An adaptive restart mechanism for continuous epidemic systems

An Adaptive Restart Mechanism For Continuous Epidemic Systems

Mosab M. Ayiad Giuseppe Di Fatta

Department of Computer Science, University of Reading
Whiteknights, Reading, Berkshire, RG6 6AY, UK
email: {mosab.ayiad, g.difatta}@reading.ac.uk

Abstract

Software services based on large-scale distributed systems demand continuous and decentralised solutions for achieving system consistency and providing operational monitoring. Epidemic data aggregation algorithms provide decentralised, scalable and fault-tolerant solutions that can be used for system-wide tasks such as global state determination, monitoring and consensus. Existing continuous epidemic algorithms either periodically restart at fixed epochs or apply changes in the system state instantly producing less accurate approximation. This work introduces an innovative mechanism without fixed epochs that monitors the system state and restarts upon the detection of the system convergence or divergence. The mechanism makes correct aggregation with an approximation error as small as desired. The proposed solution is validated and analysed by means of simulations under static and dynamic network conditions.

Keywords Distributed Computing, Continuous Systems, Decentralised Aggregation, Epidemic Protocols, Node Churn

1 Introduction

Network services in large-scale distributed systems often require decentralised solutions for monitoring system-wide state and maintaining its consistency. For example, an online service where participants join and leave independently of the service may need to track the number of active participants to successfully complete a specific task [1]. Also, distributed applications for unmanned vehicles may repeatedly attempt to coordinate a universal speed limit to optimise system performance or to avoid catastrophic scenarios [2]. In Wireless Sensor Networks (WSN), devices frequently collect data from their sensors and require to have a view of the network with fresh aggregated information [3]. Recent trends in Edge computing have considered moving Cloud services away from centralised computation and data centres towards the edges of the network: Edge computing can benefit from decentralised monitoring and processing capabilities [4]. Distributed consensus protocols enable decentralisation in Blockchain [5] and ensure that all participants collectively maintain a common transaction ledger without a central authority: these protocols also require a robust and continuous adaptation to changes in the system conditions.

Epidemic protocols are based on a Peer-to-Peer (P2P) paradigm for decentralised communication and computation in large and extreme-scale distributed systems. They adopt randomised communication models inspired by biological systems, which have natural diffusion properties [6]. In general, applications may incorporate one or more epidemic algorithms to implement a desired service and forming a single *epidemic system* for a particular task. Epidemic systems are typically adopted for two main tasks: (1) *information dissemination* and (2) *data aggregation*. In a dissemination task, information that is in the interest of other participants is propagated in the system, e.g., information of updates and failures [7, 8]. A data aggregation task uses the epidemic diffusion process to compute a global synopsis of a distributed set of data values. Data aggregation tasks are fundamental for a wide range of services, especially in computing global system properties, e.g., for estimating system size and for resource monitoring [1, 3]. Moreover, they can be used as components of more complex applications, such as failure detection [8], distributed data mining [9], system consistency and global consensus [7].

Theoretical and practical analyses have shown stochastic guarantees on the convergence to the desired target of stable epidemic systems [6, 7]. Achieving and detecting convergence is critical for many services: for example, it is a prerequisite for actions such as *termination* and *restart* [1, 10]. Moreover, real-world systems are generally asynchronous and highly dynamic. Nodes can have arbitrary start times and non-simultaneous processing cycles [7]. Also, nodes usually have access to different clocks, encounter variable communication delays and may lose messages or message order. In dynamic systems, nodes can join and leave arbitrarily (a.k.a '*Node Churn*') [10], and may suddenly fail or become unreachable [1].

Node churn and unreliable networks have a detrimental effect on the efficiency and the robustness of epidemic systems [10, 11]. In extreme case, the inherent properties of the epidemic systems cannot be guaranteed: tasks

may not converge or may converge to incorrect results. Therefore, it is essential to maintain a robust and consistent state at any time for a system to remain operational and predictable. Through constant restarting, an epidemic system can monitor its global state and achieve distributed consistency. Conventional continuous epidemic systems typically use a restart mechanism at fixed time intervals, i.e. *epochs* [1], which may unnecessarily penalise the performance or may not provide sufficient guarantees on the convergence. Some epoch-less techniques provide imprecise approximations and may take a long time for adapting the system to the correct state [3, 12, 13].

The work in this paper introduces a novel epidemic algorithm with an adaptive restart mechanism. The mechanism restarts an epidemic task upon the detection of convergence or divergence in autonomous and variant epochs. Also, the mechanism ensures correct convergence to the target for all nodes through aggregating nodes decisions and acquiring *consensus* on the restart action. Moreover, the mechanism is lightweight producing small communication overhead, which can be piggybacked on existing protocol messages.

The paper is organised as follows. The model of epidemic systems is described in the next section. In particular, the data aggregation process and the seed selection method are discussed in Section 2.1. Section 2.2 describes the intrinsic convergence property of epidemic algorithms and the common methods for detecting local and global convergence. Section 2.3 addresses convergence detection under dynamic network conditions. The proposed algorithm and the restart mechanism are described in Section 3. Experimentations and results are presented in Section 4. Finally, Section 5 lists some related work and Section 6 provides conclusive remarks.

2 The Model of Epidemic Systems

An epidemic system consists of a large number of nodes N, connected to a network infrastructure (e.g., the Internet), which cooperate by exchanging messages to provide a system-wide decentralised service. Each node is assigned a unique identifier and communicates with random peers that are selected uniformly from the system (a.k.a *uniform gossip* [6]). The *Node Cache Protocol* (NCP) [14] is a simple peer-sampling service adopted for the purpose of this work, although any other membership protocol with better properties can be used (e.g., [10]).

In this work the epidemic protocols adopt the asynchronous model of '*Symmetric Push-Sum Protocol*' (SPSP) [14]. The model is a non-atomic PUSH-PULL scheme that does not lock waiting for a response after sending a PUSH message. Message order is not guaranteed and message interleaving can be present [10]. Protocols generate a PUSH message in each '*Cycle*'. Cycles are time intervals of fixed length $\mathcal{T}$ that are typically set to be greater of the Round Trip Time (RTT) on the diameter of the network. The parameter $\mathcal{T}$ is sufficient for the delivery of most messages within a cycle [1, 7, 14]. Moreover, subsequent cycles may overlap among various nodes as global cycle synchronisation is not enforced.

The transport protocol is assumed reliable (e.g., *TCP*); this limits failure types to node churn. Churn is a collective behaviour in a system where new nodes join and existing nodes depart at arbitrary times. The voluntary or adversary departure of nodes is not specified and both cases are treated as generic node failure. Although new nodes can join the system at any time, they do not participate in the current cycle of the ongoing epidemic task: they start contributing in the following cycle [1]. The fractions of joined and departed nodes in a specific time interval define the churn rates. The distribution of churn rates over time intervals is not assumed constant and there must be a time at which the system is sufficiently stable to allow convergence.

2.1 Data Aggregation and Seed Selection

The proposed restart mechanism and many epidemic systems require global data aggregation and, in particular, the global sum of a distributed set of numeric attributes or measurements. Each node i holds a local data value x_i and for this task it initialises and updates a local tuple τ_i to perform a global aggregation process on the distributed values $\{x_i, 0 < i \leq N\}$. Let τ_i be $\langle \varsigma_i, v_i, w_i \rangle$, where v_i is initialised with x_i, w_i is the weight element of the tuple and ς_i is an identifier further described below. The initialisation of the weights determine the aggregation function. For global summation, the initial weights follows a peak distribution [6, 14]. At the start time t_0, it is required to set $w_{\hat{i},t_0} = 1$ at a single node $\hat{i}$ (*seed node*), and $w_{i,t_0} = 0$ at all other nodes. The determination of the seed node $\hat{i}$ in a real-world decentralised system is challenging and requires a leader election step.

To overcome this initialisation issue, we introduce a seed selection method as follows. The tuple identifier ς is used as '*seed*' selector. The seed is a Unique Universal Identifier (UUID) generated by a global function $\mathcal{F}()$. There are two implementations of the function $\mathcal{F}$ used in this work. $\mathcal{F}_\alpha(i,t)$ which computes a UUID given a node identifier i and the current time t. The output of $\mathcal{F}_\alpha(i,t)$ preserves the natural order, such that for any two UUIDs: $U_i = \mathcal{F}_\alpha(i,t_i)$ and $U_j = \mathcal{F}_\alpha(j,t_j)$, $U_i < U_j \iff t_i < t_j \vee t_i = t_j \wedge i < j$. On the other hand, the function $\mathcal{F}_\beta(i)$ generates a random UUID.

Initially, all nodes are seed nodes and the tuple τ_i in each node i is initialised to $\tau_i = \langle \mathcal{F}(), x_i, 1 \rangle$, where each ς_{i,t_0} identifies a unique seed in the system. The initial diffusion process selects only one seed in the system for

this epoch. During the diffusion process, seeds propagate in the system following a random-walk fashion and each node performs a selection operation. Apart from the seed initialisation, the data aggregation process is based on SPSP [14].

2.2 Convergence Detection

In epidemic systems for information dissemination or data aggregation, local convergence is achieved when the local states of nodes have reached the desired target within a marginal error. In dissemination tasks, nodes achieve and detect convergence by receiving a copy of a particular information item [7]. In aggregation tasks, each node holds a local value, such as a local attribute or measurement. The data aggregation process aims at computing a numeric target value $\mathcal{V}$ at every node, which corresponds to some global synopsis function (e.g., *average*, *sum*, *max*, *sample*, etc.) over the distributed set of local values. During the epidemic process each node i updates a local estimate $e_{i,t}$ of $\mathcal{V}$ at every cycle t. The convergence process corresponds to a reduction in the variance of the local estimates. Eventually, all nodes converge to the target value when the local estimation error $\varepsilon_{i,t}$ is smaller than a global *tolerance threshold* ϵ [1, 6, 14].

Methods for local detection of convergence in aggregation processes are heuristic and require application-specific parameters. In a general method [1, 14, 15], each node i computes the estimation error $\varepsilon_{i,t} = \frac{|e_{i,t}-\mathcal{V}|}{\mathcal{V}}$, and verifies the criterion $\varepsilon_{i,t} < \epsilon$. The method also counts the number of consecutive cycles (Υ) in which the criterion is verified. The parameter Υ is used to avoid a precocious detection of convergence, which may be caused by the fluctuation of $e_{i,t}$. However, this method requires some a priori global knowledge of the target value $\mathcal{V}$. Typically, this information is unavailable or hard to obtain in real-world conditions: it is ultimately the goal of the epidemic process. The method in [7, 16] uses a technique based on the moving-average on local and remote estimates in order to approximate the target value and the estimation error. This approach is adopted in this work to compute the Standard Error (SE). SE does not require the correct target and provides less uncertainty in error measurements around the mean in comparison to other statistical measures.

In addition to the local detection of convergence, the detection of *global convergence* may also be required in some epidemic systems [7, 15]. Global convergence is needed for acquiring local awareness on the convergence of other nodes, and it is usually achieved through a poll-alike process, in which every node places a vote after the occurrence of a local event (e.g., detection of local convergence). Ultimately, the poll result at each node provides certainty on the occurrence of the event in other nodes. This technique is applied for achieving consensus for global synchronisation in the proposed restart mechanism.

2.3 Convergence Detection under Churn

Although epidemic processes are intrinsically fault-tolerant thanks to redundancy and the lack of single points of failures, the system dynamics can have a detrimental impact on the data aggregation process due to the violation of the '*mass conservation*' invariant [6, 11, 14]. The mass refers to the ideal aggregate of the initial values of all nodes in the system, which has to be conserved at all times for the formal correctness of the aggregation process. Previous work [11] has shown that under dynamic conditions, the accuracy of local estimates cannot be guaranteed, leading to an incorrect convergence: results may significantly differ from the true target due to the violation of system mass. Nevertheless, convergence to or divergence from the correct target can still be detected under some moderate churn conditions. In this work, the proposed solution can detect the violation of the system mass invariant to validate or invalidate the aggregation results at convergence.

3 The Adaptive Restart Mechanism

Algorithm 1 is a continuous epidemic process that runs over sequential epochs, where each epoch has an incremental global identifier (ι). The epoch is the inter-restart interval, and two subsequent epoch identifiers may exist in the system for some time after restart. Nodes are enforced to join the epoch with the higher identifier. Epoch length is variant and depends on the detection of convergence or divergence. The algorithm consists of several aggregation processes, sequential and parallel. The process $\mathcal{A}$ corresponds to the intended objective of the epidemic task. The process $\mathcal{C}$ is a subsequent phase for achieving consensus. Nodes join the Consensus phase after they achieve local convergence. Also, the algorithm encompasses a tuple $\mathcal{P}$ of several aggregation processes such that each $p \in \mathcal{P}$ runs in parallel with the process $\mathcal{A}$. Processes in the tuple $\mathcal{P}$ are used for the convergence detection, and their results defines the convergence correctness state (i.e. convergence or divergence).

The intended epidemic task defines the initialisation of the process $\mathcal{A}$. The process $\mathcal{C}$ and processes in $\mathcal{P}$ are all initialised for the aggregate *count*. The process $\mathcal{C}$ counts nodes which have achieved local convergence, and each process p estimates the total number of nodes joined the process $\mathcal{A}$.

Each process $p \in \mathcal{P}$ will initially start with a different random seed identifier at each node in the system. During the aggregation process, seeds of all processes in $\mathcal{P}$ are piggybacked and propagated with the messages

from the process $\mathcal{A}$. The seed selection method makes a random selection for each process due to the random seed initialisation. Moreover, a node failure will affect a random seed of each process, and causes each process to achieve different convergence. Convergence state can be verified using local estimates $e_{p,t}$, $\forall p \in \mathcal{P}$. A correct convergence is confirmed when all local estimates in $\mathcal{P}$ converges to the same target, $\forall e_{p,t} \approx \mathcal{V}, p \in \mathcal{P}$. Otherwise, $\exists e_{p,t} \not\approx \mathcal{V}$ indicates a divergence, which implies experiencing dynamical conditions during the aggregation process.

The procedure *DetectConvergence* illustrates convergence detection method. The method calculates the average of estimates in $\mathcal{P}$ every cycle and inserts the average in the queue $\mathcal{Q}$. Eventually, estimates average will converge to an approximation result, and the error among elements of $\mathcal{Q}$ becomes very small. The method verifies the detection of convergence using the SE of $\mathcal{Q}$ and monitoring it approaching the tolerance threshold ϵ_1 for a number of consecutive cycles Υ. Next, the method verifies the state of the convergence using the SE of estimates in $\mathcal{P}$. The criterion validates that the error among local estimates of processes in $\mathcal{P}$ is above a tolerance threshold of ϵ_2. The limits ϵ_1, ϵ_2 and Υ are global application parameters as described in Section 2.2.

Upon the detection of divergence in a node i, the node initiates a global restarting process using a new epoch identified $\iota_i + 1$. The restart steps are described in procedure *Restart*. Also, upon the detection of a correct convergence, node i makes a transition to the CONSENSUS phase by starting the process $\mathcal{C}$. Other nodes may join the phase at the same time or later when they converge. The seed selection process unifies the seed elements, and each node participates in the phase by adding 1 to the total data mass in the process $\mathcal{C}$. In the CONSENSUS phase, the detection method records the estimate of the process $\mathcal{C}$ in $\mathcal{Q}$ at every cycle. Each node uses the SE of $\mathcal{Q}$ and the thresholds ϵ_1 and Υ to locally detect the convergence of the CONSENSUS phase.

Achieving convergence in the CONSENSUS phase indicates the agreement among nodes to restart the epidemic task as they all have converged to the correct target (i.e. *global convergence*). However, small number of nodes in the CONSENSUS phase are enforced to join the next epoch, although they did not yet detect convergence, which optimises the inter-times between epochs. Also, it adapts the epidemic task should it experience any dynamical conditions during the CONSENSUS phase.

Procedure *ResolveEpoch* has two duties: (1) discovering and joining new epochs, and (2) applying the seed selection method to unify seed elements for each process. Each node receives a new epoch identifier starts a new epoch and reinitialise local processes as shown in procedure *Restart*. Also, the procedure updates the local tuples upon the detection of a new seed with smaller identifier. The algorithm in lines $9-11$ continue processing the received message and responses to the sender node by a PULL message with the adopted epoch identifier and seed elements. In lines $12-14$, the algorithm updates the local tuple for each process.

4 Simulations and Experimental Analysis

The algorithm is validated via simulations using PEERSIM [17]. PEERSIM is a Java-based discrete-event *P2P* simulation tool that is particularly useful to evaluate distributed protocols in large-scale systems. The simulations are event-based and adopt the event-driven engine in PEERSIM. Three events are used in the simulations: (1) *Start Event* occurs only once at the start time of each node. At this event, each node initialises local seed and

Algorithm 1: Adaptive Restart Algorithm

Require: ϵ_1, ϵ_2, Υ, $l^{\mathcal{Q}}$, $l^{\mathcal{P}}$.
Initialisation: $\iota = 0$; $\mathcal{Q} = \emptyset$; $\widetilde{\mathcal{P}} = \{\mathcal{A}, \mathcal{C}\} \cup \mathcal{P}$; and $\forall p \in \widetilde{\mathcal{P}}, p \longrightarrow \langle \infty, 0, 0 \rangle$.

1 **At start time** t_0 **at node** i**:**
2 $Restart(1, i, t_0)$
3 $Push(i, t_0)$

4 **At each cycle** t **at node** i**:**
5 $DetectConvergence(i, t)$
6 $Push(i, t)$

7 **At event 'receive message** m **from** j**' at node** i**:**
8 $ResolveEpoch(i, t, m)$
9 **if** $m.reply$ **then**
10 | **foreach** $p \in \widetilde{\mathcal{P}}$ **do** $p \longrightarrow \langle p.\varsigma, \frac{p.v}{2}, \frac{p.w}{2} \rangle$
11 | Send $\langle \iota, \widetilde{\mathcal{P}}, reply = false \rangle$ to j `// PULL to j`
12 **if** $m.\iota == \iota$ **then** `// Update local tuples in all processes`
13 | **foreach** $p \in \widetilde{\mathcal{P}}$ **do**
14 | | **if** $m.p.\varsigma == p.\varsigma$ **then** $p \longrightarrow \langle p.\varsigma, m.p.v + p.v, m.p.w + p.w \rangle$

def avg($H = \{a_1, \ldots, a_n\}$): $\frac{1}{n}\sum a$ // Average

def se($H = \{a_1, \ldots, a_n\}$): $\frac{1}{\sqrt{n}}\sqrt{\frac{1}{n-1}\sum(a - \texttt{avg}(H))^2}$ // Standard Error

procedure *Restart(ι,i, t)*

  $\iota_i = \iota$

  $phase_i =$ AGGREGATION

  $\mathcal{A}_i \longrightarrow \langle \mathcal{F}_\alpha(i,t), x_i, 1\rangle$ // Reset processes

  $\mathcal{C}_i \longrightarrow \langle \infty, 0, 0\rangle$

  foreach $p \in \mathcal{P}_i$ **do** $p \longrightarrow \langle \mathcal{F}_\beta(i), 1, 1\rangle$

procedure *Push(i,t)*

  foreach $p \in \widetilde{\mathcal{P}}_i$ **do** // Divide data elements and copy tuples

    $p \longrightarrow \langle p.\varsigma, \frac{p.v}{2}, \frac{p.w}{2}\rangle$

  $j \longleftarrow getRandomPeer()$ // Get random peer

  Send $\langle \iota_i, \widetilde{\mathcal{P}}_i, reply = true\rangle$ to j // PUSH to node j

procedure *DetectConvergence(i,t)*

  switch *phase* **do**

    case AGGREGATION **do**

      $\mathcal{Q}_i \cup \texttt{avg}(\{p.e : \forall p \in \mathcal{P}_i\})$ // Insert estimates average of $\mathcal{P}_i$

      if se($\mathcal{Q}_i$) $< \epsilon_1$ *for* Υ *cycles* **then** // Detect local convergence

        if se($\{p.e : \forall p \in \mathcal{P}_i\}$) $> \epsilon_2$ **then** // Detect divergence

          $Restart(\iota_i + 1, i, t)$ // Start a new epoch

        else // Make transition to CONSENSUS phase

          if $\mathcal{F}_\alpha(i,t) < \mathcal{C}_i.\varsigma$ **then** $\mathcal{C}_i \longrightarrow \langle \mathcal{F}_\alpha(i,t), 1, 1\rangle$

          else $\mathcal{C}_i \longrightarrow \langle \mathcal{C}_i.\varsigma, \mathcal{C}_i.v + 1, \mathcal{C}_i.w\rangle$

          phase=CONSENSUS

    case CONSENSUS **do**

      $\mathcal{Q}_i \cup \mathcal{C}_i.e$

      if se($\mathcal{Q}_i$) $< \epsilon_1$ *for* Υ *cycles* **then** // Detect global convergence

        $Restart(\iota_i + 1, i, t)$ // Start a new epoch

procedure *ResolveEpoch(i,t,m)*

  if $m.\iota > \iota_i$ **then** $Restart(m.\iota, i, t)$ // New epoch discovered

  if $m.\iota == \iota_i$ **then** // Resolve seed elements

    if $m.\mathcal{A}.\varsigma < \mathcal{A}_i.\varsigma$ **then** $\mathcal{A}_i \longrightarrow \langle m.\mathcal{A}.\varsigma, x_i, 0\rangle$

    if $m.\mathcal{C}.\varsigma < \mathcal{C}_i.\varsigma$ **then**

      $\mathcal{C}_i \longrightarrow \langle m.\mathcal{C}.\varsigma, 0, 0\rangle$ **and if** *phase*==CONSENSUS **then** $\mathcal{C}_i.v = 1$

    foreach $p \in \mathcal{P}_i$ **do**

      if $m.p.\varsigma < p.\varsigma$ **then** $p \longrightarrow \langle m.p.\varsigma, 1, 0\rangle$

data elements. (2)*Run Event* is scheduled at every cycle for each node to detect convergence and send PUSH messages. The event at all nodes stops after a predefined number of cycles. (3) *Message Receive Event* is a notification event, in which a receiver node identifies new epochs, applies seed selection method and updates local tuples.

The process $\mathcal{A}$ is initialised for the aggregate *count* targetting system size, and hence, seed elements at all nodes are set using $\mathcal{F}_\alpha(i, t_s)$, where $t_s = [0, t_{off}[$, and t_{off} is a start time synchronisation offset. The settings of threshold parameters follow previous work recommendations in [7, 11], and they are set to $\epsilon_1 = 0.5$, $\epsilon_1 = 1$, $\Upsilon = 3$, and $l^{\mathcal{Q}} = 10$. The protocol NCP is used with k-regular overlay graph initialisation, where $k = 30$.

The cycle length $\mathcal{T}$ is defined as $\mathcal{T} = 2 \times \delta + t_{off}$, where $t_{off} = 250ms$, δ is the expected maximum propagation delay in Internet. Values of δ are randomly generated using a Weibull distribution with the parameters, $\beta = 4$ that bounds δ to $125ms$, $\eta = 70ms$ is the average delay value, and $\gamma = 25ms$ is the minimum delay. A choice of $\mathcal{T} = 500ms$ is sufficiently large for typical applications and large enough to allow most messages to be delivered within the current cycle.

The results in figure 1 illustrate the selection method and restart mechanism performance. In this experiment, node churn is disabled and the result shows the algorithm behaviour under stable conditions. The results for the processes $\mathcal{A}$ and $\mathcal{C}$ are distinguished for clarity. The figure 1.(a) and 1.(b) show the variation in initial system mass over time. In the figures, data elements approach the correct value as a result of the selection method. Particularly, the figure 1.(b) presents the decrease in weights mass due to the selection of the correct seed and the discarding of other seeds.

Figures 1.(c), 1.(d) and 1.(e) illustrate the correct reach and detection of convergence in each phase. The

results validate the algorithm and the restart mechanism efficiency. Each node makes a transition to the CONSENSUS phase after the detection of the convergence. It also restarts the task after achieving convergence in the CONSENSUS phase. The figure 1.(d) shows the variation in estimation error in all processes, which indicates the reach to a correct convergence, the transition to the CONSENSUS phase, and the voting for the global restarting. Figure 1.(e) shows that 100% of nodes achieve and detect convergence in both phases with nodes restarting and joining a new epoch asynchronously.

Figure 1.(f) summarises the results of 30 simulation runs for different system sizes. The figure shows logarithmic increase in inter-restart times (epochs) as the system size increases. It also presents the variation in the epoch length for different sizes.

Results in figures 2.(a) and 2.(b) illustrate the algorithm behaviour under dynamic conditions. Two experiments are carried out. The first experiment examines the algorithm sensitivity to a single node failure and to a modest churn rates. Figure 2.(a) shows the results for a single failure injected at cycle 5 followed by the failure of 30% of the system between cycles $[60 - 120]$. The second experiment tests the algorithm under severe condition when a system loses 75% of its nodes during a task. Figure 2.(b) shows the failure of 75% of the system between cycles $[15, 195]$. In both experiments, results prove the algorithm abilities to validate or invalidate convergence even for a single node failure. The algorithm continues restarting until the system stabilises before it can enter the CONSENSUS phase and return a correct estimation.

The impact of varying the parameter $l^{\mathcal{P}}$ has also been examined; however, results are omitted due to space limitation. The results have shown that the effect was negligible under stable conditions while causing a small increase in overhead to the underlying network. In dynamic conditions, the increase in the number of processes in $\mathcal{P}$ makes validation of convergence more accurate, especially, for the detection of divergence. Although precision is essential for the detection of small amount of churn, e.g. single node failure as shown in figure 2.(a), the accuracy can also be controlled by the tolerance thresholds, and with low cost. From another perspective, the amount of error that the algorithm can tolerate corresponds to the lost portion of initial system mass. In early cycles of an aggregation process, node failure may cause a major loss in the system mass, however, after convergence the impact of churn fades [11]. Therefore, even large churn rates at late cycles cannot result in divergence detection, the figure 2.(b) shows the impact of this scenario in the cycles $[15, 60]$.

5 Related work

A simple restarting mechanism for epidemic protocols was introduced in [1], where global restarting was achieved using fixed length epochs with incremental epoch identifiers. Authors in [12] proposed a technique that restarts two overlapping aggregation processes in epochs of fixed hops. The protocol improves results in dynamic conditions; however, results accuracy can only be validated after the next epoch. The work in [3] introduces a continuous epoch-less data aggregation protocol. The protocol is based on atomic PUSH-PULL with a timeout and requires prior information about the system to produce accurate results. The *Flow Updating* aggregation protocol [13] operates under dynamic conditions without periodic restarting. Upon failure detection, the protocol uses symmetric exchanges among neighbour nodes and recovers values instantly, and hence delays system convergence. The work in [16] introduces two heuristic methods for convergence detection. The first method uses the moving-average technique for the local detection of convergence, and the second method is used for global convergence detection by utilising parallel aggregation processes. Authors in [15] provide an analysis for local and global convergence, which has shown the need for applications-specific parameters in the detection methods.

6 Conclusions

In large and extreme-scale distributed systems, continuous epidemic tasks are useful tools for monitoring and maintaining system consistency. Through periodic restarting, epidemic processes can detect and adapt to new conditions. This work introduces a novel continuous epidemic algorithm with an adaptive restart mechanism. The process restarts either upon acquiring consensus on the global convergence of the epidemic task or upon the detection of divergence. Moreover, the mechanism is lightweight with optimised communication overhead that can be piggybacked with regular message exchange. The detection accuracy of the algorithm can be tuned according to the application preference for any quick approximation or an accurate one that takes longer to compute. Also, the algorithm introduces a decentralised selection method for data aggregation tasks that require single-point initialisation. Simulation results validated the performance of the algorithm under static and dynamic conditions. Further study is required to specify approximation quality through results of multiple aggregation processes, and use results for system-mass restoration under nodes churn.

References

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

[2] Y. Cao et al. "An Overview of Recent Progress in the Study of Distributed Multi-Agent Coordination". In: *IEEE Transactions on Industrial Informatics* 9.1 (Feb. 2013).

[3] V. Rapp and K. Graffi. "Continuous Gossip-Based Aggregation through Dynamic Information Aging". In: *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. July 2013.

[4] P. Costa and J. Leitão. "Practical Continuous Aggregation in Wireless Edge Environments". In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. Oct. 2018.

[5] A. Litke, D. Anagnostopoulos, and T. Varvarigou. "Blockchains for Supply Chain Management: Architectural Elements and Challenges Towards a Global Scale Deployment". In: *Logistics* 3.1 (2019).

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

[7] M. M. Ayiad and G. Di Fatta. "Agreement in Epidemic Data Aggregation". In: *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. Dec. 2017.

[8] A. Katti and D. J. Lilja. "Efficient and Fast Approximate Consensus with Epidemic Failure Detection at Extreme Scale". In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. Mar. 2018.

[9] G. Di Fatta et al. "Fault tolerant decentralised K-Means clustering for asynchronous large-scale networks". In: *Journal of Parallel and Distributed Computing* 73.3 (2013). Models and Algorithms for High-Performance Distributed Data Mining.

[10] P. Poonpakdee and G. Di Fatta. "Robust and efficient membership management in large-scale dynamic networks". In: *Future Generation Computer Systems* (2017).

[11] M. M. Ayiad and G. Di Fatta. "Robust Epidemic Aggregation Under Churn". In: *Internet and Distributed Computing Systems*. Cham: Springer International Publishing, 2018.

[12] H.-G. Roh and C. L. Ignat. *Rapid and Round-free Multi-pair Asynchronous Push-Pull Aggregation*. Research Report RR-8044. INRIA, 2012.

[13] P. Jesus, C. Baquero, and P. S. Almeida. "Flow updating: Fault-tolerant aggregation for dynamic networks". In: *Journal of Parallel and Distributed Computing* 78 (2015).

[14] F. Blasa et al. "Symmetric Push-Sum Protocol for decentralised aggregation". In: *Proceedings of AP2PS 2011, the Third International Conference on Advances in P2P Systems*. IARIA, 2011.

[15] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. "An Efficient and Robust Decentralized Algorithm for Detecting the Global Convergence in Asynchronous Iterative Algorithms". In: *High Performance Computing for Computational Science - VECPAR 2008*. Springer Berlin Heidelberg, 2008.

[16] P. Poonpakdee, N. G. Orhon, and G. Di Fatta. "Convergence Detection in Epidemic Aggregation". In: *Euro-Par 2013: Parallel Processing Workshops*. Vol. 8374. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014.

[17] A. Montresor and M. Jelasity. "PeerSim: A scalable P2P simulator". In: *2009 IEEE 9th Int. Conference on Peer-to-Peer Computing*. Sept. 2009.

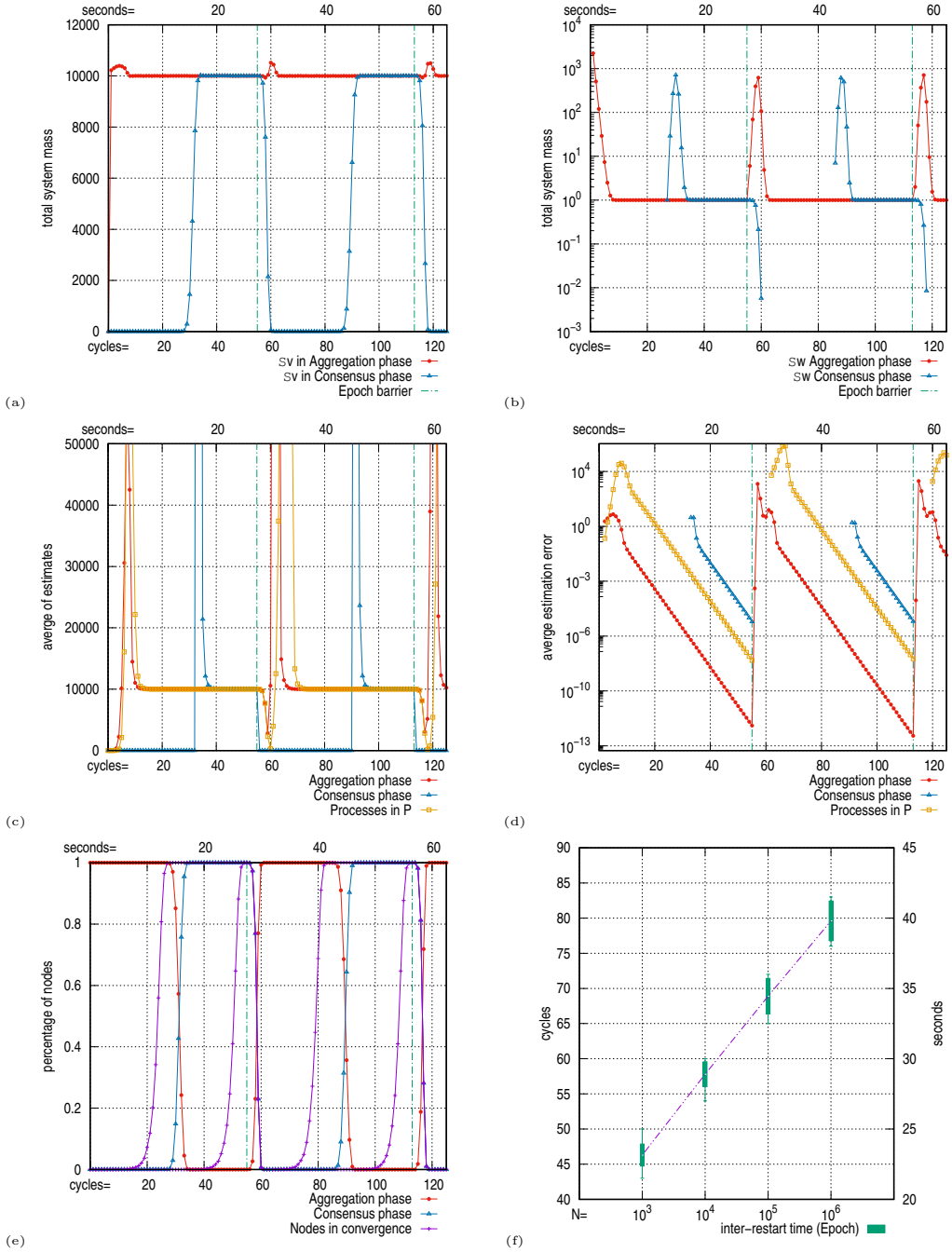
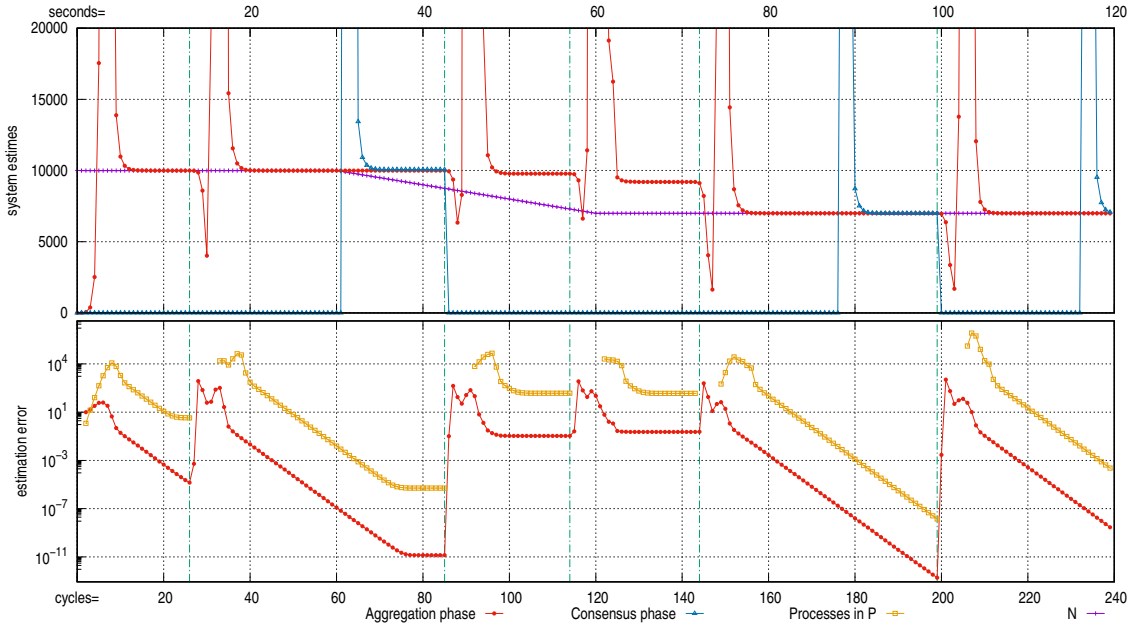
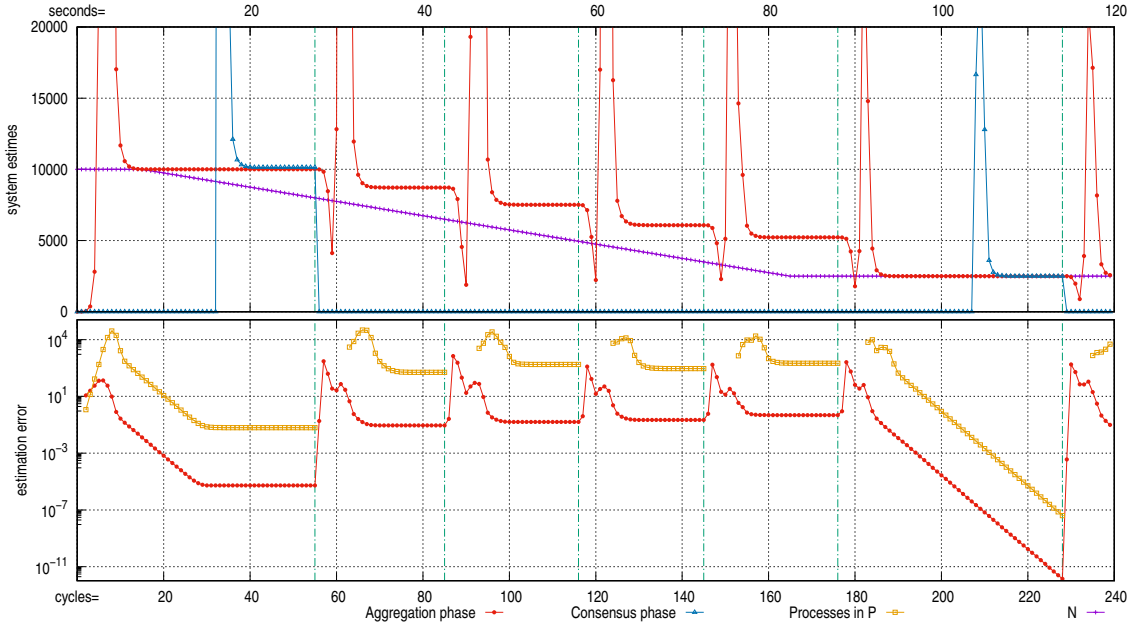


Figure 1: Algorithm performance under stable conditions, $\mathcal{V} = 10^4$, $\epsilon_1 = 0.5$, $\epsilon_2 = 1$, $\Upsilon = 3$, $l^{\mathcal{P}} = 5$, $l^{\mathcal{Q}} = 10$



(a) moderate churn



(b) severe churn

Figure 2: Algorithm performance under churn, $\mathcal{V} = 10^4$, $\epsilon_1 = 0.5$, $\epsilon_2 = 1$, $\Upsilon = 3$, $l^{\mathcal{P}} = 5$, $l^{\Omega} = 10$