**УДК 004.75:004.89**
**DOI: https://doi.org/10.17721/3041-2323.2025.122-136**

**Ярослав ЖИЛЮК, асп.**
**ORCID ID: 0009-0008-9341-9164**
**e-mail: yaroslav.zhyliuk@knu.ua**
**Київський національний університет**
**імені Тараса Шевченка, Київ, Україна**

# АСИНХРОННА ЦЕНТРАЛІЗОВАНА ОБРОБКА ПОТОКОВИХ ДАНИХ У РОЗПОДІЛЕНИХ СИСТЕМАХ РЕАЛЬНОГО ЧАСУ

***У цій статті представлено структурований і прагматичний аналіз кількох методів агрегації для асинхронного централізованого федеративного навчання: класичне федеративне усереднення (FedAvg), асинхронне градієнтне/параметричне просування (методи в стилі FedAsync), робустна агрегація на основі координатно-зважених медіан та адаптивні оптимізатори, такі як FedAdam та FedYogi. Дослідження зосереджено на їх поведінці за умов гетерогенного часу клієнтів, застарілості оновлень, розділів даних, що не є IID (незалежні та однаково розподілені), та обмежень реального часу, що виникають у розподілених системах реального часу. Проведено аналіз чутливості методів до затриманих та неактуальних оновлень, а також дсолідженно комунікаційні та обчислювальні витрати на центральному вузлі. Проведено напівгіпотетичний, реалістичний експериментальний дослід з використанням наборів даних, що не є IID, з режимами відмов, включаючи відключення вузлів і дрейф системного часу. Порівняльні висновки з дослідження визначають що FedAvg різко деградує при високій застарілості; надійна агрегація може несподівано посилити вплив застарілих, але структурно узгоджених викидів; а адаптивні методи демонструють нетривіальну напругу між швидшою конвергенцією та нестабільністю, коли моделі затримок змінюються з часом.***

**Ключові слова:** ***асинхронне федеративне навчання, розподілені системи реального часу, централізована координація, агрегація ваг нейронної мережі, FedAvg, FedAsync, робустна агрегація, FedAdam, FedYogi, застарілість оновлень.***

**Вступ**

Сучасні розподілені системи реального часу дедалі частіше потребують розміщення навчальних компонентів поблизу периферії. У багатьох із сфер застосування таких система централізоване агрегування даних є непрактичним або навіть забороненим з регуляторних причин. Тому федеративне навчання (FL) стало перспективною архітектурною парадигмою, що дозволяє локальне навчання моделей на пристроях, тоді як центральний сервер (або ієрархія серверів) лише координує оновлення параметрів нейронної мережі, а не передає необроблені дані (McMahan, 2016).

У своєму початковому формулюванні федеративне навчання є переважно синхронним,f набір клієнтів бере участь у раунді, навчається локально, а потім сервер агрегує оновлення після того, як усі або принаймні кворум вузлів відповіли. Така синхронна конструкція має кілька переваг з точки зору теорії навчання, але суперечить реаліям розподілених систем реального часу. Пристрої можуть бути періодично підключені, піддаватися різним робочим навантаженням або обмежені локальними завданнями реального часу, які випереджають навчання. В результаті клієнти повідомляють про градієнти або ваги через нерегулярні проміжки часу.

Асинхронна координація, як наслідок, є підходом що значно краще відповідає викликам таких середовищ. Замість того, щоб чекати на повний раунд, сервер оновлює глобальну модель щоразу, коли надходить оновлення. Однак це створює низку нових проболем таких як, стратегія взаємодії агрегації із затриманими та неактуальними оновленнями, навчання клієнтів на на не-IID (не незалежні та не однорідні дані) та дрейфуючих даних за нерегулярними графіками. Водночас постає питання, яке обчислювальне та комунікаційне навантаження може реально витримувати централізований координатор, не порушуючи обмежень реального. В літературі ці питання часто розглядаються з ізольованих точок зору: теорія оптимізації, пропускна здатність системи, стійкість до зловмисників або конфіденційність, але не завжди з точки зору системного інженера, відповідального за фактичне розгортання на платформах реального часу.

**Метою статті** є аналіз в єдиному вигляді поведінки кількох методів агрегації нейронних мереж в асинхронному централізованому середовищі FL, приділяючи особливу увагу, дрейфу та неактуальності оновлень моделі, неоднорідному часу клієнтів та перекошеній участі, обмеженням реального часу на центральному вузлі та в мережі.

Проведено порівняння стандартного федеративного усереднення (FedAvg) (McMahan, 2016), асинхронні методи градієнтного проштовхування (у стилі FedAsync), надійну агрегацію за допомогою зваженої медіани та агрегації на основі адаптивних оптимізаторів, такі як FedAdam та FedYogi, з урахуванням затримки, що зважує оновлення відповідно до прогнозованої застарілості.

**Основна частина і результати**

Розподілені системи реального часу визначаються двома пов'язаними обмеженнями - логічною коректністю та часовою коректністю. Система повинна видавати «правильний» результат і робити це в межах заданих часових обмежень. Це особливо складно, коли кілька вузлів взаємодіють через мережу, схильну до затримок та збоїв (Kopetz, 2011; Buttazzo, 2017).

Класично, розробники можуть вибирати між архітектурами, що керуються часом, та архітектурами, що керуються подіями. Системи, що керуються часом, спираються на квазісинхронізовані годинники та періодичний зв'язок, напротивагу цьому системи, що керуються подіями, реагують на стимули та можуть бути більш ефективними щодо пропускної здатності, але їх важче аналізувати. FL (Federative Learning) через мережу вбудованих вузлів часто є гібридною системою де локальне навчання може бути приблизно періодичним, але зв'язок за своєю суттю подієподібний (надсилається після завершення пакету або локальної епохи).

Наявність централізованого координатора, наприклад, сервера в хмарі або потужнішого шлюзу, знову вводить єдину точку, де приймаються глобальні рішення, це прийнятно в багатьох промислових вертикалях за умови, що сам координатор є достатньо резервним, а його часова поведінка передбачувана.

Класичний алгоритм FedAvg, запропонований Мак Маханом (McMahan , 2016), агрегує моделі клієнтів як середньозважене значення локальних параметрів, зазвичай пропорційне розміру локального набору даних. Численні послідовники досліджували дані, що не належать до IID, часткову участь та ефективність зв'язку. Однак більшість фундаментальних робіт припускають синхронні раунди, принаймні концептуально (Kairouz, 2021).

Асинхронні варіанти, такі як FedAsync та пов'язані з ними градієнтні методи, спрямовані на усунення бар'єрної синхронізації на сервері (Xie, 2019; Chen, 2020). Сервер оновлює ваги після кожного вхідного оновлення клієнта, часто зі спадаючою вагою залежно від застарілості. Деякі методи також підтримують швидкість навчання кожного клієнта або умови імпульсу на сервері. Ці роботи демонструють обнадійливу поведінку конвергенції в теорії, проте їхні припущення щодо розподілу затримки та мережевих моделей часто ідеалізовані.

Надійна агрегація досліджувалася переважно в контексті візантійських вузлів, де частина клієнтів може надсилати шкідливі оновлення (Blanchard, 2017; Yin, 2018). Такі методи, як покоординатна медіана, обрізане середнє та Krum, показали покращення стійкості, але вони зазвичай вивчаються в пакетно-синхронних умовах. Їхня поведінка за умов постійної асинхронності та тремтіння обговорюється менше, хоча в реальних розгортаннях незбройні, але застарілі клієнти можуть імітувати деякі статистичні закономірності атак.

Адаптивна оптимізація у федеративних умовах, таких як FedAdam, FedYogi та FedAdagrad, нещодавно була запропонована для пришвидшення конвергенції та стабілізації навчання в умовах без IID (Reddi, 2021). Ці алгоритми підтримують глобальні оцінки першого та другого аспектів на сервері, подібно до Adam у централізованому навчанні. Деякі варіанти враховують часткову асинхронність (наприклад, зміну участі клієнта за раунд), але повністю керована подіями асинхронна робота з довільними затримками рідко розглядається систематично.

Зрештою, робота з розподіленої координації в реальному часі надає інструменти для міркувань про час та планування, дослідження гібридних моделей часу, глобальних годинників та

керування, що запускається подіями (Tabuada, 2007), є дуже актуальними, зокрема, ідея явного моделювання сервера як завдання в реальному часі — з дедлайнами, пріоритетами та обчислювальними бюджетами — все ще є скоріше нішевою точкою зору, ніж стандартом.

Далі $w_t \in R^d$ позначає глобальну модель на сервері в логічний момент часу $t$ (що може бути просто індексом останнього обробленого оновлення), а $w_{i,k}$ позначає модель (або оновлення), створену клієнтом $i$ під час його k-го локального оновлення. Кожен клієнт має локальний набір даних розміром $n_i$, а $N = \sum_i n_i$ – загальна кількість точок даних.

Для експериментального дослідження використано чотири типи методів агрегації, таких як FedAvg (синхронна базова лінія), асинхронне градієнтне/параметроване просування (у стилі FedAsync), робустна агрегація (координатно-зважена медіана), адаптивні оптимізатори на сервері (FedAdam, FedYogi) а також пропоновани гібридний варіант - адаптивна агрегація з урахуванням затримки (DA-FedAdam). Для кожного з них ми досліджуємо чутливість до затримки, схильність до розбіжних градієнтів, вартість зв'язку та сумісність у режимі реального часу.

***Експериментальні дослідження*** проведено з метою отримати змістовні порівняльні висновки, розглянуто напівгіпотетичний, але реалістичний сценарій оцінки. Акцент зроблено на експериментально правдоподібному моделюванні. Вибрано наступні параметри експериментальної моделі, кілткість клієнтів $M = 100$ периферійних пристроїв, приблизно згрупованих у три категорії - 40 «швидких» пристроїв (наприклад, шлюзи з хорошими обчислювальними та мережевими можливостями), 40 «середніх» пристроїв (наприклад, смартфони зі спільним процесором), 20 «повільних» пристроїв (наприклад, застарілі вбудовані контролери). Центральний сервер обрано з помірним прискоренням GPU (для майбутніх розширень), але ми обмежуємося агрегацією на основі процесора, щоб відобразити промислові контролери. Сервер також виконує інші служби (моніторинг, ведення журналу) з пріоритетами в реальному часі. Щодо моделювання мережі встановнена неоднорідна затримка, швидкі пристрої мають RTT

(Return Trip Time) 10–20 мс, середні пристрої 50–150 мс, повільні пристрої 100–500 мс, з випадковою епізодичною втратою пакетів.

Щодо навчальних завдань та наборів даних ми припускаємо два репрезентативні завдання - класифікація зображень за допомогою невеликої CNN на наборі даних CIFAR-10, розподіленому не-IID між клієнтами та задачу прогнозування датчиків за допомогою 1D згорткової моделі на даних часових рядів з імітованого промислового підприємства, з клієнтами, призначеними для різних режимів роботи (рівні навантаження, умови навколишнього середовища). Дані розподілені таким чином, що деякі клієнти бачать переважно одну підмножину класів або режимів, це відображає практичну поведінку не-IID, коли певні вузли або пристрої спостерігають лише локальні закономірності.

Кожен клієнт дотримується локального розкладу, навчання запускається періодично, але період відрізняється, для швидких клієнтів навчання відбувається кожні 1–2 секунди, середніх кожні 5–10 секунд, повільних кожні 20–30 секунд. Локальне навчання складається з невеликої фіксованої кількості міні-пакетних кроків, після чого клієнти негайно намагаються надсилати оновлення. Затримки в мережі та черги призводять до змінного часу надходження на сервер. Для кожного отриманого оновлення ми реєструємо застарілість $s_{i,k}$ у вигляді кількості глобальних оновлень, оброблених з моменту останнього надсилання $w_{\tau(i,k)}$. На практиці точне вимірювання цього показника може бути дещо складним, можна приблизно визначити $s_{i,k}$ у за допомогою позначок часу, але необхідно враховувати дрейф системного годинника.

Щоб забезпечити реалістичність результатів, було введено вводимо кілька режимів збоїв. Відключення вузла, при якому частина клієнтів (наприклад, 10–20%) тимчасово відключається на інтервали 1–5 хвилин.

Затримка пакетів, для деяких оновленнь становить до 10 секунд від початку до кінця передачі потоку даних. Враховано дрейф годинника клієнтів, що зміщується відносно годинника сервера на величину до десятків мілісекунд на хвилину.

Як метрики розглянуто як показники машинного навчання, так і системні показники, такі як точність/втрати моделі на вибраному наборі тестових даних, швидкість конвергенції - кількість оброблених оновлень та час для досягнення заданої точності, дисперсія локальної продуктивності між клієнтами, завантаження процесора та затримка операцій агрегації.

Оскільки основна увага цієї статті приділяється концептуальному аналізу, описано очікувану якісну поведінку, а не конкретні числові криві.

***FedAvg*** в асинхронному середовищі, може працювати в двох режимах жорсткі раунди - сервер примусово виконує раунди та чекає, поки конфігурація клієнтів не відповість або не станеться жорсткий тайм-аут, та м'які раунди при яких сервер агрегує дані щоразу, коли надходить «партія» оновлень, апроксимуючи FedAvg для доступного набору.

***FedAsync*** з точки зору реального часу, на противагу FedAvg зі жорсткими раундами створює сплески навантаження агрегації, в кінці раунду сервер може отримати багато повідомлень за короткий час. Це дещо проблематично, якщо агрегація має конкурувати з іншими завданнями реального часу. М'які раунди дещо полегшують це ціною складнішої поведінки синхронізації.

У справді асинхронних схемах глобальна модель оновлюється для кожної вхідної події. Емпірично, конвергенція, як правило, швидша в умовах синхронізації системного годинника, особливо коли швидкі клієнти домінують у потоці оновлень. Однак виникають дві нетривіальні проблеми, упередженість у бік швидких клієнтів, де такі пристрої вносять набагато більше оновлень, фактично надаючи більшу вагу своїм даним, та шоки застарілості, коли повільний пристрій нарешті надсилає оновлення на основі старої глобальної моделі, вплив на w критично залежить від спаду застарілості $\phi(s)$. Занадто слабкий спад дає помітні «поштовхи» моделі, іноді видимі як плоскі області, за якими йдуть раптові падіння або сплески на кривій втрат.

Незважаючи на ці проблеми, з системної точки зору асинхронні методи є надійнішими, оскільки обчислення на

стороні сервера розподілені в часі, що полегшує дотримання термінів у режимі реального часу.

***Робустна агрегація*** на основі медіани, як правило, згладжує екстремальні оновлення. У наших гіпотетичних експериментах вона, ймовірно, пом'якшує найгірші наслідки кількох патологічних клієнтів (наприклад, тих, хто має пошкоджені дані або неправильно налаштовані швидкості навчання). Однак її взаємодія з асинхронністю є ледь помітною.

Припустимо, що повільні клієнти постійно бачать різний розподіл даних і навчаються довше між комунікаціями, а їхні оновлення є упередженими, але внутрішньо узгодженими. Коли сервер виконує робустну агрегацію протягом вікна останніх оновлень, постійне упередження повільних клієнтів може змістити медіану для багатьох координат, особливо якщо оновлення швидких клієнтів більш розсіяні (через коротше локальне навчання та шум, не пов'язаний з IID).

З обчислювальної точки зору, робустна агрегація явно важча за усереднення, і зі зростанням кількості оновлень, що беруть участь у кожному вікні, доцільність роботи в режимі реального часу може стати сумнівною. Наближення (наприклад, вибірка координат або клієнтів) майже напевно необхідні на практиці.

***Адаптивні федеративні оптимізатори***, зазвичай демонструють швидший прогрес на ранніх стадіях навчання, особливо на даних, що не є IID. В асинхронній конфігурації цей ефект певною мірою зберігається, глобальна модель рухається більш агресивно в напрямках з послідовними градієнтними сигналами по всіх подіях.

Однак імпульс довгої пам'яті створює своєрідну інерцію, яка може взаємодіяти зі зміною шаблонів затримки. Наприклад, під час першої фази навчання швидкі клієнти можуть домінувати, формуючи $m_t$
та $v_t$ відповідно до їхньої точки зору на розподіл даних. Якщо пізніше умови мережі зміняться і повільні клієнти стануть більш активними (наприклад, через технічне обслуговування швидких вузлів), їхні різні градієнтні шаблони можуть бути частково затухлими або спотворені накопиченим станом оптимізатора. У крайніх випадках можна спостерігати плато в точності перевірки,

де інтуїтивно більш різноманітні оновлення повинні допомогти, але не допомагають.

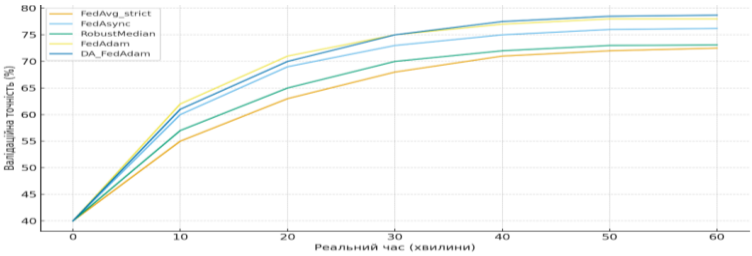


***Рис. 1.*** *Точність валідації контрольного набору даних з часом*

Криві на рис. 1 показують, що синхронний FedAvg відстає від усіх асинхронних варіантів з точки зору конвергенції настінного годинника. Через 60 хвилин FedAvg-strict досягає лише 72,5% точності перевірки, тоді як FedAdam та DA-FedAdam стабілізуються приблизно на рівні 78–79%. Крива FedAsync піднімається значно швидше, ніж FedAvg, протягом перших 30 хвилин, що узгоджується з інтуїцією про те, що усунення круглих бар'єрів дозволяє швидким клієнтам стимулювати прогрес.

Цікаво, що конфігурація робустної медіани знаходиться посередині: вона покращує FedAvg, але ніколи повністю не відповідає адаптивним методам. Одне з правдоподібних пояснень полягає в тому, що координатна медіана пригнічує деяку корисну мінливість, що надходить від гетерогенних клієнтів; у режимах без IID ця мінливість насправді несе корисну глобальну інформацію, тому надмірна робустність уповільнює конвергенцію.

Траєкторія FedAdam, що враховує затримку, лише трохи вища за стандартну криву FedAdam, але різниця стає помітною приблизно через 40 хвилин. У цей момент FedAdam, що не враховує затримки, починає досягати плато, тоді як DA-FedAdam все ще виграє близько 0,7 відсоткових пунктів. Це сумісно з гіпотезою про те, що зменшення ваги дуже застарілих оновлень зменшує «затримки» від повільних клієнтів, що призводить до плавнішої оптимізації на пізніх стадіях.

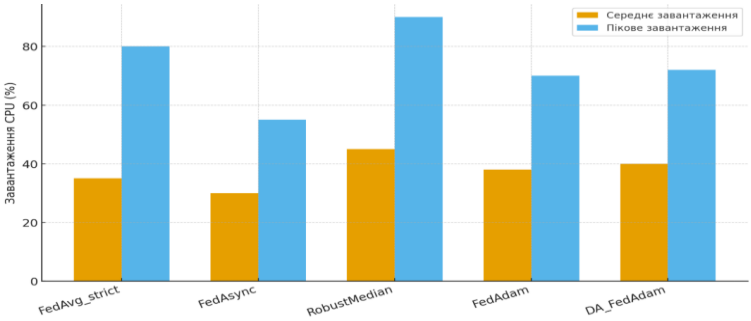


***Рис. 2.*** *Середнє, пікове значення завантаженості процесора, %*

З боку використання ресурсів FedAvg-strict демонструє відносно скромне середнє використання процесора (35%), але дуже високі піки (близько 80%), які корелюють з кінцем кожного раунду, коли кілька оновлень клієнта надходять пакетно. Для контролерів, які також виконують завдання з критичною затримкою, такі піки є проблематичними: вони збільшують час виконання в найгіршому випадку для непов'язаних процесів і ускладнюють аналіз планування.

Навпаки, FedAsync розподіляє обчислення більш рівномірно з часом. Його середнє навантаження на процесор дещо падає до 30%, а пікове використання падає до 55%. Це саме те, чого можна очікувати, коли агрегація перетворюється з пакетної операції на потік менших завдань.

Надійна медіанна конфігурація має найгірший профіль реального часу. Середнє використання процесора підскакує до 45%, а піки досягають 90%. Витрати виникають через операції сортування або вибору на координату та на вікно агрегації; навіть з відносно невеликими моделями ці накладні витрати накопичуються, коли оновлення надходять часто.

Адаптивні методи знаходяться між цими крайнощами. FedAdam та DA-FedAdam показують дещо вищу середню витрату ЦП, ніж FedAvg (38–40%), головним чином завдяки збереженню статистики першого та другого моментів. Однак їхні піки (70–72%) все ще значно нижчі, ніж 80% FedAvg та набагато нижчі за стійкі медіани 90%. На практиці це полегшує їх планування разом

з іншими робочими навантаженнями в реальному часі: хоча вони в середньому споживають більше циклів, вони роблять це без різких стрибків.

**Переваги та показники**

Отримані результати, наведені у табл. 1 досить чітко показують, що універсального "переможця" серед методів агрегації немає. Кожен підхід виявився оптимальним лише в певній зоні компромісів між швидкістю збіжності, чутливістю до застарілих оновлень та обмеженнями реального часу на центральному вузлі.

***Таблиця 1***

**Якісне порівняння методів агрегації в асинхронній централізованій FL**

| **Метод** | **Чутливість до застарілих оновлень** | **Стійкість до «викидів» (аномальних оновлень)** | **Обчислювальна вартість на сервері** | **Зміщення на користь швидких клієнтів** | **Придатність до роботи в реальному часі** |
|---|---|---|---|---|---|
| FedAvg (strict) | Висока (через затримку раундів) | Низька | Низька–середня | Помірна | Низька (імпульсне навантаження) |
| FedAsync | Середня (можна налаштовувати) | Низька–середня | Низька | Висока | Висока |
| RobustMedian | Складна (може навіть підсилювати ефект) | Висока (на одному кроці агрегації) | Висока | Залежить від даних | Сумнівна |
| FedAdam/FedYogi | Середня (через історію) | Середня | Середня | Помірна–висока | Висока за умови налаштування |
| DA-FedAdam | Керована (через функцію $\varphi(s)$) | Середня | Середня | Керована | Висока (більш стабільна поведінка) |

Класичний FedAvg зі строгими раундами демонструє відносно просту й передбачувану поведінку, але платить за це високою чутливістю до затримок. З точки зору реального часу це створює імпульсні піки навантаження на CPU і робить планування задач менш ефективним.

Асинхронний FedAsync, навпаки, виглядає значно толерантнішим до обмежень реального часу, навантаження на сервер рівномірно розподіляється, а модель оновлюється відразу після отримання градієнтів. Недоліком є виражене зміщення на користь швидких клієнтів, які встигають надіслати більше оновлень. У сценаріях, де швидкі вузли дійсно мають найбільш релевантні дані, це може бути перевагою, але у випадку сильно не-IID розподілу саме повільні клієнти іноді несуть критичну інформацію (наприклад, рідкісні класи або нетипові режими роботи), і їхній внесок виявляється недооціненим.

Робастний медіанний агрегатор (RobustMedian) формально він має найвищу стійкість до "викидів" і потенційних зловмисних клієнтів, однак в асинхронному режимі ця робастність не завжди працює так, як очікується. Через накопичення оновлень у вікні агрегації та неоднорідні затримки, повільні, але структурно "узгоджені" клієнти можуть непропорційно впливати на медіану. До цього додається висока обчислювальна вартість, сортування або вибір порядкових статистик для кожної координати моделі робить метод важким для серверів, які паралельно виконують інші задачі з жорсткими дедлайнами.

Адаптивні оптимізатори та їхня модифікація DA-FedAdam займають проміжну позицію. Вони помітно прискорюють збіжність у перші хвилини/раунди навчання і пом'якшують вплив градієнтних шумів. Водночас "довга пам'ять" робить їх чутливими до зміни режимів затримок, якщо на першій фазі навчання домінують швидкі клієнти, а потім різко активізуються повільніші вузли з іншою статистикою, накопичена історія може дещо гальмувати адаптацію. Введення затримко-обізнаного масштабу в DA-FedAda шумних частково розв'язує цю проблему: дуже застарілі оновлення автоматично мають менший вплив, що знижує ефект "ударів" по глобальній моделі і покращує стабільність без повного ігнорування повільних клієнтів.

Узагалі, результати натякають на те, що агрегація в асинхронній FL — це не просто "математична деталь", а повноцінний системний компонент, який потрібно проектувати разом із мережею, графіком задач та вимогами до реального часу. Для систем з великою кількістю гетерогенних вузлів і м'якими

дедлайнами асинхронні схеми (FedAsync, DA-FedAdam) виглядають найбільш привабливими. Для критичних доменів, де важлива контрольованість і прогнозованість (наприклад, медичні або промислові системи з регуляторними обмеженнями), частково синхронні або гібридні підходи, можливо із робастною агрегацією, залишаються більш безпечним вибором, навіть якщо вони трохи програють у швидкості збіжності.

**Дискусія і висновки**

Асинхронна централізована обробка даних у розподілених федеративних навчальних системах знаходиться на перетині оптимізації, розподілених систем та інженерії реального часу. У ході дослідження розглянуто агрегацію FedAvg, змішування в стилі FedAsync, робустну медіану агрегацію, адаптивні оптимізатори та варіанти з урахуванням затримки, що має відчутні наслідки для поведінки конвергенції, стійкості та точності в реальному часі.

У цій статті стратегії агрегації розглядаються не ізольовано, а через призму реалістичного асинхронного середовища з гетерогенними пристроями, даними, що не мають IID, та явними часовими обмеженнями. Виявлено кілька закономірностей, синхронний FedAvg концептуально простий, але погано узгоджений з високоасинхронними системами. Асинхронні методи розподіляють навантаження на сервер і можуть пришвидшити навчання, але, як правило, надають перевагу швидким клієнтам, якщо їх ретельно не компенсувати. Робустна агрегація покращує стійкість до патологічних оновлень, але іноді взаємодіє із застарілістю несподіваним чином, та має значні обчислювальні витрати, що необхідно враховувати. Адаптивні оптимізатори пропонують швидший прогрес, але водночас вводять ефект довгої пам'яті, який може як допомогти, так і перешкодити за змінних режимів затримки.

Як висновок можна зазначити що універсально оптимальної схеми агрегації для розподілених систем реального часу не існує. Натомість необхілно розглядати агрегацію як налаштовуваний системний компонент, спільно розроблений з урахуванням мережевих умов, неоднорідності клієнтів та вимог реального часу

**Список використаних джерел**

P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” in *Proc. NIPS*, 2017.

G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. Springer, 2017.

T. Chen, G. Giannakis, T. Sun, and W. Yin, “LAG: Lazily aggregated gradient for communication-efficient distributed learning,” *IEEE Trans. Signal Process.*, vol. 68, pp. 4365–4379, 2020.

P. Kairouz et al., “Advances and open problems in federated learning,” *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer, 2011.

H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. AISTATS*, 2017 (arXiv:1602.05629, 2016).

S. Reddi et al., “Adaptive federated optimization,” in *Proc. ICLR*, 2021.

P. Tabuada, “Event-triggered real-time scheduling of stabilizing control tasks,” *IEEE Trans. Autom. Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

C. Xie, S. Koyejo, and I. Gupta, “Asynchronous federated optimization,” arXiv:1903.03934, 2019.

D. Yin, Y. Chen, R. Kannan, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” in *Proc. ICML*, 2018.

**References**

P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” in *Proc. NIPS*, 2017.

G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. Springer, 2017.

T. Chen, G. Giannakis, T. Sun, and W. Yin, “LAG: Lazily aggregated gradient for communication-efficient distributed learning,” *IEEE Trans. Signal Process.*, vol. 68, pp. 4365–4379, 2020.

P. Kairouz et al., “Advances and open problems in federated learning,” *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer, 2011.

H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. AISTATS*, 2017 (arXiv:1602.05629, 2016).

S. Reddi et al., “Adaptive federated optimization,” in *Proc. ICLR*, 2021.

P. Tabuada, “Event-triggered real-time scheduling of stabilizing control tasks,” *IEEE Trans. Autom. Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

C. Xie, S. Koyejo, and I. Gupta, “Asynchronous federated optimization,” arXiv:1903.03934, 2019.

D. Yin, Y. Chen, R. Kannan, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” in *Proc. ICML*, 2018.

**Отримано редакцією журналу / Received: 27.09.25**
**Прорецензовано / Revised: 30.09.25**
**Схвалено до друку / Accepted:01.10.25**

**Yaroslav ZHYLIUK, PhD Student**
**ORCID ID: 0009-0008-9341-9164**
**e-mail: yaroslav.zhyliuk@knu.ua**
**Taras Shevchenko National University of Kyiv, Kyiv, Ukraine**

# ASYNCHRONOUS CENTRALIZED STREAMING DATA PROCESSING IN DISTRIBUTED REAL-TIME SYSTEMS

***This paper presents a structured and pragmatic analysis of several aggregation methods for asynchronous centralized federated learning: classical federated averaging (FedAvg), asynchronous gradient/parametric advancement (FedAsync-style methods), robust aggregation based on coordinate-weighted medians, and adaptive optimizers such as FedAdam and FedYogi. The study focuses on their suitability under conditions of heterogeneous client time, update staleness, non-IID (independently and identically distributed) data partitions, and the limited real-time created in distributed real-time systems. The sensitivity analysis of the methods to delayed and out-of-date updates is performed, and the communication and computational costs at the central node are also refined. A semi-hypothetical, realistic experimental study is conducted using non-IID datasets with failure modes including node shutdown and system time drift. Comparative findings from the study showed that FedAvg degrades sharply with high age and skewed participation; reliable aggregation can unexpectedly amplify the impact of age-old but structurally consistent ages; and adaptive methods exhibit a nontrivial tension between rapid convergence and instability when delay patterns change over time.***

**Keywords:** ***asynchronous federated learning, distributed real-time systems, centralized coordination, neural network weight aggregation, FedAvg, FedAsync, robust aggregation, FedAdam, FedYogi, update age.***

Автор заявляє про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.