***Sudnik Yuriy Alexandrovich**, PhD (Eng), Professor*
(e-mail: sudnikya@mail.ru)
Russian State Agrarian University – Moscow Agricultural Academy named after K.A., Moscow, Russian Federation

MONITORING OF SOIL FERTILITY BY ITS ELECTRICAL INDICATORS

***Abstract.** This article presents the requirements for developing a device for monitoring the content of soil fertility components, including humus in the field. An electronic scheme for measuring the electrical resistance of soil at constant current is proposed, manufactured and investigated.*

***Keywords:** humus, soil, methods, measurement, electrical activity.*

РЕАЛИЗАЦИЯ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛЕНИЙ НА ЯЗЫКЕ PYTHON С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ DOCKER *

***Кузнецов Максим Сергеевич**, студент*
***Андрианов Игорь Александрович**, к.т.н., доцент*
Вологодский государственный университет

В статье описывается способ реализации распределённых вычислений на языке Python с применением технологии Docker-контейнеров. Разработанная система позволяет запускать на каждом узле набор изолированных процессов с собственными настройками среды выполнения, балансировать нагрузку между узлами кластера, автоматически перезапускать задания в случае сбоев. Приводится описание и результаты вычислительных экспериментов.

Ключевые слова: распределённые вычисления, кластер, Docker, Python.

Распределённые вычисления представляют собой способ решения трудоемких вычислительных задач с использованием нескольких компьютеров, чаще всего объединённых в параллельную вычислительную систему [1].

Одно из первых упоминаний распределенных вычислений относится к 1973 году. Сотрудники научно-исследовательского центра Xerox PARC Джон Шох и Джон Хапп написали программу, которая рассылала себя по другим работающими компьютерам через локальную сеть PARC.

Впоследствии, в связи с развитием и ростом количества персональных компьютеров, распределённые вычисления стали использоваться всё более и более широко. Так, в конце 1980-х годов Арьен Ленстра и Марк Менес написали программу для факторизации длинных чисел. Она рассылала задания на компьютеры участников по электронной почте и таким же образом принимала ответы.

Ещё одним значимым событием было создание проекта SETI@Home (Search for Extra-Terrestrial Intelligence at Home) для поиска внеземного разума путём анализа данных с радиотелескопов, в том числе на домашних компьютерах участников. Данный проект был запущен в 1999 году и оста-

*Работа частично поддержана грантом РФФИ 18-47-350001 р_а

новлен в 2020-м. Эта распределенная система была построена на платформе BOINC, созданной в университете Беркли.

В дальнейшем разработки по созданию различных распределённых систем активно продолжались, и в настоящее время они применяются в самых различных областях. В частности, распределённые вычисления широко используются для математических задач. Типичным примером является факторизация чисел (разложение их на произведение простых множителей).

Ещё одной важной областью применения распределённых вычислений является обработка больших данных с использованием методов машинного обучения и Data Mining. В качестве языка программирования для этой цели в последние годы на лидирующие позиции выходит язык Python. По состоянию на март 2020 года, согласно рейтингу TIOBE, Python находится на третьем месте, хотя ещё в 2015 году занимал лишь седьмое.

Одной из известных проблем языка Python является относительно низкая производительность в сравнении с компилируемыми языками – такими как C++. Данный недостаток является дополнительным поводом применять параллельное и распределённое программирование в процессе разработки.

Также во второй половине второй декады 21 века стали набирать популярность такие технологии, как контейнеризация и микросервисы. Благодаря программному обеспечению Docker и изолированным пространствам исполнения стало возможным относительно легко запускать на одном сервере несколько приложений полностью изолированно друг от друга и с собственными настройками среды выполнения. В настоящее время технология Docker используется всё более широко и для самых разных целей, в том числе при разработке программного обеспечения: она позволяет проводить различные эксперименты с зависимостями ПО, не “засоряя” основную систему.

Основной инструмент кластеризации для Docker носит название Docker Swarm. Он позволяет объединять узлы в единое кластерное пространство и распространять контейнеры по этому кластеру. Представленная в данной статье разработка основывается на изолированных Docker-контейнерах – “исполнителях”.

Каждый исполнитель представляет из себя асинхронный TCP-сервер, работающий следующим образом. На первом шаге он десериализует полученные данные и получает объект задания. Каждый такой объект содержит код функции, которую необходимо выполнить, а также значения её входных данных. Исполнитель запускает заданную функцию, используя определённую стратегию исполнения, после чего сериализует результат её работы и возвращает полученный результат [2].

Под упомянутым выше термином «стратегия исполнения» понимается в первую очередь способ организации параллельной работы. В языке Python доступны два основных варианта – многопроцессный и многопоточный.

При использовании стратегии множественных процессов каждая функция запускается в отдельном процессе. В случае же многопоточной стратегии функции, соответственно, запускаются в отдельных потоках.

Здесь стоит учитывать, что потоки в Python по умолчанию являются не совсем полноценными – в отличие, например, от потоков в языке C++. Дело в том, что интерпретатор Python (CPython) написан на языке C с использованием некоторых библиотек, не являющихся потокобезопасными. Из-за этого используемый в CPython механизм GIL (Global Interpreter Lock) не позволяет потокам исполняться по-настоящему параллельно даже на многоядерных процессорах. Вместо этого интерпретатор постоянно переключается между потоками с интервалом примерно 5 миллисекунд. В связи с этим, на современных компьютерах в большинстве случаев более эффективным вариантом будет использование стратегии множественных процессов.

Разрабатываемая библиотека имеет следующую архитектурную особенность. В ней применён механизм Docker Swarm, позволяющий организовать единый вход и выполнять балансировку нагрузки между исполнителями. Наличие единой точки входа для клиентов позволяет разработчикам не заботиться о настройке множественных подключений и конфигурации каждого исполнителя в отдельности.

Кроме того, Docker Swarm позволяет после создания контейнера реплицировать его на доступные узлы. Также он позволяет легко увеличивать количество репликаций исполнителя, способен балансировать нагрузку между исполнителями, автоматически перезапускает исполнителей при их отключении, автоматически подключает компьютеры, которые ранее были отключены от кластера.

Дополнительно отметим, что в представленной библиотеке оставлена возможность работы и без использования Docker Swarm, так как в некоторых случаях его применение будет избыточным. Примером может быть ситуация, когда у нас имеется ограниченное количество “слабых” машин, имеющих одноядерный процессор и ограниченный объём оперативной памяти.

Для проверки производительности данной разработки были выбраны две следующие задачи: вычисление факториала (цикл с умножением на каждой итерации) и расчёт простых чисел, используя алгоритм “решето Эратосфена”.

Тестирование проводилось на двух объединённых в локальную сеть компьютерах (6 ядер на первом и 2 – на втором) с суммарным объёмом ОЗУ 20 ГБ, под управлением ОС Linux.

Были выбраны четыре вычислительные стратегии: с использованием множества процессов, многопоточная, асинхронная и последовательная. Стратегия с использованием множества процессов подразумевала запуск каждого экземпляра функции как отдельного процесса. Многопоточная стратегия предполагала запуск каждой функции в отдельном потоке (пом-

ня о механизме GIL). Асинхронная стратегия – это стратегия с применением событийного цикла, где каждая функция создается как отдельная сопрограмма (“корутина”) и опрашивается в таком цикле. Последовательная стратегия – это стратегия запуска, при которой функции вызываются по очереди.

Результаты выполнения представлены в следующей таблице.

Стратегия/Задача	Расчет факториала	Вычисление простых чисел
Множественные процессы (8 процессов)	7.3525 с	39.3731 с
Многопоточный (8 потоков)	54.3255 с	42.0415 с
Последовательная	43.4656 с	41.4426 с
Асинхронная	43.5361 с	43.9102 с

Исходя из данных результатов, можно сделать вывод, что запуск функций в отдельных процессах позволяет максимально загрузить процессор и достичь наибольшей скорости выполнения задач. Многопоточный вариант уступает даже последовательному исполнению из-за особенностей реализации многопоточности в языке Python. Асинхронный вариант показал результаты, не слишком сильно отличающиеся от последовательного. Но, в целом, это хороший показатель для математических задач, так как в этих функциях отсутствует, например, ожидание ответа базы данных. Также благодаря асинхронному серверу в основе каждого исполнителя, можно запускать эти задачи совместно, не блокируя запуск других функций.

В дальнейшем развитии данной разработки планируется добавить поддержку виртуальных окружений, что позволит использовать сторонние модули и фреймворки Python, а также избежать конфликта версий одного модуля с разными функциями.

Одним из возможных применений разработки является организация распределённой проверки решений при проведении соревнований по программированию [3].

Список литературы

1. Распределённые вычисления [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Распределённые_вычисления

2. Кузнецов, М.С. Разработка механизма распараллеливания кода на языке Python с использованием Docker-контейнеров / М.С. Кузнецов – материалы межрегиональной научной конференции «XIII ежегодная научная сессия аспирантов и молодых учёных», 2019. – С. 165-168.

3. Андрианов, И.А. Анализ и моделирование производительности автоматизированной системы проведения олимпиад по информатике / И.А. Андрианов, Н.О. Менухова – Бизнес. Наука. Образование: проблемы, перспективы, стратегии: материалы Российской научно-практической конференции с международным участием. – Вологда, Вологодский институт бизнеса, 2015. – С. 426-429.