УДК 004.451
Мельник В.М., Мельник К.В., Багнюк Н.В., Пишук Ю.С.
Луцький національний технічний університет

МОДЕЛЮВАННЯ ПІДХОДУ ЗМЕНШЕННЯ ЗАТРАТ НА КОПІЮВАННЯ ПІД ЧАС ОБМІНУ ПОВІДОМЛЕННЯМИ НА БАЗІ TCP ПРОТОКОЛІВ

Melnyk V.M., Melnyk K.V., Bahnyuk N.V., Pyshuk Y.S. Modeling of the copying overheads reduction based on TCP protocols during message exchange. The extension of the traditional socket interface is simplified for messaging based on TCP/IP protocols with using a new mechanism for finding the messages instead of receiving them at the set queue. Finding a message through a TCP socket allows the user to receive the expected message by skipping the order of the previous packets in the connection turn. By using the messaging search procedure, the user program can view the TCP socket as a list of messages that can be received or deleted with their data not just from the socket top, but from any arbitrary position in the queue. The simulated search mechanism allows you to bypass the operation of unwanted data copying into the library buffer before it is received and further displaying in the socket buffer. The interface for socket search messages can be used on the basis of the Linux kernel, and the results of an experimental verification can be obtained by using a set of simple micro-benchmark method.
Keywords: TCP/IP network interface, socket, message search, copying procedure, receiving time

Мельник В.М., Мельник К.В., Багнюк Н.В., Пишук Ю.С. Моделювання підходу зменшення затрат на копіювання під час обміну повідомленнями на базі TCP протоколів. Змодельовано розширення традиційного сокетного інтерфейсу для здійснення обміну повідомленнями на основі TCP/IP-протоколів з залученням нового механізму пошуку повідомлень замість їх отримання за встановленою чергою. Пошук повідомлення через TCP-сокет дозволяє користувачу отримувати очікуване повідомлення, обминаючи чергу проходження попередніх пакетів в порядку з'єднання. З застосуванням процедури пошуку під час обміну повідомленнями програма користувача може розглядати TCP-сокет як список повідомлень, що можуть отримуватися чи видалятися із їхніми даними не тільки з вершини сокета, а з будь-якої довільної позиції в ньому. Змодельований механізм пошуку дозволяє обійти операцію небажаного копіювання даних в буфер бібліотеки перед його отриманням і подальшим відображенням в буфері сокета. Інтерфейс для сокетів пошуку повідомлень може застосовуватись на базі ядра Linux, а результати експериментальної перевірки можуть бути отримані за допомогою набору простих мікро-тестових міток.
Ключові слова: TCP/IP мережевий інтерфейс, сокети, пошук повідомлень, процедура копіювання, час отримання.

Мельник В.М., Мельник К.В., Багнюк Н.В., Пышук Ю.С. Моделирование подхода уменьшения потерь на копирование в ходе обменеа сообщениями на базе TCP протоколов. Смоделировано расширение традиционного сокетного интерфейса для осуществления обмена сообщениями на основе TCP/IP-протоколов с использованием нового механизма поиска сообщений вместо их получения согласно установленной очереди. Поиск сообщения через TCP-сокет позволяет пользователю получать ожидаемое сообщение, минуя очередь прохождения предыдущих пакетов в порядке соединения. С применением процедуры поиска при обмене сообщениями программа пользователя может рассматривать TCP-сокет как список сообщений, получаемых или удаляемых с их данными не только с вершины сокета, но и с любой произвольной позиции в нем. Смоделирован механизм поиска позволяет обойти операцию нежелательного копирования данных в буфер библиотеки перед его получением и последующим отображением в буфере сокета. Интерфейс для сокетов поиска сообщений может применяться на базе ядра Linux, а результаты экспериментальной проверки могут быть получены с помощью набора простых микро-тестовых меток.
Ключевые слова: TCP/IP сетевой интерфейс, сокеты, поиск сообщений, процедура копирования, время получения.

1 Вступ. Високопродуктивні кластерні обчислення в основному використовуються для виконання довготривалих громіздких розрахунків наукового чи обчислювально-прикладного характеру. Особливо велику користь комп'ютерні мережеві кластери приносять в швидкодії виконання прикладних задач розподілених та локалізованих обчислень, які застосовуються в наукових мікроскопічних та телескопічних зйомках, які вимагають досить великої роздільної здатності. Подібні комп'ютерні системи здебільшого і потребують залучення у їх інтегровану роботу спеціалізованих операційних систем, спеціалізованого програмного та апаратного забезпечення [1]. Однак, домінуючу роль в обчисленнях подібного виду відіграє швидкодія, оскільки розподілені порції інформації передаються до кожного комп'ютера в системі на відповідну обробку, а параметр швидкодії прямо пропорційний швидкодії обміну пакетів між обчислювальними машинами, тобто отриманні та відсиланні повідомлень, які доставляють інформацію для обробки.

Крім того, окремі комп'ютери з відповідною частотою надсилають дані та повідомлення один одному чи на комп'ютер, що об'єднує оброблену інформацію з метою оновлення сукупної її цілісності, або надання іншим комп'ютерам даних та інформації, необхідних як параметрів чи доповнень для наступних розрахункових частин.

Базові комп'ютерні структури на основі кластерних компонентів можуть забезпечити гнучке та ефективне середовище для додатків з інтенсивною обробкою даних на базі розподілених платформ [2–5]. Додатки для таких структур розробляються спеціальним чином з набору підібраних взаємодіючих програмних компонентів, розміщення яких, поряд з ресурсами обчислення, відіграє важливість в ступені гнучкості та оптимізації продуктивності роботи прикладних додатків.

Наприклад, в роботі [5] згадується також, що в багатьох додатках з інтенсивною обробкою даних об'єм їх може розподілятися на визначені користувачем субблоки, обробка яких може проходити конвеєрно. Відмічається, якщо в ході організації роботи додатків обробка і зв'язок можуть перекриватися, то покращення продуктивності також буде залежати від розподілу обчислень, блоковості і розміру повідомлень, тобто субблоків даних. У відповідності з літературними даними блоки середніх та малих розмірів призводять до покращення балансу навантаження і конвеєризації. В практиці комунікації багато повідомлень генеруються саме блоками малих (до 1024 байт) розмірів. Однак блоки більшого розміру зменшують кількість повідомлень і досягають вищої пропускної здатності каналу зв'язку [6, 7]. Проте, ймовірно, вони можуть створювати дизбаланс навантаження і зменшення конвеєрності.

Додатки, написані з використанням інтерфейсу сокетів на базі *kernel* та TCP/IP поряд із вимогами продуктивності та інтенсивності обробки даних мають також і інші вимоги, такі як гарантія продуктивності, масштабованість цих гарантій і пристосовність до гетерогенних мереж. Щоб дозволити таким додаткам мати перевагу у використанні високоефективних протоколів, дослідники використали ряд підходів, в тому числі і високопродуктивні сокетні рівні через протоколи на рівні користувача. До них відносяться і реалізація *архітектури віртуального інтерфейсу* та AHB [10]. Додатки, які використовують їх, повинні бути написані з урахуванням збереження продуктивності зв'язку на базі TCP/IP протоколів.

Як показують результати окремих досліджень [5,10,11], в ході реорганізації окремих компонентів додатків досягаються значні покращення в продуктивності, яке приводить до підвищення масштабованості додатків з гарантіями виконання і дрібноблокового балансування навантаження. Таке навантаження, в свою чергу, робить додатки більш адаптованими до гетерогенних мереж. За архітектурою віртуального інтерфейсу різні характеристики роботи сокетів дозволяють більш ефективний поділ даних на вихідних вузлах, що призводить також до покращення продуктивності на порядок. Разом з високою продуктивністю сокети з низькими накладними затратами надають можливість додаткам досягти і якісних показників у багатьох напрямках вимірювання і мають чітке застосовуються у проектуванні, розробці та реалізації додатків з інтенсивною обробкою на сучасних кластерах.Не зважаючи на те, що структура кластера дозволяє багатьом комп'ютерам бути якомога повністю використаними з метою зменшити затрати часу на обчислення [5,6], все ж кластерні комп'ютерні системи зазнають накладних затрат в ході необхідної високопродуктивної комп'ютерної комунікації. Накладні затрати на виконання подібних обчислень залежать також і від кількості комп'ютерів в кластері, бібліотек використання для полегшення зв'язку, від вибору комунікаційного інтерфейсу між комп'ютерами та іншими складовими блоками всередині обчислювального кластера. Незважаючи на різноманітні експериментальні результати для кластерних взаємозв'язків останні дослідження продемонстрували взаємозалежність ефективного дизайну бібліотеки для організації обміну повідомленнями в кластерах, що використовують складові компоненти TCP/IP та Ethernet. Вони дозволили досягти продуктивності, співмірної із власними з'єднаннями [12, 13].

Отже, в даній роботі дослідження буде також спрямоване на прийомі та передачі повідомлень на основі TCP/IP протоколів.

Рис. 1. Залежність ефективності тримання очікуваного повідомлення від попередніх в черзі та неочікуваних повідомлень

Залежність витрат, які виникають під час використання повідомлень на основі протоколів сімейства TCP/IP, залежить від розміру повідомлень та їх частоти надходження на кожен робочий комп'ютер, і від того, чи повідомлення є очікуваним чи неочікуваним (несподіваним) для користувача. Повідомлення може надходити як неочікуване у тому випадку, якщо його дані надходять до приймача перед стартом системного виклику бібліотекою з метою одержати таке повідомлення в буфер пам'яті на рівні користувацького додатка. Відомо, що спочатку неочікувані дані копіюються в тимчасовий буфер бібліотеки [13, 14]. Для передачі повідомлень на базі TCP/IP-протоколів його можна вважати отриманим, коли дані про нього з'являються в мережі, а TCP-стек розміщує його в буфер сокета з'єднання між двома хостами комунікації.

Рисунок 1 висвітлює ту ситуацію, коли вона очікує на конкретне повідомлення поряд з неочікуваними повідомленнями, які надійшли до системи в той же момент. Наприклад, система очікує на повідомлення з комунікаційним інтерфейсом та з певним типом тега [8]. Інтерфейс передавання повідомлень (ІПП) представляє собою стандартизовану і портативну систему передачі повідомлень, розроблену групою наукових та промислових дослідників для реалізації на різноманітних спаралелених обчислювальних архітектурах.

Для кластерної комунікації існує декілька досить ефективних реалізацій інтерфейсу передавання повідомлень, багато з яких є вільними і доступними чи відкритими для використання. Вищеописаний підхід сприяв також поштовху до розпаралелення програмного забезпечення з метою розробки великомасштабних і портативних додатків, які призначені до виконання паралельних обчислень.

В ході мережевої комунікації сокетний інтерфейс операційної системи для TCP-протоколів отримує певні байти або інформацію від утвореного з'єднання в установленому відповідному порядку, використовуючи один із системних мережевих викликів recv() або read(). Враховуючи це, щоб в наступному отримати доступ до очікуваного повідомлення, спочатку потрібно звільнити сокет від попередніх непередбачених повідомлень. В загальному процесі отримання повідомлень робочими додатками, ці дані, швидше за все, будуть потрібні пізніше. Кожне з них повинно бути скопійоване в спеціальну призначену пам'ять – пул прийому неочікуваних повідомлень. Такі операції перекопіювання неочікуваних повідомлень спричиняють суттєві накладні витрати (збільшується час затримки в ході отримання повідомлення) на копіювання. Далі, перед отриманням повідомлення на рівні додатка, потрібно перевірити пул прийому неочікуваних повідомлень, і лише після цього викликати до виконання функцію прийому очікуваного повідомлення для додатка користувача на рівні сокета з'єднання.

Метою даної роботи є моделювання розширення звичайного сокетного інтерфейсу на базі операційної системи Linux, за рахунок якого можна отримувати доступ до випадкового очікуваного повідомлення, тобто до місця його розміщення в сокетному буфері, обминаючи почерговість вибірки всіх попередніх перед ним повідомлень, які надійшли в чергу. Оновлений розширений інтерфейс повинен виконувати знаходження потрібної інформації в сокетному буфері і дозволяти додатку користувача отримувати очікуване повідомлення з будь-якого стекового положення, обминаючи операцію копіювання попередніх повідомлень в пул неочікуваних повідомлень, які надійшли раніше перед зазначеним повідомленням очікування додатком.

Збоку реалізації подібної розробки потрібно застосовувати багаторазовий пошук. Програма обміну повідомленнями або бібліотека використання повинна розглядати сокет TCP як послідовний список повідомлень, що можуть отримуватися додатком користувача і при реалізації процедури отримання зразу видаляти дані не тільки з вершини, як це робиться в порядку черги, але і з будь-якої довільної точки сокетного буфера.

Удосконалений діючий інтерфейс для сокетів, які відшукують повідомлення в стеку у відповідності з вищеописаним методом, є взаємодоповнюючим для тих робіт, які своєю метою напрямлені на підвищення продуктивності процесу обміну повідомленнями з використанням протоколів TCP. Серед них можна виділити два напрямки – це ті, які спрямовані на використання перебудованої бібліотеки з удосконаленим або більш ефективним її дизайном чи конфігурацією та керовані подіями збоку власної доповненої архітектури або такі, які використовують спеціальну підтримку збоку втіленої апаратної перебудови, відмінної від загальноприйнятої та інтерфейсу мережевої карти, чи такі, які базуються на підходах розщеплення TCP-рівня [9,13,15]. Однак, дана робота базується на ефективності використання інтерфейсу операційної системи саме на рівні сокета для TCP-повідомлень і описана модель повинна співпрацювати з ідеями, які концентруються на підтримці інтерфейсу мережевої карти або на перебудованій архітектурі

діючої бібліотеки використання з метою підвищення продуктивності роботи інших системних компонентів.

Інтерфейс для здійснення необхідної операції пошуку очікуваних повідомлень в сокетному буфері згідно моделі може бути розроблений і реалізований на базі ядра Linux. Згідно літературних даних передбачається, що експериментальна перевірка може здійснюватися за допомогою використання тестового способу набору мікро-міток, дані для якого повинні отримуватися поза запитом, реалізованим для даного сокета. Результати деяких уже пророоблених теоретичних розрахунків та припущень дають можливість виявити, що скорочення часу обробки з використанням сокетів, які прямим способом відшукують повідомлення в стеку, повинно економити більш як третину часу. Зростання продуктивності системи, яка б функціонувала згідно з запропонованою моделлю, буде також прямо залежати від отримання більших за об'ємом повідомлень очікування чи від кількості попередніх повідомлень, які потрібно обійти в черзі отримання. Патч сокетів прямого пошуку повідомлень є доступний за адресою http://www.ece.purdue.edu/~vpai/ssocks/, який і можна використати в ході досліджень.

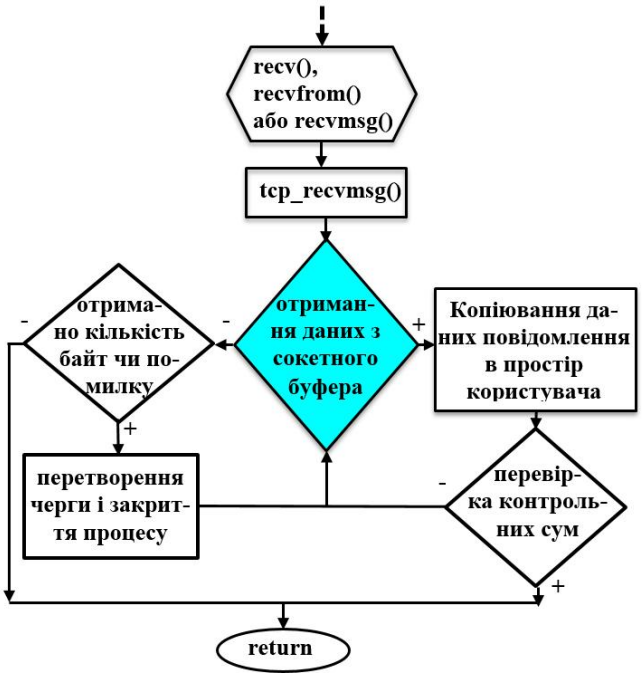

Рис. 2. Алгоритм функції recvmsg() для обміну TCP-повідомленнями на базі Linux

2. Основи Linux TCP

Згідно літературних джерел [9,13,14] TCP-стек під управлінням ядра Linux працює з сокетними буферами sk_buff's (sock_buf's чи skb's). Оскільки пакети даних приймаються мережевим пристроєм в процесі їх надходження, то вони розміщаються в так званий кільцевий буфер, а сокетний буфер sock_buff призначений для розміщення даних і повністю співвідноситься до них. Сокетний буфер sock_buff зберігає також метадані для кожного пакета, а стек операційної системи Linux для протоколів TCP/IP обробляє дані пакета, взаємодіючи з sock_buff. Для конкретного з'єднання, асоційованого кожного з сокетним буфером sock_buff проходить розміщення його в загальній черзі сокетних буферів. Для полегшення процедури прийому їх в правильному порядку та управління прийнятими пакетами кожному з sock_buff присвоюється послідовний номер в цій черзі, згідно якого вказується кількість байтів, надісланих для кожного пакета зокрема в ході з'єднання. Таке втілення дозволяє в процесі повторного запиту на рівні мережі можливість відновлювати пакети на стороні прийому та вилучати дані з сокетних буферів. Додаток користувача «не знає» або повинен би знати впорядкованість даних в пакетах чи почерговість надходження їх в мережу. Завдання, які асоціюються виключно з додатком користувача, полягають у тому, яким чином елементи даних надсилаються з джерела та визначення фіксованих їх довжин.

Коли програма користувача здійснює системні виклики `recv()`, `recvfrom()` або `recvmsg()` на TCP-сокеті, то функція `tcp_recvmsg()` виконується в ядрі Linux (`net/ipv4/tcp.c`). Рисунок 2 демонструє уривок наведеної блок-схеми алгоритму роботи цих функцій згідно запропонованої моделі. Функція `tcp_recvmsg()` починає копіювання даних з буферів `sock_buff's` у черзі сокета, які вказують на фактичні дані, розміщені в кільцевому буфері. Функція перевіряє перший `skb` у буфері сокета, а потім копіює дані в буфер простору користувача. Якщо `skb` має більше даних, ніж запитується, то функція `tcp_recvmsg()` залишає про це "мітку-нагадування" на черзі буфера сокета. Якщо користувач запитує більше даних, ніж об'єм даних, що розміщені на першому `skb`, то він виділяється разом з відповідними даними на кільцевому буфері, а вказані кроки, описані вище, продовжуються з наступним `skb` який знаходиться наступним в черзі. За замовчуванням функція `tcp_recvmsg()` повертає всі дані запиту із буфера сокета після процедури повного зчитування, вилучаючи всі буфери `skb`'s, які містили очікувані дані отримання в повному об'ємі, і поновлює номер послідовності першого байта для здійснення наступної операції зчитування на сокеті.

Для відстеження треку, за яким здійснювалося зчитуваня з буфера сокетів TCP також використовує номери послідовностей і визначає порядок продовження процедури зчитування. На кожному кроці системна змінна `copied_seq` чітко фіксує порядок копіювання і отримує номер послідовності, за яким буде здійснюватися зчитування даних. Іншими словами, ця змінна міститиме впорядковану послідовність того, що вже було скопійовано, і що буде ще зчитуватися далі з черги прийому. В такому випадку, якщо послідовність номерів `copied_seq` була більша, ніж номер першої базової послідовності `skb`, а частина вже була зчитана з неї, то повна довжина запиту даних буде копіюватися, починаючи з номера послідовності, зазначеного в `copied_seq`. Отже, через використання зафіксованих номерів послідовностей будуть здійснюватися визначення даних, які запит прийому повинен зчитувати на сокеті.

3. Особливості додаткової процедури пошуку для запропонованої моделі

Основна мета застосування сокетів прямого пошуку повідомлень в черзі полягає в отриманні даних, які розміщуються на приймальному буфері сокета у будь-якому порядку. Коли дані копіюються з сокета, то відповідні суббуфери `skb`, які задіяні для шуканого повідомлення зі скопійованими даними, повинні бути вилученими з буфера, а поточний список їх (`skb`'ів) також повинен бути виправлений. Оскільки дані, які в номерах послідовності було зчитано, більше не наявні і не доступні з-за їх відсутності, то наступні звернення для їх отримання на сокеті повинні «знати про це» і враховувати, що ці дані більше не є доступними в буфері.

Це реалізується через список з'єднань, який містить початкові та кінцеві номери послідовностей для кожного "вибраного отвору" в сокетному буфері прийому. Створення отвору звільняє простір пам'яті у буфері сокета і дозволяє нормалізувати керування поведінкою TCP-потоку незалежно від місця в буфері, з якого були прийняті користувачем і видалені дані. Коли запит прийому починає процедуру копіювання даних користувачеві, він пропускає на шляху будь-яку дірку, яка йому зустрічається, і продовжує отримувати дані нормально, згідно порядкового номера, який слідує після дірки. В процесі отримування даних список дірок зростає, проходить їх зливання, а разом з тим і пов'язане з їх видаленням динамічне скорочення черги.

Запропонована реалізація прямого пошуку повідомлень в даній роботі вимагає створити оновлений потоковий протокол SOCK_SEEK_STREAM, який використовував би той же самий стек, що і TCP та звичайні сокети типу SOCK_STREAM. Однак базові функції повинні би бути дещо видозмінені таким чином, щоб можна було здійснювати прямий пошук сокетів типу SOCK_SEEK_STREAM.

У випадку, якщо сокетний запит здійснюється на звичайний сокет, який не здійснює процедури пошуку, або якщо виклик не відшукує пакет, який очікується користувачем для отримання, то шлях через стек TCP та його функції повинні бути майже ідентичними до коду, що використовується звичайним ядром Linux. Однак, якщо запит пошуку здійснюється на запропонованому сокеті пошуку згідно наведеної нами моделі, то шлях через стек TCP залишається однаковим, проте може змінюватися шлях коду через окремі функції. В основному зміни стосуються функції `tcp_recvmsg()`, її коду та субфункцій, які викликаються цією функцією.

Всі необхідні нововведені модифікації повинні також бути застосовними для керування списком дірок, їх порядковими номерами і `skb`'ами (буферами), які були вже зчитаними. Ще одна

вагома зміна повинна бути внесена до функції `tcp_recvmsg()`, яка стосується процедури отримання сокета, на якому здійснюється пошук. Вона полягає в відключенні механізму попереднього завантаження черги TCP. Хоча механізм попереднього завантаження черги TCP дозволяє краще керувати потоковими ресурсами в процесі обміну повідомленнями, однак він все ж спричиняє невелике зниження продуктивності роботи. Також неможливо легко змінювати чергу завантаження в загальній процедурі обміну повідомленнями для того, щоб дозволити реалізацію подальшого пошуку. В зв'язку з усім вищесказаним, механізм попереднього завантаження повинен бути вимкнений саме в момент прийому на сокет, який здійснює прямий пошук очікуваного повідомлення.

Після того, як змодельований сокет типу `SOCK_SEEK_STREAM` буде створений, відомі нам мережеві функції `recv()` і `recvmsg()` можна використовувати в роботі як звичайні функції. Нова розроблена функція `seek_recv()` реалізується у вигляді системного виклику, якому необхідно надати наступні аргументи:

```
ssize_t seek_recv(int s, void * buf, size_t len, int flags, size_t
offset);
```

Аргументи для виклику функції `seek_recv()` ідентичні, як і для функції `recv()`, зі зміною, доданою для вказівки передавання кількості байтів для переведення в потік. Як відомо, таке зміщення завжди вказує на перший байт, який повинен бути отриманий через застосування системного виклику recv().

Так як системний виклик `seek_recv()` змінює базову структуру `msghdr`, а потім викликає загальну функцію `sock_recvmsg()`, то для пошуку повідомлень очікування для конкретних одержувачів також можна використовувати функцію стандартної бібліотеки `recvmsg()`. Змінна `msg_seek` була додана до структури `msghdr` для того, щоб вказати зміщення пошуку. Модифікуючи структуру `msghdr`, передану в функцію `recvmsg()`, можна легше здійснювати пошук пакета, який очікується для отримання, без залучення в роботу нової спеціальної функції.

6 Висновки

У даній роботі запропоновано нове змодельоване розширення сокетного рівня, яке дозволяє отримувати випадковий доступи з одного TCP-з'єднання. Новий інтерфейс для сокетів прямого пошуку повідомлень, який отримує дані шуканого повідомлення поза запитом, може, на думку авторів, бути реалізований в системі Linux і протестований з використанням набору простих мікроміток. Модельні передбачення доводять про суттєве скорочення часу обробки отримання очікуваних повідомлень, а, за одно, і зростання загальної продуктивності обміну повідомлень.

Змодельовано новий сокетний інтерфейс, використання якого, як видно з наведеного алгоритму та удосконалення системних викликів, свідчить про те, що на початковому етапі розробки коду додатків потрібні лише деякі не значні зміни. На практиці використання цей інтерфейс потрібно буде інтегрувати в повнофункціональну бібліотеку обміну повідомленнями або бібліотеку, яка відповідала б за мережеву комунікацію, перш ніж робити подальші висновки.

Дією наступних досліджень послужить вплив повідомлень різного об'єму (байт) або різної їх кількості за встановленим порядком отримання на параметри отримання їх кінцевим користувачем та продуктивності роботи обміну взагалі на базі ядер системи Linux. Досвід використання цього нового інтерфейсу свідчить про те, що на початковому етапі розробки коду додатків потрібні лише деякі зміни.

1. Мельник В.М. Використання звичайних сокетів API для файлових систем типу Hadoop. / Міжнародна науково-практична конференція молодих вчених та студентів. «Інформаційне, програмне та технічне забезпечення систем управління організаційно-технічними комплексами» (5-6 квітня 2015 року). // Міжвузівський збірник «Комп'ютерно-інтегровані технології: освіта, наука, виробництво». – Луцьк. – 2015, №18. – с. 40-49.
2. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE MICRO*, 15(1):29–36, 1995.
3. P. Buonadonna and D. Culler. Queue Pair IP: A Hybrid Architecture for System Area Networks. In *Proceedings of the 29th International Symposium on Computer Architecture*, pages 247–256, May 2002.
4. D. Dunning, G. Regnier, G.McAlpine, D. Cameron, B. Shubert, F. Berry, A. Merritt, E. Gronke, and C. Dodd. The Virtual Interface Architecture. *IEEE MICRO*, 18(2):66–76, March 1998.
5. В. М. Мельник, Н. В. Багнюк, К. В. Мельник. Вплив високопродуктивних сокетів на інтенсивність обробки даних / Науковий журнал «ScienceRise» №6/2(11)2015. – с. 38-48.

6. Melnyk V.M. The Full Performance Effects for Parallel TCP Sockets on a Wide-Area Network. // Міжвузівський збірник «Комп'ютерно-інтегровані технології: освіта, наука, виробництво». – Луцьк. – 2014, № 16-17 ст. 32-39.
7. V. Melnyk, N. Bahnyuk, K. Melnyk, O. Zhyharevych, N. Panasyuk. Implementation of the simplified communication mechanism in the cloud of high performance computations. East-European journal of Enterprise Technologies. – Kharkiv (DOI: 10.15587/1729-4061.2017.98896). – 2017. – № 2/2/86. – p. 24-32.
8. T. M. P. I. Forum. MPI: A Message-Passing Interface Standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994.
9. P. Gilfeather and A. B. Macabe. Making TCP Viable as a High Performance Computing Protocol. In *Proceedings of the Third LACSI Symposium*, October 2002.
10. M. Lin, J. Hsieh,D.H. C. Du, and J. A. MacDonald. Performance of High-Speed Network I/O Subsystems: Case Study of a Fibre Channel Network. In *Proceedings of the 1994 conference on Supercomputing*, pages 174–183. IEEE Computer Society Press, 1994.
11. В.М. Мельник, П.А. Пех, К.В. Мельник, Н.В. Багнюк, О.К. Жигаревич. Побудова та використання міждоменного механізму зв'язку для високопродуктивної обробки даних. // Східно-європейський журнал передових технологій. – Харків. – 2016. – № 1/9/79. – с. 10-15.
12. S. Majumder and S. Rixner. Comparing Ethernet and Myrinet for MPI Communication. In *Proceedings of the Seventh Workshop on Languages, Compilers, and Run-time Support for Scalable Systems (LCR 2004)*, pages 83–89, October 2004.
13. S. Majumder, S. Rixner, and V. S. Pai. An Event-driven Architecture for MPI Libraries. In *Proceedings of the 2004 Los Alamos Computer Science Institute Symposium*, October 2004.
14. S. H. Rodrigues, T. E. Anderson, and D. E. Culler. High-Performance Local Area Communication With Fast Sockets. In *Proceedings of the 1997 USENIX Technical Conference*, pages 257–274, January 1997.
15. V. Melnyk, N. Bahnyuk, K. Melnyk, O. Zhyharevych, N. Panasyuk. Implementation of the simplified communication mechanism in the cloud of high performance computations. East-European journal of Enterprise Technologies. – Харків (Scopus DOI: 10.15587/1729-4061.2017.98896). – 2017. – № 2/2/86. – p. 24-32.