Викторов А.С. Метод и алгоритм обнаружения сбоев в работе вычислительного кластера периферийного сервиса для предотвращения каскадных отказов // Информационно-экономические аспекты стандартизации и технического регулирования, 2018. № 6(46).

УДК 004.65

МЕТОД И АЛГОРИТМ ОБНАРУЖЕНИЯ СБОЕВ В РАБОТЕ ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА ПЕРИФЕРИЙНОГО СЕРВИСА ДЛЯ ПРЕДОТВРАЩЕНИЯ КАСКАДНЫХ ОТКАЗОВ

Викторов А.С., соискатель, ВИНИТИ РАН.

В статье рассматривается реализация механизма предотвращения каскадных отказов и обнаружения нарушений в функционировании узлов вычислительной сети периферийного сервиса распределенной обработки данных, а также набор метрик, используемый для оценки состояния вычислительных узлов. При разработке реализации рассматриваемого механизма предотвращения отказов на основе анализа различных подходов был произведен выбор метода, обеспечивающего надежный уровень защиты вычислительной сети периферийного сервиса от каскадных отказов и стабильную производительность.

Ключевые слова: каскадный отказ, fail-fast принцип, отказоустойчивая система, шаблон проектирования «Circuit Breaker», «Netflix Hystrix», периферийный сервис.

UDC 004.65

THE FAULT DETECTION METHOD IN A COMPUTING CLUSTER OF EDGE SERVICE FOR CASCADING FAILURES PREVENTION

Viktorov A.S., post-graduate student, VINITI RAS.

This paper presents cascading failures prevention and fault detection system implementation developed for providing a high fault tolerance and resilience for computing cluster of edge service designed for unmanned aerial vehicle telemetry data processing, and also a metrics set used for nodes state evaluation. Design of the considered cascading failure prevention and fault detection system based on a resulting approach which was developed by the analysis of various approaches described in modern literature and which be able to provide a stable performance and reliable protection level of the computing cluster of the edge service from cascading failures.

Keywords: cascading failures, fail-fast principle, fault-tolerant system, «Circuit Breaker» design pattern, «Netflix Hystrix», edge service.

В статье рассматривается реализация механизма предотвращения каскадных отказов и обнаружения нарушений в функционировании узлов вычислительной сети, что обеспечивает стабильную производительность и надежную работу распределенного приложения, выполняющегося в распределенной среде вычислительной сети. Для обеспечения стабильной и надежной работы распределенного приложения во время его выполнения необходимо стремиться к повышению отказоустойчивости и достижению согласованности в обработке пакетов данных на вычислительном кластере [1]. Так как при выполнении распределенного приложения возможно возникновение нештатных ситуаций, при возникновении которых вычислительная сеть перестанет функционировать полностью или начинает функционировать неправильно, в результате отказа компонентов вычислительной сети или предоставлении вычислительными узлами некорректной информации. Существуют различные сценарии нештатного функционирования вычислительной сети [2], которые либо связаны с полным отказом или частичным выходом из строя отдельных компонентов вычислительной сети, например:

1. выходом из строя отдельных вычислительных узлов сети;
2. выходом из строя коммуникационного оборудования;
3. отказом программного обеспечения, установленного на вычислительном узле;
4. возможно возникновение ситуации, когда узел продолжает работать, но функционирует с ошибками и предоставляет неверную информацию.

При этом наиболее серьезной проблемой, в связи с неочевидностью ее проявлений и вычислительными затратами, связанными с ее детектированием, является представление вычислительными узлами искаженной информации или ее искажение во время передачи между узлами, возникающие в результате ошибок в работе коммуникационного оборудования. Решением проблемы представления вычислительными узлами

искаженной информации является применение техники дублирования вычислений (применение избыточных вычислений), суть которой заключается в выполнении одного и того же задания разными вычислительными узлами, при этом если результаты вычислений совпадают, то все узлы, выполняющие одно и тоже задание, работают корректно. Но данный подход ресурсоемкий, поэтому целесообразно детектировать подозрительную активность вычислительных узлов и после чего производить проверку подозрительных узлов. Также возможна проверка вычислительных узлов в соответствии с заданным расписанием, при этом для проверки правильности функционирования узла могут использоваться тестовые задания с известным решением.

Также в распределенных вычислительных системах особенно важно предотвращение возникновения ситуаций каскадного отказа вычислительных узлов. Причиной каскадного отказа является выход из строя одного или нескольких вычислительных узлов системы и/или аномальное отсутствие длительное время отклика [5], от вышедшего из строя узла или узлов. Например, вычислительные узлы, ожидающие отклика от вышедшего из строя узла, вынуждены простаивать в ожидание отсутствующего отклика, что приводит в свою очередь к простаиванию узлов, зависящих от работоспособности ожидающих узлов и так далее по цепочке [3], что приводит к потере работоспособности остальных узлов вычислительной сети. Массовая потеря работоспособности вычислительных узлов и отсутствие отклика от них в итоге приводит к нарушению нормального функционирования вычислительной сети или полному ее отказу. Для предотвращения возникновения ситуации каскадного отказа используются различные техники, одной из широко используемых техник является применение программного шаблона «Circuit Breaker» , диаграмма состояний, которого изображена на рисунке 1.

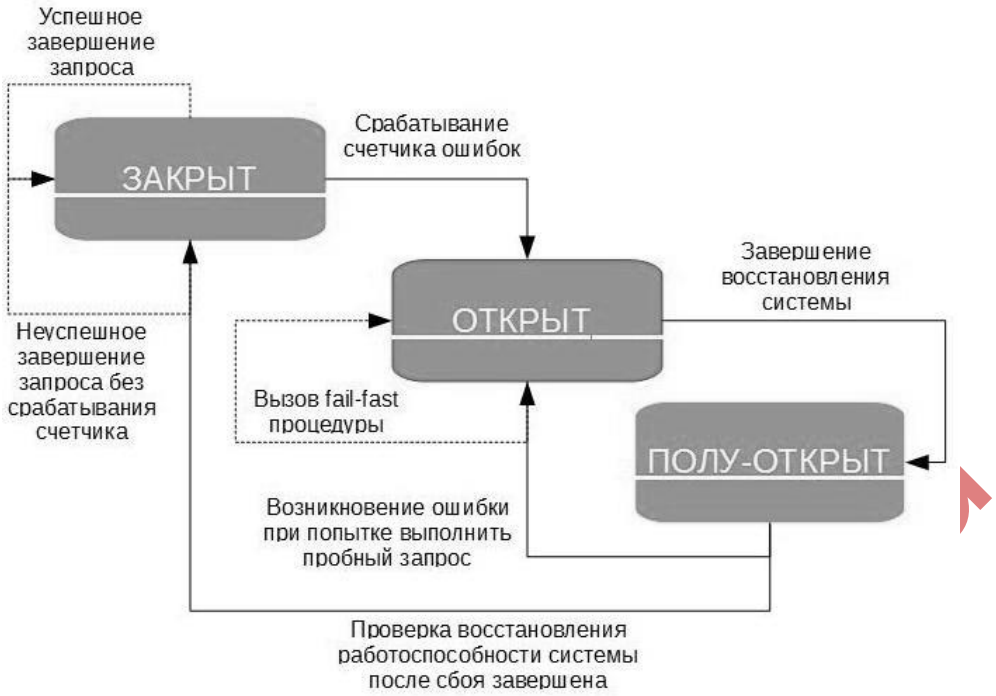

Рисунок 1. Диаграмма состояний «Circuit Breaker»

Логика работы механизма, описанного шаблоном проектирования «Circuit Breaker», основана на «fail-fast» принципе, который описывает поведение системы в случае возникновения ошибки в работе системы или при выявлении перехода системы в предаварийное состояние [5], которое может привести к ошибке в работе системы. Данное поведение характеризуется тем, что при возникновении ошибки или выявлении предаварийного состояния системы с целью предотвращения простоя и не рационального использования критически важных аппаратных ресурсов вычислительных узлов, взаимодействующих с системой, система немедленно завершает свою работу и как возможно быстрее для предотвращения негативных последствий уведомляет об обнаружении ошибки в работе системы или предаварийном состоянии взаимодействующих с ней вычислительных узлов.

Шаблон проектирования «Circuit Breaker» позволяет предотвращать попытки приложения выполнить операцию, которая с высокой долей

вероятности завершится неудачно, тем самым позволяя экономить вычислительные ресурсы. Работу программного шаблона «Circuit Breaker» можно описать в виде алгоритма функционирования конечного автомата [7], который может находиться в следующих возможных состояниях:

1. Состояние «Closed» характеризуется тем, что запрос от клиентского приложения напрямую направляется к целевому сервису. Если операция завершилась не успешно, либо по причине, связанной с превышением заданного лимита времени ожидания отклика, или при получении уведомления от целевого сервиса об ошибке, увеличивается значение счетчика ошибок. Если значение счетчика ошибок за некоторые временной интервал превышает заранее заданное пороговое значение, то «Circuit Breaker» переходит в состояние «Open».

2. Состояние «Open» характеризуется тем, что запрос от клиента к сервису блокируется и немедленно завершается ошибкой. Клиенту отправляется сообщение о недоступности сервиса в данный момент времени. При переходе в данное состояние из состояния «Closed» может производиться запуск механизма восстановления сервиса [6] после произошедшего сбоя. При переходе в данное состояние производиться запуск таймера, при срабатывании которого по истечении заданного интервала времени «Circuit Breaker» переходит в состояние «Half-Open», или может периодически производится опрос сервиса, при этом при получении отклика о успешном восстановлении сервиса «Circuit Breaker» переходит в состояние «Half-Open».

3. Состояние «Half-Open» характеризуется тем, что ограниченное число запросов от клиентского приложения разрешается направить на обработку сервису. Сервис на каждый полученный запрос отправляет отклик, содержащий информацию о результатах выполнения запроса. Если сервис производит обработку всех сообщений без сбоев, то механизм, основанный на шаблоне «Circuit Breaker», переходит в состояние «Open» при этом

производится сброс счетчиков, производящих учет ситуаций, указывающих на возможность отказа сервиса. Но в случае получения хотя бы одного отклика о неудачном завершении обработки запроса механизм «Circuit Breaker» переходит в открытое состояние. Нахождение механизма «Circuit Breaker» в данном переходном состоянии обеспечивает плавное восстановление работоспособности сервиса, предотвращая быстрый рост количества запросов к сервису, а также позволяет удостовериться в полном восстановлении сервиса перед возвращением сервиса в режим нормального функционирования.

Программная реализация механизма, основанного на шаблоне «Circuit Breaker», может быть разработана [7]: для выполнения на вычислительных мощностях, на которых выполняется клиентское приложение; для выполнения на вычислительных мощностях, на которых выполняется сервисное приложение; в виде программного обеспечения для отдельного прокси-сервера, выступающего в роли посредника между клиентским приложением и сервисом. При реализации механизма «Circuit Breaker» для выполнения на стороне клиентского приложения для каждого внешнего сервиса, к которому отправляет запрос клиентское приложение, создается отдельный объект «Circuit Breaker». Преимуществом данной стратегии является разгрузка сервиса от дополнительной вычислительной нагрузки и отсутствие запросов к сервису при нахождении «Circuit Breaker» в открытом состоянии. В качестве недостатка данного подхода можно указать зависимость актуальности состояния «Circuit Breaker» от того насколько часто производится запрос к сервису от клиентского приложения. При реализации шаблона «Circuit Breaker» для выполнения на стороне сервисного приложения при получении запроса от клиентского приложения на основании состояния объекта «Circuit Breaker» принимается решение произвести обработку запроса или нет. Преимуществом данного подхода является возможность обнаружения злонамеренной активности и изоляции

клиентского приложения, которое производит отправку таких запросов. Также стоит отметить, что встроенные механизмы сервиса могут производить статистический анализ сетевой активности и при необходимости предотвращать перегрузку сервиса путем блокировки части клиентских запросов. В качестве недостатка данного подхода можно указать необходимость задействования вычислительных ресурсов для анализа входящих запросов даже при условии нахождения механизма «Circuit Breaker» в открытом состоянии. При реализации шаблона «Circuit Breaker» в виде программного обеспечения для отдельного прокси-сервера, для каждого клиента и сервиса создается отдельный объект «Circuit Breaker», при этом для запуска процесса обработки запроса оба объекта «Circuit Breaker», соответствующих клиенту и сервису, должны находиться в закрытом состоянии. Преимуществом данного подхода является то, что клиент и сервис непосредственно изолированы друг от друга, что предотвращает неблагоприятное воздействие клиента и сервиса друг на друга. Существенным недостатком данного подхода является то, что отдельный прокси-сервер является единственной точкой отказа.

Причиной отказа распределенной вычислительной системы может являться перегрузка вычислительных узлов распределенного сервиса клиентскими запросами. Причиной перегрузки вычислительных узлов [7] может являться выход из строя одного или нескольких вычислительных узлов, производящих обработку клиентских запросов, что в свою очередь приводит к перераспределению вычислительной нагрузки сервисом балансировки на оставшиеся работоспособными вычислительные узлы. Увеличение вычислительной нагрузки, приходящейся на оставшиеся работоспособными вычислительные узлы, приводит к перегрузке аппаратных ресурсов узлов и увеличению вероятности отказа оставшихся работоспособными узлов и соответственно всего сервиса. В частности, при увеличении вычислительной нагрузки, приходящейся на обрабатывающий

запросы вычислительный узел: увеличивается время отклика узла, что приводит к нерациональному использованию вычислительных мощностей узлов, ожидающих отклика; возможно переполнение доступной памяти перегруженного вычислительного узла. Для предотвращения отказа сервиса из-за перегрузки клиентскими запросами, используется специальная техника «Load Shedding» («Сброс Нагрузки»), смысл которой заключается в игнорировании части клиентских запросов и сохранении работоспособности сервиса.

Для мониторинга работоспособности вычислительных узлов, предотвращения каскадных отказов и обнаружения нарушений в функционировании вычислительных узлов, которые возникают в результате отказа оборудования или программного обеспечения вычислительной сети используется специальный механизм, реализованный на основе библиотеки «Netflix Hystrix» [8], которая содержит реализацию программного шаблона «Circuit Breaker». В библиотеке «Netflix Hystrix» используется поведенческий шаблон проектирования «Command», который предназначен для реализации программных компонентов, в которых класс-отправитель и класс-получатель не зависят друг от друга напрямую. В библиотеке «Netflix Hystrix» также предусмотрена возможность вызова предопределенного метода в случае неудачного выполнения клиентского запроса. По результатам обработки получаемых запросов сервис «Netflix Hystrix» производит расчет набора метрик необходимых для анализа правильности функционирования вычислительных узлов кластера. Ниже будет рассмотрен мониторинг на основе данных метрик.

Стандартная реализация механизма предотвращения сбоев предусмотренная в библиотеке «Hystrix» основана на отслеживании вызовов защищенных команд сервиса внутри распределенной вычислительной сети, каждая из которых представлена классом, который расширяет один из предусмотренных в библиотеке «Hystrix» абстрактных классов, и вычислении

возможности отказа сервиса, производящего обработку вызовов, генерируемых различными узлам. При возникновении исключения или неприемлемой временной задержки во время обработки вызовов производится инкрементация счетчика, производящего учет ситуаций, указывающих на возможность отказа сервиса. При превышении значения счетчика заданного порогового значения или выполнении иного предопределенного условия, происходит смена состояния «Circuit Breaker» на «Open». В этом состоянии все запросы, идущие к сервису, вышедшему из строя, аннулируются и клиенту узлу отправляются сообщение о произошедшем отказе. Для реализации механизма генерации, отслеживания состояния и анализа вызовов внутри распределенной вычислительной сети в библиотеке «Hystrix» используются различные шаблоны проектирования, например «Command» и «Observer». Библиотека «Hystrix» [8] содержит реализацию программного компонента для мониторинга нагрузки внешних систем, что позволяет произвести оптимизацию пулов вычислительных процессов в распределенном приложении. Для каждой группы команд «Hystrix» установлены ограничения на уровень конкуренции, то есть максимальное количество команд из одной группы. Библиотека «Hystrix» поддерживает различные режимы исполнения команд: синхронный, асинхронный, реактивный режим. Особенно стоит рассмотреть реактивный режим исполнения команды [4], который позволяет производить мониторинг исполнения команды. Данный режим исполнения команды реализован с использованием механизмов библиотеки «RxJava», которая реализует принципы реактивного программирования. В частности фундаментальными инструментами, которые обеспечивает функционал реализации концепции реактивного программирования в «RxJava», являются класс «Observable» и интерфейс «Observer». Рассматриваемые инструменты реализуют программный паттерн «Observer». Класс «Observable» при изменении своего состояния отвечает за оповещение о произошедшем событии всех зависящих

от него объектов, реализующих интерфейс «Observer» и подписанных на оповещение. В интерфейсе «Observer» описано поведению по умолчанию, которое определяет порядок оповещения наблюдателя о произошедшем событии заданного типа. При этом класс, реализующий интерфейс «Observer», является объектом, который при получении сообщения от объекта типа «Observable» реагирует на него в соответствии с заложенной в нем логикой. Любое событие в потоке событий, генерируемом объектом типа «Observable», относятся к одному из следующих типов:

1. событие типа «Next», которое генерируется каждый раз, когда становиться доступна очередная порция данных;
2. событие типа «Error», которое генерируется при возникновении ошибки;
3. событие типа «Completed», которое генерируется при завершении процесса обработки вызова.

Работу механизма предотвращения отказов «Hystrix» при отправке запроса от одного сервиса к другому можно представить в виде следующей последовательности действий:

1. Отправленный запрос перехватывается механизмом предотвращения отказов «Hystrix»;
2. Проводится проверка состояния «Circuit Breaker»;
3. Если «Circuit Breaker» не в состоянии «Closed», то производится вызов резервного метода;
4. Если «Circuit Breaker» в состоянии «Closed», то производится проверка на наличие свободного потока в пуле потоков;
5. Если свободных потоков нет, то производится вызов резервного метода;
6. При наличии свободного потока в пуле выбранному потоку назначается на исполнение задание на обработку запроса;

7. Если сервис, которому отправлен запрос, не отвечает в течение заданного интервала времени или происходит сбой, то производится вызов резервного метода. Значение счетчика ошибок инкрементируется при этом, если количество ошибок за определенный интервал времени превышает заданное пороговое значение, то механизм предотвращения отказов «Hystrix» меняет состояние «Circuit Breaker» на «Open»;

8. Если запрос завершился удачно и был обработан во время, отклик на запрос передается вызвавшему сервису.

Изменение поведения по умолчанию механизма предотвращения отказов «Hystrix» возможно при помощи регистрации пользовательских плагинов посредством загрузчика «HystrixPlugins». Для оценки состояния системы во время ее функционирования необходимо выделить набор событий, появление которых указывает на неправильное функционирование системы, и на их основе сформировать набор метрик, позволяющих идентифицировать аварийное состояние системы. В «Netflix Hystrix» определены следующие типы событий, оповещение о которых предусмотрено в случае вызова удаленного метода командой «HystrixObservableCommand»:

1. «EMIT» – в результате вызова команды «Hystrix» было возвращено значение;

2. «SUCCESS» – команда завершена без ошибок;

3. «FAILURE» – во время выполнения команды было выброшено исключение;

4. «TIMEOUT» – исполнение команды началось, но не было завершено вовремя;

5. «BAD_REQUEST» – запрос не был выполнен из-за некорректного сформированного содержимого, было выброшено исключение «HystrixBadRequestException». Данное событие не влияет на состояние счетчика ошибок и не может привести к его срабатыванию;

6. «SHORT_CIRCUITED» – «Circuit Breaker» находится в состоянии «Open», попытка выполнения запроса была мгновенно заблокирована;

7. «THREAD_POOL_REJECTED» – запрос не был выполнен из-за того, что все потоки в пуле потоков заняты;

8. «SEMAPHORE_REJECTED» – выполнение запроса было заблокировано семафором.

При реализации механизма предотвращения отказов для учета задержек в обработке запросов был выбран нестандартный подход, учитывающий частоту возникновения отказов и интенсивность запросов. Перед началом работы алгоритма на основании исторических данных о сбоях в работе сервиса, к которому будет отправляться запрос, высчитывается среднее время выполнения запроса данным сервисом, на основании чего вычисляется пороговое время ожидания выполнения запроса. Причиной сбоя при выполнении запроса может быть как превышение заданного интервала ожидания, на которое влияет интенсивность запросов получаемых обрабатывающим сервисом, так и возникновение исключений или ошибок, на которое имеет сильное влияние трудно поддающиеся учету факторы. Поэтому для предсказания сбоя при выполнении запроса вероятность его возникновения можно представить в виде суммы:

$$P_{failure} = P(t_{exec} > t_{timeout}) + P_{other} \quad (1)$$

где $P(t_{exec} > t_{timeout})$ – вероятность того, что время выполнения запроса превысит время ожидания; P_{other} – вероятность того, что во время выполнения запроса возникнет исключение или ошибка. Так как время, за которое запрос будет обработан сервисом, зависит от интенсивности запросов, поступающих на обрабатывающий сервис от различных узлов вычислительной сети, то значение вероятности $P(t_{exec} > t_{timeout})$ динамически изменяется во время функционирования сети. Из чего можно сделать вывод, что в процессе

работы алгоритма предотвращения сбоев возникает необходимость модификации параметров используемой для прогнозирования вероятностной модели. На основе анализа различных вероятностных моделей, используемых для предсказания сбоев в работе различных систем, было принято решение использовать нормальное вероятностное распределение для оценки вероятности $P(t_{exec} > t_{timeout}) = 1 - F_{t_{exec}}(t_{timeout})$, где $F_{t_{exec}}(t_{timeout})$ – функция распределения времени выполнения запроса для нормального закона распределения. В разработанном механизме предотвращения сбоев используется стратегия блокировки запроса к обрабатывающему сервису, если существуют высокая вероятность того, что запрос завершится сбоем. Каждый раз после выполнения запроса производится обновление распределения используемого для оценки времени выполнения запросов. Если после обновления параметров распределения вычисленное значение $P_{predict}(t_{exec} > t_{timeout})$ больше предопределенного порогового значения вероятности сбоя $P_{tresh}(t_{exec} > t_{timeout})$, то состояние «Circuit Breaker» изменяется на «Open». Обновление параметров распределения, используемого для оценки времени выполнения запросов, с целью уменьшения пространственной сложности (которая увеличивается при хранении замеров времени выполнения запросов) производится по следующим формулам:

$$M_{new}(t_{exec}) = M_{old}(t_{exec}) - \frac{t_{sample,\,exec}}{N} + \frac{t_{last,\,exec}}{N} \qquad (2)$$

$$D_{new}(t_{exec}) = D_{old}(t_{exec}) - \frac{(t_{sample,\,exec})^2}{N} + \frac{(t_{last,\,exec})^2}{N} + (M_{old}(t_{exec}))^2 - (M_{new}(t_{exec}))^2 \qquad (3)$$

, где $M_{old}(t_{exec})$ – значение математического ожидания времени выполнения запроса до обновления; $M_{new}(t_{exec})$ – значение математического ожидания времени выполнения запроса после обновления; где $D_{old}(t_{exec})$ – значение дисперсии времени выполнения запроса до обновления; $D_{new}(t_{exec})$ – значение дисперсии времени выполнения запроса после обновления; $t_{sample,\,exec}$ –

значение полученное при помощи метода сэмплирования из распределения времени выполнения запроса (для упрощения можно принять равным $M_{old}(t_{exec})$); $t_{last,exec}$ – время выполнения последнего запроса; N – количество наблюдений, используемое для вычисления параметров распределения времени выполнения запроса.

Так же «Circuit Breaker» переходит в состояние «Open», если время обработки запроса превысило интервал ожидания. Оценка времени нахождения «Circuit Breaker» в состоянии «Open» T_{open} производится таким образом, чтобы минимизировать суммарное время нахождения в данном состоянии и максимизировать интервал времени между сбоями за заданное число наблюдений N. При этом принимаем, что значение интервала времени между запросами, распределено в соответствии с нормальным распределением. При совершении сбоя производится замер интервала времени между новым и предыдущим сбоем и обновление параметров распределения. Также принимаем, что значение интервала времени необходимого для восстановления сервиса после сбоя, распределено в соответствии с логнормальным распределением. После каждого сбоя производим обновление параметров логнормального распределения, после чего находим значение интервала восстановления из распределения, как наиболее вероятное значение. Так как значение случайной величины, моделируемой логнормальным распределением, неотрицательны и распределение характеризуется правосторонней асимметрией, то логнормальное распределение подходит для реализации алгоритма, осуществляющего поиск оптимального значения интервала времени восстановления. Для поиска оптимального значения интервала времени необходимого для восстановления работоспособности обрабатывающего сервиса осуществляем поиск параметров логнормального распределения. Поиск параметров осуществляется методом стохастического градиента, при

этом минимизируем функционал $\log\left(chx(\frac{T_{srt}}{T_{tbf}}\cdot\frac{N_f}{N})\right)$, где T_{srt} – суммарное время восстановления сервиса после сбоев (когда сервис был недоступен) за заданное число наблюдений N; T_{tbf} – суммарное время между сбоями, когда сервис был доступен, за заданное число наблюдений N; N_f – количество сбоев за заданное число наблюдений N. Для предотвращения сбоев, не связанных с перегрузкой вычислительными задачами обрабатывающего сервиса, используется стандартная реализация алгоритма предотвращения отказов, предусмотренная в библиотеке «Hystrix».

В результате анализа различных подходов, применяемых для обнаружения и предсказания сбоев в работе узлов распределенной вычислительной сети с целью предотвращения каскадных отказов, был разработан алгоритм, при помощи которого можно предсказать вероятность того, что обработка входящего вызова сервисом не будет завершена в срок. Реализация предложенного алгоритма повышает отказоустойчивость вычислительной сети и предотвращает возникновение каскадных отказов.

Список использованных источников и литературы

1. Артамонов Ю.С., Востокин С.В. Разработка распределенных приложений сбора и анализа данных на базе микросервисной архитектуры // Известия Самарского научного центра РАН. 2016. №4-4. С. 688-693.
2. Наумов А.В., Рыбалко А.А. Модель обеспечения отказоустойчивости контейнерных виртуальных сервисов в центрах обработки данных // Труды МАИ. 2017. №97. С. 20-43.
3. Фетцер К. Создание критически важных приложений на основе микросервисов // Открытые системы. СУБД Издательство «Открытые системы». - 2017. - №1. - С. 25-27.
4. Bonér J. Reactive Microsystems The Evolution of Microservices at Scale. - 1 изд. - California, USA: O'Reilly Media, 2017. - 74 с.
5. Heorhiadi V., Jamjoom H., Rajagopalan S., Reiter M.K., Sekar V. Gremlin: Systematic Resilience Testing of Microservices // IEEE 36th International Conference on Distributed Computing Systems. - Nara, Japan: IEEE, 2016. - С. 57-66.

6. Jamjoom H., Rajagopalan S. App–Bisect: Autonomous Healing for Microservice-based Apps // 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15). - Santa Clara, CA: USENIX Association, 2015. - С. 217-224.

7. Montesi F., Weber J. Circuit Breakers, Discovery, and API Gateways in Microservices // arXiv.org URL: https://arxiv.org/pdf/1609.05830.pdf.

8. Nastic S. Programming, Provisioning and Governing IoT Cloud Systems: Doktor der Technischen Wissenschaften: 0527493. - Vienna, Austria, 2016. - 251 с.