Заключение. В результате исследования был разработан микроконтроллерный регулятор скорости вращения вентилятора персонального компьютера, в зависимости от температуры. Данная система может быть применена в компьютере для охлаждения важных частей, а также в других устройствах, где требуется регулирование скорости вращения кулера в зависимости от температуры.

Список цитируемых источников

1. Компьютерная грамотность для начинающих [Электронный ресурс]. — Режим доступа: https://www.pc-school.ru/sistema-oxlazhdeniya-kompyutera/ . — Дата доступа: 29.04.2020.

2. *Баранов, В. Н.* Применение микроконтроллеров AVR / В. Н. Баранов. — М. : Додэка-XXI, 2004. — 288 с.

УДК 004.75

М. С. Кузнецов

Федеральное государственное бюджетное образовательное учреждение высшего образования «Вологодский государственный университет», Вологда, Российская Федерация

РЕАЛИЗАЦИЯ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛЕНИЙ НА ЯЗЫКЕ PYTHON С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ DOCKER

Введение. 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 работающим 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 остановлен 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, 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 XXI 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 программного обеспечения, не «засоряя» основную систему.

Основная часть. 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 оперативного запоминающего устройства 20 Гб под управлением операционной системы 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 (помня 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 таблице 1.

Таблица 1 — Результаты выполнения

Стратегия/задача	Расчет факториала	Вычисление простых чисел
Множественные процессы (8 процессов)	7,3525	39,3731
Многопоточный (8 потоков)	54,3255	42,0415
Последовательная	43,4656	41,4426
Асинхронная	43,5361	43,9102

Заключение. 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.

Список цитируемых источников

1. Распределённые вычисления [Электронный ресурс]. — Режим доступа: https://ru.wikipedia.org/wiki/Распределённые_вычисления/ . — Дата доступа: 03.05.2020.

2. *Кузнецов, М. С.* Разработка механизма распараллеливания кода на языке Python с использованием Docker-контейнеров / М. С. Кузнецов // XIII Ежегодная научная сессия аспирантов и молодых учёных : материалы межрегион. науч. конф., 2019. — С. 165—168.

УДК 004.021

А. А. Мишин
Федеральное государственное бюджетное образовательное учреждение высшего образования «Оренбургский государственный университет», Оренбург, Российская Федерация

ПРИОРИТИЗАЦИЯ В ПРОЦЕССЕ СОПРОВОЖДЕНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Введение. В имеющихся стандартах жизненного цикла программного обеспечения (далее — ПО), таких как ГОСТ 34.601-90 и ISO/IEC 12207:2008, существует этап сопровождения ПО. Это одна из фаз, следующая за фазой передачи ПО в эксплуатацию.

Сопровождение является одним из основных этапов жизненного цикла ПО, который занимает от 50 % затрат на разработку и внедрение программного средства (ПС). От эффективности работ на этапе поддержки и сопровождения зависит непрерывность бизнес-процессов и сохранность информации, необходимой компании для выполнения стратегических, тактических, оперативных задач [1].

Существуют ПС для процесса сопровождения ПО, например, такие как Freshdesk и vsDesk.

Freshdesk — это сервис, который дает возможность ускорить поддержку клиентов, автоматизировать некоторые трудоёмкие задачи, в том числе с помощью распределения заявок. Это решение помогает компаниям, заинтересованным в снижении человеко-часов, которые хотят сконцентрировать своих сотрудников только на самых важных задачах.

vsDesk — это Service Desk (Help desk) для автоматизации работы как внутреннего ИТ-отдела, так и внешней службы технической поддержки. Используя сервисный подход и SLA (Service Level Agreement), система позволяет упорядочить и автоматизировать процессы управления потоком заявок пользователей (клиентов). Однако как сформировать хороший SLA? Этот вопрос остается открытым. В предлагаемой статье рассматривается процесс сопровождения и определения приоритета заявок на обслуживание ПО.

Основная часть. В ходе сопровождения ПО возникает необходимость вносить изменения, исправлять недоработки и дефекты, представляющие собой ошибки в коде или недоработки в интерфейсе, добавлять новый функционал и многое другое с целью повысить качество ПО.

Сопровождения ПО можно представить в виде черного ящика, который показан на рисунке 1.

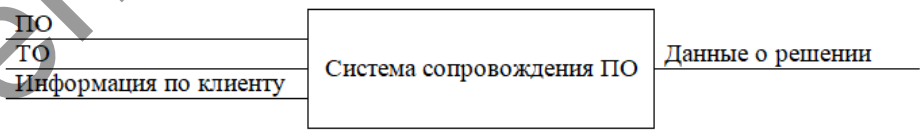

Рисунок 1 — Черный ящик системы сопровождения ПО

Приведем функциональное моделирование рассматриваемого процесса. Сопровождение ПО можно разделить на несколько частей: принятие (создания) заявки, приоритизация и исполнение заявки. В виде SADT-модели они представлены на рисунках 2 и 3.

ГОСТ Р ИСО/МЭК 14764-2002 «Информационная технология. Сопровождение программных средств» представляет собой рекомендации по управлению (или выполнению) процессом сопровождения. Стандарт определяет использование (привлечение) процесса сопровождения в процессах заказа и эксплуатации [2].