УДК 004.4

І. Ф. ШАЄХОВА, Ю. О. ОЛІЙНИК
Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського»

ПІДХІД ДО РОЗРОБКИ АРХІТЕКТУРИ ГЕТЕРОГЕННОЇ МУЛЬТИКОМП'ЮТЕРНОЇ СИСТЕМИ ПЛАНУВАННЯ ЗАДАЧ

При розробці архітектури розподіленої системи планування задач розробники неминуче стикаються з проблемою забезпечення роботи системи як єдиного цілого. Хоча гетерогенні мультикомп'ютерні системи, які будуються на основі вузлів з мережевими операційними системами, і мають перевагу в гнучкості масштабування системи, їх розробники стикаються з проблемами при забезпеченні безпеки та прозорості функціонування системи. Правильний архітектурний підхід при розробці таких систем допоможе нівелювати їхні недоліки, а також забезпечити відмовостійкість і консистентність даних. При проектуванні розподілених систем планування задач потрібно зважати на проблему збалансування навантаження на вузлах, які беруть участь у виконанні задач, тому була розглянута література, присвячена цій тематиці. У статті також проаналізовано іншу сучасну літературу, присвячену розробці розподілених систем в цілому. Проектування розподіленої системи розглянуто з позиції аналізу моделей взаємодії, брокерів повідомлень, різних типів архітектури та алгоритмів консенсусу. Виділено різні моделі взаємодії в розподілених системах: виклик відділених процедур (RPC), віддалений виклик методів (RMI), обмін повідомленнями (MOM), використання потоків (streaming) і визначено найбільш гнучкі з них для побудови розподіленої системи планування задач. В статті зроблено порівняння різних брокерів (RabbitMQ, Apache Kafka, ZeroMQ) для маршрутизації повідомлень в межах розподіленої системи з акцентом на надійність доставки повідомлень. Крім цього, розглянуті такі архітектури як: грід та кластер, узагальнено їх ключові особливості і представлено характеристики розробленої системи. Також, охарактеризовані методи забезпечення узгодженості даних в розподілених системах Paxos і Raft. Запропоновано використовувати failover модель для спрощення розробки і запуску системи. Крім цього, представлено BPMN схему запуску задачі в межах розробленої розподіленої системи, а також схему архітектури розробленої системи. В статті представлені результати проведеного експерименту з визначення можливості масштабування розробленої системи, а також проаналізовано особливості роботи мови програмування Golang, якою написана розроблена система.

Ключові слова: гетерогенні мультикомп'ютерні системи, архітектура програмного забезпечення, відмовостійкість, модель взаємодії, алгоритми консенсусу.

И. Ф. ШАЕХОВА, Ю. О. ОЛЕЙНИК
Национальный технический университет Украины «Киевский политехнический институт им. Игоря Сикорского»

ПОДХОД К РАЗРАБОТКЕ АРХИТЕКТУРЫ ГЕТЕРОГЕННОЙ МУЛЬТИКОМПЬЮТЕРНОЙ СИСТЕМЫ ПЛАНИРОВАНИЯ ЗАДАЧ

При разработке архитектуры распределенной системы планирования задач разработчики неизбежно сталкиваются с проблемой обеспечения работы системы как единого целого. Несмотря на преимущество гетерогенных мультикомпьютерных систем на основе узлов с сетевыми операционными системами в гибкости масштабирования, разработчики сталкиваются с проблемами при обеспечении безопасности и прозрачности функционирования системы. Правильный архитектурный подход при разработке таких систем поможет нивелировать их недостатки, а также обеспечить отказоустойчивость и согласованность данных. При проектировании распределенных систем планирования задач нужно учитывать проблему сбалансирования нагрузки на узлах, которые принимают участие в выполнении задач, поэтому была рассмотрена литература, посвященная этой тематике. В статье также проанализированы другие современные источники, посвященные разработке распределенных систем в целом. Проектирование распределенной системы рассмотрено с позиции анализа моделей взаимодействия, брокеров сообщений, различных типов архитектуры и алгоритмов консенсуса. Выделены различные модели взаимодействия в распределенных системах: вызов отделенных процедур (RPC), удаленный вызов методов (RMI), обмен сообщениями (MOM), использование потоков (streaming) и определены наиболее гибкие из них

для построения распределенной системы планирования задач. В статье сделано сравнение различных брокеров (RabbitMQ, Apache Kafka, ZeroMQ) для маршрутизации сообщений в пределах распределенной системы с акцентом на надежность доставки сообщений. Кроме этого, рассмотрены такие архитектуры как: грид и кластер, обобщены их ключевые особенности и представлены характеристики разработанной системы. Также, охарактеризованы методы обеспечения согласованности данных в распределенных системах Paxos и Raft. Предложено использовать failover модель для упрощения разработки и запуска системы. Кроме этого, представлены BPMN схема запуска задачи в рамках разработанной распределенной системы, а также схема архитектуры разработанной системы. В статье представлены результаты проведенного эксперимента по определению возможности масштабирования разработанной системы, а также проанализированы особенности работы языка программирования Golang, на котором написана разработанная система.

Ключевые слова: гетерогенные мультикомпьютерные системы, архитектура программного обеспечения, отказоустойчивость, модель взаимодействия, алгоритмы консенсуса.

I. F. SHAIEKHOVA, Y.O. OLIINYK
Igor Sikorsky Kyiv Polytechnic Institute

APPROACH TO DEVELOPING ARCHITECTURE OF A HETEROGENEOUS MULTICOMPUTER TASK PLANNING SYSTEM

When developing the architecture of a distributed task scheduling system, developers inevitably face the problem of ensuring the operation of the system as a whole unit. Despite the scalability advantage of heterogeneous multicomputer systems based on nodes with network operating systems, developers face challenges in ensuring the security and transparency of the system. The correct architectural approach in the development of such systems will help to level their shortcomings, as well as ensure fault tolerance and data consistency. When designing distributed task planning systems, it is necessary to consider the problem of load balancing at the nodes that participate in the execution of tasks, so the literature on this topic was considered. The article also analyzes other modern literature on the development of distributed systems in general. The design of a distributed system is considered from the standpoint of analysis of interaction models, message brokers, different types of architecture and consensus algorithms. Different models of interaction in distributed systems are also considered – remote procedure call (RPC), remote method invocation (RMI), message-oriented middleware (MOM), streams (streaming) - and the most flexible for building a distributed task scheduling system is identified. The article compares different brokers (RabbitMQ, Apache Kafka, ZeroMQ) for routing messages within a distributed system with an emphasis on the reliability of message delivery. In addition, such architectures as grid and cluster are considered, their key features are generalized, and the characteristics of the developed system are presented. Methods for ensuring data consistency in distributed systems Paxos and Raft are also described. The failover model is presented to simplify system development and startup. In addition, the BPMN scheme of running the task within the developed distributed system, as well as the scheme of the architecture of the developed system are presented. The article presents the results of an experiment to determine the scalability of the developed system, as well as analyzes the features of the Golang programming language on which the developed system is written.

Keywords: heterogeneous multicomputer systems, software architecture, fault tolerance, interaction model, consensus algorithms.

Постановка проблеми

При розробці архітектури системи розподіленого планування задач необхідно визначитися з типом архітектури, моделлю взаємодії, типом зв’язку між вузлами, способом синхронізації, реплікацією та способами забезпечення узгодженості даних. При цьому потрібно мати на увазі, що операції в системі повинні бути атомарними і сама система повинна бути відмовостійкою. Для побудови такої системи необхідно вирішити наступні задачі:

1. Визначитися з типом архітектури розроблюваної системи з урахуванням призначення системи.
2. Обрати ефективну модель взаємодії в системі.

3. Визначитися з типом зв'язку між вузлами в розроблюваній системі.
4. Забезпечити певний погоджений рівень відмовостійкості за рахунок реплікації даних та алгоритмів вибору лідера.
5. Протестувати розроблену систему і її здатність до масштабування.

Аналіз останніх досліджень і публікацій

Сутність розподілених систем розкрито в книзі [1] авторами Ендрю Таненбаумом та Маартеном ван Стеєном, викладачами університету Вріє в Амстердамі. В книзі доволі скрупульозно розглянуто різні аспекти роботи розподілених систем – архітектуру, зв'язок між процесами, синхронізацію, реплікацію, безпеку. Серед розподілених систем розрізняють мультипроцесорні та мультикомп'ютерні системи, останні в свою чергу можна поділити на гетеро- та гомогенні [1]. Особливості планування задач в гомо- та гетерогенних розподілених системах описані в дослідженні [2]. Автори виділяють статичні та динамічні алгоритми. При цьому, статичні алгоритми означають, що час виконання задач, витрати на комунікацію відомі наперед, а динамічні означають, що рішення приймаються в режимі реального часу таким чином, щоб мінімізувати час виконання задач та витрати на комунікацію. До статичних відносять Rate Monotonic Algorithm (пріоритет надається в залежності від частоти виконання), Earliest Deadline First Algorithm (пріоритет надається в залежності від часу дедлайну). Прикладом динамічного алгоритму є Maximum Urgency First Algorithm, який використовує комбінацію статичних та динамічних властивостей задач.

Особливості архітектури розподілених систем описані в книзі [1] і статтях [3-6]. В джерелах проаналізовано особливості побудови класичної клієнт-серверної архітектури та багатошарових архітектур (three-tier, multi-tier), в яких за рахунок додавання проміжних шарів вдається досягти безстановості (statelessness). Окремо виділено однорангову архітектуру.

Різні підходи до проектування розподілених систем висвітлено в статтях [7-8], а також в стандарті OGSA [4]. Розрізняють такі форми проектування розподілених систем – грід, кластер, хмарна система.

В розподілених системах імплементовані різні моделі взаємодії між вузлами. Враховуючи вимогу до забезпечення прозорості, такі моделі взаємодії мають створювати враження роботи системи як єдиного цілого. Особливості моделі RPC (модель виклику віддалених процедур), моделі виклику віддалених об'єктів описані в книзі [1], статтях [9-10]. Синхронність цих моделей виправдовує доцільність взаємодії з використанням сокетів або черг повідомлень. Особливості побудови зв'язку між вузлами в розподіленій системі описано в статті [5] від компанії Microsoft, у якій проаналізовано використання синхронних (HTTP) та асинхронних (AMQP) протоколів під час реалізації системи. Особливості застосування брокера RabbitMQ описані в серії статей від розробників [11], а також в статтях [12-14]. Порівняльний аналіз роботи RabbitMQ і Kafka здійснено в статті [13], відповідно до якого RabbitMQ має перевагу в забезпеченні надійності при передачі повідомлень. Також розглянуто роботу сучасного фреймворку для обміну повідомленнями ZeroMQ [15]. Робота з потоками даних описана в статтях [16-17].

Забезпеченню відмовостійкості в розподілених системах присвячено багато статей. Дослідження [18] виділяє такі типи відмов: мережеві неполадки, неполадки фізичних ресурсів (пам'ять, CPU), неполадки в процесах (баги в програмному забезпеченні), неполадки при використанні спільних ресурсів. Ці відмови можуть призвести як до повного, так і до часткового припинення роботи системи. Оскільки в системі навіть при

найбільш ретельному плануванні її архітектури можуть трапитися відмови, ключовою проблемою є забезпечення відмовостійкості системи. Відмовостійкість забезпечується як реактивними, так і проактивними діями. До реактивних заходів належать такі методи, як: створення checkpoint, міграція задач при необхідності на інші фізичні машини, реплікація, повторення спроби виконання задачі. Проактивні заходи передбачають настання відмов. До відповідних методів можна віднести, наприклад, метод збалансування навантаження, коли програмне забезпечення постійно аналізує навантаження на пам'ять та CPU, і у випадку перевищення лімітів задачі переносяться на іншу машину. В дослідженні [6] висвітлено проблеми відмовостійкості в контексті мікросервісної архітектури і запропоновано архітектурний підхід до побудови таких систем.

Для досягнення відмовостійкості застосовуються алгоритми консенсусу. Алгоритм консенсусу Paxos був запропонований Леслі Лемпортом. Його сутність розкрита в статті [19]. В дослідженні [20] запропоновано удосконалення існуючого алгоритму з використанням batching та pipeline.

Алгоритм Paxos був складним та подекуди потребував вдосконалення, тому було розроблено більш простий алгоритм консенсусу Raft. Особливості роботи алгоритму описані в дисертації Дієго Онгаро [21]. Розробка алгоритму була спрямована на подолання недоліків Paxos: слабкої форми лідерства, необхідності в об'єднанні значень, вибраних в процесі одноступеневого Paxos, в один лог для роботи в режимі Multi-Paxos і таке інше. За рахунок відокремлення виборів лідера, реплікації логу та безпеки вдалося побудувати більш прозорий алгоритм.

Один із способів досягнення синхронізації в розподіленій системі пов'язаний з використанням алгоритму розподіленого блокування. Один із таких алгоритмів, Redlock з використанням Redis, описаний в роботі [22].

Мета дослідження

Метою дослідження є узагальнення підходу до побудови розподілених систем планування сервісних задач.

Викладення основного матеріалу дослідження

З точки зору архітектури програмного забезпечення доцільніше розробляти розподілену систему планування сервісних задач як клієнт-серверну архітектуру з моделлю взаємодії МОМ. Це твердження можна обґрунтувати тим, що у випадку планування сервісних задач виникає потреба в централізованому управлінні задачами, що робить беззмістовним використання однорангової архітектури. Сервер не зберігатиме сесії клієнтів (stateless).

Проаналізовано різні моделі взаємодії в розподілених системах (табл. 1). Зважаючи на результати аналізу, модель МОМ є найбільш оптимальним варіантом для розподілених систем планування сервісних задач, оскільки асинхронна взаємодія, гарантована доставка повідомлень і гнучка маршрутизація дозволяють провести налаштування системи таким чином, щоб забезпечити відмовостійкість та зменшити затримки.

Таблиця 1

Порівняння моделей взаємодії

Моделі взаємодії	**RPC (Remote Procedure Call)**	**RMI (Remove Method Invocation)**	**MOM (Message Oriented Middleware)**	**Потоки (streams)**
Синхронність/ Асинхронність	синхронна	синхронна	асинхронна	асинхронна
Сутність	Виклик віддалених процедур на віддаленому вузлі так, неначе це локальні процедури.	Робота через локальну заглушку, яка є проксі для віддаленого об'єкта.	Взаємодія відправників та одержувачів через черги повідомлень.	Обмін потоками даних, які не є самостійними елементами, в режимі реального часу.
Переваги	1.Однаковий інтерфейс виклику процедур. 2.Стандартний формат повідомлень. 3. Спільний сервіс для обробки запитів-повідомлень, прихований «внизу» віддаленої процедури.	1.Об'єктно-орієнтований стиль програмування. 2.Передача віддалених об'єктів за посиланням, яке формується з адреси віддаленого вузла, порту і ідентифікатора віддаленого об'єкта. 3.Стандартний формат повідомлень.	1.Гнучка маршрутизація. 2.Послаблення зв'язку в системі за рахунок появи додаткового шару між відправниками і отримувачами. 3.Гарантія доставки повідомлень. 4.Доступність і механізми збалансування навантаження.	1. Обробка даних в режимі реального часу. 2.Можливість налаштування гарантій доставки повідомлення.
Недоліки	1.Передача параметрів за посиланням неможлива (процеси не мають спільної пам'яті). 2.Сумісність форматів обох систем по відношенню до порядку байтів, кодування символів.	1.Неможливо розрізнити локальні і віддалені об'єкти. 2. Проблеми з блокуванням віддалених об'єктів.	1.Необхідність нового елемента в системі, який в перспективі може стати її вузьким місцем.	1.Специфічне призначення розподілених систем з використанням потоків даних. 2.Складність обробки все більшої кількості потоків даних з різних джерел.

Продовження таблиці 1

Приклади систем	DCE RPC	DCE, Java RMI	Системи, які використовують Apache Kafka, RabbitMQ, IBM MQ (IBM MQSehes)	Системи, які використовують Apache S4, Apache Storm, Apache Samza, Spark Streaming, Twitter's Heron, Google Millwheel, Azure Stream Analytics, Amazon Kinesis.

Таблиця 1 складена на основі джерел [1, 9-10, 12, 16-17].

Оскільки брокер є центральним елементом при застосуванні моделі взаємодії МОМ, було проаналізовано роботу різних брокерів для визначення найбільш надійного брокера для систем планування сервісних задач (табл. 2). Було визначено, що найбільш гнучким брокером для побудови транспортної інфраструктури є фреймворк ZeroMQ, однак він потребує значних зусиль з боку розробників для забезпечення гарантії доставки, персистентності повідомлень. Найбільш стабільним варіантом є застосування брокера RabbitMQ, який є повноцінним самостійним елементом системи, що забезпечує надійну доставку повідомлень і гнучку маршрутизацію.

Таблиця 2

Порівняння брокерів повідомлення

Назва брокера	**RabbitMQ**	**Apache Kafka**	**ZeroMQ**
Мова програмування, на якій написаний брокер	Erlang	Scala	C++
Формат повідомлень	Бінарний (краще для стискання даних)	байти	Бінарний
Протокол	AMQP	TCP	ZMTP (побудований над TCP)
Гарантії доставки повідомлень	At-least-once, most-once	Всі	Відсутня гарантія доставки, існує лише гарантія атомарної доставки multipart-повідомлення

Продовження таблиці 2

Персистентність повідомлень	Налаштовується при створенні черги (оперативна пам'ять/диск)	Повідомлення пишуться в файлову систему	Не підтримується. Відповідальність за підтримання персистентності повідомлень лежить на кінцевому користувачу.
Модель доставки повідомлень	Push/Pull	Pull	Push/Pull
Модель отримання повідомлень	P2P, Pub/Sub	P2P, Pub/Sub	P2P, Pub/Sub
Маршрутизація	Є спеціальні об'єкти для маршрутизації повідомлень – exchange.	Немає посередників, повідомлень напряму йдуть до брокера	Використовується Router-сокет.
Масштабування	Додавання великої кількості клієнтів може вплинути на роботу брокера, оскільки статуси клієнтів зберігаються в пам'яті.	Додавання клієнтів, які зчитують інформацію, (reader) не впливає на роботу брокера.	Підтримується
Відмовостійкість	Підтримує реплікацію повідомлень по різних вузлах.	Підтримує реплікацію повідомлень по різних вузлах.	Оскільки це фреймворк, залежить від імплементації.
Сфера застосування	Фінансові і банківські сервіси	Агрегування логів, аналітика великих даних	Розподілені системи, в яких втрата повідомлень не є критичною

Таблиця 2 складена на основі джерел [11-17].

Для узагальнення характеристик розробленої системи та визначення типу архітектури було представлено основні характеристики кластерної та грід систем в порівнянні з розробленою системою (табл. 3). Різні призначення цих систем обумовили їх архітектуру, зокрема, оскільки грід система виникла для здійснення масштабних наукових обчислень з використанням неспеціалізованих комп'ютерних ресурсів, то в основі архітектури грід системи лежить техніка «виділення циклів пам'яті» (cycle scavenging) на віддалених гетерогенних вузлах. Оскільки розроблена система призначена для запуску і планування сервісних задач з різним рівнем споживання ресурсів, то акцент в розробці

даної системи полягає в тому, щоб ефективно розподілити задачі між вузлами. Задачі розподілятимуться відповідно до зайнятості вузлів.

Таблиця 3

Порівняння типів архітектури

Тип архітектури	**Кластер**	**Грід**	**Розроблена система**
Суть	Суміщені вузли, зв'язані між собою високошвидкісною мережею. Гомогенна розподілена система.	Поєднання віддалених вузлів, ресурсів, кластерів між собою. Гетерогенна розподілена система. Використання техніки «виділення цикла пам'яті» (cycle scavenging)	Поєднання віддалених вузлів, ресурсів. Гетерогенна розподілена система.
Функції	Агрегування обчислювальних потужностей. Забезпечення надійного доступу у випадку часткових збоїв. Збалансування навантаження.	Використання віддалених комп'ютерних ресурсів. Агрегування обчислювальних ресурсів невикористовуваних вузлів. Високий рівень надійності та здатність до масштабування без значних зусиль.	Ефективне використання віддалених комп'ютерних ресурсів в залежності від заданих статичних конфігурацій. Високий рівень надійності та здатність до масштабування.
Призначення	Підходить для систем, які виконують невеликі незалежні обчислення.	Підходить для систем, які спільно виконують задачі, що потребують великих обчислювальних потужностей (здебільшого для наукових обчислень).	Підходить для комерційних систем, які не є монолітною системою, а представляють набір мікросервісів, яким потрібна координація при розподілі задач.
Недоліки	Висока вартість. Простої більшості вузлів в кластері. Великі навантаження, незважаючи на обчислювальні потужності кластера, можуть призвести до його падіння та збоїв.	Необхідність чекати результати від усіх вузлів, які беруть участь в обчисленнях. Проблеми з безпекою. Відсутність мережевих збоїв є критичною для правильної роботи гріда.	Використання статичної конфігурації не дозволяє побачити реальне навантаження на віддалених вузлах і може спричиняти неефективний розподіл задач.

Таблиця 2 складена на основі джерел [4, 7-8]

Важливим етапом побудови розподіленої системи є вибір алгоритмів, методів та технік, які б забезпечили надійність розподіленої системи у випадку часткового падіння вузлів системи. Було проаналізовано роботу систем Paxos і Raft, які є повноцінними системами, спрямованими на забезпечення узгодженості даних в кластерах і потребують

значних ресурсів при розгортанні (табл. 4). В результаті запропоновано використовувати модель Failover, яка потребує мінімум ресурсів і є більш простою у використанні.

Таблиця 4

Порівняння різних методів забезпечення відмовостійкості

Забезпечення відмовостійкості	**Алгоритм консенсусу Paxos**	**Алгоритм консенсусу Raft**	**Модель Failover з використанням Redlock алгоритму**
Модель	Використовується для реплікації машини стану. Передбачає існування вузлів з наступними ролями – ті, хто висувають пропозицію з запитом, ті, хто схвалюють пропозицію, ті, хто виконують запит.	Використовуються для реплікації машини стану. Передбачає існування лідера і послідовника. Фази вибору лідера і реплікації розділені.	Використовується для обмеження доступу інших вузлів до ресурсу (в даному випадку до стартового методу запуску обробника задач). У випадку падіння сервера, один із серверів, який очікує звільнення блокування, запустить обробника задач і заблокує інші сервери.
Реплікація	Лідер відправляє запит послідовникам з вимогою приєднати запит під певним індексом до лога.	Лідер відправляє запит послідовникам з вимогою приєднати запит під певним індексом до лога.	Немає, система не підтримує копіювання машини стану на інші вузли. Система містить числення перевірки, щоб уникнути повторного виконання запитів у випадку падіння вузла.
Узгодження даних	Якщо у послідовника і лідера є відмінності в записах, лідер переписує записи послідовника.	Якщо у послідовника і лідера є відмінності в записах, лідер переписує записи послідовника.	Узгодження даних не потрібно, оскільки не підтримується машина станів.
Переваги	Забезпечує прийняти пропозиції з найвищим номером для узгодження даних між вузлами.	Простіший алгоритм прийняття пропозиції. Алгоритм більш зрозумілий ніж Paxos, тому що його робота як за нормальних умов, так у за умови неузгодженості чи падіння вузлів є прописаною в роботах, присвячених алгоритму [14]. Просте управління кластером.	Проста реалізація. Виникає менша кількість непростих ситуацій, які потребують вирішення. Потрібно менше ресурсів. Не обов'язково мати 2*N + 1 вузлів в якості майстра та потенційних майстрів.

Обмеження	Потрібен головний вузол, що висуває пропозицію, інакше між вузлами може виникнути гонка пропозицій.	Підтримує вибір тільки одного лідера.	У разі використання мінімальної кількості вузлів, які є потенційними майстрами, збільшується ризик падіння всієї системи.

Таблиця 2 складена на основі джерел [19, 21-22].

Виділено наступні характеристики архітектури розробленої системи:

- Гетерогенна розподілена система.
- Використання моделі Failover.
- Використання статичної конфігурації для розподілу задач.
- Побудова транспортної системи на основі обміну повідомленнями (MOM) з використанням брокера RabbitMQ.
- Вибір лідера здійснюється за допомогою алгоритму Redlock, що побудований на основі Redis (в перспективі планується використовувати алгоритм консенсусу Raft).
- Дані, отримані під час обробки задач, зберігаються в базі даних Postgres.
- Майстер запускає тільки ті задачі, які були отримані в повідомленні від вузлів. Сторонні користувачі не можуть відправити фальшиві повідомлення для підривання роботи системи, оскільки необхідно знати користувача і пароль для встановлення зв'язку з транспортною системою.

Задачі будуть рухатися по наступному циклу. Майстер запускатиме задачі на виконання залежно від розкладу, обраного користувачами, або користувач може запустити задачу на одноразове виконання. Задачу можна буде скасувати і перезапустити. Залежно від конфігурації можна буде налаштувати максимальну кількість задач, які одночасно виконуватимуться на робочому вузлі, дедлайн для задачі (рис. 1).

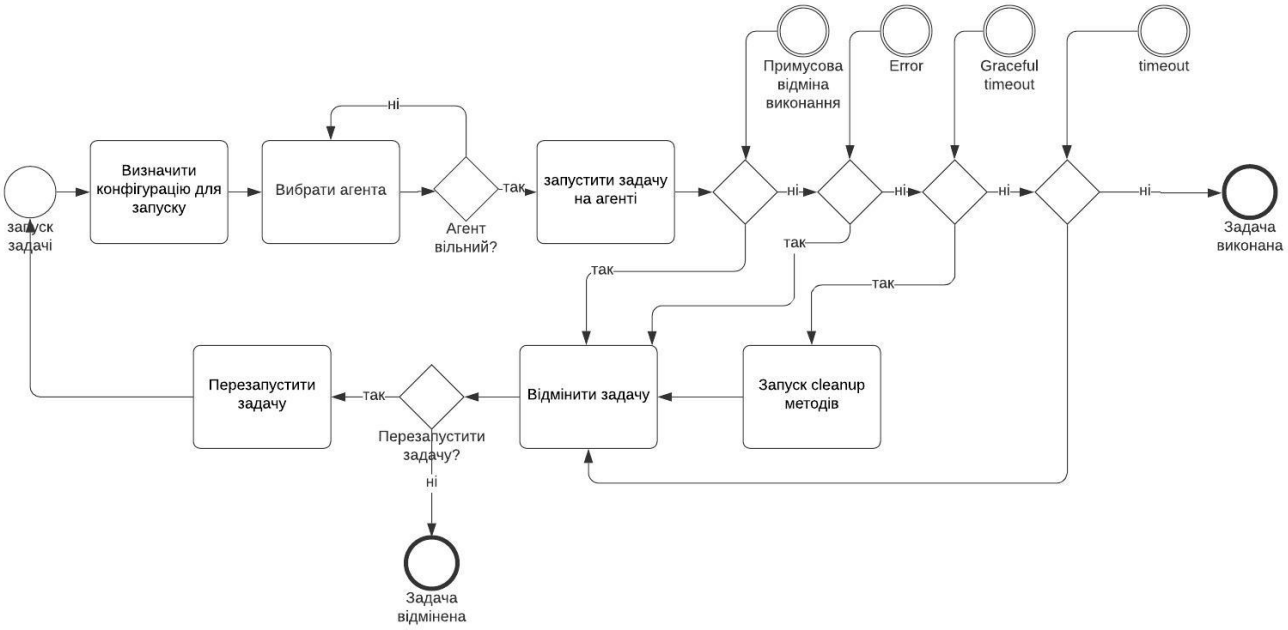

Рис 1. BPMN діаграма запуску задачі в розроблюваній системі

Опис експерименту.

Склад розподіленої системи (рис. 2) – два майстер-вузли та 3 робочих вузли з такими характеристиками:

Майстер:

- 16 x Intel(R) Xeon(R) E-2288G CPU @ 3.70GHz (1 Socket);
- 128 GB DDR4 ECC 2,666 MHz.

Робочий вузол:

- AMD Ryzen 7 4800U;
- 128 GB DDR4 2,666 MHz.

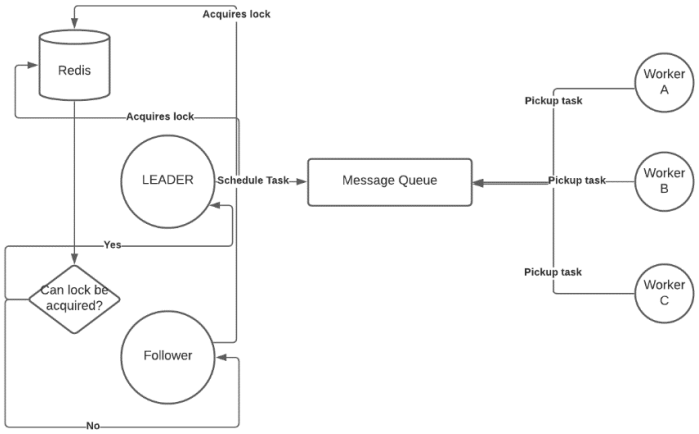

Рис 2. Схема архітектури системи в ході експерименту

В ході експерименту генерується навантаження в 10 – 50 тисяч задач на один робочий вузол. Затримки при виконанні задачі в основному складаються з мережевої затримки і затримки при створенні нової goroutine – потоку в Golang. Для імітації роботи реальної задачі була створена імплементація задачі на робочих вузлах, яка зупиняє потік на 10 секунд. Результат експерименту наведено на рис. 3.

Рис 3. Середня затримка виконання задачі на вузлі (µс)

В ході експерименту виявлено, що в результаті збільшення кількості конкурентних задач на одному вузлі витрачається більше часу на запуск кожної наступної goroutine. Goroutine – це приклад використання спрощеного паралелізму. В Golang є планувальник, за допомогою якого планується запуск goroutine (задач). У кожної goroutine є стек, мінімальний розмір якого - 2048 байтів, максимальний розмір залежить від архітектури і становить - 1 ГБ для 64-розрядних систем, або 250 МБ для 32-розрядних систем. Важливою частиною конфігурації є налаштування GOMAXPROCS, яке визначає максимальну кількість паралельних потоків операційної системи для виконання коду. За замовчуванням цей параметр дорівнює кількості доступних логічних процесорів [23-24]. Також, для покращення роботи системи з високим навантаження бажано використовувати пули goroutine [25]. В розробленій системі використовуються поки що необмежена кількість goroutine, в майбутньому планується використати пул.

В Go потоки використовують стеки скромних розмірів. Під час виконання програми стек goroutine збільшується і зменшується залежно від потреби. Таким чином goroutine живуть в одному адресному просторі і за рахунок цього не витрачають стільки ресурсів як повноцінні потоки. Таким чином, використовуючи переваги таких потоків, вдалося добитися значної оптимізації роботи розподіленої системи.

Висновки

У роботі представлено підхід до проектування розподілених систем планування задач, які є доволі розповсюдженими в корпоративному середовищі. Проаналізовано різні моделі взаємодії. В результаті прийнято рішення використовувати асинхронну модель, яка базується на обміні повідомленнями. Розглянуто переваги і недоліки роботи різних брокерів. Прийнято рішення використовувати брокер RabbitMQ, оскільки він забезпечує надійну доставку повідомлень. Було проаналізовано такі типи архітектури систем як: грід та кластер, і в результаті прийнято рішення побудувати свою архітектуру, призначення якої полягатиме в плануванні сервісних задач корпоративними користувачами. Також, було розглянути способи забезпечення відмовостійкості, що передбачає застосування алгоритмів консенсусу. Оскільки ці алгоритми є доволі громіздкими надбудовами поверх існуючої системи, було прийнято рішення використовувати механізм розподіленого блокування. Таким чином, визначено набір ключових характеристик для розробленої архітектури. Проведено експеримент з розробленою системою для визначення можливості розширення розподіленої системи в сторону збільшення кількості виконуваних задач. Відповідно до результатів експерименту було виявлено, що відбувається незначне збільшення часу виконання задач, і їх виконання можна ще більш оптимізувати за рахунок використання пулу goroutine.

Список використаної літератури

1. Стеен М., Таненбаум Е. Распределенные системы. Принципы и парадигмы: уч. пособ. Санкт-Петербург: Питер, 2003. 877 с.
2. Vucha M. A Case Study: Task Scheduling Methodologies for High Speed Computing Systems. *International Journal of Embedded Systems and Applications.* 2015. URL: https://www.researchgate.net/publication/270593930_A_Case_Study_Task_Scheduling_Methodologies_for_High_Speed_Computing_Systems.

3. Shakirat Haroon-Sulyman. Client-Server Model. *IOSR Journal of Computer Engineering.* 2014. № 16. С. 57-71. URL: https://www.researchgate.net/publication/271295146_Client-Server_Model.
4. Berry D., Djaoui A., Grimshaw A. та ін. The Open Grid Services Architecture, Version 1.5. 2006. URL: https://ogf.org/documents/GFD.80.pdf.
5. Communication in a microservice architecture. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture.
6. Baboia M., Iftene A., Gîfu D. Dynamic Microservices to Create Scalable and Fault Tolerance Architecture. *Procedia Computer Science*. 2019. № 159. URL: https://www.sciencedirect.com/science/article/pii/S187705091931467X.
7. Chee Shin Yeo, Rajkumar Buyya, Hossein Pourreza та ін. Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers *Handbook of Nature-Inspired and Innovative Computing.* 2006. С. 521-551. URL: https://www.researchgate.net/publication/226533607_Cluster_Computing_High-Performance_High-Availability_and_High-Throughput_Processing_on_a_Network_of_Computers.
8. Barkallah H. Evolution of the Distributed Computing Paradigms: Brief Road Map. *IJCDS Journal.* № 6(5). 2017. URL: https://www.researchgate.net/publication/319352828_Evolution_of_the_Distributed_Computing_Paradigms_a_Brief_Road_Map.
9. Java RMI. URL: https://docs.oracle.com/javase/7/docs/platform/rmi/spec/rmiTOC.html.
10. Lecture 3: RPC and RMI. URL: https://cseweb.ucsd.edu/classes/sp16/cse291-e/applications/ln/lecture3.html.
11. RabbitMQ Tutorials. URL: https://www.rabbitmq.com/getstarted.html.
12. Curry E. Message-Oriented Middleware. *Middleware for Communications.* 2005. С. 1-28. URL: https://www.researchgate.net/publication/220035284_Message-Oriented_Middleware.
13. Vineet J, Xia Liu. A Survey of Distributed Message Broker Queues. 2017. URL: https://www.researchgate.net/publication/315764651_A_Survey_of_Distributed_Message_Broker_Queues.
14. Guo Fu., Yanfeng Zhang, Ge Yu. A Fair Comparison of Message Queuing Systems. 2020.URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9303425.
15. ZeroMQ Documentation URL: https://zeromq.org/.
16. Kamburugamuve S. Survey of Distributed Stream Processing. 2016. URL: https://www.researchgate.net/publication/299411481_Survey_of_Distributed_Stream_Processing
17. Isah H., Abughofa T., Mahfuz S., Ajerla D. A Survey of Distributed Data Stream Processing Frameworks *IEEE Access.* № 7. 2019. URL: https://www.researchgate.net/publication/336430459_A_Survey_of_Distributed_Data_Stream_Processing_Frameworks.
18. Kumari P., Kaur P. A survey of fault tolerance in cloud. *Journal of King Saud University Computer and Information Sciences*. 2018. URL: https://www.sciencedirect.com/science/article/pii/S1319157818306438.
19. Lamport L. Paxos Made Simple. 2001. URL: https://www.microsoft.com/en-us/research/uploads/prod/2016/12/paxos-simple-Copy.pdf.
20. Santos N., Schiper A. Optimizing Paxos with batching and pipelining. *Theoretical Computer Science.* № 496. 2013.

URL: https://www.sciencedirect.com/science/article/pii/S0304397512009097.
21. Ongaro D. Consensus: bridging theory and practice : дис. докт. / Stanford University. 2014.
22. Kleppmann M. How to do distributed locking.
URL: https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html.
23. Golang. Frequently Asked Questions. URL: https://golang.org/doc/faq
24. Что такое горутины и каков их размер?
URL: https://habr.com/ru/company/otus/blog/527748/
25. Ants. URL: https://github.com/panjf2000/ants
26. Розроблена розподілена система управління задачами. URL: https://github.com/dev-renshark/demo_taskdealer

References

1. Steen, M., & Tanenbaum, E. (2003). Raspredelennyie sistemyi. Printsipyi i paradigmyi: uch. posob. Sankt-Peterburg: Piter.
2. Vucha, M. (2015). A Case Study: Task Scheduling Methodologies for High Speed Computing Systems. *International Journal of Embedded Systems and Applications.* URL: https://www.researchgate.net/publication/270593930_A_Case_Study_Task_Scheduling_Methodologies_for_High_Speed_Computing_Systems.
3. Shakirat Haroon-Sulyman. (2014). Client-Server Model. *IOSR Journal of Computer Engineering*. **16**, 57-71. URL: https://www.researchgate.net/publication/271295146_Client-Server_Model.
4. Berry, D., Djaoui, A., & Grimshaw, A. (2006). The Open Grid Services Architecture, Version 1.5. URL: https://ogf.org/documents/GFD.80.pdf .
5. Communication in a microservice architecture. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture.
6. Baboia, M., Iftene, A., & Gîfu D. (2019) Dynamic Microservices to Create Scalable and Fault Tolerance Architecture. *Procedia Computer Science*. **159**. URL: https://www.sciencedirect.com/science/article/pii/S187705091931467X.
7. Chee Shin Yeo, Rajkumar Buyya, & Hossein Pourreza. (2006). Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers *Handbook of Nature-Inspired and Innovative Computing.* 521-551. URL: https://www.researchgate.net/publication/226533607_Cluster_Computing_High-Performance_High-Availability_and_High-Throughput_Processing_on_a_Network_of_Computers.
8. Barkallah, H. (2017). Evolution of the Distributed Computing Paradigms: Brief Road Map. *IJCDS Journal.* **6**(5). URL: https://www.researchgate.net/publication/319352828_Evolution_of_the_Distributed_Computing_Paradigms_a_Brief_Road_Map.
9. Java RMI. URL: https://docs.oracle.com/javase/7/docs/platform/rmi/spec/rmiTOC.html.
10. Lecture 3: RPC and RMI. URL: https://cseweb.ucsd.edu/classes/sp16/cse291-e/applications/ln/lecture3.html.
11. RabbitMQ Tutorials. URL: https://www.rabbitmq.com/getstarted.html.
12. Curry, E. (2005). Message-Oriented Middleware. *Middleware for Communications.* 1-28. URL: https://www.researchgate.net/publication/220035284_Message-Oriented_Middleware.

13. Vineet, J, & Xia Liu. (2017). A Survey of Distributed Message Broker Queues. URL: https://www.researchgate.net/publication/315764651_A_Survey_of_Distributed_Message_Broker_Queues.
14. Guo Fu., Yanfeng Zhang, & Ge Yu. (2020). A Fair Comparison of Message Queuing Systems. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9303425.
15. ZeroMQ Documentation URL: https://zeromq.org/.
16. Kamburugamuve, S. (2016). Survey of Distributed Stream Processing. URL: https://www.researchgate.net/publication/299411481_Survey_of_Distributed_Stream_Processing
17. Isah, H., Abughofa, T., Mahfuz, S., & Ajerla, D. (2019). A Survey of Distributed Data Stream Processing Frameworks *IEEE Access.* **7**. URL: https://www.researchgate.net/publication/336430459_A_Survey_of_Distributed_Data_Stream_Processing_Frameworks.
18. Kumari, P., & Kaur, P. (2018). A survey of fault tolerance in cloud. *Journal of King Saud University Computer and Information Sciences*. URL: https://www.sciencedirect.com/science/article/pii/S1319157818306438.
19. Lamport, L. (2001). Paxos Made Simple. URL: https://www.microsoft.com/en-us/research/uploads/prod/2016/12/paxos-simple-Copy.pdf.
20. Santos, N., & Schiper, A. (2013). Optimizing Paxos with batching and pipelining. *Theoretical Computer Science.* **496**. URL: https://www.sciencedirect.com/science/article/pii/S0304397512009097.
21. Ongaro, D. (2014). Consensus: bridging theory and practice / Stanford University.
22. Kleppmann, M. How to do distributed locking. URL: https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html.
23. Golang. Frequently Asked Questions. URL: https://golang.org/doc/faq
24. Chto takoe gorutinyi i kakov ih razmer? URL: https://habr.com/ru/company/otus/blog/527748/
25. Ants. URL: https://github.com/panjf2000/ants
26. Rozroblena rozpodilena systema upravlinnia zadachamy. URL: https://github.com/dev-renshark/demo_taskdealer

Шаєхова Ірина Фанусівна – магістрант кафедри АСОІУ Національного технічного університету України «КПІ ім. Ігоря Сікорського», email: irynashaiekhova@gmail.com, ORCID: 0000-0001-6098-3297.

Олійник Юрій Олександрович – старший викладач кафедри автоматизованих систем обробки інформації і управління, Національний технічний університет України «КПІ ім. Ігоря Сікорського», e-mail: oliyura@gmail.com, ORCID: 0000-0002-7408-4927.