necessarily true (Novak 1993). Thus, the algorithm for reaching BA in this case proceeds in the following two phases:

TPS1 Each processor $p_i$ sends its message to all the other processors

TPS2 Verification of message correctness.

Thus BA is reached, according to the above specifications, and $p_1$ can claim to have received message X from $p_4$ as a result of the TPS2 phase, whereas $p_4$, being "malicious" can claim that it sent $m_4$. Thus we have reached "Byzantine" agreement, agreeing, within reason, to disagree, while still getting the job done. Comparison TPS1 and TPS2 with the **two-phase commit** protocol for distributed databases (McFadden 1994: p.481) reveals a similar pattern, with some modification relating to the application domain, etc. Suffice it for our purposes to interpret "site" as "processor" and "commit" as "agree on value a common v as per BA1" For explication of the broadcast variant of BA, see (Wang,1995).

TPC1 A message is broadcast to every participating site, asking whether that site is willing to commit to its portion of the transaction at that site. Each site returns an "OK" or "not OK" message.

TPC2 The originating site collects messages from all sites. If all are "OK," it broadcasts a message to all sites to commit the transaction (come to agreement). If one or more responses are "not OK," it broadcasts a message to all sites to abort the transaction.

Obviously, this is a "no non-sense" (fault intolerant) specification which could be generalized to admit one or more failures to respond "OK" in which case it would amount to a direct solution to BA. However, non-sense at the level of automated teller machines savings/checking account transactions and corresponding account balance updates is not taken lightly by the customer on the street who is likely to complain angrily at the slightest "irregularity", let alone fault. At this level, fault-tolerance and k-resilience are not to affect the end user, but rather only the processing system and its administrators.

## 3 Hopfield's Linear Associator (LA) and its Convergence Properties

In this section, we define Hopfield's linear associator (LA) in terms of discrete-time dynamical systems (effectively, finite state automata) and weighted graphs, thus avoiding this systems usual interpretation as a neural network. The reason for this will become apparent in the final two sections of this paper.

### 3.1 Specification of the Hopfield Linear Associator

The topology and architecture of Hopfield's LA (HLA) are specified as follows (Bruck 1990:p.247):

HLA1 A Hopfield LA is a discrete-time dynamical system represented by a weighted graph

HLA2 Each edge of the graph is assigned a weight and each node a threshold value.

HLA3 The weighted graph is fully connected.

The *order* of the HLA is defined to be the number of nodes in the corresponding graph. Thus, if N is an HLA of order n, written henceforth N∈HLA(n), then, according to HLA1-3, the ordered pair (W,T) determines the topology and architecture N uniquely to within a renaming, where W and T are defined as follows. For natural number n:

- W is an n x n matrix, the elements $w_{ij}$ of which represent the weight assigned to the edge joining the i-th and j-th nodes.
- T is a vector of dimension n, the components $\mathbf{t}_i$ of which denote the threshold assigned to the i-th node.

The processing element represented by any node can be in one of two possible states, either -1 or +1, and the state is denoted by $s_i(t)$, where t is a natural number to be interpreted as a certain discrete number of ticks of a clock analogous to the global clock defined above. The n-dimensional vector $s(t)$ of all elements $s_i(t)$ represents the state of N(n). The vector $s$ is called the *state vector* of the HLA N(n). Thus, the order of N is determined by the dimensionality of its state vector. Furthermore, the state vector represents the states of all individual processing elements ("processors", for short) represented by the nodes of the graph. Thus, the state vector $s(t)$ is defined by the equation

$$s(t) := (s_1, s_2, \cdots, s_n). \tag{1}$$

All the set of all possible state vectors of a suitably-chosen set of individual processor states $s_i$, where i varies from 1 to n by ones and n is some finite natural number, forms a *state space.* The evolution equation of the dynamical system, i.e., the equations governing the transition of the system from one state $s(t_0)$ to the next state $s(t_0 + 1)$ are given componentwise as follows:

$$s_i(t_0 + 1) = sgn(H_i(t_0)), \tag{2}$$

where the function $sgn(x)$ is the sign of the number $x$ defined as +1, if $x \geq 0$ and -1, otherwise, and

$$H_i(t_0) = \sum_{j=1}^{n} w_{j,i} s_j(t_0) - \mathbf{t_j}. \tag{3}$$

Accordingly, each processor at a given node is a linear threshold element, which adds its input signals from all other elements, at all other nodes in the system. Thus, each linear threshold element acts here as an "adder". To complete the specification of HLA, we must state a Hebbian weight adjustment rule and an energy function as follows (Hopfield 1982: p.2556).

$$\Delta w_{j,i} = [s_i(t)s_j(t)]_{average}, \tag{4}$$

where the average is calculated over past history, and

$$E{=}-\tfrac{1}{2}\sum\sum_{i \neq j} w_{j,i} s_i s_j \tag{5}$$

and the change in energy $\Delta E$ due to a change in state $\Delta E$ due to change in the state of an individual processor $\Delta s_i$ is given by

$$\Delta E = -\Delta s_i \sum_{j \neq i} w_{j,i} s_j. \tag{6}$$

Apart from particular initial conditions, which depend on the specific problem instance in question, this completes our specification of HLA.

### 3.2 Convergence Properties of the HLA

The feature property of the HLA for the purposes of reaching BA and, more specifcally, SBA, is that since its state space is finite, the HLA dynamical system will *always* coverge to either a stable state/cycles in state space (Bruck 1990). A specific value of the state vector $s(t)$ at a given time $t$ is called a *configuration.* Stable states (or for a given $t$, configurations) of the HLA are called *patterns* (or *pattern configurations*, respectively). Thus, *the process of n processors reaching SBA reduces to the convergence of the HLA to a pattern or pattern configuration.*

## 4 Reaching Simultaneous Byzantine Agreement with Hopfield's LA

In this section, we state the main result of this paper, which in the terminology introduced above, may be stated as **SBA can be represented by a pattern in an HLA.** This implies that the process of reaching SBA can be represented by convergence of an HLA to a stable configuration, i.e., to a pattern. Now, we must just state some facts about the stability of states and configurations in order to indicate corresponding conditions for achieving SBA as well as flush out a key property of HLA's which corresponds to k-resilience and realizes fault tolerance.

### 4.1 Stability of Configurations

A necessary and sufficient condition for a state of an HLA to be stable is the following (Bruck 1990: p.247) (in vectorial form):

$$s(t) \; is \; stable \Leftrightarrow s(t) = sgn(Ws(t) - T), \qquad (7)$$

where W is an n x n matrix of weights and T is an n-dimensional vector of node thresholds. This is critical in determining patterns, since for a given time $t$, the state $s(t)$ becomes a configuration, which, if it is stable, is a pattern.

### 4.2 k-resilience and Fault Tolerance in HLA

As is evident from the HLA specification given above, the next state at time $t_0 + 1$ of an HLA is computed componentwise from its current state at some time $t_0$ applying equation (2) to each component $s_i(t)$ of $s(t)$. This yields the state of the HLA at time $t_0 + 1$, i.e., $s(t_0 + 1)$. One of the most prominent features of the HLA model which make it well suited to the implementation of fault-tolerant distributed protocols is that $s(t_0 + 1)$ can be computed from a *proper subset*[6] of all processors in the system (Bruck 1990). This is analogous to k-resilient protocols where, by definition (Kilmer 1995), at most k processors could fail, yet the protocol continues to execute and terminates correctly. Specifically, if S is the set of indices of processors from whose states the state of $N(n)$ is computed, then

$$cardinality(S) \leq n \qquad (8)$$

and

$$cardinality(complement(S)) = k, \qquad (9)$$

where $n$ and $k$ have the same meanings as in the PDP and SBA specifications in the previous sections.[7]

## 5 Conclusion

In this paper, we have demonstrated that Simultaneous Byzantine Agreement can be represented by a Hopfield linear associator. We have done so without interpreting Hopfield's system as a neural network, but rather as a discrete dynamical system. This allows for implementations in other physical systems, which have possibly greater computational power (as measured by the classes of problems which can be solved by means of such systems) than the current computational models and systems used to implement artificial neural networks. This will be the topic of a forthcomming paper.

## Acknowledgements

The author is delighted to acknowlege the many eye-opening discussions he has had in the past few years on the subject of PDP and Byzantine Agreement. Special thanks go to Dieter Gawlick of Oracle (California), who shared with me his enthusiasm for new ideas and creative solutions to computer science problems, Bob Jackson of IBM Santa Teresa Laboratory (California) for drawing my attention to two-phase commit in this context, to my mentor of many years, Prof. Dr. Andrej Ule, for introducing me to Byzantine Agreement in the first place, to Mitja Peruš, who drove the point home about the interrelation between neural networks and quantum systems, to Dr. Joseph Halpern of IBM's Almaden Research Center (California), who, in an inspiring telephone conversation when I needed it most, made me aware of uncharted territory in the Byzantine Agreement enterprise, to Dr. Franc Novak, of IJS, for a stimulating discussion on identifying faulty processors in Byzantine environments, to Damjan Bojadžiev, of Institute Josef Stefan, in Ljubljana, who read through and corrected an earlier version of this paper, and to one of editors, Prof. Dr. Matjaž Gams, of IJS, for his advice on formatting and style as well as encouraging me to submit a paper in the first place. Most of all, I would like to thank my father, who introduced me to computers many years ago, when they were not at all commonplace household items and, over the years, gave me an osmotic course in information processing. To these and many others, the author extends warmest gratitude.

[6]If set $A$ is a *subset* of set $B$, then every element of $A$ is an element of $B$. A set C is a *proper subset* of set B if B contains at least one element which is not contained in set C.

[7]The *cardinality* of a set is the number of elements contained in the set, and the *complement* of a given set is the set of all those elements which are not contained in the given set.

## References

[1] Bruck J. (1990) On the Covergence Properties of the Hopfield Model *Proceedings IEEE*, Vol. 78, No. 10, Oct., p. 1579-1585.

[2] Chakrabarti B.K., Dutta A., & Sen P. (1996) *Quantum Ising Phases and Transitions in Transverse Ising Models*, Berlin: Springer-Verlag.

[3] Halpern J. (ed.) (1986) *Theoretical Aspects of Reasoning about Knowledge*, Proceedings of the 1986 Conference, Los Altos: Morgan Kaufmann.

[4] Kilmer W. L. (1995) Self-Repairing Processor Modules *IEEE TRANSACTIONS ON RELIABILITY*, Vol. 44, No. 2, June

[5] Koch C. (1997) Computation and the single neuron, *Nature*, V ol. 35, 16 January, p. 207-210.

[6] Lamport L., *et al* (1982) The Byzantine Generals Problem *ACM Trans. Programing Languages, Syst.*, Vol.4, No.3, pp. 382-401, July.

[7] McFadden F.R. & Hoffer, J.A. (1994) *Modern Database Management*, 4th ed., Redwood City: The Benjami/Cummings Publishing Company, Inc.

[8] Novak F. & Klavžar S. (1993) An Algorithm for Identification of Maliciously Faulty Units, *International Journal of Computer Mathematics*, Vol. 48, p.21-29.

[9] Pease M., Shostak R., and Lamport L. (1980) Reaching agreement in the presence of faults JACM, Vol.27, No.2, pp.228-234, April.

[10] Wang S.C., Chin Y.H., & Yan K.Q. (1995) *IEEE Transactions on Parallel and Distributed Systems*, Vol.6, No.4, April, p.420-427.