# Сравнительный анализ решений на базе .NET для итерационного выполнения задач в распределенных отказоустойчивых системах

Т.Э. Зейналлы

***Аннотация*—В работе представлены результаты сравнительного анализа решений на базе .NET для итерационного выполнения задач в распределенных отказоустойчивых системах. Для сравнения рассматриваются одноранговые распределенные системы, способные итерационно выполнять задачи по расписанию. Среди сравниваемых решений были отобраны quartz, hangfire и recurrent worker service. Исследуется зависимость измеряемых показателей, таких как точность выполнения итераций, минимальный интервал выполнения и потребление ресурсов, от внешних дестабилизирующих факторов, включая инфраструктурные (выходы узлов из строя) и сетевые (пропускная способность, задержки, потери пакетов). Методологической основой работы является эксперимент. Эксперимент проводится на двух серверах, на одном из которых размещены узлы приложения, а на другом — узлы синхронно реплицированного хранилища. Проводится ряд испытаний, где на узлах приложения контролируемо эмулируются сетевые и инфраструктурные дестабилизирующие факторы. В ходе эксперимента производятся замеры аппаратных показателей и сбор телеметрии с приложений. По собранным данным проводится анализ и расчет статистических параметров, после чего результаты представляются в виде таблиц и диаграмм. Результаты текущей работы могут быть применены при проектировании и оптимизации распределенных систем, требующих высокой степени отказоустойчивости и эффективности.**

***Ключевые слова*—распределенные отказоустойчивые системы, итерационное выполнение задач, одноранговые распределенные системы, точность выполнения итераций.**

## I. Введение

В настоящее время в корпоративном сегменте широко распространены распределенные системы, использующие в своей основе итеративное выполнение задач. Логика итеративного выполнения и распределения задач реализуется в виде подключаемых библиотек или пакетов, которые распространяются через системы управления пакетами для различных языков программирования. Эти инструменты решают проблему итерационного или отложенного выполнения произвольной полезной нагрузки (задачи). Под полезной нагрузкой понимается алгоритм, направленный на выполнение соответствующего бизнес-процесса. К примерам полезных нагрузок можно отнести отправку уведомлений, построение аналитических отчетов, ETL (Extract-Transform-Load) процессы и т.д.

Статья получена 10 апреля 2024.

Зейналлы Теймур Эйюб оглы, Московский политехнический университет, z.teymur.e@gmail.com

Важным аспектом в системах является обеспечение их отказоустойчивости. Для обеспечения отказоустойчивости в микросервисных архитектурах широко применяется горизонтальное масштабирование. Этот подход обеспечения отказоустойчивости делает систему распределенной [1,2]. При проектировании распределенных систем возникает вопрос о координации выполнения полезных нагрузок, т.к. в ряде случаев их параллельное выполнение недопустимо. Примером может послужить процесс отправки уведомлений: если два разных узла одновременно отправят одно и то же уведомление, то пользователь получит два одинаковых уведомления. Для координации в распределенных системах используются различные механизмы и архитектурные решения, основанные на очередях, распределенных блокировщиках, системах управления базами данных и др [3]. Решения, существующие на текущий момент, учитывают этот аспект и предлагают встроенные механизмы на базе различных подходов.

В работе [4] были предложены алгоритмы координации и обеспечения отказоустойчивости систем, итерационно выполняющих задачи в условиях одноранговости узлов. Для этой конфигурации позже в работе [5] был описан механизм приоритезации. Также была оценена эффективность предложенных решений [6].

В рамках данного исследования проводится сравнительный анализ решений, предложенных в работе [4] с аналогичными решениями на базе платформы .NET. Выбор .NET обусловлен широким набором встроенных и подключаемых библиотек для разработки распределенных и отказоустойчивых систем, включая продвинутую поддержку многопоточности и асинхронности [7]. Сравнение решений, выполненных на различных языках программирования, не всегда представляется корректным ввиду фундаментальных различий в особенностях языков: способы управления памятью, сборка мусора, многопоточность, системы типов и т.д. Эти факторы оказывают влияние на производительность алгоритмов в решениях, делая их сравнение нерелевантным без учета контекста конкретного языка. Для сравнения решений вне контекста конкретного языка обычно применяется теоретический метод, сравнивающий сложность алгоритмов, например, в О-нотации. Такое сравнение дает информацию о производительности алгоритмов по скорости и по памяти. Однако, оно не отражает эффективность алгоритмов в контексте распределенных систем. Функционирование рассматриваемых решений

зависит от взаимодействия с разными персистентными хранилищами данных, алгоритмы работы которых могут быть закрытыми или содержать дополнительные механизмы кеширования, обработки транзакций и согласованности данных [8]. Помимо этого, в реальных условиях параллельной и распределенной обработки, важны такие факторы как синхронизация данных между узлами и управление конкурентным доступом. С учетом всех этих факторов, оценка рассматриваемых решений, опираясь исключительно на теоретический анализ, окажется недостаточно полной. В данной работе основное внимание уделяется экспериментальной оценке выбранных решений.

При подходе к выбору решений для анализа в данной работе, основное внимание уделяется отказоустойчивости. Отказоустойчивость в данном контексте означает, что все компоненты решения, включая хранилища данных, должны обеспечивать непрерывную работу даже при возникновении сбоев на отдельных узлах. Таким образом, если работа решения основана на хранилище с асинхронной репликацией, то при выходе из строя основного узла, возможна потеря важных данных о выполнении задач [9]. Следовательно, такие решения не рассматриваются в рамках данного исследования.

Среди научных работ, описывающих системы для итерационного выполнения задач, можно отметить следующие. Bryk, P., Malawski, M., Juve, G., & Deelman, E. [10] исследовали алгоритмы, учитывающие персистентность данных для планирования ансамблей рабочих процессов в облаках, что позволило улучшить управление ресурсами и эффективность выполнения задач. Garg, R., Mittal, M., & Son, L. H. [11] сконцентрировались на надежности и энергоэффективности при планировании процессов в облачных средах. Их подход предлагает методы уменьшения энергопотребления при сохранении высокого уровня выполнения задач. Sharma, R., Nitin, N., AlShehri, M. A. R., & Dahiya, D. [12] разработали приоритетный алгоритм планирования EDF-RM для индивидуальных реальных задач в распределенных системах. Алгоритм учитывает различные приоритеты задач, что оптимизирует распределение ресурсов и повышает эффективность выполнения задач в распределенных системах. Li, C., Tang, J., Ma, T., Yang, X., Luo, Y. [13] предложили алгоритм планирования заданий в облачных системах на основе балансировки нагрузки. Этот алгоритм направлен на оптимальное распределение заданий между ресурсами, что способствует более эффективной работе распределенных облачных систем.

В научной литературе описаны теоретические принципы распределения задач, но часто без конкретных реализаций на языках программирования. Реализация этих теоретических моделей для целей сравнения представляет собой трудоемкий процесс. В связи с этим, в качестве базы для поиска решений была выбрана платформа GitHub. В процессе анализа было изучено множество репозиториев, большинство из которых было отклонено по причинам, таким как использование принципов, не соответствующих исходным требованиям, наличие отказоустойчивого хранилища данных, или отсутствие функциональности для итерационного выполнения задач. В результате были отобраны три подходящих решения: quartz [14], hangfire [15] и recurrent worker service (rws) [16]. Решения quartz и hangfire используют систему управления базами данных MSSQL для координации задач, а rws основан на использовании etcd. Алгоритмы работы rws описываются в работе [4]. Quartz и hangfire являются наиболее популярными решениями в среде dotnet. Алгоритмы rws поддерживают выполнение задач в двух режимах: базовом (далее будет обозначаться rws_v1) и добавленном позднее режиме с захватом блокировки на несколько итераций (rws_v2). В данной работе представлено сравнение обоих режимов.

## II. Постановка цели и задач

Целью работы является определение наиболее эффективных по производительности и надежности решений на базе .NET для итерационного выполнения задач в распределенных отказоустойчивых системах.

Исходя из того, что основным назначением рассматриваемых решений является итеративное выполнение полезных нагрузок, задача работы заключается в оценке зависимости измеряемых показателей от ряда внешних факторов. К показателям относятся: точность выполнения итераций, минимальный интервал выполнения, потребляемые ресурсы для всех компонентов. К внешним факторам относятся: инфраструктурные дестабилизирующие факторы (выход узлов из строя) и сетевые дестабилизирующие факторы (низкая пропускная способность, сетевые задержки, потери пакетов).

## III. Материалы и методы

### *A. Методология*

Методологической основой текущего исследования является эксперимент. Он включает в себя подготовку и развертывание экспериментальной среды с тестируемым решением, выполнение испытаний, проведение измерений и последующий анализ данных с применением статистических методов.

### *B. Условия проведения эксперимента*

Экспериментальная среда состоит из двух серверов. На первом сервере разворачиваются узлы приложения тестируемого решения. На втором сервере располагаются узлы хранилищ, требуемые для тестируемого решения. Каждый сервер разворачивался с нуля, чтобы исключить влияние сторонних процессов на ход эксперимента.

Сервера имеют следующие аппаратно-программные характеристики. Сервер приложений работает под управлением ОС Linux, версия ядра 5.15-146-1, оснащен 4-ядерным процессором Intel (R) Core (TM) i5-6200U CPU @ 2.30GHz и 12 гигабайтами оперативной памяти с частотой 3200 MHz. Сервер хранилищ, в свою очередь, оснащен 8-ядерным процессором 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz, SSD 1Tb Kingston NVMe M2, а также 16 гигабайтами оперативной памяти частотой 2133 MHz. Работает также под управлением ОС Linux, версия ядра 5.15-0-101. Приложения и хранилища разделены по разным серверам для обеспечения более эффективного распределения ресурсов и изоляции

хранилищ от тестируемых решений. Хранилища чувствительны к дисковым операциям ввода-вывода, поэтому разворачиваются на сервере с наиболее подходящим аппаратным обеспечением. На сервере, в зависимости от испытания, разворачивались хранилища etcd v3.5.12, mssql 2022 v16.0.

Во всех испытаниях, независимо от типа используемого хранилища, само хранилище разворачивается в отказоустойчивой конфигурации с тремя узлами в режиме синхронной репликации данных.

На каждом сервере установлен docker версии 24.0.7, в нем разворачиваются как приложения, так и хранилища. Docker позволяет изолировать узлы выполняющихся процессов (как процессов приложений, так и процессов хранилищ) [17]. Также он предлоставляет возможность создавать виртуальные подсети и эмулировать на них сетевые дестабилизирующие факторы. Docker позволяет переразворачивать все компоненты тестируемого решения с нуля перед каждым испытанием, чтобы исключить влияние испытаний друг на друга. Если этого не делать, то, в частности, хранилища могут аллоцировать память на одном испытании и не освободить ее перед следующим, в таком случае показатели памяти на следующем испытании будут завышены. Резюмируя, docker позволяет перед каждым испытанием разворачивать все компоненты тестируемого решения в требуемом количестве, и с заданными сетевыми дестабилизирующими факторами, согласно плану испытания.

На обоих серверах в docker разворачиватся контейнер с telegraf версии 1.19.1. Его основная функция состоит в измерении показателей использования аппаратных ресурсов каждым развернутым внутри docker компонентом тестируемого решения [18]. Информацию об использовании аппаратных ресурсов telegraf получает от docker. Полученная информация записывается в файл в виде метрик формата influx line protocol. Измеряются следующие показатели: утилизация процессора, использование ОЗУ, операции на сети, операции на файловой системе.

Каждое приложение, развернутое в docker на сервере приложений, также производит измерения на выполняемых операциях и записывает в файл в виде метрик формата influx line protocol. Детальное описание этих измерений будет изложено в разделе, посвященном проведению измерений.

На сервере приложений в docker разворачивается docker-tc версии 18.12. Функция docker-tc состоит в эмуляции дестабилизирующих факторов в сети для отдельных контейнеров приложения [19]. Данный инструмент способен контролируемо эмулировать в заданном диапазоне следующие факторы: потери, искажения, дублирования пакетов, ограничение пропускной способности, сетевые задержки.

Основным этапом в проведении эксперимента является выполнение испытаний. Список испытаний будет приведен в разделе плана испытаний. В текущем разделе необходимо отметить, что за выполнение испытаний отвечает отдельная утилита "runner", специально разработанная для выполнения плана всех испытаний для данного эксперимента. На вход она получает файл плана испытаний в формате JSON. В файле испытаний перечисляется список испытаний, где для каждого испытания описывается тестируемое решение, количество узлов, сетевые дестабилизирующие факторы, характер неожиданных отказов узлов, длительность испытания. Утилита в соответствии с планом испытания разворачивает инфраструктуру на обоих серверах. В ходе своей работы утилита фиксирует в файле временные метки начала и окончания каждого испытания. Утилита запускается на стороннем сервере, который имеет сетевой доступ до обоих серверов экспериментальной среды, чтобы исключить влияние утилиты на ход эксперимента.

После завершения всех испытаний собранные метрики с обоих серверов загружаются в InfluxDb версии 2.7.5 средствами утилиты influx cli версии 2.7.3. InfluxDb также развернута на стороннем сервере и не оказывает влияния на ход эксперимента. InfluxDb предоставляет широкий функционал для работы с временными рядами [20]. В частности, в текущей работе используется функционал подсчета некоторых статистических показателей, запросы данных за определенный диапазон и визуализация данных.

Для анализа собранных данных разработана специальная утилита "analyser", она используя информацию из лога утилиты "runner" для каждого испытания запрашивает данные из InfluxDb и отвечает за их обработку и формирование итоговых результатов. Обработка данных подробно описана в разделе методов обработки полученных данных.

Визуализация данных была представлена с использованием Grafana версии 10.3.3, используемая панель nline-plotlyjs-panel версии 1.6.5.

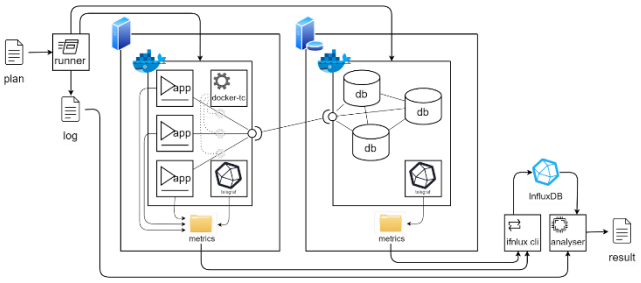


Рис. 1. Схема эксперимента

Все описанные действия подробно визуализированы на рисунке 1. Направленными стрелками на рисунке обозначены потоки данных. Это могут быть как управляющие потоки, так и потоки передачи данных. Прямыми ненаправленными линиями обозначены сетевые взаимодействия.

### *C. Методы измерений и установление их потребной точности*

Для проведения измерений на самих приложениях используются средства телеметрии. Эти средства широко распространены и имеют свою реализацию для большинства языков программирования. Они позволяют установить временную метку старта и длительность каждой операции в стеке вызовов. Измерения осуществляются с использованием наносекундных интервалов.

Тем не менее, в соответствии с технической документацией .Net, заявленная точность таких измерений составляет приблизительно 0.1 микросекунды [21]. Указанного уровня точности достаточно для выполнения задач, представленных в данном исследовании, поскольку анализируемые значения измеряются в миллисекундах.

В текущем исследовании тестируемые решения параллельно выполняют ряд полезных нагрузок. Каждая полезная нагрузка покрывается телеметрией. Это позволяет проводить измерения, которые включают в себя временную метку начала выполнения полезной нагрузки и ее длительность. Процесс измерения не оказывает существенного влияния на длительность выполнения полезных нагрузок ввиду того, что результаты измерения записываются в оперативную память, а затем фоновым потоком сохраняются в файл.

### *D. Реализация приложений*

Каждое приложение осуществляет выполнение трех типов полезных нагрузок. Принцип работы полезных нагрузок основан на механизме ожидания в течение заданного временного интервала. В контексте .NET такой подход позволяет освобождать поток выполнения обратно в пул потоков, минимизируя при этом влияние на использование аппаратных ресурсов.

Полезные нагрузки в приложении классифицируются на три типа: моментальная, быстрая и медленная. Моментальная нагрузка выполняется незамедлительно, быстрая нагрузка занимает от 3 до 8 секунд, а медленная — от 2 до 4 минут.

Моментальная нагрузка выполняется с интервалом, равным нулю. Быстрая и медленная полезные нагрузки имеют интервал в 1 секунду.

В текущей работе моментальная полезная нагрузка позволит измерить минимально возможный интервал выполнения. А остальные полезные нагрузки позволяют измерить точность выполнения операций.

### *E. План испытаний*

План испытаний включает серию испытаний с различными настройками сетевого окружения для каждого приложения, варьирование параметров пропускной способности, задержек и потерь пакетов, а также имитацию сбоев узлов. Длительность каждого испытания - 40 минут.

Первое испытание - контрольное. В нем количество узлов приложения равно 3 и не производится никаких эмуляций.

Далее идут испытания на пропускную способность. Пропускная способность уменьшается на всех 3 узлах следующим образом: 10 МБ/сек, 7 МБ/сек, 3 МБ/сек, 1 МБ/сек, 512 КБ/сек, 256 КБ/сек, 128 КБ/сек, 1 КБ/сек, 512 Б/сек. Также проводятся испытания с ограничением пропускной способности только на части узлов: по 1 КБ/сек и 512 Б/сек на одном и на двух узлах приложений.

Помимо этого, проводятся испытания с эмуляцией задержек на сети для трех узлов со следующими значениями задержек: 100 мс, 300 мс, 500 мс, 700 мс, 900 мс, 1100 мс. И по 1100 мс на одном и на двух узлах приложений.

Следующий ряд испытаний проводится с эмуляцией потерь пакетов на трех узлах с вероятностями: 10%, 30%, 50%. А также по 50% на одном и на двух узлах приложений.

Далее идут испытания на отказоустойчивость. Сценарий следующий. Разворачиваются 3 узла приложения. После этого, с некоторым интервалом времени будут последовательно отключаться несколько узлов на определенное время. Таким образом, предусмотрены испытания, где производится прерывание работы одного узла каждые 5 минут на 30 секунд и 1 минуту соответственно, а также испытания, в которых прерывается работа двух узлов с теми же интервалами, но на 30 и 50 секунд. Предусмотрены прерывания двух типов: штатное завершение и резкая остановка.

### *F. Методы обработки полученных данных*

В работе основным анализируемым показателем является точность выполнения операций. Она рассчитывается исходя из данных о начале выполнения операции и длительности ее выполнения, которые поставляются телеметрией. Для оценки точности, необходимо рассчитать временную разницу между ожидаемым временем выполнения и фактическим.

Пусть $t$ - время начала операции, $d$ - длительность, $p$ - интервал выполнения полезной нагрузки. В таком случае, ожидаемое время выполнения $Et_i = t_{i-1} + d_{i-1}$, при $d_{i-1} < p$. При $d_{i-1} \geq p$, $Et_i = t_{i-1} + p$. Таким образом, разница во времени вычисляется по формуле $\Delta t_i = t_i - Et_i$.

Имея результаты в виде массива значений $\Delta t$, необходимо оценить его статистические характеристики, такие как среднее $\mu = \frac{1}{n}\sum_{i=1}^{n} \Delta t_i$ и среднеквадратическое отклонение $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\Delta t_i - \mu)^2}$. Итоговые результаты будут представлены в виде $\mu \pm \sigma$.

При подготовке экспериментального стенда были приняты меры по минимизации влияния внешних факторов на итоговые данные. Тем ни менее, в полученных результатах не исключена возможность наличия выбросов, которые могут быть обусловлены внутренней работой алгоритмов проверяемых решений, хранилищ данных, сетевыми факторами или другими неучтенными параметрами. Прежде чем приступить к исключению выбросов, проводится оценка достаточности объема данных. Из-за эмуляции проблем на сети, в некоторых испытаниях распределение значений $\Delta t$ не всегда может соответствовать нормальному закону. Поэтому доверительный интервал в данной работе определяется неравенством Чебышева $P(|\Delta t - \mu| \geq k\sigma) \leq \frac{1}{k^2}$. Где при выбранном $k = 3$, доверительный интервал составляет $\mu - 3\sigma \leq \Delta t \leq \mu + 3\sigma$.

## IV. Результаты

На следующих графиках визуализирован показатель разницы между ожидаемым и фактическим временем выполнения (Δt) для моментальной, быстрой и медленной полезным нагрузкам. По оси ординат отмечен показатель Δt, по оси абсцисс перечислены испытания

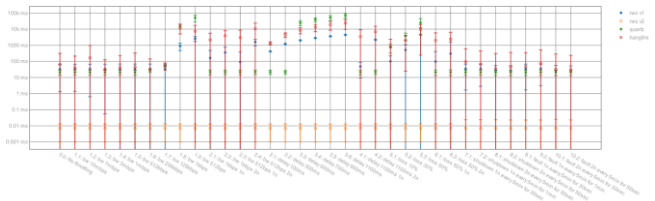

Рис. 2. Δt для моментальной полезной нагрузки

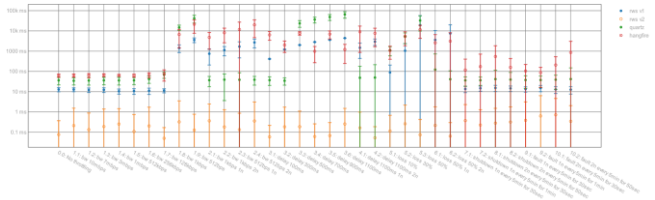

Рис. 3. Δt для быстрой полезной нагрузки

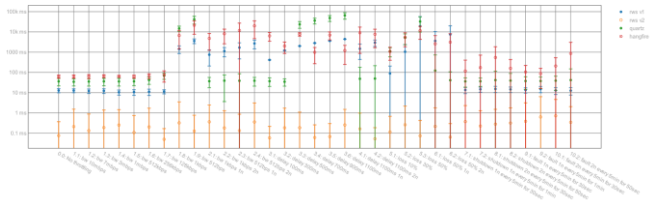

Рис. 4. Δt для медленной полезной нагрузки

У всех систем в испытаниях без эмуляции сетевых дестабилизирующих факторов показатели Δt находятся в пределах 100 миллисекунд. Ни одна система не показывает сильных отклонений по времени, которые можно было бы считать неприемлемым. Выделяется на данном фоне rws_v2 с показателем разницы между ожидаемым и фактическим временем выполнении (Δt) моментальной полезной нагрузки 0.007±0.006 мс, когда у quartz 20.764±6.735 мс, у hangfire 63.938±248.808 мс, и у rws_v1 33.441±32.112 мс. Если смотреть на количество выполненных итераций, то здесь также сильное отличие наблюдается у rws_v2 с показателем в 261 млн операций, в то время как у quartz 113 тыс, у hangfire 4 тыс, и у rws_v1 71 тыс. Что касается минимального интервала выполнения, то он равен Δt моментальной полезной нагрузки.

Полученные показатели Δt не имеют радикальных отличий и находятся в пределах 100 мс. Это позволяет сделать вывод о том, что метод замеров подходит для всех систем, и расчет ожидаемого времени корректно построен под каждую систему с учетом их конфигураций.

Для rws_v1 и hangfire в испытаниях с ограничением пропускной способности до 128 КБ/сек на всех узлах приложения, полученные показатели Δt совпадают с контрольными в пределах погрешности. Это связано с тем, что для этих систем показатели нагрузки на сеть составляют для hangfire 5.252±13.515 КБ/сек, а для rws_v1 13.981±1.163 КБ/сек. Для rws_v1 и hangfire систем резкий рост в показателях Δt наблюдается в испытаниях с пропускной способностью 1 КБ/сек и 512 Б/сек. Для rws_v2 показатель Δt остается в пределах контрольного значения во всех испытаниях, т.к. нагрузка на сеть для него составляет 0.425±0.062 КБ/сек. Для quartz, с его показателем нагрузки на сеть в 119.593±120.485 КБ/сек рост показателей начинается в испытаниях с ограничением пропускной способности в 256 КБ/сек. В испытании с пропускной способностью 256 КБ/сек рост Δt незначителен, однако в испытаниях с пропускной способностью 128 КБ/сек, 1 КБ/сек и 512 Б/сек наблюдается рост показателей пропорционально пропускной способности.

При ограничении пропускной способности до 1 КБ/сек и 512 Б/сек на одном и на двух узлах приложения, наблюдаются резкий рост показателей только у rws_v1 и hangfire. Показатели для quartz и rws_v2 находятся в пределах контрольных значений. В этих испытаниях всегда есть хотя бы один узел, на котором не эмулируются проблемы на сети.

В испытаниях с эмуляцией задержек на сети на всех узлах приложения наблюдается рост показателя Δt пропорционально эмулируемым задержкам, кроме quartz и rws_v2. Для rws_v2 во всех испытаниях показатели Δt находятся в пределах контрольных значений. Для quartz в пределах контрольного значения показатель Δt наблюдается только в испытаниях с задержками 100 мс и 300 мс, а при увеличении задержки, начиная с 500 мс показатель Δt увеличивается пропорционально.

При эмуляции задержек в 1100 мс на одном и на двух узлах приложения, для quartz и rws_v2 показатель Δt находится в пределах контрольного, как и в случае с испытаниями на пропускную способность. Для rws_v1 в испытании с задержками на одном узле показатель Δt отличается незначительно, но только для моментальной полезной нагрузки. Оставшиеся показатели Δt для rws_v1 и hangfire резко возрастают.

В испытаниях на эмуляцию потерь у rws_v1, quartz и hangfire наблюдается увеличение показателя Δt пропорционально эмулируемому проценту вероятности потери. Для rws_v2 Δt находится в пределах контрольного значения. Наблюдаемое поведение можно соотнести с поведением в испытаниях на сетевые задержки. Ввиду того, что приложения осуществляют сетевые взаимодействия через протокол TCP, потери пакетов приводят к сетевым задержкам. Это обусловлено спецификой работы протокола, а именно повторной отправкой пакета в случае, если принимающая сторона не подтвердила его получения.

В испытаниях с эмуляцией потерь пакетов с вероятностью 50% на одном и на двух узлах приложения наблюдаемые показатели также сопоставимы с аналогичными испытаниями на эмуляцию сетевых задержек. Наблюдается рост показателей Δt для rws_v1 и hangfire. Для quartz и rws_v2 показатель Δt в пределах контрольного значения.

В испытаниях на эмуляцию периодического отключения и выхода из строя узлов приложения показатели разнятся в зависимости от типа выполняемой полезной нагрузки. Для моментальной полезной нагрузки показатели Δt для всех систем сопоставимы с контрольными. Исходя из этого, можно сделать вывод, что все тестируемые системы справляются с выходом из строя или отключением узлов и продолжают обработку данных. Однако, в текущей работе из исходных данных исключаются выбросы, и выход из строя узла может расцениваться как выброс и не отразиться на полученных показателях. Для того, чтобы оценить влияние остановок и

выходов из строя узлов на итоговые показатели Δt, необходимо сравнить максимальное значение Δt до исключения выбросов с максимальным Δt в контрольном испытании для всех систем. Разница этих показателей составляет для hangfire 109.093 мс, для quartz 17137.524 мс, для rws_v2 5144.981 мс, и для rws_v1 4814.034 мс. Значение для hangfire наименьшее, для остальных полученное значение означает время восстановления после сбоя. Данное время для указанных систем конфигурируемо, для quartz это параметр "org.quartz.jobStore.clusterCheckinInterval" равный 15 сек, для rws_v2 это параметр HeartbeatExpirationTimeout равный 5 сек. Полученные показатели сопоставимы с параметрами из конфигурации.

В испытаниях с быстрой и медленной полезной нагрузкой, показатель Δt может отличаться от контрольного на длительность выполнения самой полезной нагрузки. Например, медленная полезная нагрузка может выполняться до 4 мин. Если остановка или выход из строя приложения произошло на 2-й минуте работы, то показатель Δt будет отличаться от контрольного на 2 минуты + время восстановления. Это поведение и наблюдается на графиках.

Таблица 1. Утилизация аппаратных ресурсов узлами приложений

| Система | Процессор, % | ОЗУ, МБ | Сеть, КБ/сек |
|---|---|---|---|
| hangfire | 1.464±3.807 | 58.948±3.714 | 5.252±13.515 |
| quartz | 48.49±6.203 | 52.927±2.477 | 119.593±120.485 |
| rws v1 | 18.886±3.469 | 36.112±1.501 | 13.981±1.163 |
| rws v2 | 37.045±4.836 | 53.683±15.926 | 0.425±0.062 |

По значениям утилизации ресурсов сети выделятся rws_v2 по наименьшему количеству переданной информации по сети. Однако, при относительном сравнении утилизации сети, следует учитывать, что quartz и hangfire используют для координации MSSQL, а rws_v1 и rws_v2 используют etcd. Тем не менее, результаты показывают, что даже при использовании одного и того же хранилища (MSSQL) для координации, сетевая нагрузка может значительно различаться, как видно на примере систем quartz и hangfire. Это подчеркивает, что архитектура и способы реализации взаимодействия с базой данных имеют ключевое значение для показателя сетевой нагрузки.

Выводы о потреблении оперативной памяти делать затруднительно из-за того, что она заполнена буфером телеметрии (процесс сбора телеметрии был описан ранее).

Ожидаемая нагрузка на процессор предполагалась на уровне 25-30%. Это предположение основывается на логике функционирования приложения, которое выполняет один итерационный процесс с моментальной полезной нагрузкой. Поскольку этот процесс должен выполняться последовательно и как можно чаще, предполагается, что одно из ядер процессора сервера приложения будет загружено на 100%. Учитывая наличие четырех ядер в системе, общая загрузка процессора для моментальной полезной нагрузки ожидалась на уровне 25%. Наиболее близко к этому значение утилизации процессора для системы rws_v2.

Таблица 2. Утилизация аппаратных ресурсов узлами хранилищ

| Система | Проц., % | ОЗУ, МБ | Сеть, КБ/сек | Диск, КБ/сек |
|---|---|---|---|---|
| hangfire | 2.942±1.202 | 938.057±64.921 | 58.486±112.804 | 54.158 ±785.885 |
| quartz | 7.52±1.154 | 1070.99±85.958 | 1530.27±220.594 | 1524.87 ±4190.92 |
| rws v1 | 3.224±0.437 | 109.495±33.396 | 116.172±54.991 | 429.174 ±430.438 |
| rws v2 | 0.467±0.047 | 10.954±1.67 | 3.853±1.879 | 0.014±0.414 |

При анализе значений утилизации ресурсов хранилища также следует учитывать типы хранилищ. Например, MSSQL аллоцирует часть доступной оперативной памяти, независимо от объема данных или числа запросов. Показатель использования сетевого ресурса для хранилищ отличается от показателя сетевого ресурса для приложений, поскольку хранилища, помимо взаимодействия с приложениями, также занимаются синхронной репликацией данных.

Показатели утилизации аппаратных ресурсов узлами хранилищ коррелируют с показателями утилизации аппаратных ресурсов приложений. Здесь также можно сделать вывод о том, что ключевую роль в утилизации ресурсов хранилищами играют алгоритмы, используемые в сравниваемых решениях.

## V. Выводы

Результаты показали, что все системы достигли высокой точности выполнения итераций в стандартных условиях. Это свидетельствует о корректности замеров и адекватности расчетов ожидаемого времени для каждой системы.

В ходе тестирования на устойчивость к отключению и выходу из строя узлов, все системы продемонстрировали способность к восстановлению в пределах заданных временных параметров.

Во всех испытаниях с сетевыми дестабилизирующими факторами, rws v2 поддерживал показатели времени выполнения в пределах контрольных значений, демонстрируя стабильность работы даже в условиях нестабильной сети. Также, наименьший минимальный интервал выполнения и наилучшее соотношение утилизации ресурсов между приложением и хранилищем продемонстрировал rws_v2, что указывает на его высокую эффективность в использовании аппаратных ресурсов.

Системы quartz, hangfire и rws_v1 показали чувствительность к сетевым помехам, что приводило к значительному влиянию на точность операций. Их производительность снижалась в условиях сетевых дестабилизаций, особенно при эмуляции задержек и потерь пакетов. Однако, при условии отсутствия проблем на сети хотя бы на одном узле, quartz сохранял показатели времени выполнения в пределах контрольных значений, что свидетельствует о его способности адаптироваться к частичным сетевым нарушениям.

Система, реализованная на базе hangfire, продемонстрировала ограниченные возможности по частоте выполнения итераций в сравнении с другими рассмотренными решениями, показав наименьший показатель среди них.

## VI. Заключение

В ходе данной работы был проведен сравнительный анализ решений на базе .NET для итерационного выполнения задач в распределенных отказоустойчивых системах. В ходе исследования были оценены такие показатели, как точность выполнения итераций, минимальный интервал выполнения и потребление ресурсов компонентами системы в условиях инфраструктурных и сетевых дестабилизирующих факторов.

Результаты исследования могут быть применены при проектировании и оптимизации распределенных систем, требующих высокой степени отказоустойчивости и эффективности. Это, в свою очередь, способствует повышению надежности и производительности критически важных приложений, работающих на платформе .NET.

## Библиография


[1] H. Mykhailyshyn, N. Pasyeka, V. Sheketa, M Pasyeka., O. Kondur, & M. Varvaruk, “Designing network computing systems for intensive processing of information flows of data,” In *Lecture Notes on Data Engineering and Communications Technologies*, vol. 48, 2021 https://doi.org/10.1007/978-3-030-43070-2_18

[2] G. Blinowski, A. Ojdowska, & A. Przybylek, “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation,” *IEEE Access*, vol. 10, pp. 20357-20374, 2022, https://doi.org/10.1109/ACCESS.2022.3152803

[3] T. Distler, C. Bahn, A. Bessani, F. Fischer, & F. Junqueira, “Extensible distributed coordination,” In *Proceedings of the 10th European Conference on Computer Systems, EuroSys 2015*, 2015. https://doi.org/10.1145/2741948.2741954

[4] T. Zeynally, & D. Demidov, “Fault tolerance of distributed worker processes in corporate information systems and technologies,” *Communications in Computer and Information Science*, vol. 1703, pp.32-41, 2022, https://doi.org/10.1007/978-3-031-21340-3_4

[5] T. Zeynally, Demidov, & L. Dimitrov, “Prioritization of Distributed Worker Processes Based on Etcd Locks,”. *Communications in Computer and Information Science*, vol. 1703, pp. 93-103, 2022. https://doi.org/10.1007/978-3-031-21340-3_9

[6] T. Zeynally, & D. Demidov, “Evaluation of the Efficiency of Fault Tolerance Algorithms for Distributed Peer-To-Peer Worker Processes Connected Through a Key-Value Store,” Communications in Computer and Information Science, vol. 1821, pp. 87-98, 2023. https://doi.org/10.1007/978-3-031-31353-0_8ol 1821.

[7] A. Troelsen, & P. Japikse, “Introducing C# and .NET (Core) 5,” In *Pro C# 9 with .NET 5*, 2021. https://doi.org/10.1007/978-1-4842-6939-8_1

[8] I. C. Schuszter, & M. Cioca, “An implementation of a fault-tolerant database system using the actor model,” *MATEC Web of Conferences*, vol. 342, 2021. https://doi.org/10.1051/matecconf/202134205001

[9] A. Natanzon, & E. Bachmat, “Dynamic synchronous/asynchronous replication,” ACM Transactions on Storage, vol. 9, no. 3, pp. 1-19, 2013. https://doi.org/10.1145/2508011

[10] Bryk, P., Malawski, M., Juve, G., & Deelman, E. (2016). Storage-aware Algorithms for Scheduling of Workflow Ensembles in Clouds. Journal of Grid Computing, 14(2). https://doi.org/10.1007/s10723-015-9355-6

[11] R. Garg, M. Mittal, & L. H. Son, “Reliability and energy efficient workflow scheduling in cloud environment,” *Cluster Computing*, vol. 22, no. 4, pp. 1283-1297, 2019. https://doi.org/10.1007/s10586-019-02911-7

[12] R. Sharma, N. Nitin, M. A. R., AlShehri & D. Dahiya, “Priority-based joint EDF–RM scheduling algorithm for individual real-time task on distributed systems,” *The Journal of Supercomputing*, vol. 77, pp. 890-908, 2021. https://doi.org/10.1007/s11227-020-03306-x0.1007/s11227-020-03306-x

[13] C. Li, J. Tang, T. Ma, X. Yang, & Y. Luo, “Load balance based workflow job scheduling algorithm in distributed cloud,” *Journal of Network and Computer Applications*, vol. 152, p. 102518, 2020. https://doi.org/10.1016/j.jnca.2019.102518

[14] HangfireIO/Hangfire: сайт. – URL: https://github.com/HangfireIO/Hangfire (дата обращения: 08.01.2024)

[15] quartznet/quartznet.: сайт. – URL: https://github.com/quartznet/quartznet (дата обращения: 08.01.2024)

[16] TeymurZeynally/RecurrentWorkerService. : сайт. – URL: : https://github.com/TeymurZeynally/RecurrentWorkerService (дата обращения: 08.01.2024)

[17] Docker overview: сайт. – URL: https://docs.docker.com/get-started/overview/ (дата обращения: 08.01.2024)

[18] Telegraf Docker Input Plugin: сайт. – URL: https://github.com/influxdata/telegraf/blob/master/plugins/inputs/docker/README.md (дата обращения: 08.01.2024)

[19] Docker Traffic Control: сайт. – URL: https://github.com/lukaszlach/docker-tc (дата обращения: 08.01.2024)

[20] InfluxDB key concepts: сайт. – URL: https://docs.influxdata.com/influxdb/v1/concepts/key_concepts/ (дата обращения: 08.01.2024)

[21] DateTimeOffset Ticks: сайт. – URL: https://learn.microsoft.com/en-us/dotnet/api/system.datetimeoffset.ticks?view=net-7.0 (дата обращения: 08.01.2024)

# Comparative analysis of .NET-based solutions for iterative task execution in distributed fault-tolerant systems

T. Zeynally

***Annotation*—This paper presents the results of a comparative analysis of .NET-based solutions for iterative task execution in distributed fault-tolerant systems. The study examines peer-to-peer distributed systems capable of iteratively performing scheduled tasks. Among the compared solutions are Quartz, Hangfire, and Recurrent Worker Service. The research investigates the dependency of measurable metrics such as iteration execution accuracy, minimum execution interval, and resource consumption on external destabilizing factors, including infrastructure (node failures) and network issues (bandwidth, delays, packet loss). The methodological basis of the study is experimental. The experiment is conducted on two servers, with one hosting the application nodes and the other hosting nodes of a synchronously replicated storage system. A series of tests are conducted on the application nodes where network and infrastructure destabilizing factors are controlled and simulated. During the experiment, hardware metrics are measured, and telemetry is collected from the applications. An analysis and calculation of statistical parameters are conducted based on the collected data, after which the results are presented in the form of tables and diagrams. The results of this work can be applied in the design and optimization of distributed systems requiring high levels of fault tolerance and efficiency.**

***Keywords*—distributed fault-tolerant systems, iterative task execution, peer-to-peer distributed systems, iteration execution accuracy.**

REFERENCES

[1] H. Mykhailyshyn, N. Pasyeka, V. Sheketa, M Pasyeka., O. Kondur, & M. Varvaruk, “Designing network computing systems for intensive processing of information flows of data,” In *Lecture Notes on Data Engineering and Communications Technologies*, vol. 48, 2021 https://doi.org/10.1007/978-3-030-43070-2_18

[2] G. Blinowski, A. Ojdowska, & A. Przybylek, “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation,” *IEEE Access*, vol. 10, pp. 20357-20374, 2022, https://doi.org/10.1109/ACCESS.2022.3152803

[3] T. Distler, C. Bahn, A. Bessani, F. Fischer, & F. Junqueira, “Extensible distributed coordination,” In *Proceedings of the 10th European Conference on Computer Systems, EuroSys 2015*, 2015. https://doi.org/10.1145/2741948.2741954

[4] T. Zeynally, & D. Demidov, “Fault tolerance of distributed worker processes in corporate information systems and technologies,” *Communications in Computer and Information Science*, vol. 1703, pp.32-41, 2022, https://doi.org/10.1007/978-3-031-21340-3_4

[5] T. Zeynally, Demidov, & L. Dimitrov, “Prioritization of Distributed Worker Processes Based on Etcd Locks,”. *Communications in Computer and Information Science*, vol. 1703, pp. 93-103, 2022. https://doi.org/10.1007/978-3-031-21340-3_9

[6] T. Zeynally, & D. Demidov, “Evaluation of the Efficiency of Fault Tolerance Algorithms for Distributed Peer-To-Peer Worker Processes Connected Through a Key-Value Store,” Communications in Computer and Information Science, vol. 1821, pp. 87-98, 2023. https://doi.org/10.1007/978-3-031-31353-0_8ol 1821.

[7] A. Troelsen, & P. Japikse, “Introducing C# and .NET (Core) 5,” In *Pro C# 9 with .NET 5*, 2021. https://doi.org/10.1007/978-1-4842-6939-8_1

[8] I. C. Schuszter, & M. Cioca, “An implementation of a fault-tolerant database system using the actor model,” *MATEC Web of Conferences*, vol. 342, 2021. https://doi.org/10.1051/matecconf/202134205001

[9] A. Natanzon, & E. Bachmat, “Dynamic synchronous/asynchronous replication,” ACM Transactions on Storage, vol. 9, no. 3, pp. 1-19, 2013. https://doi.org/10.1145/2508011

[10] Bryk, P., Malawski, M., Juve, G., & Deelman, E. (2016). Storage-aware Algorithms for Scheduling of Workflow Ensembles in Clouds. Journal of Grid Computing, 14(2). https://doi.org/10.1007/s10723-015-9355-6

[11] R. Garg, M. Mittal, & L. H. Son, “Reliability and energy efficient workflow scheduling in cloud environment,” *Cluster Computing*, vol. 22, no. 4, pp. 1283-1297, 2019. https://doi.org/10.1007/s10586-019-02911-7

[12] R. Sharma, N. Nitin, M. A. R., AlShehri & D. Dahiya, “Priority-based joint EDF–RM scheduling algorithm for individual real-time task on distributed systems,” *The Journal of Supercomputing*, vol. 77, pp. 890-908, 2021. https://doi.org/10.1007/s11227-020-03306-x0.1007/s11227-020-03306-x

[13] C. Li, J. Tang, T. Ma, X. Yang, & Y. Luo, “Load balance based workflow job scheduling algorithm in distributed cloud,” *Journal of Network and Computer Applications*, vol. 152, p. 102518, 2020. https://doi.org/10.1016/j.jnca.2019.102518

[14] HangfireIO/Hangfire.: website. – URL: https://github.com/HangfireIO/Hangfire (2024, April 12)

[15] quartznet/quartznet.: website. – URL: https://github.com/quartznet/quartznet (2024, April 12)

[16] TeymurZeynally/RecurrentWorkerService: website. – URL: : https://github.com/TeymurZeynally/RecurrentWorkerService (2024, April 12)

[17] Docker overview: website. – URL: https://docs.docker.com/get-started/overview/ (2024, April 12)

[18] Telegraf Docker Input Plugin: website. – URL: https://github.com/influxdata/telegraf/blob/master/plugins/inputs/docker/README.md (2024, April 12)

[19] Docker Traffic Control: website. – URL: https://github.com/lukaszlach/docker-tc (2024, April 12)

[20] InfluxDB key concepts: website. – URL: https://docs.influxdata.com/influxdb/v1/concepts/key_concepts/ (2024, April 12)

[21] DateTimeOffset Ticks: website. – URL: https://learn.microsoft.com/en-us/dotnet/api/system.datetimeoffset.ticks?view=net-7.0 (2024, April 12)

About of Author

Teymur Zeynally, Moscow Polytechnic University, z.teymur.e@gmail.com