УДК 656+004.9 DOi 10.30837/ bi.2024.1(100).01

С. М. Неронов[1], Г. А. Плєхова[2], М. В. Костікова[3]

[1]ХНАДУ, м. Харків, Україна, sernikner@gmail.com, ORCID iD: 0000-0003-2381-1271
[2]ХНАДУ, м. Харків, Україна, plehovaanna11@gmail.com, ORCID iD: 0000-0002-6912-6520
[3]ХНАДУ, м. Харків, Україна, kmv_topaz@ukr.net, ORCID iD: 0000-0001-5197-7389

УДОСКОНАЛЕННЯ АРХІТЕКТУРИ NEURONET АВТОТРАНСФЕРУ

Для створення єдиного інформаційного простору транспортних послуг без зайвих капітальних витрат на створення спеціальної IT-інфраструктури, що є основою підвищення конкурентній спроможності транспортних та дорожніх організації надані практичні рекомендації з удосконалення архітектури Neuronet автотрансферу.

АРХІТЕКТУРА, ВІРТУАЛІЗАЦІЯ, КЛАСТЕР, СЕРВЕР, САЙТ

S. M. Neronov, G. A. Pliekhova, M. V. Kostikova. Improvement of Neuronet autotransfer architecture. In order to create a unified information space for transport services without unnecessary capital costs for the creation of a special IT infrastructure, which is the basis of increasing the competitiveness of transport and road organizations, practical recommendations are provided for improving the Neuronet architecture of auto transfer.

ARCHITECTURE, VIRTUALIZATION, CLUSTER, SERVER, SITE

Вступ

Рекомендації з удосконалення архітектури Neuronet автотрансферу базуються на твердженні про те, що програмна платформа Neuronet автотрансферу є гетерогенною розподіленою системою. Вертикальне масштабування такої системи можна здійснити завдяки збільшенню серверної потужності базової системи. Робочу версію транспортного порталу (з урахуванням попереднього тестування елементів програмних, технічних та апаратних рішень взаємодії бортового інформаційно-комунікаційного комплексу з транспортним порталом та результатів обчислювального експерименту з визначення інформаційних потоків та обсягу комп'ютерних ресурсів, необхідних для моніторингу транспортних ситуацій) не зважаючи на існування більш сучасних зразків було розгорнуто на сервері на базі двох процесорів Intel Xeon 3.00 GHz, 2 Гбайт оперативної пам'яті, двох жорстких дисків за технологією SCSI 7 Gb у апаратному RAID-масиві [1, 2], але це не вплинуло на швидкодію системи.

Виклад основного матеріалу

Завдяки експериментальним дослідженням при навантаженні серверу запитами користувачів визначено, що основним недоліком обраної архітектури був надто малий об'єм оперативної пам'яті, що є критичним у разі серверної віртуалізації. Тому для здійснення масштабування серверних ресурсів було придбано та введено до експериментальної дії сервер самостійної зборки української компанії (рис. 1) на базі процесору Quad-Core 3 ГГц INTEL Xeon QC E3-1240V2, материнської плати INTEL S1200BTLR, 4-ох модулів пам'яті Kingston 8 Gb із загальним об'ємом 32 Гбайт DDR3, 3-ох накопичувачів HDD 1 Тбайт Seagate ES ST1000N 3, оптичного пристрою DVD-RW ASUS DRW-24B5ST та корпусу серверного 2U CSV UNI (400 Вт).

Рис. 1. Сервер R-Line на базі INTEL Xeon E3

Така архітектура є масштабованою «вертикально», тобто дозволяє додавати кількість модулів пам'яті, накопичувачі. Однак цей тип масштабування звичайно є обмеженим за «горизонтальне», що передбачає додавання однотипних чи гетерогенних вузлів до ферми або кластеру транспортного порталу.

Поруч із цим масштабування на рівні застосування технологій віртуалізації дозволяє виконувати фактичне прозоре збільшення ресурсів як за моделлю «вертикального» масштабування, так й «горизонтального». Для виконання завдань платформи віртуалізації транспортного порталу обрано систему віртуалізації Proxmox VE, яка використовує тільки x86_64 архітектуру. Proxmox VE базується на дистрибутиві Debian Linux та є Web-інтерфейсом до застосування технологій віртуалізації типу KVM та LXC.

Для використання KVM-віртуалізації центральний процесор повинен підтримувати апаратну віртуалізацію (Intel VT або AMD-V). Система KVM (Kernel-based Virtual Machine) дозволяє запускати у середовищі віртуальних машин фактично будь-яку операційну систему, наприклад, Linux, Windows чи FreeBSD. Для LXC (Linux Containers) апаратна віртуалізація не

потрібна, оскільки контейнери Linux функціонують як запуск одного ядра операційної системи Linux для кожного контейнеру. Контейнери є механізмом захисту Web-додатків та створюють оточення навколо кожного додатку. Це призводить до покращення надійності системи та дозволяє проектувати за модульним принципом вміст віртуальної машини.

Установка Proxmox VE є дуже простою та швидкою. Упевнившись, що перший сервер працює, слід виконати установку додаткового сервера або вузла, процес установки ідентичний процесу установки «першого», тільки вказуємо відповідне ім'я сервера та його IP-адресу. У самому кластері можна заходити на кожен вузол та отримати ідентичний Web-інтерфейс керування системою. Таким чином можна створити на базі системи віртуалізації кластер високої доступності. Позначаються абревіатурою HA (англ. High Availability – висока доступність). Створюються для забезпечення високої доступності сервісу, що надається кластером. Надмірна кількість вузлів, що входять в кластер, гарантує надання сервісу у разі відмови одного або декількох серверів. Типове число вузлів – два, це мінімальна кількість, що приводить до підвищення доступності. У разі кластеру високої доступності слід застосовувати декілька фізичних систем з Proxmox VE (за даними: http://habrahabr.ru/post/122338).

Принцип дії кластеру розподілу навантаження будується на розподілі запитів через один або кілька вхідних вузлів, які перенаправляють їх на обробку в інші, обчислювальні вузли. Початкова мета такого кластера – продуктивність, однак, у них часто використовуються також і методи, що підвищують надійність. Подібні конструкції називаються серверними фермами. Для такого типу кластеру можна визначити ряд віртуальних машин на одному фізичному сервері з Proxmox VE, однак, у такому разі слід пам'ятати про загрозу відмови системи в цілому.

Комбінація кластерів на базі віртуальних машин дозволить значно підвищити надійність роботи транспортного порталу, а також його продуктивність та готовність до навантажень. На рис. 2 наведено зовнішній вигляд панелі управління Proxmox VE. Відповідна система дозволяє уявити всі інформаційні ресурси та віртуальні мережі у якості певного віртуалізованого центру обробки даних. Перевагою застосування кластеру на базі Proxmox VE є можливість швидкої міграції віртуальних машин у різні обчислювальні середовища та розташування їх накопичувачів у певних мережевих сховищах даних.

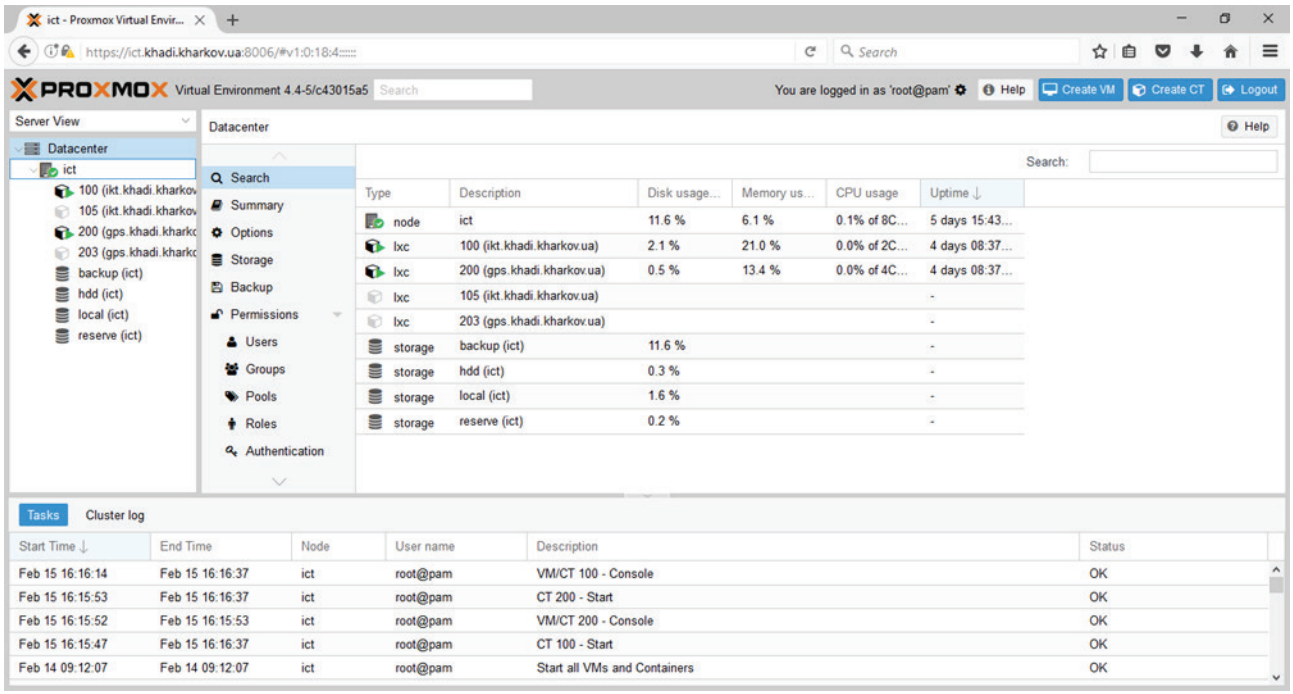

Рис. 2. Середовище адміністрування Proxmox VE

Слід розділяти поняття: масштабованість – здатність своєчасно реагувати на безперервне зростання навантаження і непередбачені напливи користувачів; доступність – надання доступу до додатку навіть у разі надзвичайних обставин; продуктивність – навіть найменша затримка в завантаженні сторінки може залишити негативне враження у користувача (за матеріалами: http://www.insight-it.ru/masshtabiruemost/masshtabiruemye-veb-arkhitektury/).

Розробка фактично кожного сайту (порталу) спрямована на функціонування із максимальною стабільністю, тобто мати доступність для абсолютно всіх потенційних відвідувачів у кожен момент часу, однак завжди трапляються непередбачені ситуації,

які можуть стати причиною тимчасової недоступності. Для мінімізації потенційного збитку доступності додатка необхідно уникати наявності компонентів в системі, потенційний збій у яких привів би до недоступності, зменшення функціональності або втрати даних (сайту в цілому або частини порталу). Таким чином, кожен сервер або будь-який інший компонент системи повинен мати хоча одного дублера (не важливо в якому режимі вони працюватимуть: паралельно або один «страхує» інший, перебуваючи при цьому в пасивному режимі), а дані повинні бути репліковаными як мінімум у двох примірниках (причому бажано не на рівні RAID, а на різних фізичних машинах). Зберігання декількох резервних копій даних десь окремо від основної системи (наприклад, на спеціальних сервісах або на окремому кластері) також допоможе уникнути багатьох проблем. Не варто забувати і про фінансову сторону питання: «страхування» на випадок збоїв вимагає додаткових істотних вкладень в устаткування, які має сенс намагатися мінімізувати.

Масштабованість прийнято розділяти на два напрямки: вертикальна масштабованість – збільшення продуктивності кожного компонента системи з метою підвищення загальної продуктивності та горизонтальна масштабованість – розбиття системи на більш дрібні структурні компоненти та рознесення їх по окремих фізичних машинам (або їх груп) та/чи збільшення кількості серверів, які паралельно виконують одну й ту ж функцію.

При розробці стратегії росту системи слід шукати компроміс між ціною, часом розробки, підсумкової продуктивності, стабільністю та багатьма іншими критеріями. З фінансової точки зору вертикальна масштабованість є далеко не самим привабливим рішенням, оскільки ціни на сервера з великою кількістю процесорів завжди ростуть практично експоненціальне щодо кількості процесорів. Саме по цьому найбільш цікавий є горизонтальний підхід. Але вертикальна масштабованість має право на існування, особливо в ситуаціях, коли основну роль відіграє час і швидкість вирішення завдання, а не фінансове питання. Це продиктовано тим, що купити потужній сервер істотно швидше, ніж практично заново розробляти додатки, адаптуючи їх до роботи на великій кількості паралельно працюючих серверів.

Архітектуру транспортного порталу обрано таким чином, що кожен компонент системи є окремим та незалежним від інших. Тому актуальною є задача рівномірного розподілення запитів між доступними серверами додатків. Запуск типового сайту починається з дуже простої архітектури – один Web-сервер (звичайно в його ролі виступає Apache), який виконує всю роботу по обслуговуванню HTTP-запитів, що надходять від відвідувачів. Він віддає клієнтам так звану «статику», тобто файли, що лежать на диску сервера і не потребують обробки: картинки (gif, jpg, png), листи стилів (css), клієнтські скрипти (javascript). Той же сервер відповідає на запити, що вимагають обчислень – це формування html-сторінок, хоча іноді динамічно створюються і зображення та інші документи. Найчастіше відповіді на такі запити формуються скриптами, написаними на java, php або іншими мовами (за матеріалами: http://habrahabr.ru/post/15362/).

Недоліком простої схеми роботи сайту в тому, що різні за характером запити (віддача файлів з диска і обчислювальна робота скриптів) обробляються одним й тим же Web-сервером. Обчислювальні запити потребують збереження в пам'яті сервера багато інформації (інтерпретатор скриптової мови, самі скрипти, дані, з якими вони працюють) та можуть займати багато обчислювальних ресурсів. Видача статики, навпаки, не вимагає багато ресурсів процесора, але може займати тривалий час, якщо у клієнта низька швидкість зв'язку. Внутрішній устрій сервера Apache припускає, що кожне з'єднання обробляється окремим процесом. Це зручно для роботи скриптів, однак неоптимально для обробки простих запитів. Відповідно «важкі» процеси Apache багато часу проводять в очікуванні (спочатку при отриманні запиту, потім при формуванні відповіді), не оптимально займаючи пам'ять сервера.

Вирішення цієї проблеми – розподіл роботи по обробці запитів між двома різними програмами – тобто поділ на frontend (сторона користувача ресурсу або зовнішнє представлення) та backend (бізнес-логіка роботи додатка). Легкий frontend-сервер виконує завдання по віддачі статики, а решта запитів перенаправляє на backend (застосовує режим проксі), де виконується формування сторінок. Очікування повільних клієнтів також бере на себе frontend, і якщо він використовує мультиплексування (коли один процес обслуговує кількох клієнтів, наприклад, nginx або HAproxy), то очікування практично не впливає на навантаження обладнання серверу (рис. 3).

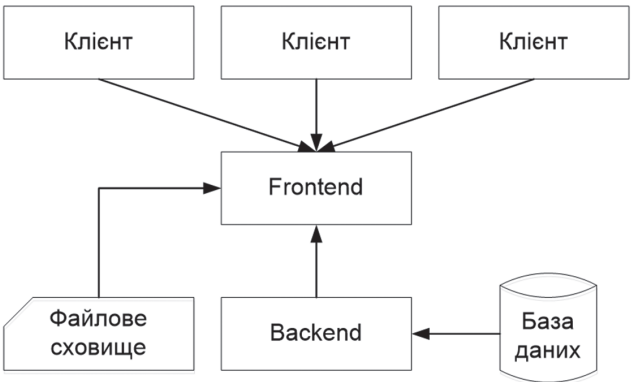

Рис. 3. Типова архітектура Web-ресурсу

Слід зазначити, що сучасний Web-ресурс фактично завжди застосовує базу даних, в якої зазвичай зберігаються основні дані системи, наприклад, MySQL

та PostgreSQL. Часто окремо можна виділити окреме сховище бінарних файлів, де містяться картинки (наприклад, ілюстрації до статей сайту та фотографії) або інші файли.

У більшості випадків на початку життєвого циклу сайту всі компоненти його архітектури розташовуються на одному сервері. Якщо він перестає справлятися з навантаженням, то є просте рішення – винести найбільш легко відокремлювані частини на інший сервер.

Найбільш продуктивним рішенням є застосування статичних сайтів (тільки frontend, що автоматично генерується одноразово на стороні серверу чи засобів розробника), однак, хоча вони є найбільш швидкісними, для рішення порталу такий підхід не задовольнить. У разі рішення масштабу Web-порталу звичайно будуть застосовані стандартні технології рівня frontend-backend із можливістю кешування статики.

Типовою ситуацією для зростаючого сайту є стан, коли база даних вже винесена на окрему машину і виконано поділ на frontend та backend, однак навантаження продовжує збільшуватися і backend не встигає обробляти запити. Це означає, що необхідним є розподіл обчислень на кілька серверів. Зробити це відносно просто – достатньо ввести до експлуатації другий сервер (певну віртуальну машину) і розгорнути на ньому програми та скрипти, що необхідні для роботи backend. Після цього треба зробити так, щоб запити користувачів розподілялися (балансувалися) між наявними серверами.

Крім цього мають місце менш поширені варіанти, засновані на DNS, тобто в процесі визначення клієнтом IP-адреси сервера з необхідним йому Інтернет-ресурсом адрес для нього формується з урахуванням навантаження на доступні сервери, а також з певних географічних міркувань (рис. 4).

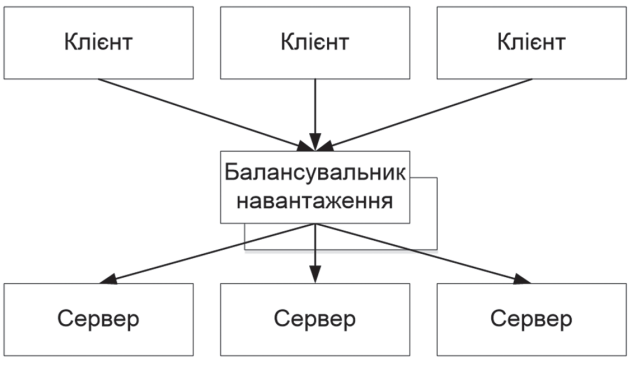

Рис. 4. Типова архітектура балансування навантаження Web-ресурсу

Мережеве устаткування, що дозволяє розподіляти навантаження між декількома серверами, зазвичай коштує досить значні суми, але серед інших варіантів саме цей підхід пропонує найвищу продуктивність та стабільність (в основному завдяки якості, також таке обладнання іноді поставляється парами, що працюють за принципом HeartBeat або визначення працездатності вузлів). Іншим варіантом є застосування програмного забезпечення, зазвичай це просто HTTP-сервери, що перенаправляють запити іншим Web-серверам на інших вузлах замість відправки безпосередньо на обробку інтерпретатору мови програмування. Всі запити проходять через балансувальник, який визначає кому з серверів віддати запит на обробку. При отриманні запиту від клієнта балансувальнику потрібно визначити якому з Web-серверів переслати запит. Алгоритм прийняття рішення називається методом або стратегією балансування (за матеріалами: http://www.phphighload.com/ 2012/ 08/ blog-post.html; http://habrahabr.ru/post/147390/). Найбільш поширені стратегії:

– Round Robin де з доступних серверів будуються черги і балансувальник вибирає перший в черзі. Після виконання запиту сервер переміщається в кінець черги;

– менша кількість з'єднань. Балансувальник веде облік кількості незакритих з'єднань та обирає той сервер, у якого кількість таких з'єднань менша;

– використання «ваги» серверів. Кожному серверу в залежності від потужності присвоюється вага, яка використовується для ранжирування.

Стратегія балансування, що не включає перевірку стану серверів або хоча б працездатності, не придатна для використання, так як не гарантує обробку запиту. Тому, балансувальнику слід перевіряти стан сервера, його завантаженість і вибирати найбільш здібний.

При балансуванні часто виникає проблема зберігання сесій, адже сесія доступна тільки на тому сервері, який створив її. Це слід враховувати в алгоритмі перенаправлення запиту або додаток повинен зберігати сесії на окремому сервері або в базі даних.

Для реалізації програмного балансувальника є багато рішень, наприклад: nginx – вільний Web-сервер і проксі-сервер, який має версії для сімейства Unix-подібних операційних систем (FreeBSD, Linux та ін.) й Microsoft Windows. Основні функції – це HTTP-сервер для обслуговування статичних запитів, акселерування проксіювання з підтримкою кешування, акселерована підтримка FastCGI та memcached серверів, має простий розподіл навантаження і відмовостійкість, модульність. Apache HTTP-сервер – відкритий веб-сервер для UNIX-подібних, Microsoft Windows та інших операційних систем. На сьогодні є найуживанішим Web-сервером мережі Інтернет.

Більш простим рішенням є застосування проксі-серверу, наприклад, HAproxy. Для його конфігурування потрібно вказати IP-адреси веб-серверів. Налаштування балансування зводиться до запису в конфігурації проксі-сервера /etc/haproxy/haproxy.cfg – «mode http» режиму роботи проксі, який стосується обробки пакетів, «balance leastconn» – алгоритм

вибору сервера (перенаправляємо запит серверу з найменшою кількістю з'єднань). Параметр «cookie serv insert» застосовується для забезпечення роботи сесій. Таким чином клієнт завжди буде потрапляти на один сервер і мати доступ до даних сесії. Параметр «option httpclose» автоматично закривати з'єднання після завершення передачі даних. Запускається проксі сервер командою: sudo haproxy-f/etc/haproxy/haproxy.cfg. Конфігурація HAproxy звичайно сильно залежить від завдання, наприклад, для роботи із статичним контентом або роботи сайту без сесій немає потреби прив'язувати конкретний сервер до клієнта.

Тому не випадково все більшою популярністю користуються Web-сервери, наприклад, nginx, які можуть працювати автономно або в зв'язці з Apache в якості кешу для статичних запитів. Однак можна розглянути дещо інший підхід, пов'язаний з використанням HTTP-прискорювача Varnish, який дозволяє помітно розвантажити Web-сервери і зменшити загальний час відгуку (згідно електронному ресурсу: http://webperformance.ru/2011/06/03/varnish-speed-up/).

Varnish є безкоштовним рішенням для кешування як статичного, так і динамічного контенту. Працює він як балансувальник до будь-якого Web-сервера або сервера додатків і повністю орієнтований на високу продуктивність, багатопоточність та максимально ефективне використання можливостей операційних систем сімейства Linux.

Альтернативою до визначених підходів виступають так звані Content Delievery Network (CDN) – зовнішні сервіси, що забезпечують доступність контенту користувачам. Перевага очевидна – немає необхідності організовувати власну інфраструктуру для вирішення цього завдання, але з'являється інша додаткова стаття витрат.

Таким чином, для рішення завдань балансування навантаження на транспортний портал оптимальним вибором є система Varnish, що забезпечує як балансування, так й кеш даних. Поставивши між користувачем і Web-сервером прозорий проксі-сервер, можна надавати користувачу дані з кеша проксі (який може бути як в оперативній пам'яті, так і дисковим), не доводячи запити навіть до HTTP-серверів. У більшості випадків цей підхід актуальний тільки для статичного контента, в основному різних форм медіа-даних: зображень, відео і тому подібного. Це дозволяє Web-серверам зосередитися тільки на роботі з самими сторінками.

У разі здійснення заходів оптимізації додаток все одно може не впоратись з навантаженням. У такому випадку рішенням проблеми, очевидно, може послужити рознесення його по декільком вузлам, з метою збільшення загальної продуктивності додатка за рахунок збільшення доступних ресурсів.

Більшість Web-додатків апріорі є розподіленими, оскільки в їх архітектурі можна виділити мінімум три шари: Web-сервер, бізнес-логіка (додаток), дані (база даних, статика). Кожен із цих шарів можна масштабувати. Тому, якщо у системі додаток і база даних розміщені на одному хості – першим кроком, безсумнівно, має стати рознесення їх по різних хостам (за матеріалами: http://habrahabr.ru/post/113992/, http://www.phphighload.com/2012/10/mysql-scaling-strategies.html).

Починаючи масштабування системи, варто визначити, який із шарів є «вузьким місцем» – тобто працює повільніше за решту системи. Для цього можна скористатися простими утилітами типу top (htop) для оцінки завантаження процесора/пам'яті та df, iostat – для оцінки продуктивності дискової підсистеми серверу. Однак, бажано виділити окремий хост, з емуляцією промислового навантаження (наприклад, за допомогою Apache JMeter), на якому можна буде профілювати роботу додатка. Для виявлення вузьких запитів до бази даних можна скористатися утилітами на основі логів з журналу сервера, що знаходиться у промислової експлуатації.

Більшість залежить від архітектури Web-додатку, але найбільш вірогідними кандидатами на «вузьке місце» в загальному випадку є бази даних та код. Якщо додаток працює з великим об'ємом даних користувача, то «вузьким місцем», відповідно, швидше за все буде зберігання статики.

Часто вузьким місцем в сучасних додатках є бази даних. Проблем з базами даних діляться, як правило, на два класи: продуктивність і необхідність зберігання великої кількості даних. Знизити навантаження на базу даних можна за умови поширення її на декілька вузлів. При цьому гостро постає проблема синхронізації між вузлами.

Висновки

Результат полягає в усуненні протиріч наявності загальних вартісних обмежень та потрібних комп'ютер-них ресурсів раціональної організації клієнт-серверної технології перевізного процесу. Своєрідною вільною новою нішею відповідних розробок є синергетичний, інформаційний розвиток ринку транспортних послуг.

Список літератури:

[1] Aleksiyev O. Development of automotive computer systems based on the virtualization of transportation processes management / O. Aleksiyev, V. Aleksiyev, D. Klets, V. Khabarov et al. // Eastern-European Journal of Enterprise Technologies. – 2017. – Vol. 6, No 3 (90). – P. 14 – 25. – DOI: 10.15587/1729-4061.2017.116351.

[2] Алексієв О. П. Інформаційний розвиток порталу віртуального управління процесами транспортного обслуговування / О. П. Алексієв, В. О. Алексієв, // Інформаційні технології: проблеми та перспективи: монографія [Текст]. – Х.: Вид-во: Рожко С. Г., 2017. – Розд. 2. – С. 32 – 47. URI: http://www.repository.hneu.edu.ua/jspui/handle/ 123456789/16051.

Надійшла до редколегії 15.01.2024