УДК 004.62:681.3.06

**Д. І. Харчук**
**О. К. Колесницький**

# АНАЛІЗ ОСОБЛИВОСТЕЙ ПОТОКОВОЇ ОБРОБКИ ДАНИХ ТА СПОСОБІВ ПІДВИЩЕННЯ ЇЇ ПРОДУКТИВНОСТІ

Вінницький національний технічний університет

***Анотація***
*Проведено аналіз особливостей потокової обробки даних та запропоновано способи підвищення її продуктивності, використовуючи мову програмування Elixir на базі віртуальної машини Erlang.*
**Ключові слова:** потік, черга, система реального часу, модуль, паралелізм, аналіз.

***Abstract***
*The analysis of features of streaming data processing is carried out and the ways of increasing its productivity are offered, using programming language Elixir on the basis of the Erlang virtual machine.*
**Keywords**: steam, queue, real time system, module, parallelism, analysis.

## Вступ

Системи реального часу з'явилися давно, але, протягом багатьох років, режим реального часу і потокова обробка залишались спадщиною апаратних систем. Приклади потокової обробки зустрічаються скрізь: соціальні мережі, ігри, розумні міста, інтелектуальні вимірювальні прилади, навіть пральна машина.

Метою роботи є аналіз особливостей потокової обробки даних та огляд способів підвищення її продуктивності.

## Результати дослідження

Потокова обробка даних являє собою обробку даних в реальному часі. Існує три типи систем реального часу: жорсткого реального часу, м’якого реального часу та майже рального часу.

Класифікація систем реального часу наведена в табл. 1.

| Клас часу | Приклад | Одиниця виміру затримки | Терпимість до затримок |
|---|---|---|---|
| Жорсткий | Кардіостимулятор, гальмівна система | Мікросекунди - мілісекунди | Нульова - повна відмова системи, потенційна смерть людини |
| М’який | Система резервування білетів, електронні біржові торги | Мілісекунди - секунди | Низька - без відмови системи, без загроз життю |
| Майже реальний | Відео в YouTube, домашня автоматика | Секунди - хвилини | Висока - без відмови системи, без загрози життю |

В багатьох ситуаціях обчислювальна частина системи працює в режимі нежорсткого реального часу, але користувачі споживають дані не в реальному часі через мережеві затримки, дизайн додатку чи тому, що додаток просто не запущений. Іншими словами, існує служба нежорсткого реального часу і клієнти, що споживають дані, коли їм це буде потрібно. Це і називається системою потокової

обробки даних - система нежорсткого реального часу, що надає доступ до даних, в момент коли клієнт їх потребує. Це не система м’якого або майже реального часу, це потокова система.

Концепція потокової обробки даних виключає будь-які неточності, пов’язані з неправильним розумінням різниці між системами м’якого реального часу, майже реального і зовсім нереального часу, та дозволяє сфокусуватись на розробці системи, що доставляє повідомлення користувачу в момент їх запиту.

Після ознайомлення з різновидами систем реального часу та принципом потокової обробки даних, слід описати основні процеси та модулі, з який складається потокова обробка даних. Діаграма потокової обробки даних зображена на рис. 1.

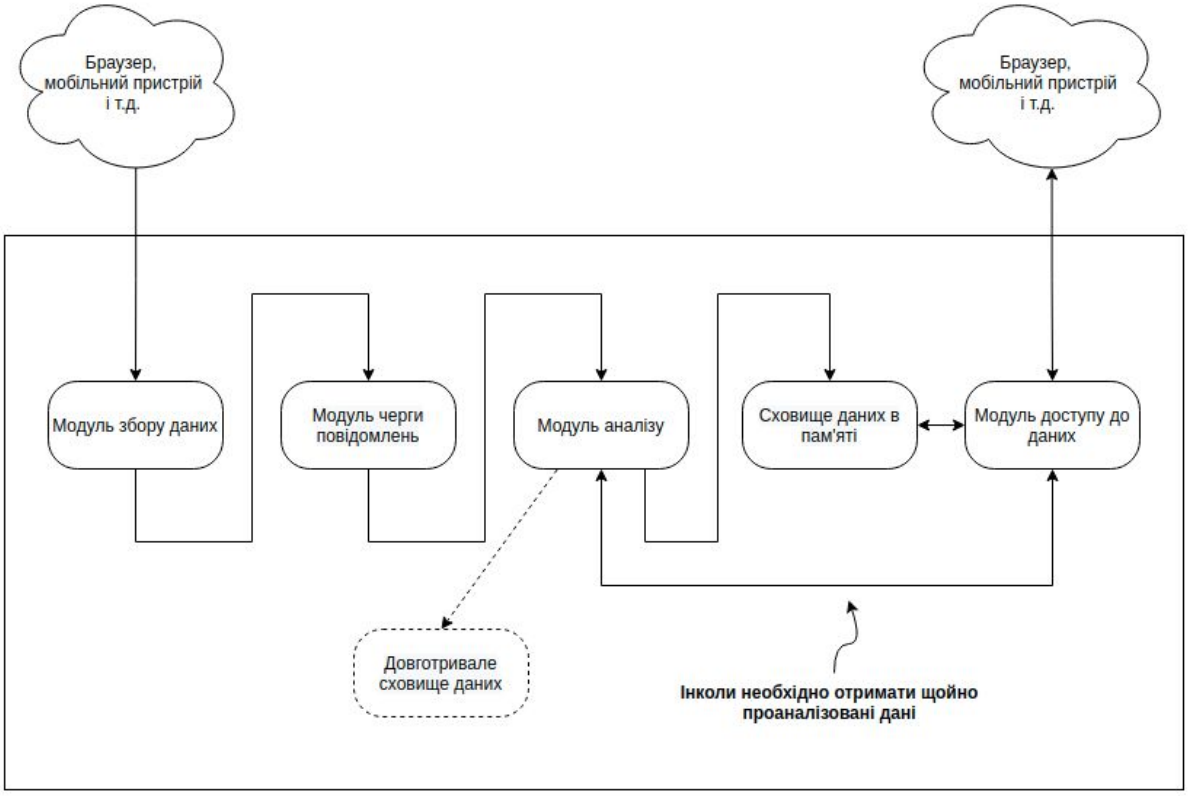


Рис.1. Архітектурна діаграма потокової обробки даних

Хоч і на показаній вище діаграмі кожен модуль має свою назву та місце, слід розуміти, що кожна ланка потокової обробки даних визначена не так строго, як в деяких інший архітектурах. Хоч вони й називаються модулями, їх можна вважати деталями конструктора, що дає змогу сконструювати таке архітектурне рішення, що відповідає конкретній задачі. В багатьох випадках модулі навіть можуть фізично знаходитись в різних місцях[1].

Щоб впоратись із постійним потоком даних, кожен модуль має виконувати поставлену перед ним задачу за допустимий час, для уникнення колапсу системи. Зважаючи на те, що обробка та аналіз потребують найбільше системного часу та ресурсів, слід звернути увагу на покращення продуктивності та стабільності саме модуля аналізу даних. Тому, даний модуль варто реалізовувати з використанням паралельних обчислень. Як приклад, можна використати мову програмування Elixir, що працює на базі віртуальної машини Erlang та має потужні механізми паралельної обробки даних і високі показники відмовостійкості. Часто ресурси системи бувають обмежені та постійне масштабування просто неможливе, тому даний підхід розв'язує цю проблему також.

Ще одним із способів підвищення продуктивності є реалізація кластера нодів (екземплярів) модуля аналізу. Вбудовані механізми Elixir/Erlang дозволяють легко комунікувати між нодами в кластері та рівномірно розподіляти ресурси[2, 3].

Комбінування цих підходів та продумане розподілення ресурсів дозволить підтримувати систему працездатною навіть зі збільшенням потоку даних.

## Висновки

Встановлено, що системи потокової обробки даних бувають різних типів та, залежно від задачі, до системи ставлять різні вимоги. Одним із способів підвищення продуктивності системи є впровадження паралельних обчислень та налаштування кластеру для модуля аналізу.

### Список використаної літератури


1. Andrew G. Psaltis — M: Streaming Data, 2017. — 19-29 с.
2. Elixir School [Електронний ресурс] – Режим доступу до ресурсу: https://elixirschool.com/ru/lessons/advanced/otp-concurrency/
3. Saša Jurić. Elixir in Action — M. : Concurrency primitives, 2019. — 149-179 с.



***Харчук Денис Ігорович*** — студент групи 1КН-20м, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: kharchuk.dennis@gmail.com

Науковий керівник: **Колесницький Олег Костянтинович** — кандидат технічних наук, доцент кафедри комп'ютерних наук, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця

***Kharchuk Denys Igorovich*** — student of Information Technologies and Computer Engineering Department, Vinnytsia National Technical University, Vinnytsia, e-mail: kharchuk.dennis@gmail.com

Supervisor: **Kolesnytskyy Oleg K.** — Candidate of Science (Engineering), associate professor of Computer Science Department, Informations Technologies and Computer Engineering Faculty, Vinnytsia National Technical University, Vinnytsia