The 9th International Conference on Internet and Distributed Computing Systems (IDCS 2016), Wuhan, China, Sept. 28-30, 2016. (in press)

Agreement in Epidemic Information Dissemination

Mosab Ayiad, Amogh Katti and Giuseppe Di Fatta
Department of Computer Science, University of Reading
Reading, Berkshire, RG6 6AH, United Kingdom
Email: {m.m.ayiad@pgr., a.p.katti@pgr., g.difatta@}reading.ac.uk

Abstract

Consensus is one of the fundamental problems in multi-agent systems and distributed computing, in which agents or processing nodes are required to reach global agreement on some data value, decision, action, or synchronisation. In the absence of centralised coordination, achieving global consensus is challenging especially in dynamic and large-scale distributed systems with faulty processes. This paper presents a fully decentralised phase transition protocol to achieve global consensus on the convergence of an underlying information dissemination process. The proposed approach is based on Epidemic protocols, which are a randomised communication and computation paradigm and provide excellent scalability and fault-tolerant properties. The experimental analysis is based on simulations of a large-scale information dissemination process and the results show that global agreement can be achieved without deterministic and global communication patterns, such as those based on centralised coordination.

Index Terms

distributed consensus, epidemic protocols, gossip-based protocols, large-scale distributed computing, decentralised algorithms

I. Introduction

In distributed computing and multi-agent systems, nodes (processes/agents) are often required to agree on some value or some action. Achieving agreement in large-scale and dynamic distributed systems is a challenging task. Such challenge forms one of the fundamental problems in distributed computing, the so-called *Consensus Problem* [1]. A solution to the consensus problem is often a critical component in many distributed applications, e.g. transactions in distributed databases, leader election, consent on replicas, synchronisation, load balancing.

In a typical formulation of the consensus problem, each participant holds a value and exchanges it with other participants. All participants then decide (*agree*) on a common output which must be one of the held values [2]. The challenge is to achieve and detect agreement among all participants (*Global Consensus*) from only locally available information at each participant when a centralised coordinator is not available.

Conceptually, the consensus problem involves the following properties [1]. All non-faulty nodes should eventually decide on some value (*Termination*). The selected value is the same for all non-faulty nodes (*Agreement*). The final decision should be valid, i.e. within the set of proposed and exchanged values (*Validity*). These properties rely on the safety and liveness of the distributed system, where safety implies that the nodes never propose incorrect values and liveness implies that all nodes perform exactly as intended [1].

A basic Epidemic (a.k.a. *Gossip-based*) system consists of a large set of nodes that adopt a randomised communication strategy to implement network services and applications. Epidemic protocols are typically formulated as periodic processes with a fixed cycle length. At each cycle, each node sends its local state to a random peer. During each cycle, a node receives the local state of some other peers and updates the local state. Random pairwise communication provides stochastic guarantees that the nodes in the system ultimately converge to a common state [3]. Two fundamental global operations that can be implemented by means of Epidemic processes are information dissemination (broadcast) and data aggregation. In data aggregation a global synopsis function (*average*, *count*, *sum*, etc.) is computed in parallel and consistently at every node.

A typical Epidemic algorithm is formulated by combining the appropriate aggregation function with a particular communication model, e.g. Push-Sum, Push-Pull Averaging, Symmetric Push-Sum [4, 5]. Aggregation algorithms may estimate a global value across the neighbours, a group of nodes, or an entire network (*Uniform Gossiping*) [3]. To achieve Uniform Gossiping, an Epidemic membership protocol is adopted to provide a peer sampling service [6].

Interestingly, Epidemic approaches for consensus obtain several advantages over approaches based on deterministic and centralised communication, as they inherited the fault tolerance, scalability, decentralisation and lightweight properties of Epidemic protocols [7]. Moreover, it is found that Epidemic algorithms can support asynchronous applications better, as they exhibit loose coupling and converge in lesser time with acceptable cost [4].

In this paper, we propose a novel Epidemic approach to the consensus problem for information dissemination. For the sake of simplicity and to emphasise the key features of the proposed solution, we adopt a simple Information Dissemination Application (*IDA*) to simulate the underlying information propagation process on which global consensus is required. In *IDA*, each node generates and propagates information in the network. Information items are uniquely identified, though the same item identifier may be generated at several nodes. *IDA* must provide a mechanism to remove duplicates, to ensure propagation of the information to all nodes and to establish a system-wide consensus for each information item.

The rest of the paper is organised as follows. Section II defines the model of the distributed system that has been adopted in this work. Section III describes *IDA*, the information dissemination application. The *IDA* protocols are presented in Section IV. The experimental results of the simulations are described and discussed in Section V. Section VI discusses some related work. Finally, conclusions and future work are drawn in Section VII.

II. Model of the Distributed System

The *impossibility result* [8] (a.k.a. the *FLP result*) refers to the impossibility of detecting consensus in distributed systems of asynchronous and unreliable processes. Achieving consensus is not possible in an asynchronous distributed system with no prior bounds δ and ϕ on, respectively, the communication delay and the relative process speed [2]. The *FLP result* has motivated the identification of the minimal properties of distributed systems that are necessary to solve the consensus problem [9]. Following the work in [2], the proposed solution in this paper is provided for a partial synchrony setting so that upper bounds δ and ϕ are defined but unknown to the nodes. The message exchange is subject to random delays within δ and all processes run at a bounded relative speed ϕ and perform at least one operation in each cycle. The simulations adopt discrete events scheduled with random offsets within fixed time intervals (*cycles*). Within each process, cycles are consecutive and do not overlap. A uniformly distributed synchronisation offset is used for starting each process [5]. Processes perform separate send and receive operations: no complex atomic communication operations are required. As a consequence, interleaved message exchanges exist and message order is neglected. No process lock or exclusive access is present.

The next section introduces the information dissemination application for which global consensus has to be provided.

III. The Information Dissemination Application (IDA)

IDA is adopted to simulate the distributed generation of information items for which propagation and global consensus is required. In this section, we present the conceptual design and practical scenario of *IDA* for the generation and propagation of the information items. In Section IV the proposed protocols for propagation and consensus are described in details.

Information items in *IDA* are assigned unique (sequential) identifiers (ID). At each cycle, a node generates a new item with a given probability. A new item is given the next locally unique ID with no global or centralised coordination. This way, items with a specific ID can be generated simultaneously at different nodes and ID duplication must be resolved.

At each node, an information item is associated with one of three possible states: *Propagation*, *Agreement* and *Commit*. The state diagram is shown in Figure 1. The same item can be associated with different states at different nodes. The ultimate goal is for each item to reach the final state (*Commit*) at all nodes, which corresponds to global consensus on that item.

Initially, each node is started with an empty information cache C. At each node, an information item is created at each cycle with a given probability. The node at which the item is created is called the *originator*. The item is represented by a tuple, which includes the item ID, the originator ID and the item state. The tuple of a newly generated item is added to the local cache C with initial state *Propagation*. Each tuple also contains some numerical variables that are used by the consensus protocol and are described in Section IV. Each node periodically disseminates the items that are present in its local cache C in the system by sending it to a randomly chosen peer. When a node receives a message with a remote information cache, it updates its local cache by merging the local and remote entries, aggregating identical items and resolving ID duplicates.

Nodes have no explicit or prior knowledge of the system size n. Thereby, each node runs a specific protocol to estimate the current system size. This protocol is detailed in Section IV-A.

The proposed consensus protocol is a concatenation of two Epidemic aggregation operations, which are used to estimate the number of nodes in the system which have a particular item in a specific state, respectively, at *Propagation* and *Agreement*. When this estimated count corresponds to the system size within some tolerance, the state of the local copy of the item is updated to the next state (*phase transition*). This protocol is detailed in Section IV-B. The action taken at the transition to the *Commit* state depends on the specific application and is out of the scope of this work.

IDA is a simplified and sufficiently general model which may find applications in diverse domains, such as failure detection and consensus, transactions in distributed databases, consent on replicas, etc.

The next section describes the protocols employed for the estimation of the system size and for the two phase transitions.

IV. IDA Protocols

In *IDA*, a connected physical topology, routing and transport protocols with no packet loss are assumed. *IDA* employs three protocols: the simple Node Cache Protocol (*NCP*) [5] for membership management; the System Size Estimation Protocol (*SSEP*) for the estimation of the current system size n and the Phase Transition Protocol (*PTP*) for determining the state transition of the local copies of information items.

The protocol *NCP* implements and exports the function $getRandomPeer()$, a peer sampling service with uniform random probability (Uniform Gossiping). In the simulations, we have adopted *NCP* with a random k-regular overlay topology initialisation. Any other Epidemic Membership Protocols could also be employed and for further details on *NCP* we refer the reader to [5]. *IDA* nodes have no knowledge of the system size n. Therefore, each node adopts an Epidemic aggregation protocol, the

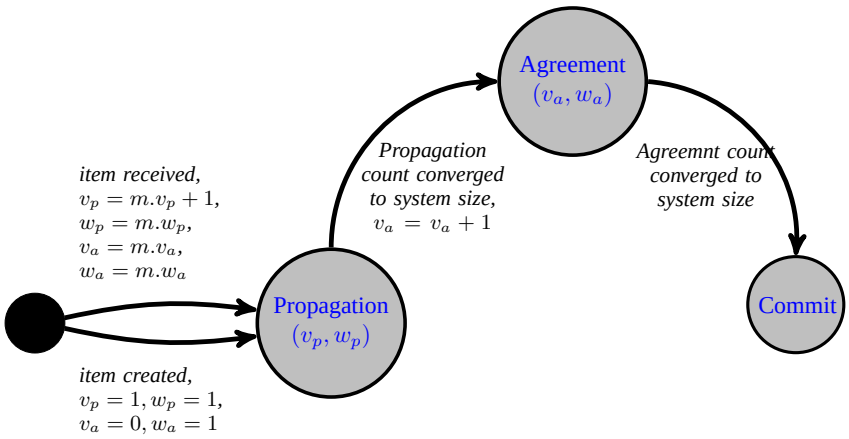


Fig. 1. *IDA* state diagram for an information item

System Size Estimation Protocol (*SSEP*), to estimate the current system size n. The Phase Transition Protocol (*PTP*) provides an Epidemic solution to the distributed consensus problem following a three-phase commit protocol approach. The protocols *SSEP* and *PTP* are described in the next two sections.

A. The System Size Estimation Protocol (SSEP)

The protocol *SSEP* implements the Symmetric Push-Sum Protocol for data aggregation [5]. Precisely, it estimates the global function '*count*'. In Algorithm 1, the pair (v, w) is used, where v is the aggregation value and w the aggregation weight. Initially, all node values are set to 1, and weights are set to 0, except for one node that has $w = 1$. At each cycle, a node i halves its pair values (v, w) and sends the pair to a random node (PUSH). When a node receives a PUSH message from a remote node j, it halves its local value and weight and sends them to j (PULL). Finally, it combines the local and remote pairs and updates the estimated system size.

SSEP propagates aggregation pairs in the system. The global sum of w will be evenly distributed to all nodes converging to $\frac{1}{n}$, where n is the system size. Similarly, v is aggregated and distributed, and will converge to 1. At each cycle, the local estimation of n can be calculated by $\frac{v}{w}$. Although interleaved messages are present in the system, as long as the mass invariant holds in the system, $\frac{v}{w}$ will quickly converge to n with a relative error as small as desired [3, 5]. The estimated system size will hold as long as the system size remains static [5, 7]. However, since the protocol is continuously executed, it can also adapt to changes in the system size. Nevertheless, the simulation of dynamic network conditions (e.g., node churn) is out of the scope of this work.

B. The Phase Transition Protocol (PTP)

Each *PTP* instance maintains a local cache C of information items. The local cache is initially empty and will be used to store items either created locally or received from other nodes. Each item in C is represented by a tuple, as described in section III. The tuple contains two aggregation pairs (v, w). One aggregation pair, (v_p, w_p), is used to estimate the number of nodes which have received the item, i.e. the $(p)ropagation$ count. The second pair, (v_a, w_a), is used to estimate the $(a)greement$ count, that is the number of nodes holding the item at the second phase. The protocol is a cascade of two Epidemic aggregations based on the Symmetric Push-Sum Protocol for the global function '*sum*' [5].

As shown in Algorithm 2, at each cycle, node i halves the two aggregation pairs of each tuple in C. Then, it sends a copy of the local C to a random peer. The protocol *PTP* checks if an agreement is reached for the transition from a phase to the successive. On the event of receiving a message from a peer, the algorithm adds new items, updates the tuple of the items already stored in C and resolves duplicates by keeping either the oldest tuple or keeping the one with the lowest originator ID if the tuples have the same creation time.

At the creation of a new information item at node i, *PTP* selects the next unique identifier (line 4) and inserts a new tuple into C (line 5). *PTP* obtains the estimated system size n from *SSEP* protocol using the function $size()$. For each τ in C, the criterion in line 14 decides upon the transition from *Propagation* to *Agreement*. The aggregation count is compared to the estimated system size with a relative error tolerance (ϵ). The test requires a minimum number of consecutive cycles (MIN) within tolerance to ensure robust transition to the next phase, avoiding early false transitions. The transition to *Commit* is associated with a similar test condition on the *Agreement* count. The transition to *Commit* is shown in lines 18-19. When receiving a message with a remote cache, *PTP* merges local and remote items in lines 25-32. Tuples related to the same

Algorithm 1: System Size Estimation Protocol (SSEP)

1 **Initialisation:**
2 $v = 1.0$ *at all nodes,* $w = 1.0$ *at one node and* $w = 0.0$ *at all other nodes*
3 *At each node* i*:* $est = 0.0$ `// size estimation`

---

4 **At each cycle at node** i**:**
5 $j \leftarrow getRandomPeer()$
6 $v = \frac{v}{2.0}$; $w = \frac{w}{2.0}$
7 $send(j, v, w, reply = true)$ `// PUSH message`

---

8 **At event 'receive message** m **from** j**' at node** i**:**
9 **if** $m.reply$ **then**
10 $v = \frac{v}{2.0}$; $w = \frac{w}{2.0}$
11 $send(j, v, w, reply = false)$ `// PULL message`
12 $v = v + m.v$; $w = w + m.w$
13 **if** $w > 0.0$ **then** $est = \frac{v}{w}$

---

14 **Method** $size()$ *:* $\mathbb{R}$
15 **return** est

information item are aggregated in line 28, duplicate IDs are resolved in line 30 and new items are added to the local cache C in line 32.

V. Simulations and Experimental Results

Simulations are carried out using PeerSim [10], a Java-based discrete-event P2P simulation tool. PeerSim is flexible, scalable and easy to configure. *SSEP*, *PTP* and *NCP* protocols are implemented in dedicated modules for PeerSim.

The simulation common settings are as follows. Different random seeds are used in each simulation run to validate performance and enforce randomisation. The system default size is $n = 10000$ nodes and the maximum experiment length is 100 cycles. Membership is managed by *NCP*, which maintains the overlay topology with $k = 10$. The generation of new items in *PTP* is interrupted after the completion of 50% of the simulation cycles to observe the protocol performance in the residual cycles.

Simulation cycles are time intervals of fixed length, which adopt the cycle structure used in [5]. For experimental purposes, we define Δt, a cycle length that is long enough for all nodes to finish send, receive and aggregate operations, such that $\Delta t = t_1 + t_2 + t_2 + t_3$, where t_1 limits PUSH offsets, t_2 limits transmission delays, and t_3 limits initial synchronisation offsets. In Δt, the portion $t_2 + t_2$ is the maximum communication latency corresponding to the round trip time on the diameter of the network. However, some messages may take very long to arrive ($\delta > \Delta t$) and arrive in later cycles (*Out of Cycle Message*). Out of cycle messages slightly delay the convergence in *SSEP* and *PTP* protocols due to potential loss of aggregation mass. Nevertheless, the aggregation mass is restored when out of cycle messages reach destination.

All protocols are based on the event-driven engine of PeerSim, where three common events are defined as follows.

1) The Activate Event occurs at every cycle. At the beginning of the simulation, the event is scheduled by a specific initialiser to occur after a random offset within t_3. The event is then scheduled to occur at every Δt. The item generation procedure and the phase transition tests are executed at this event. The cyclic event stops when a maximum number of cycles is reached.
2) The Push Event is scheduled at a random time $t < t_1$ from the Activate Event. At this event, a node sends a PUSH message to a random peer.
3) The Message Receive Event occurs when a node receives a message from a peer. At this event the incoming message is processed.

The protocol *SSEP* is tested with different system size values (n) and the convergence of the estimated system size is monitored. Figure 2.a shows the percentage of nodes which have locally estimated n within an error tolerance (ϵ); while Figure 2.b shows the average of the estimated size for all nodes. Figure 2 confirms that 100% of nodes correctly estimate the system size n after a sufficient number of cycles.

The protocol *PTP* is tested with the generation of a single information item and of multiple items. The diffusion of a single item is shown in Figure 3, where the percentage of nodes that have achieved a particular phase is illustrated. Additional

Algorithm 2: Phase Transition Protocol (PTP)

Require: $size()$, *the* SSEP *estimation;* ϵ*, an error tolerance;* MIN*, a minimum number of consecutive cycles; a local cache of items* $C = \{\tau = \langle id, o, t, v_p, w_p, v_a, w_a, state\rangle, ...\}$*, where* id *is the item identifier,* o *the originator identifier,* t *the creation time,* (v_p, w_p) *the propagation pair,* (v_a, w_a) *the agreement pair and* $state$ *the item state.*

Initialisation: *at each node* i*:* $C \longleftarrow \{\}$

At event 'new $item$ **generated' at node** i**:**

$id \longleftarrow$ next locally unique identifier

$C \longleftarrow C \cup \{\langle id, i, current\_cycle, 1, 1, 0, 1, PROPAGATION\rangle\}$

At each cycle at node i**:**

$j = getRandomPeer()$

foreach $\tau \in C$ **do**

  $\tau = \langle \tau.id, \tau.o, \tau.t, \frac{\tau.v_p}{2}, \frac{\tau.w_p}{2}, \frac{\tau.v_a}{2}, \frac{\tau.w_a}{2}, \tau.state\rangle$

$send(j, C, reply = true)$ `// PUSH message`

foreach $\tau \in C$ **do**

  switch $\tau.state$ **do**

    case $PROPAGATION$

      if $size() > 0.0$ **and** $\left|\frac{size() - \frac{\tau.v_p}{\tau.w_p}}{size()}\right| \leq \epsilon$ *for at least* MIN *cycles* **then**

        $\tau.state = AGREEMENT$

        $\tau.v_a = \tau.v_a + 1$

    case $AGREEMENT$

      if $size() > 0.0$ **and** $\left|\frac{size() - \frac{\tau.v_a}{\tau.w_a}}{size()}\right| \leq \epsilon$ *for at least* MIN *cycles* **then**

        $\tau.state = COMMIT$

        `// and may take some application-specific action.`

At event 'received m **message from** j**' at node** i**:**

if $m.reply$ **then**

  foreach $\tau \in C$ **do**

    $\tau = \langle \tau.id, \tau.o, \tau.t, \frac{\tau.v_p}{2}, \frac{\tau.w_p}{2}, \frac{\tau.v_a}{2}, \frac{\tau.w_a}{2}, \tau.state\rangle$

  $send(j, C, reply = false)$ `// PULL message`

foreach $\tau_0 \in m.C$ **do**

  if C **contains** τ_1 **where** $\tau_0.id == \tau_1.id$ **then**

    `// Resolve duplicate item ID`

    if $(\tau_0.t == \tau_1.t$ **and** $\tau_0.o == \tau_1.o)$ **then**

      $\tau_1 = \langle \tau_1.id, \tau_1.o, \tau_1.t,$

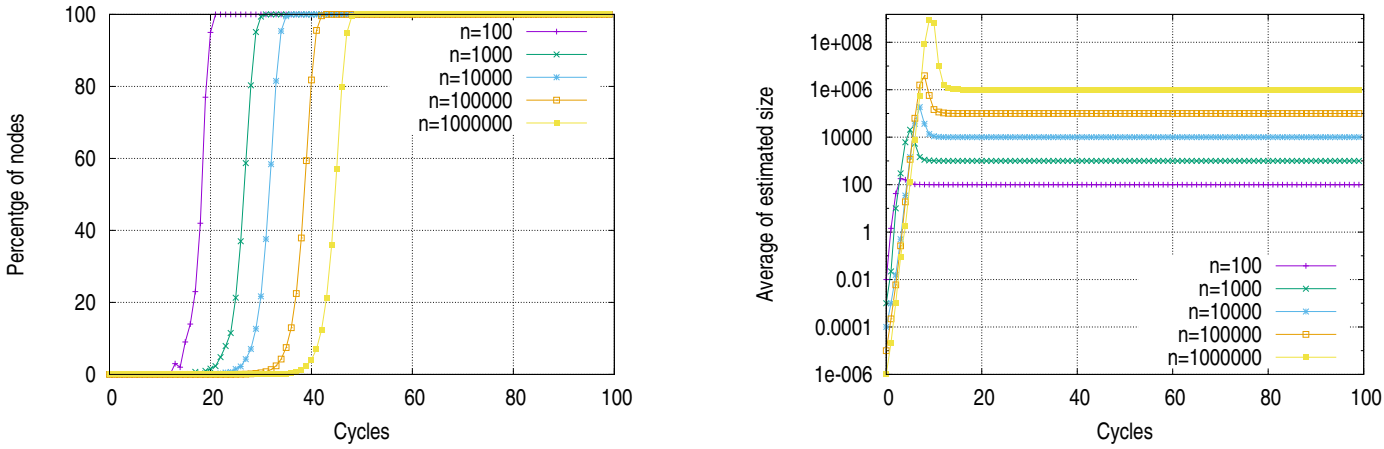
      $\tau_1.v_p + \tau_0.v_p, \tau_1.w_p + \tau_0.w_p, \tau_1.v_a + \tau_0.v_a, \tau_1.w_a + \tau_0.w_a, \tau_1.state\rangle$

    else if $(\tau_0.t == \tau_1.t$ **and** $\tau_0.o < \tau_1.o)$ **or** $(\tau_0.t < \tau_1.t)$ **then**

      $\tau_1 = \langle \tau_0.id, \tau_0.o, \tau_0.t, \tau_0.v_p + 1, \tau_0.w_p, \tau_0.v_a, \tau_0.w_a, \tau_0.state\rangle$

  else

    $C \longleftarrow C \cup \{\langle \tau_0.id, \tau_0.o, \tau_0.t, \tau_0.v_p + 1, \tau_0.w_p, \tau_0.v_a, \tau_0.w_a, \tau_0.state\rangle\}$



(a) Percentage of nodes converged to n

(b) Average of size estimates

Fig. 2. System size estimation in *SSEP* converges to actual system size n ($\epsilon = 0.01$)

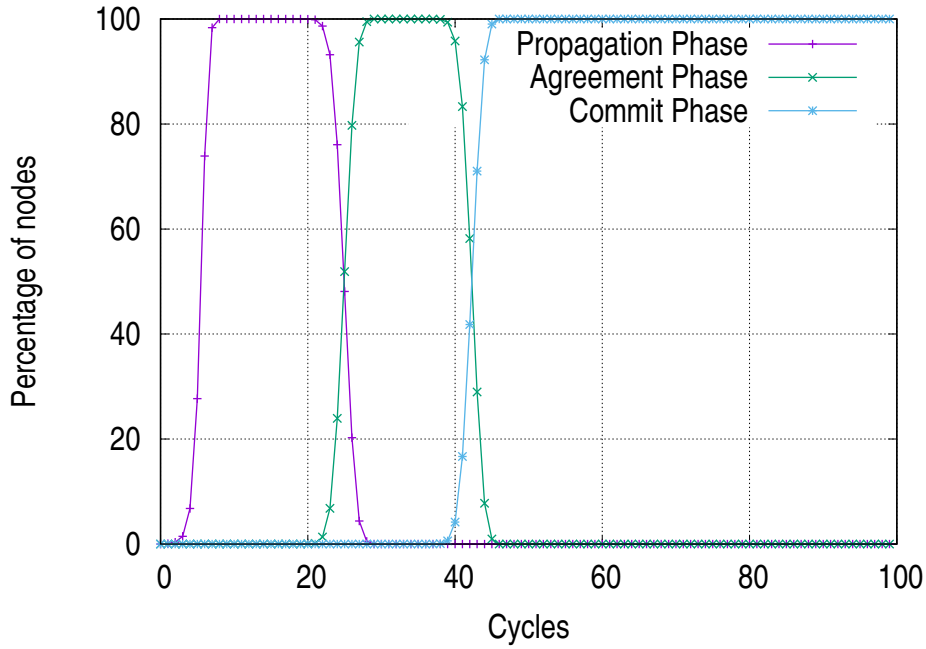
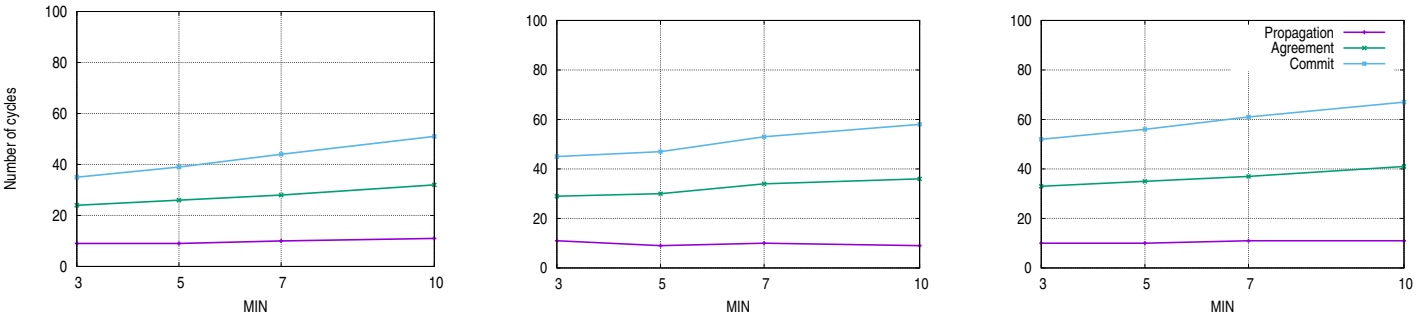


Fig. 3. Percentage of nodes at each phase for a single information item in *PTP* ($n = 10000$, $\epsilon = 0.001$, $MIN = 5$)

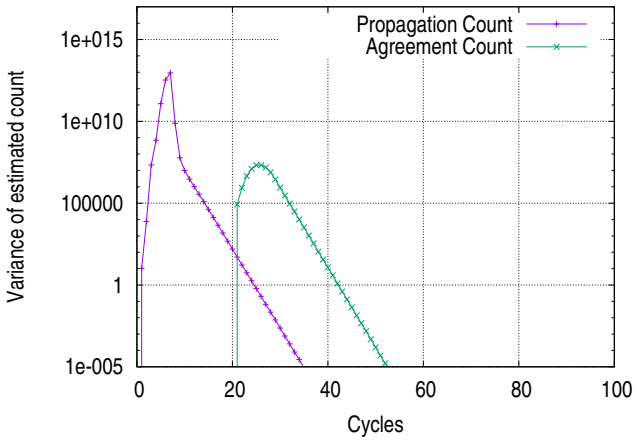


(a) $\epsilon = 0.01$

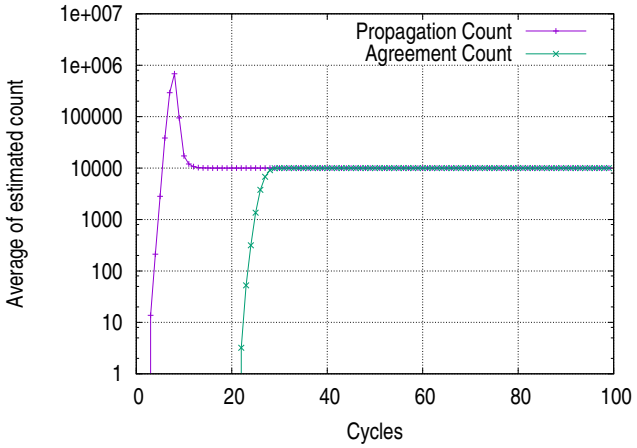
(b) $\epsilon = 0.001$

(c) $\epsilon = 0.0001$

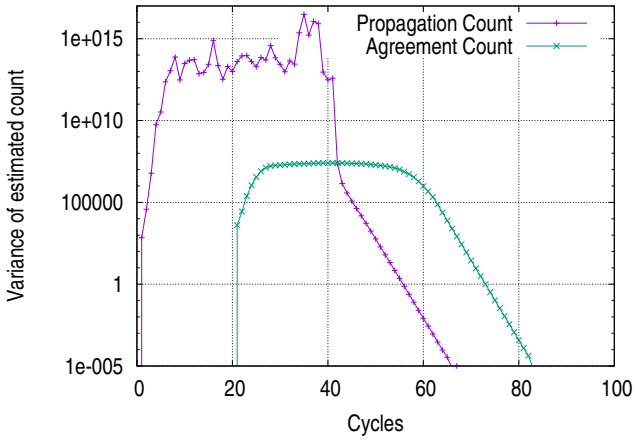
Fig. 4. Number of cycles to complete a phase transition in *PTP* for a single item varying the minimum number of consecutive cycles MIN and for three values of the error tolerance ϵ ($n = 10000$)



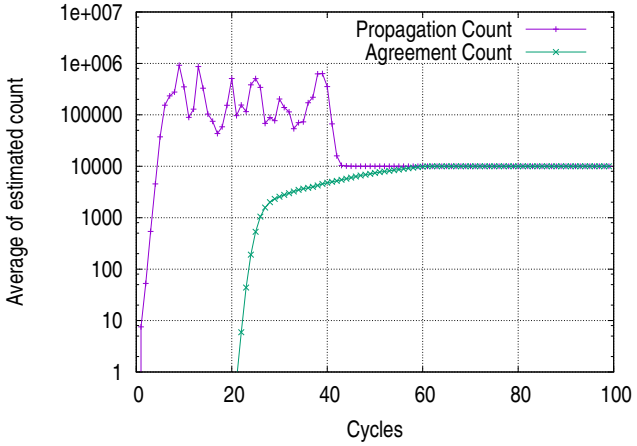
(a) Variance of estimates for a single item



(b) Average of estimates for a single item



(c) Variance of estimates for multiple items (50 distinct items)



(d) Average of estimates for multiple items (50 distinct items)

Fig. 5. Convergence of the *Propagation* and *Agreement* count estimates in *PTP* ($n = 10000$, $\epsilon = 0.001$, $MIN = 5$)

experiments on single item diffusion are conducted with several values of tolerance ϵ and minimum number of consecutive cycles MIN. Figure 4 summarises the results and shows the number of cycles to complete a phase transition for an item when varying MIN and for a few values of ϵ. The number of cycles required to complete a phase transition linearly increases with both parameters.

The convergence of the *Propagation* count and of the *Agreement* count in *PTP* is demonstrated in Figure 5. For a single information item, Figure 5.a shows the variance of the estimates over all nodes; while Figure 5.b shows the average of the estimates in the system.

PTP is also tested for the propagation of 50 distinct items in the presence of of item ID duplication. Figure 5.c shows the variance of the estimates over all items and all nodes; while Figure 5.d shows the average of the estimates. It can be inferred that the protocol correctly manages item ID duplicates and that count estimates in the nodes correctly converge to the system size n.

The proposed Epidemic three-phase approach provides a solution to the consensus problem without any centralised coordination. Global agreement and synchronisation can be achieved without global deterministic communication patterns. Moreover, Epidemic protocols have excellent scalability and fault-tolerance, which are important properties for large-scale distributed systems.

VI. Related work

Solutions for the consensus problem based on Epidemic approaches have been applied to various applications in distributed systems. The work in [11] proposed general consensus and average consensus algorithms for quantisation of states in randomised networks using the asynchronous Epidemic-based communications of agents. A survey on the consensus problem in multi-agent cooperative control is provided in [12]. The work in [13] uses Epidemic-based ping and time-out mechanism to detect and

propagate failures. Correctly operating processes reach consensus when all of them detect the failed ones. In [14], an Epidemic-based aggregation protocol is used to perform a global synchronisation and reduction operation for a fully decentralised K-Means clustering without global communication patterns under node churn and message loss. The work in [15] investigated heuristic methods to detect the convergence of Epidemic aggregation and those methods could be used to build a consensus protocol for data aggregation.

VII. Conclusions

This paper presented a phase transition protocol (*PTP*) to achieve global consensus on the convergence of information dissemination. The proposed solution assumes the minimal properties needed to solve the consensus problem in partially synchronous distributed systems. The solution is based on Epidemic protocols and so it inherits their advantages. A simple application scenario (*IDA*) of global information dissemination is used to demonstrate the key idea of Epidemic consensus in large-scale distributed systems and could be extended in more complex scenarios, such as data aggregation. In *IDA*, two main algorithms are introduced, the protocol *SSEP* to estimate the system size and the protocol *PTP* to manage phase transitions and achieve global agreement on each information item.

The experimental analysis based on simulations has shown that *PTP* is able to achieve global consensus on the propagation of information items over a large number of nodes with no centralised coordination, no prior knowledge of system size, and no assumption of the a priori global uniqueness of the item identifiers. However, the solution currently assumes that system size holds during the consensus transition period and that no message loss or churn are present. Future research may address these limitations, introduce optimisations of the performance and communication overhead. A further direction of future investigations is the extension of Epidemic consensus to the data aggregation problem and to dynamic network conditions.

VIII. Acknowledgements

The authors Mosab Ayiad and Amogh Katti are supported for their PhD projects, respectively, by the Merit Scholarship Program, Islamic Development Bank, and by the Felix Scholarship.

References

[1] R. Guerraoui, M. Hurfinn, A. Mostefaoui, R. Oliveira, M. Raynal, and A. Schiper. “Consensus in Asynchronous Distributed Systems: A Concise Guided Tour”. In: *Advances in Distributed Systems: From Algorithms to Systems*. Springer Berlin Heidelberg, 2000.

[2] C. Dwork, N. Lynch, and L. Stockmeyer. “Consensus in the Presence of Partial Synchrony”. In: *Journal of the ACM* 35.2 (Apr. 1988).

[3] D. Kempe, A. Dobra, and J. Gehrke. “Gossip-based computation of aggregate information”. In: *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*. 2003.

[4] I. Rao, A. Harwood, and S. Karunasekera. “Impacts of Asynchrony on Epidemic-Style Aggregation Protocols”. In: *Parallel and Distributed Systems (ICPADS), IEEE 16th International Conference on*. 2010.

[5] F. Blasa, S. Cafiero, G. Fortino, and G. Di Fatta. “Symmetric Push-Sum Protocol for decentralised aggregation”. In: *Proceedings of AP2PS, the Third International Conference on Advances in P2P Systems*. 2011.

[6] P. Poonpakdee and G. Di Fatta. “Expander Graph Quality Optimisation in Randomised Communication”. In: *Data Mining Workshop (ICDMW), IEEE International Conference on*. Dec. 2014.

[7] M. Jelasity, A. Montresor, and O. Babaoglu. “Gossip-based Aggregation in Large Dynamic Networks”. In: *ACM Transactions on Computer Systems* 23.3 (Aug. 2005).

[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson. “Impossibility of Distributed Consensus with One Faulty Process”. In: *Journal of the ACM* (1985).

[9] D. Dolev, C. Dwork, and L. Stockmeyer. “On the Minimal Synchronism Needed for Distributed Consensus”. In: *Journal of the ACM* (1987).

[10] A. Montresor and M. Jelasity. “PeerSim: A scalable P2P simulator”. In: *Peer-to-Peer Computing. P2P’09. IEEE Ninth International Conference on*. IEEE. 2009.

[11] K. Cai and H. Ishii. “Gossip consensus and averaging algorithms with quantization”. In: *American Control Conference (ACC)*, 2010.

[12] W. Ren, R. W. Beard, and E. M. Atkins. “A survey of consensus problems in multi-agent coordination”. In: *Proceedings of the 2005, American Control Conference*. June 2005.

[13] A. Katti, G. Di Fatta, T. Naughton, and C. Engelmann. “Scalable and Fault Tolerant Failure Detection and Consensus”. In: *Proceedings of the 22Nd European MPI Users’ Group Meeting*. EuroMPI ’15. Bordeaux, France: ACM, 2015.

[14] G. Di Fatta, F. Blasa, S. Cafiero, and G. Fortino. “Fault tolerant decentralised K-Means clustering for asynchronous large-scale networks”. In: *Journal of Parallel and Distributed Computing* 73.3 (2013).

[15] P. Poonpakdee, N. G. Orhon, and G. Di Fatta. “Convergence Detection in Epidemic Aggregation”. In: *Euro-Par 2013: Parallel Processing Workshops*. Lecture Notes in Computer Science. Springer, 2014.