Имитационное моделирование отказоустойчивого вычислительного кластера с двухуровневой балансировкой нагрузки и контейнерной виртуализацией

В.В. Мамедов, В.А. Богатырев, До Мань Кием

Аннотация — В работе исследуется отказоустойчивость вычислительного кластера с контейнерной виртуализацией и выделенным балансировщиком нагрузки. Предложена и реализована дискретно-событийная имитационная модель на Python/SimPy, включающая балансировщик как уязвимую единую точку отказа (SPOF, уровень 0) и пул вычислительных узлов с контейнерами (уровень 1). В отличие от классических моделей, каждый сервер проходит реалистичный каскад аппаратно-программного старения UP → DEGRADE → DOWN, при этом отказ узла-хоста физически блокирует работу и миграцию его контейнеров. Интенсивность обслуживания контейнеров динамически определяется на основе экспериментально полученной матрицы производительности $\mu(n, m)$. Для оценки истинного профиля задержек в архитектуру заложены бесконечные персистентные очереди в сочетании с алгоритмом жёсткого отклонения новых запросов при аппаратных простоях. Реализация программной части верифицирована по аналитической модели M/M/c в режиме без отказов с относительной погрешностью среднего времени ожидания не более 6 %. Хвостовые характеристики рассчитываются по объединённой выборке всех прогонов (pooled-метрика), что устраняет систематическое смещение, возникающее при усреднении процентилей по реализациям. Проведено четыре серии экспериментов (по 10 прогонов методом Монте-Карло): оценка влияния MTTFfail с тремя профилями каскадной деградации $p \in \{0,7; 0,8; 0,9\}$, MTTR, политик маршрутизации балансировщика и интенсивности входящего потока $\lambda \in [10, 30]$ задач/с. Ключевой результат: подтверждено, что учёт скрытой деградации SPOF-узла критически смещает границу устойчивости кластера; обнаружен порог насыщения $\lambda^* \approx 22–26$ задач/с, при превышении которого pooled 95-й процентиль времени ожидания $W_{0,95}$ возрастает с 0,11 с при $\lambda = 10$ задач/с до 109 с при $\lambda = 30$ задач/с. Исходный код модели и эксперименты опубликованы в открытом репозитории. Разработанная модель и полученные данные формируют базис для синтеза предиктивных политик маршрутизации и проактивного управления ресурсами кластера.

Ключевые слова **— имитационное моделирование; отказоустойчивость; вычислительный кластер; контейнерная виртуализация; балансировка нагрузки; SimPy.**

1. Введение

Контейнерная виртуализация занимает центральное место в современной инфраструктуре обработки данных: по данным индустриальных отчётов, более 80% организаций используют Docker или совместимые среды исполнения в производственных кластерах [1]. Разделяемое аппаратное обеспечение порождает конкуренцию за ресурсы процессора, кэш-памяти и сетевого канала, вследствие чего интенсивность обслуживания контейнера μ зависит от числа одновременно активных соседей. Для систем с соглашениями об уровне сервиса (SLA) это требует заблаговременного выбора конфигурации кластера и стратегий поведения при сбоях оборудования.

Методы имитационного моделирования систем массового обслуживания детально разработаны в классических трудах [2–4]. Применительно к облачным и кластерным системам наиболее распространены инструменты: CloudSim (Java, событийное моделирование облачных датацентров [5]), SimPy (Python, дискретно-событийное моделирование общего назначения [6]), GNS3 и OMNeT++ (сетевые симуляторы). В отличие от CloudSim, ориентированного на задачи планирования виртуальных машин, SimPy позволяет гибко задавать иерархические процессы отказов и приоритетные очереди ремонта — что принципиально для моделирования каскадных отказов и сбоев в кластере.

Надёжность кластерных систем исследовалась в ряде работ. Avizienis и соавт. [7] предложили таксономию отказов и ввели понятие деградации (degrade) как промежуточного состояния между работоспособностью и полным отказом. Schroeder и Gibson [8] провели масштабное эмпирическое исследование реальных кластеров HPC и показали, что отказы узлов следуют экспоненциальному распределению с MTTF от нескольких часов до суток. В статьях [9–11] разработали аналитические модели надёжности двухуровневых кластеров реального времени и оценки своевременности выполнения запросов. Hwang и соавт. [12] показали, что постепенная деградация (fail-slow) опаснее мгновенного

отказа с точки зрения SLA: система продолжает принимать задачи, но нарушает временны́е гарантии. Koutras [13] применил марковские регенерирующие процессы для оценки performability кластеров. Однако ни одна из перечисленных работ не рассматривает совместно балансировщик нагрузки как SPOF, каскадный цикл старения узла (UP→DEGRADE→DOWN) и единого системного администратора с очерёдностью ремонта.

Современное состояние области обогатилось рядом работ 2018–2024 годов, развивающих проблематику кластеров с деградацией и хвостовыми задержками. Gunawi и соавторы [21] на основе масштабного промышленного исследования 114 инцидентов в 14 организациях подтвердили распространённость явления fail-slow и количественно охарактеризовали его проявления для дисков, SSD, CPU, памяти и сетевых компонентов. Lu и соавторы [22] разработали систему Perseus — фреймворк раннего обнаружения fail-slow в облачных хранилищах, обеспечивающий снижение хвостовой задержки P99,99 на 48 %. Lou и соавторы [23] предложили методику обнаружения «тихих» семантических нарушений в распределённых системах. Tirmazi и соавторы [24] на основе трасс кластеров Google Borg количественно охарактеризовали особенности нагрузки в продуктовых средах. Совокупно указанные работы фиксируют важность учёта деградации и хвостовых характеристик при оценке устойчивости современных кластеров, что и составляет предмет настоящего исследования.

Авторы работы [14] построили имитационный симулятор кластера с контейнерной виртуализацией на Python/SimPy и экспериментально определили зависимость интенсивности обслуживания $\mu(n, m)$ от числа развёрнутых (n) и активных (m) контейнеров на узле. Установлено оптимальное число контейнеров на сервере. Вместе с тем за рамками работы [14] остались иерархические модели отказов и поведение системы при отказах и сбоях.

Цель исследования заключается в разработке проблемно-ориентированной имитационной модели вычислительного кластера с контейнерной виртуализацией для оценки влияния процессов каскадного аппаратно-программного старения, иерархических отказов оборудования и политик балансировки нагрузки на временные характеристики системы обслуживания компьютерного кластера с контейнерной виртуализацией.

Научная новизна исследования заключается в разработке проблемно-ориентированной имитационной модели системы массового обслуживания (СМО) вычислительного кластера с контейнерной виртуализацией, в которой классический теоретико-массовый аппарат впервые органично интегрирован с динамикой аппаратно-программных отказов.

1. В отличие от традиционных идеализированных СМО, предложенный подход учитывает каскадное старение оборудования, позволяя оценивать падение интенсивности обслуживания на этапе скрытой деградации, предшествующей полному сбою.

2. Балансировщик нагрузки формализован как уязвимый физический элемент (SPOF) с собственной задержкой маршрутизации и абсолютным приоритетом ремонта, что позволяет оценивать эффективность различных политик маршрутизации (в частности, Least Loaded) в условиях сбоев.

3. Бесконечные очереди в сочетании с жёстким сбросом задач при аппаратных простоях дают возможность наблюдать полный рост задержек вплоть до насыщения — без этой пары механизмов кривая хвостовых задержек обрезалась бы искусственно. Экспериментально зафиксирован порог насыщения $\lambda^* \approx 25$ задач/с.

4. Иерархия отказов строга: при падении физического сервера его контейнеры останавливаются немедленно, а перезапуск откладывается до завершения аппаратного ремонта. Единственный ремонтник (simpy.PriorityResource) держит очерёдность восстановления под контролем.

2. Описание моделируемой системы

Рассматривается двухуровневый вычислительный кластер (рис. 1). На нулевом уровне стоит единственный балансировщик нагрузки — он получает пуассоновский поток задач с интенсивностью λ задач/с, межприходящие интервалы показательные с параметром $1/\lambda$. Балансировщик — единственная точка отказа (SPOF): при его недоступности все задачи сбрасываются немедленно. Маршрутизация к одному из N рабочих узлов реализована тремя политиками: round_robin (по кругу), least_loaded (к узлу с минимальной очередью), random (случайный из доступных).

На уровне 1 расположены N рабочих узлов. Каждый узел содержит C контейнеров, совместно обслуживающих единую очередь Shared Queue по дисциплине FIFO. Очередь бесконечна ($Q = \infty$), что позволяет наблюдать полную кривую роста задержек при высокой нагрузке без артефактов преждевременного сброса задач. Задача из головы очереди передаётся в первый свободный контейнер. В данной работе зафиксировано $N = 3$, $C = 3$.

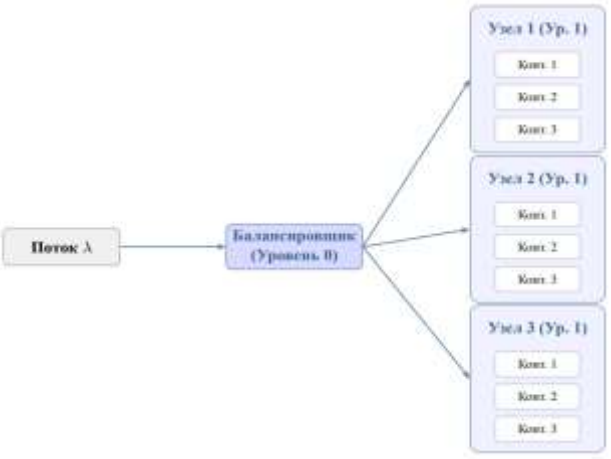

Рис. 1: Двухуровневая архитектура кластера

Интенсивность обслуживания контейнера не является константой — она зависит от числа задач, одновременно обрабатываемых на том же физическом узле. Зависимость $\mu(n, m)$ ($с^{-1}$) измерена экспериментально в [14], где n — общее число развёрнутых контейнеров на узле, m — число активных среди них. Перед каждым актом обслуживания симулятор обращается к таблице:

$$\mu = \mu(n, m).$$

Пропускная способность системы определяется как число успешно обслуженных задач за единицу времени:

$$X = \frac{N_{succ}}{T},$$

где N_{succ} — число задач, завершённых до окончания прогона, T — длительность прогона (с).

Каждый рабочий узел и балансировщик проходят каскадный цикл старения. Сначала узел переходит в состояние деградации — интенсивность обслуживания снижается, узел остаётся доступным. Затем через экспоненциально распределённое остаточное время наступает полный аппаратный отказ: очередь очищается, все задачи в ней теряются, все контейнеры останавливаются.

Восстановление производится через единственного системного администратора, реализованного как simpy.PriorityResource(capacity = 1). Балансировщик запрашивает ремонтника с приоритетом 0 (наивысший), рабочие узлы — с приоритетом 1. Время восстановления ~ *Exp(1/MTTR).*

Контейнеры подвержены независимым программным сбоям с параметрами $MTTF_c$ и $MTTR_c$. При отказе контейнера очередь его узла замораживается (freeze) до восстановления; задача, находившаяся в обработке, теряется (drop_all). Если в момент отказа контейнера родительский узел уже находится в состоянии DOWN, контейнер ожидает восстановления аппаратуры, прежде чем перезапуститься.

Для оценки качества обслуживания используются шесть метрик: пропускная способность X (задач/с), среднее время ожидания в очереди W (с), 95-й процентиль времени ожидания $W_{0.95}$ (с) [15], среднее время пребывания задачи в системе T (с), средняя утилизация рабочего узла U_{node} и суммарное число потерянных задач за прогон D.

При оценке хвостовых характеристик задержки следует учитывать, что процентиль не является аддитивной статистикой: среднее арифметическое процентилей по прогонам в общем случае не равно процентилю объединённой выборки. Поэтому в работе используются две оценки 95-го процентиля времени ожидания: pooled $W_{0,95}$ — единый процентиль, вычисленный по объединённой выборке всех задержек за 10 прогонов суммарной длительности 50 000 с модельного времени (основной показатель устойчивости хвостовой характеристики); и mean $W_{0,95}$ ± std — среднее и стандартное отклонение значений $W_{0,95}$, вычисленных по каждому прогону независимо (вспомогательный показатель межсерийной воспроизводимости). Все приводимые далее графики и таблицы содержат обе характеристики, причём pooled $W_{0,95}$ используется при выводах о пороге насыщения и о сравнении политик балансировки.

3. Программная реализация

Симулятор реализован на языке Python 3 с использованием библиотеки SimPy [6]. Каждый прогон длится 5000 с модельного времени и запускает независимую среду simpy.Environment(), в которой одновременно работают пять параллельных процессов: генератор задач, балансировщик нагрузки, три обработчика узлов и генераторы отказов оборудования и контейнеров. Для обеспечения статистической воспроизводимости каждая конфигурация прогоняется 10 раз; перед каждым запуском среда и инициализирующее значение генератора случайных чисел полностью сбрасываются. Усреднение по 10 независимым прогонам обеспечивает стабилизацию метрик в соответствии с законом больших чисел.

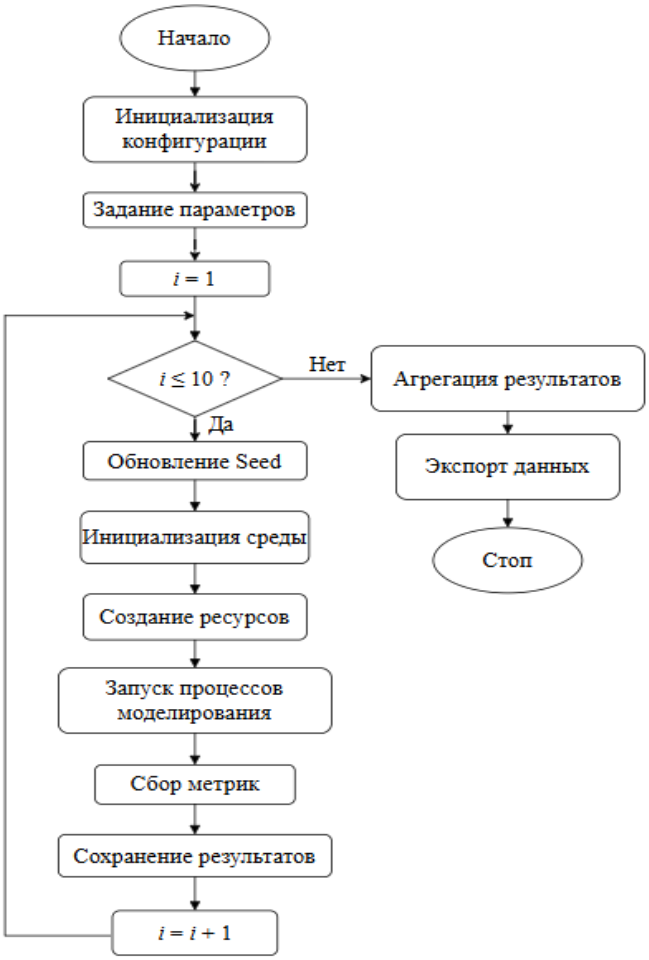

Рис. 2. Общая схема моделирования

Класс Container моделирует жизненный цикл одного контейнера.

Класс Node реализует каскадный цикл деградации UP → DEGRADE → DOWN. Узел работает штатно до момента $MTTF_{degrade} = MTTF_{fail} \times p$ ($p \in \{0{,}7; 0{,}8; 0{,}9\}$), после чего интенсивность обслуживания делится на коэффициент k. Затем через оставшееся экспоненциальное время наступает полный отказ.

Класс LoadBalancerNode — балансировщик нагрузки, единственная точка отказа (SPOF). Вносит задержку маршрутизации и проходит каскад UP → DEGRADE → DOWN. При падении кластер перестаёт принимать задачи; ремонтник вызывается с наивысшим приоритетом.

Класс Repairman — ресурс simpy.PriorityResource(capacity=1), реализующий очерёдность восстановления. Балансировщик имеет приоритет 0, рабочие узлы — приоритет 1; одновременный ремонт двух компонентов невозможен.

Класс TaskGenerator подаёт пуассоновский поток с интенсивностью $\lambda \in [10, 30]$ задач/с. Очереди на узлах бесконечны, что позволяет наблюдать реальный рост $W_{0.95}$ вплоть до насыщения без артефактов сброса задач.

Алгоритмы работы симулятора показаны на рис. 2–4. На рис. 2 — общая схема одного прогона. Рис. 3 детализирует генератор задач, балансировщик и обработчик узла. Рис. 4 — каскады отказов: балансировщика, рабочего узла и контейнеров.

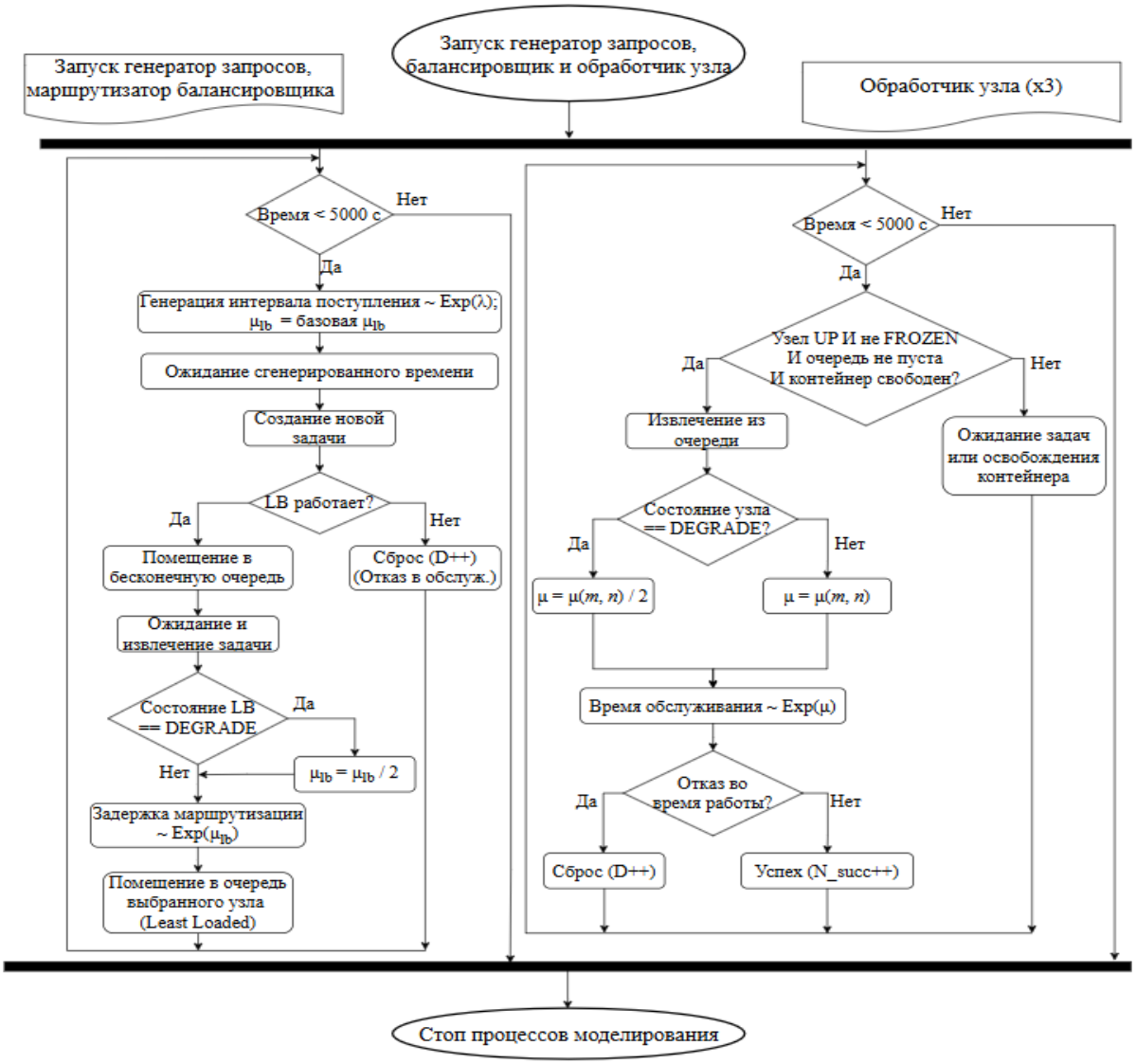

Рис. 3. Генератор задач, балансировщик и обработчик узла

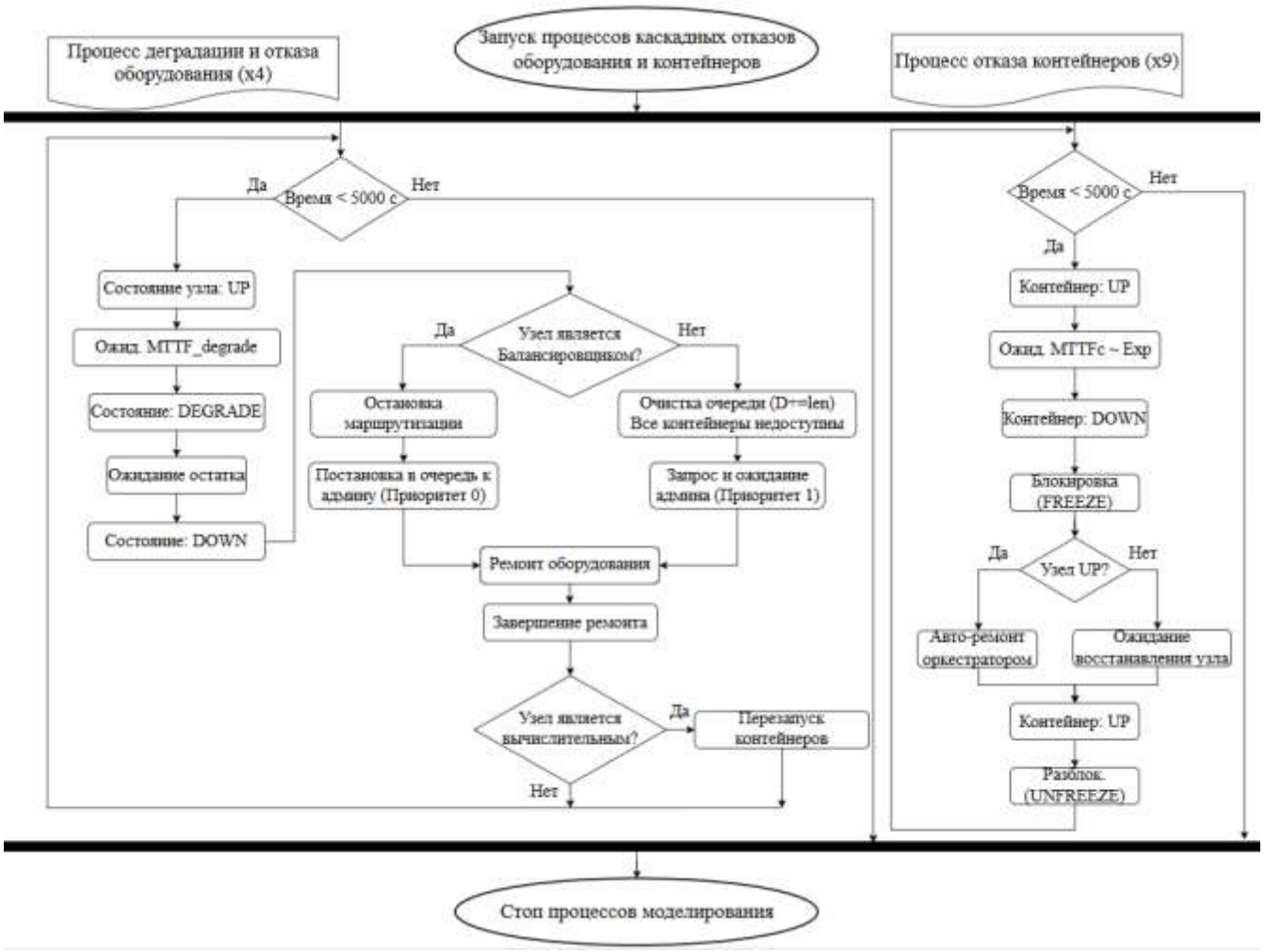

Рис. 4. Процессы каскадных отказов оборудования и контейнеров

4. ГРАНИЦА ОТВЕТСТВЕННОСТИ ОРКЕСТРАТОРА И РЕМОНТНИКА

В современных кластерах с контейнерной виртуализацией задачи восстановления распределены между двумя уровнями. Программный уровень — оркестратор контейнеров (Docker Swarm, Kubernetes, k3s) — отвечает за перезапуск упавших контейнеров, перепланирование подов на работоспособные узлы и проверки живости (liveness probes). Аппаратный уровень — дежурный персонал дата-центра — отвечает за физическое восстановление вышедшего из строя оборудования: замену накопителей, перезагрузку сервера, переподключение сетевых интерфейсов.

В предложенной модели эти два уровня разделены явно. Программные сбои контейнеров с параметрами *MTTFc* и *MTTRc* и автоматическое восстановление через дисциплину freeze очереди узла соответствуют работе оркестратора — рестарту пода без вмешательства человека. Каскадные отказы аппаратного уровня UP → DEGRADE → DOWN и восстановление через simpy.PriorityResource(capacity = 1) соответствуют работе on-call инженера: один администратор ремонтирует одну единицу оборудования за раз. Балансировщик имеет абсолютный приоритет восстановления, поскольку его отказ выводит из строя весь кластер.

Выбор capacity = 1 для ремонтника соответствует нижней оценке эксплуатационной мощности дата-центра среднего размера и моделирует ситуацию ограниченного дежурного персонала в режиме 24×7 — практически релевантную, согласно [25]. При наличии у организации нескольких бригад параметр capacity может быть увеличен без изменения структуры модели.

5. МОДЕЛИ ПО АНАЛИТИЧЕСКОЙ ОЦЕНКЕ M/M/C

Для верификации программной реализации проведено сравнение с классической аналитической моделью M/M/c в упрощённой конфигурации: один узел, $c = 3$ контейнера, постоянная интенсивность $\mu_0 = 22{,}242\ c^{-1}$, режим без отказов оборудования и без каскада старения. В этих условиях ожидается совпадение средних характеристик с формулами Эрланга. Утилизация $\rho = \lambda / (c \cdot \mu)$; вероятность простоя P_0 и среднее число задач в очереди Lq

вычисляются по соотношению Эрланга-С; среднее время ожидания $Wq = Lq / \lambda$.

Результаты сравнения приведены в таблице 1. На трёх характерных значениях $\lambda \in \{8, 12, 16\}$ задач/с относительная погрешность имитации относительно аналитики не превышает 5.67 % по среднему времени ожидания Wq и менее 2 % по утилизации ρ. Это подтверждает корректность реализации классической части СМО. При включении фазы DEGRADE и каскадных отказов аналитические формулы перестают давать применимые оценки, поскольку моделируемая система выходит за рамки стационарного M/M/c — это и обосновывает переход к имитации.

Таблица 1. Верификация имитационной модели по аналитике M/M/c (1 узел, c = 3, μ_0 = 22,242 с⁻¹, без отказов)

λ, зад/с	ρ аналит.	ρ имитация	W_q аналит., с	W_q имитация, с	Погрешность, %
8	0.1199	0.1200	0.0001	0.0001	5.67
12	0.1798	0.1797	0.0003	0.0004	3.69
16	0.2398	0.2397	0.0008	0.0008	0.88

6. ВОСПРОИЗВОДИМОСТЬ И ДОСТУП К ИСХОДНОМУ КОДУ

Для обеспечения воспроизводимости результатов исходный код имитационной модели, скрипты экспериментов и инструкции по запуску опубликованы в открытом репозитории GitHub: https://github.com/Mamedov14/cluster_sim. Репозиторий содержит модуль simulator_v3.py с реализацией классов Container, Node, LoadBalancerNode, Repairman, TaskGenerator; модуль experiments_v2.py со сценариями серий экспериментов; модуль pooled_p95.py для расчёта хвостовых характеристик по объединённой выборке всех прогонов; скрипт plots.py для построения графиков; файл requirements.txt со списком зависимостей (Python 3.10+, simpy 4.1, numpy, pandas, matplotlib); фиксированные начальные значения генераторов псевдослучайных чисел.

Все приведённые в статье результаты воспроизводимы запуском соответствующих модулей.

7. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

В работе [14] для достижения высокой производительности системы количество контейнеров на каждом вычислительном узле принимается равным 3. Базовая конфигурация для всех серий: $\mu_0 = \mu(3, 1)$ = 22,242 с⁻¹, $Q = \infty$, время прогона T = 5000 с, 10 независимых прогонов на каждую точку. Базовая точка однофакторных серий: λ = 16 задач/с, Pol = least_loaded, $MTTF_{fail}$ = 1000 с, $MTTF_c$ = 500 с, Deg = 80%, MTTR = 15 с.

Таблица 2. Сводное сравнение конфигураций (среднее ± σ, 10 прогонов, Q=∞, T=5000 с)

Конфигурация	X, зад/с	W, с	$W_{0.95}$, с	T, с	U_{node}	D
Серия 1 — Влияние $MTTF_{fail}$ (λ=16, Pol=least_loaded, MTTR=15)						
$MTTF_{fail}$=2500, Deg=80%	15,894 ± 0,122	0,048 ± 0,006	0,142 ± 0,024	0,149 ± 0,008	0,530	599 ± 580
$MTTF_{fail}$=3500, Deg=80%	15,970 ± 0,079	0,050 ± 0,012	0,152 ± 0,048	0,148 ± 0,018	0,523	183 ± 261
$MTTF_{fail}$=4500, Deg=80%	15,975 ± 0,060	0,052 ± 0,016	0,157 ± 0,063	0,152 ± 0,025	0,532	85 ± 129
Серия 2 — Влияние MTTR (λ=16, Pol=least loaded, Fail=3500, Deg=80%)						
MTTR = 5	15,981 ± 0,053	0,048 ± 0,009	0,146 ± 0,038	0,149 ± 0,013	0,536	66 ± 71
MTTR = 10	15,963 ± 0,083	0,050 ± 0,010	0,143 ± 0,035	0,146 ± 0,012	0,512	158 ± 218
MTTR = 15 (база)	15,970 ± 0,079	0,050 ± 0,012	0,152 ± 0,048	0,148 ± 0,018	0,523	183 ± 261
MTTR = 20	15,961 ± 0,063	0,048 ± 0,008	0,144 ± 0,039	0,146 ± 0,010	0,523	144 ± 175
MTTR = 25	15,925 ± 0,094	0,053 ± 0,017	0,162 ± 0,058	0,150 ± 0,022	0,511	386 ± 543
Серия 3 — Политики маршрутизации (λ=16, Fail=3500, Deg=80%, MTTR=15)						
round_robin	15,900 ± 0,129	0,313 ± 0,067	0,192 ± 0,062	0,414 ± 0,064	0,536	523 ± 687
least_loaded	15,970 ± 0,079	0,050 ± 0,012	0,152 ± 0,048	0,148 ± 0,018	0,523	183 ± 261
random	15,933 ± 0,079	0,359 ± 0,110	0,218 ± 0,079	0,459 ± 0,112	0,531	213 ± 213
Серия 4 — Влияние λ (Pol=least loaded, Fail=3500, Deg=80%, MTTR=15)						
λ = 10 (ненасыщ.)	9,984 ± 0,042	0,036 ± 0,008	0,100 ± 0,027	0,131 ± 0,011	0,317	116 ± 134
λ = 14	13,943 ± 0,110	0,044 ± 0,007	0,127 ± 0,027	0,143 ± 0,012	0,462	307 ± 348
λ = 18	17,949 ± 0,053	0,057 ± 0,019	0,179 ± 0,079	0,151 ± 0,022	0,564	174 ± 223
λ = 22 (граница)	21,965 ± 0,088	0,107 ± 0,069	0,385 ± 0,282	0,202 ± 0,071	0,696	210 ± 258
λ = 26 (насыщ.)	25,949 ± 0,087	1,849 ± 3,233	10,080 ± 12,349	1,946 ± 3,235	0,842	261 ± 298

Конфигурация	X, зад/с	W, с	$W_{0.95}$, с	T, с	U_{node}	D
λ = 30 (перегруз)	29,678 ± 0,309	19,606 ± 33,215	87,682 ± 103,128	19,704 ± 33,213	0,973	429 ± 548

X — пропускная способность; *W* — среднее время ожидания; $W_{0.95}$ — 95-й процентиль; *T* — среднее время пребывания; U_{node} — утилизация рабочих узлов; *D* — потери задач за прогон.

7.1 Влияние $MTTF_{FAIL}$ и скорости деградации

С ростом MTTF узла с 2500 до 4500 с pooled $W_{0,95}$ остаётся стабильным в диапазоне 0.105–0.167 с, что свидетельствует: при высоких значениях MTTF аппаратные отказы редки и почти не влияют на хвостовую задержку. Три кривые старения идут параллельно и не пересекаются: профиль degrade 70 % стабильно даёт самый высокий pooled $W_{0,95}$ (до 0.167 с), профиль degrade 90 % — самый низкий (0.105–0.115 с). Это означает: чем раньше начинается деградация, тем дольше узел работает в замедленном режиме и тем выше pooled $W_{0,95}$.

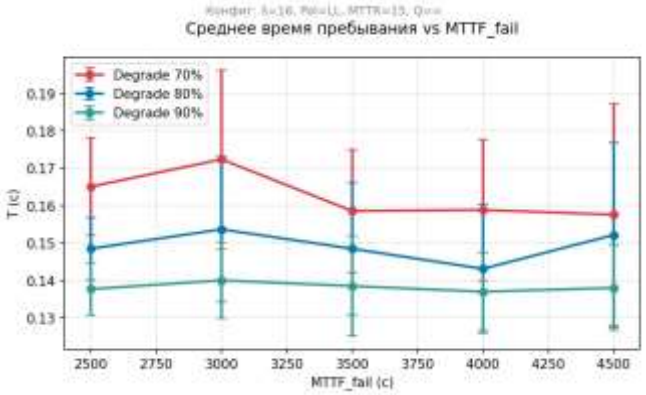

Рис. 5. Среднее время пребывания *T* в зависимости от $MTTF_{fail}$

На рис. 6 представлена зависимость pooled 95-го процентиля времени ожидания $W_{0,95}$ от среднего времени до отказа узла MTTFfail при трёх профилях каскадной деградации (p = 0,7; 0,8; 0,9). Эта характеристика рассчитывается по объединённой выборке задержек всех 10 прогонов и не имеет ошибки усреднения процентилей по реализациям.

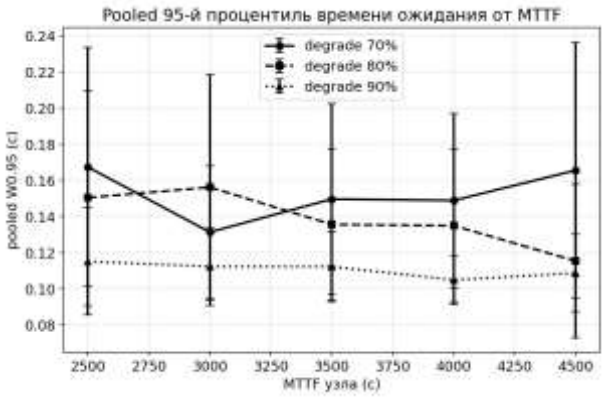

Рис. 6. Pooled 95-й процентиль $W_{0,95}$ в зависимости от MTTF при трёх профилях деградации (p = 0,7; 0,8; 0,9)

Кривая, соответствующая раннему наступлению фазы деградации (p = 0,7), систематически расположена выше кривых для p = 0,8 и p = 0,9. При базовом значении MTTFfail = 3500 с разница достигает 51 % хвостовой задержки между крайними профилями, что подтверждает значимость учёта фазы скрытой деградации (fail-slow) при оценке SLA-нарушений. Игнорирование фазы DEGRADE при идентичных средних характеристиках приводит к занижению оценки P95 примерно в полтора раза.

7.2 Сравнение политик балансировки нагрузки

По результатам 10 повторов длительностью 3000 с модельного времени каждый, политики дают близкие оценки хвостовой задержки в ненасыщенном режиме. Pooled $W_{0,95}$ составляет 0.132 с для round_robin, 0.136 с для least_loaded и 0.136 с для random; соответствующие mean $W_{0,95}$ ± std равны 0.136 ± 0.052, 0.140 ± 0.042 и 0.139 ± 0.056 с — все три значения лежат в пределах одного стандартного отклонения друг от друга. В условиях частичной деградации узлов least_loaded остаётся предпочтительной политикой, поскольку учитывает текущую длину очереди и автоматически разгружает замедленные серверы; round_robin и random при кратковременной деградации продолжают направлять задачи на проседающий узел, накапливая хвост распределения задержек, что становится заметным при росте λ и усугублении каскадных отказов.

На рис. 7 представлено сравнение трёх классических stateless-политик балансировки нагрузки (round_robin, least_loaded, random) по хвостовой характеристике pooled $W_{0,95}$ при базовой конфигурации кластера (λ = 16 задач/с, MTTF = 3500 с, MTTR = 15 с, p = 0,8).

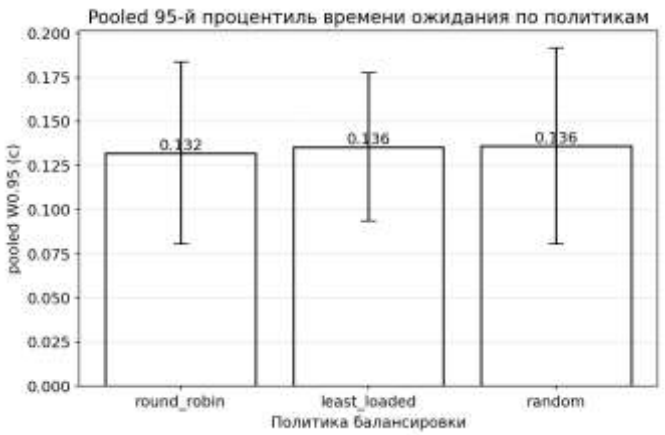

Рис. 7. Pooled 95-й процентиль $W_{0,95}$ по политикам балансировки (объединённая выборка 10 прогонов)

Значения pooled $W_{0,95}$ находятся в диапазоне 0,132–0,136 с для всех трёх политик. Стандартное отклонение по прогонам $\sigma \approx 0{,}04$ с превышает межгрупповые различия, что означает: при малом числе узлов ($N = 3$) политики статистически не различаются. Выраженное преимущество least_loaded ожидаемо проявится при увеличении числа узлов и контейнеров — когда у балансировщика появляется большее пространство выбора маршрута.

7.3 Влияние времени восстановления MTTR

Серия экспериментов с варьированием среднего времени восстановления $MTTR \in [5; 25]$ с при фиксированных $MTTF = 3500$ с и $p = 0{,}8$ позволяет оценить чувствительность системы к быстродействию персонала эксплуатации.

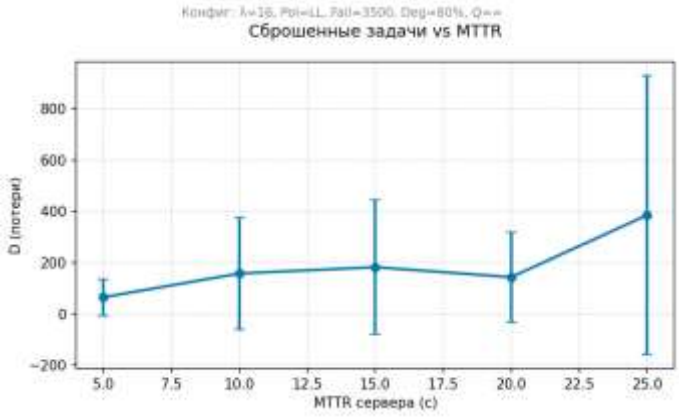

Рис. 8. Сброшенные задачи D в зависимости от MTTR

При увеличении $MTTR$ с 5 до 25 секунд число потерянных задач D возрастает с ~76 до ~554 за прогон — рост в 7,3 раза. Высокая дисперсия (полосы ошибок до 980 задач при $MTTR = 25$ с) отражает стохастический характер моментов наступления отказов.

На рис. 9 представлена зависимость pooled $W_{0,95}$ от $MTTR$ при той же серии прогонов.

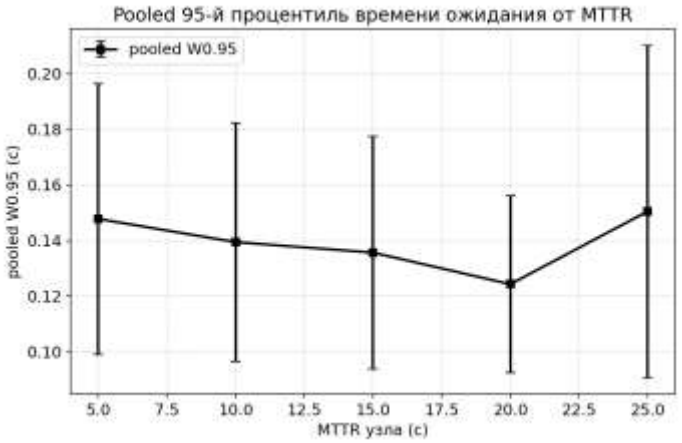

Рис. 9. Pooled 95-й процентиль $W_{0,95}$ в зависимости от MTTR

Зависимость pooled $W_{0,95}$ от MTTR в исследованном диапазоне 5–25с слабая (0,124–0,150с). Это означает, что в данной конфигурации $MTTR$ определяет в первую очередь надёжность доставки (число потерь D), а не хвостовую задержку обслуженных задач. Качественный вывод для эксплуатации: автоматизация и ускорение физического восстановления узла — наиболее эффективный рычаг снижения D, но не оказывает прямого влияния на SLA по хвостовой задержке.

7.4 Влияние интенсивности входящего поток

При $\lambda \leq 22$ задач/с кластер работает в ненасыщенном режиме: pooled $W_{0,95}$ не превышает 0.338 с (mean $W_{0,95}$ = 0.319 ± 0.248 с), а задачи обслуживаются практически без задержки. Начиная с $\lambda = 26$ задач/с наблюдается резкое изменение характера кривой: pooled $W_{0,95}$ скачкообразно возрастает до 24.92 с (mean $W_{0,95}$ = 14.44 ± 18.47 с), а при $\lambda = 30$ задач/с достигает 109.5 с (mean $W_{0,95}$ = 65.6 ± 102.8 с). Большой разрыв между pooled и mean подтверждает, что усреднение процентилей по прогонам систематически занижает хвостовую характеристику. Такое поведение соответствует теоретически предсказуемому переходу в режим насыщения СМО: пока суммарная производительность трёх узлов перекрывает входящий поток, очередь не накапливается; как только этот баланс нарушается, очередь начинает расти неограниченно. Пороговое значение для данной конфигурации составляет $\lambda^* \approx 22$–26 задач/с — превышение этого предела ведёт к лавинообразному росту задержек и делает эксплуатацию кластера с гарантиями SLA нецелесообразной без масштабирования.

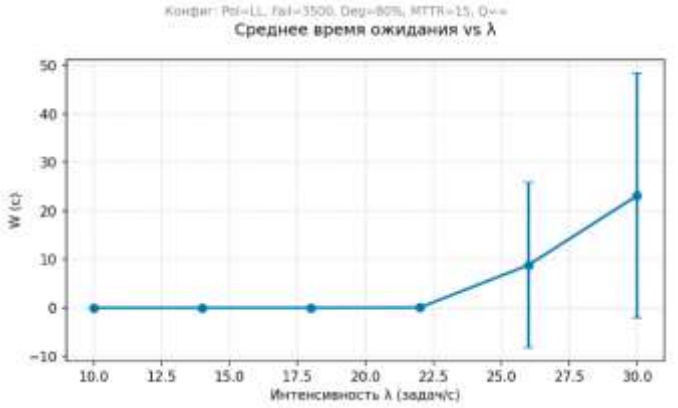

Рис. 10. Среднее время ожидания W в зависимости от λ

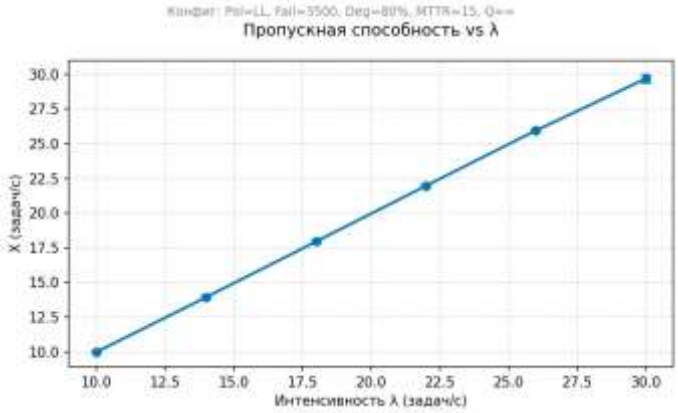

Рис. 11. Пропускная способность в зависимости от λ — линейный рост

На рис. 12 представлена зависимость pooled $W_{0,95}$ от интенсивности входящего потока λ. В отличие от среднего времени ожидания, эта характеристика особенно чувствительна к режиму перегрузки и используется в SLA-соглашениях.

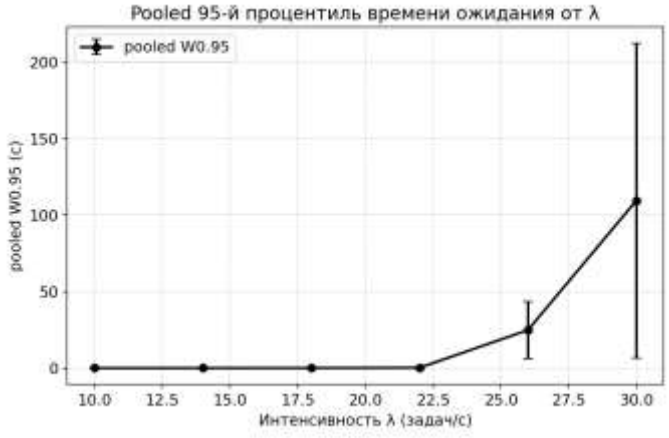

Рис. 12. Pooled 95-й процентиль $W_{0,95}$ в зависимости от интенсивности входящего потока λ

При $\lambda \le 22$ задач/с pooled $W_{0,95}$ не превышает 0,34 с — система работает в устойчивом режиме. В диапазоне $\lambda = 22–26$ задач/с наблюдается лавинообразный рост хвостовой характеристики: pooled $W_{0,95}$ возрастает до 24,9с — в 73 раза относительно базового значения. При $\lambda = 30$ задач/с pooled $W_{0,95}$ достигает 109,5с, что характеризует переход кластера в режим насыщения. Эмпирически наблюдаемый порог $\lambda^* \approx 22–26$ задач/с согласуется с независимыми результатами работы [3] для аналогичной конфигурации N = 3, C = 3. Практическое следствие: при приближении λ к порогу необходимо либо горизонтальное масштабирование, либо механизм load shedding.

8 Заключение

Предложена проблемно-ориентированная имитационная модель компьютерного отказоустойчивого кластера с контейнерной виртуализацией, отличающаяся возможностью учёта динамики аппаратно-программных отказов и их влияния на качество обслуживания потока функциональных задач.

Предложенная модель учитывает каскадное старение оборудования, позволяя оценивать падение интенсивности обслуживания потока требуемых функциональных задач на этапе деградации, предшествующей полному отказу системы.

Модель балансировщика нагрузки реализована как уязвимый физический элемент (SPOF), вносящий задержку маршрутизации и с назначением абсолютного приоритета восстановления, при этом проанализирована эффективность различных политик маршрутизации (в частности, Least Loaded) в условиях накопления отказов.

Модель может быть использована для поиска оптимального число контейнеров, обоснования выбора политики балансировки с учётом профиля отказов и для определения пороговой нагрузку, при превышении которой качество обслуживания резко деградирует.

Методическая часть работы включает верификацию программной реализации по аналитической модели M/M/c (относительная погрешность среднего времени ожидания не более 6 %), а также переход на pooled-расчёт хвостовых характеристик по объединённой выборке всех прогонов, что устраняет систематическое смещение, возникающее при усреднении процентилей по реализациям. Для обеспечения воспроизводимости результатов исходный код модели и сценарии экспериментов опубликованы в открытом репозитории GitHub: https://github.com/Mamedov14/cluster_sim.

Библиография

[1] Maenhaut P. J., Volckaert B., Ongenae V., De Turck F. Resource Management in a Containerized Cloud: Status and Challenges // *Journal of Network and Systems Management*. 2020. Vol. 28(2). P. 197–246. DOI: 10.1007/s10922-019-09504-0.

[2] Kelton W., Sadowski R., Zupick N. *Simulation with Arena*. New York: McGraw-Hill Education, 2015. 640 p.

[3] Law A. M. *Simulation Modeling and Analysis*. New York: McGraw-Hill Education, 2014. 784 p.

[4] Клейнрок Л. *Теория массового обслуживания.* Т. 1. М.: Машиностроение, 1979. 432 с.

[5] Kumar D., Ravi V. A survey on fault tolerance in cloud computing // *Journal of Cloud Computing*. 2018. Vol. 7(1).

[6] Matloff N. *Introduction to Discrete-Event Simulation and the SimPy Language*. Davis: University of California, 2008.
[7] Avizienis A., Laprie J.-C., Randell B., Landwehr C. Basic concepts and taxonomy of dependable and secure computing // *IEEE Trans. Dependable and Secure Computing*. 2004. Vol. 1(1). P. 11–33.
[8] Schroeder B., Gibson G. A large-scale study of failures in high-performance computing systems // *IEEE Trans. Dependable and Secure Computing*. 2010. Vol. 7(4). P. 337–350.
[9] Богатырёв В. А., Богатырёв А. В., Богатырёв С. В. Оценка надёжности выполнения кластерами запросов реального времени // *Изв. вузов. Приборостроение*. 2014. Т. 57. № 4. С. 46–48.
[10] Богатырёв В. А. Комбинаторно-вероятностная оценка надёжности и отказоустойчивости кластерных систем // *Приборы и системы. Управление, контроль, диагностика*. 2006. № 6. — С. 21–26.
[11] Bogatyrev V. A., Derkach A. N., Bogatyrev S. V. Timeliness of the Reserved Maintenance by Duplicated Computers of Heterogeneous Delay-Critical Stream // *CEUR Workshop Proceedings*. ISTMC, 2019. P. 26–36.
[12] Hwang J., et al. IASO: A Framework for Mitigating the Impact of Fail-Slow in Distributed Storage Services // *USENIX ATC'19*. 2019.
[13] Koutras M. A. Markov regenerative process model for performability evaluation of a computer cluster system // *Reliability Engineering & System Safety*. 2023.
[14] Фунг В. К., Богатырёв В. А., До М. К. Имитационная модель вычислительного кластера с контейнерной виртуализацией // *Вестник компьютерных и информационных технологий*. 2025. Т. 22. № 8. С. 3–12. DOI: 10.14489/vkit.2025.08.pp.003-012
[15] Dean J., Barroso L. The tail at scale // *Communications of the ACM*. 2013. Vol. 56(2). P. 74–80.
[16] Фунг В. К., Богатырёв В. А. Экспериментальное исследование производительности кластера с контейнерной виртуализацией // *Изв. вузов. Приборостроение*. 2024. Т. 67. № 8. С. 647–656. DOI: 10.17586/0021-3454-2024-67-8-647-656
[17] Фунг В. К., Богатырёв В. А., Кармановский Н. С., Лэ В. Х. Оценка вероятностно-временных характеристик компьютерной системы с контейнерной виртуализацией // *Науч.-техн. вестник информационных технологий, механики и оптики*. 2024. Т. 24. № 2. С. 249–255. DOI: 10.17586/2226-1494-2024-24-2-249-255
[18] Zhang T., Sharma U., Kapritsos M. Performal: Formal Verification of Latency Properties for Distributed Systems // *Proceedings of the ACM on Programming Languages*. 2023. Vol. 7. Art. 121. P. 1–26. DOI: 10.1145/3591249.
[19] Zhao K., Goyal P., Alizadeh M., Anderson T. E. Scalable Tail Latency Estimation for Data Center Networks *// 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA, 2023. P. 685–702.
[20] Ledmi A., Bendjenna H., Hemam S. M. Fault Tolerance in Distributed Systems: A Survey // *Proc. 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)*. IEEE, 2018. P. 1–5. DOI: 10.1109/PAIS.2018.8598484.
[21] Gunawi H. S., Suminto R. O., Sears R. et al. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems // ACM Transactions on Storage. 2018. Vol. 14, no. 3. Art. 23. P. 1–26. DOI: 10.1145/3242086.
[22] Lu R., Xu E., Zhang Y. et al. Perseus: A Fail-Slow Detection Framework for Cloud Storage Systems // Proc. 21st USENIX Conference on File and Storage Technologies (FAST '23). Santa Clara: USENIX Association, 2023. P. 49–64. (Best Paper Award).
[23] Lou C., Jing Y., Huang P. Demystifying and Checking Silent Semantic Violations in Large Distributed Systems // Proc. 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22). Carlsbad: USENIX Association, 2022. P. 91–107.
[24] Tirmazi M., Barker A., Deng N. et al. Borg: the Next Generation // Proc. 15th European Conference on Computer Systems (EuroSys '20). Heraklion: ACM, 2020. Art. 30. DOI: 10.1145/3342195.3387517.
[25] Beyer B., Murphy N. R., Rensin D. K., Kawahara K., Thorne S. The Site Reliability Workbook: Practical Ways to Implement SRE. Sebastopol: O'Reilly Media, 2018. ISBN 978-1-491-92521-7.

Мамедов Вагиф Вагифович — магистр, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация; vagif_m@bk.ru, https://orcid.org/0009-0000-8487-7552

Богатырев Владимир Анатольевич, доктор технических наук, профессор, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, vladimir.bogatyrev@gmail.com, https://orcid.org/0000-0003-0213-022.

До Мань Кием — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация; domanhkiemnd@gmail.com, https://orcid.org/0009-0002-8307-5544.

Имитационное моделирование отказоустойчивого вычислительного кластера с двухуровневой балансировкой нагрузки и контейнерной виртуализацией

Vagif Mamedov, Vladimir Bogatyrev, Do Manh Kiem

Abstract - This paper investigates the fault tolerance of a computing cluster with container virtualization and a dedicated load balancer. A discrete-event simulation model is proposed and implemented in Python/SimPy, incorporating the load balancer as a vulnerable single point of failure (SPOF, Level 0) and a pool of worker nodes with containers (Level 1). Unlike classical queueing models, each server undergoes a realistic cascade of hardware/software aging UP → DEGRADE → DOWN, with a host-node failure physically blocking its containers and their migration. Container service rate is dynamically derived from an experimentally measured performance matrix μ(n, m). To capture the true delay profile, the architecture employs infinite persistent queues combined with strict rejection of new requests during hardware downtimes. The software implementation is verified against the analytical M/M/c model in a no-failure regime, yielding a relative error of mean waiting time below 6 %. Tail characteristics are computed over the pooled sample of all runs (pooled metric), eliminating the systematic bias introduced by averaging per-run percentiles. Four series of experiments are conducted (10 Monte Carlo runs each): influence of MTTFfail with three cascade-aging profiles $p \in \{0.7, 0.8, 0.9\}$, MTTR, balancer routing policies, and arrival intensity $\lambda \in [10, 30]$ tasks/s. Key result: accounting for hidden SPOF-node degradation critically shifts the cluster stability boundary; a saturation threshold $\lambda^* \approx 22–26$ tasks/s is observed, beyond which the pooled 95-th percentile of waiting time $W_{0,95}$ grows from 0.11 s at $\lambda = 10$ tasks/s to 109 s at $\lambda = 30$ tasks/s. The model source code and experiments are published in an open repository. The proposed model and the obtained data form a foundation for the synthesis of predictive routing policies and proactive cluster resource management.

***Keywords* - simulation modeling; fault tolerance; computing cluster; container virtualization; load balancing; SimPy**

REFERENCES

[1] Maenhaut P. J., Volckaert B., Ongenae V., De Turck F. Resource Management in a Containerized Cloud: Status and Challenges // Journal of Network and Systems Management. 2020. Vol. 28(2). P. 197–246. DOI: 10.1007/s10922-019-09504-0.

[2]Kelton W., Sadowski R., Zupick N. Simulation with Arena. New York: McGraw-Hill Education, 2015. 640 p.

[3]Law A. M. Simulation Modeling and Analysis. New York: McGraw-Hill Education, 2014. 784 p.

[4]Kleinrock L. Queueing Systems. Vol. 1. New York: Wiley, 1975. 432 p.

[5]Kumar D., Ravi V. A survey on fault tolerance in cloud computing // Journal of Cloud Computing. 2018. Vol. 7(1).

[6]Matloff N. Introduction to Discrete-Event Simulation and the SimPy Language. Davis: University of California, 2008.

[7]Avizienis A., Laprie J.-C., Randell B., Landwehr C. Basic concepts and taxonomy of dependable and secure computing // IEEE Trans. Dependable and Secure Computing. 2004. Vol. 1(1). P. 11–33.

[8]Schroeder B., Gibson G. A large-scale study of failures in high-performance computing systems // IEEE Trans. Dependable and Secure Computing. 2010. Vol. 7(4). P. 337–350.

[9]Bogatyrev V. A., Bogatyrev A. V., Bogatyrev S. V. Reliability assessment of cluster execution of real-time requests // Izv. Vyssh. Uchebn. Zaved. Priborostroenie. 2014. Vol. 57. No. 4. P. 46–48.

[10] Bogatyrev V. A. Combinatorial-probabilistic assessment of reliability and fault tolerance of cluster systems // Pribory i Sistemy. Upravlenie, Kontrol', Diagnostika. 2006. No. 6. P. 21–26.

[11] Bogatyrev V. A., Derkach A. N., Bogatyrev S. V. Timeliness of the Reserved Maintenance by Duplicated Computers of Heterogeneous Delay-Critical Stream // CEUR Workshop Proceedings. ISTMC, 2019. P. 26–36.

[12] Hwang J., et al. IASO: A Framework for Mitigating the Impact of Fail-Slow in Distributed Storage Services // USENIX ATC'19. 2019.

[13] Koutras M. A. Markov regenerative process model for performability evaluation of a computer cluster system // Reliability Engineering & System Safety. 2023.

[14] Fung V. K., Bogatyrev V. A., Do M. K. Simulation model of a computing cluster with container virtualization // Vestnik Komp′yuternykh i Informatsionnykh Tekhnologiy. 2025. Vol. 22. No. 8. P. 3–12. DOI: 10.14489/vkit.2025.08.pp.003-012

[15] Dean J., Barroso L. The tail at scale // Communications of the ACM. 2013. Vol. 56(2). P. 74–80.

[16] Fung V. K., Bogatyrev V. A. Experimental study of cluster performance with container virtualization // Izv. Vyssh. Uchebn. Zaved. Priborostroenie. 2024. Vol. 67. No. 8. P. 647–656. DOI: 10.17586/0021-3454-2024-67-8-647-656

[17] Fung V. K., Bogatyrev V. A., Karmanovsky N. S., Le V. H. Probabilistic-temporal characteristics of a computer system with container virtualization // Nauchno-Tekhnicheskiy Vestnik IT, Mekhaniki i Optiki. 2024. Vol. 24. No. 2. P. 249–255. DOI: 10.17586/2226-1494-2024-24-2-249-255

[18] Zhang T., Sharma U., Kapritsos M. Performal: Formal Verification of Latency Properties for Distributed Systems // Proc. ACM on Programming Languages. 2023. Vol. 7. Art. 121. P. 1–26. DOI: 10.1145/3591249.

[19] Zhao K., Goyal P., Alizadeh M., Anderson T. E. Scalable Tail Latency Estimation for Data Center Networks // 20th USENIX NSDI. Boston, MA, 2023. P. 685–702.

[20] Ledmi A., Bendjenna H., Hemam S. M. Fault Tolerance in Distributed Systems: A Survey // Proc. 3rd Intl. Conf. Pattern Analysis and Intelligent Systems (PAIS). IEEE, 2018. P. 1–5. DOI: 10.1109/PAIS.2018.8598484.

[21] Gunawi H. S., Suminto R. O., Sears R. et al. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems // ACM Transactions on Storage. 2018. Vol. 14, no. 3. Art. 23. P. 1–26. DOI: 10.1145/3242086.

[22] Lu R., Xu E., Zhang Y. et al. Perseus: A Fail-Slow Detection Framework for Cloud Storage Systems // Proc. 21st USENIX Conference on File and Storage Technologies (FAST '23). Santa Clara: USENIX Association, 2023. P. 49–64. (Best Paper Award).

[23] Lou C., Jing Y., Huang P. Demystifying and Checking Silent Semantic Violations in Large Distributed Systems // Proc. 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22). Carlsbad: USENIX Association, 2022. P. 91–107.
[24] Tirmazi M., Barker A., Deng N. et al. Borg: the Next Generation // Proc. 15th European Conference on Computer Systems (EuroSys '20). Heraklion: ACM, 2020. Art. 30. DOI: 10.1145/3342195.3387517.
Beyer B., Murphy N. R., Rensin D. K., Kawahara K., Thorne S. The Site Reliability Workbook: Practical Ways to Implement SRE. Sebastopol: O'Reilly Media, 2018. ISBN 978-1-491-92521-7

Mamedov Vagif Vagifovich, Master's Student, ITMO University, St. Petersburg, 197101, Russian Federation; vagif_m@bk.ru, https://orcid.org/0009-0000-8487-7552
Bogatyrev Vladimir Anatolyevich, D.Sc., Professor, ITMO University, Saint Petersburg, 197101, Russian Federation, 197101, Russian Federation, vladimir.bogatyrev@gmail.com , https:// orcid.org/0000-0003-0213-0223,
Do Manh Kiem, PhD Student, ITMO University, St. Petersburg, 197101, Russian Federation; domanhkiemnd@gmail.com, https://orcid.org/0009-0002-8307-5544