МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И КОМПЬЮТЕРНЫХ СЕТЕЙ

УДК 004.75; 656.1

О ЗАДАЧЕ ПОСТРОЕНИЯ ДЕЦЕНТРАЛИЗОВАННОЙ ИНТЕЛЛЕКТУАЛЬНОЙ ТРАНСПОРТНОЙ СИСТЕМЫ НА ОСНОВЕ ПРОТОКОЛА RAFT И КЛАСТЕРИЗАЦИИ ПО СЕТЕВОМУ РАССТОЯНИЮ

М. Г. Городничев, к.т.н., доцент, декан факультета «Информационные технологии», заведующий кафедрой «Математическая кибернетика и информационные технологии» МТУСИ, Москва, Россия;
orcid.org/0000-0003-1739-9831, e-mail: m.g.gorodnichev@mtuci.ru
Е. А. Саксонов, д.т.н., профессор, профессор кафедры «Математическая кибернетика и информационные технологии» МТУСИ, Москва, Россия;
orcid.org/0000-0002-2015-6956, e-mail: saksmiem@mail.ru
В. П. Кулагин, д.т.н., профессор, профессор кафедры «Математическая кибернетика и информационные технологии» МТУСИ, Москва, Россия;
orcid.org/0000-0001-8826-6724, e-mail: vpkulagin@mail.ru
А. Ф. Каперко, д.т.н., профессор, профессор департамента электронной инженерии НИУ ВШЭ, Москва, Россия;
orcid.org/0000-0003-2253-1083, e-mail: akaperko@hse.ru

Статья посвящена разработке и экспериментальной оценке децентрализованной архитектуры интеллектуальной транспортной системы (ИТС), основанной на протоколе консенсуса Raft и методе кластеризации серверов по метрике сетевого расстояния (RTT). Показано, что существующие решения либо требуют ручной конфигурации и централизованной координации, либо не оптимизированы по задержке при географическом распределении узлов, что снижает отказоустойчивость и увеличивает время реакции системы. Предлагается метод динамического формирования подкластеров с автоматическим выбором лидеров, в котором каждый сервер присоединяется к ближайшему по RTT лидеру. Переизбрание лидера и реконфигурация выполняются без участия администратора. Представлен прототип микросервисной системы, реализующий контроль доступов токенами и автоматическую реконфигурацию. Эксперименты демонстрируют ускорение подготовки сервера к включению в кластер на 87,5 %, сокращение времени загрузки 10 ГБ данных на 47,6 % и время переключения лидера порядка 730 мс при отказе узлов.

Ключевые слова: *интеллектуальная транспортная система, Raft, RTT, кластеризация, отказоустойчивость, микросервисы, RethinkDB, ZeroMQ.*

DOI: 10.21667/1995-4565-2025-94-59-67

Введение

Интеллектуальные транспортные системы эволюционируют к обработке больших данных в реальном времени, объединяя телеметрию от сенсоров, GPS-трекеров и видеопотоков. Для интеграции пакетной и потоковой аналитики широко применяются лямбда- и каппа-архитектуры [1]. Однако архитектурный уровень координации инфраструктуры (кластеризация, реконфигурация, отказоустойчивый консенсус) остаётся узким местом при географическом распределении узлов.

Консенсус в распределённых системах традиционно обеспечивается Paxos [2], ZAB (ZooKeeper) [3, 4] и Raft [3, 4]. Paxos теоретически строг, но сложен для реализации и сопровождения. ZAB надёжен, но требует ручной настройки кворума и не ориентирован на оптимизацию по сетевой задержке. Raft предложен как модульная и понятная альтернатива, получившая широкое распространение (etcd, Consul, CockroachDB [5]), но его практические реализации, как правило, предполагают статически заданную топологию кластера без учёта RTT.

Актуальные ИТС нередко используют статическую кластеризацию или централизованные оркестраторы, которые масштабируют количество вычислительных узлов, но не перестраивают топологию в соответствии с сетевой близостью узлов, что ведёт к росту латентности и ухудшению устойчивости при межрегиональном взаимодействии [6]. Исследования по учёту RTT в распределённых БД подтверждают пользу сетевой ориентированной кластеризации, но редко интегрируют протоколы строгого консенсуса в логику самоорганизации [7, 8]. Этим подчёркивая актуальность исследования, а именно объединение строгого алгоритма консенсуса Raft [9, 10] и динамической кластеризации по RTT [11], минимизирующей задержки и повышением устойчивости к отказам.

Постановка задачи

Рассмотрим множество серверов ИТС $S = \{s_1, \ldots, s_n\}$, размещённых в разных сетевых зонах. Пусть сетевое расстояние между узлами определяется двусторонней задержкой $\mathrm{RTT}(s_i, s_j)$, образующей веса полного графа $G = (S, E)$.

Требуется найти разбиение S на подкластеры $C = \{C_1, \ldots, C_k\}$, $C_m \subseteq S$, и множество лидеров $L = \{\ell_1, \ldots, \ell_k\}$, где $\ell_m \in C_m$, при ограничениях:

$$C_m \cap C_{m'} = \varnothing, \quad \bigcup_{m=1}^{k} C_m = S,$$

$$|C_m| \geq 3 \quad \text{(минимум для большинства в Raft)},$$

$$\forall s_i \in C_m : \mathrm{assign}(s_i) = \ell_m.$$

Определим функционал качества топологии:

$$J(C, L) = \alpha \sum_{m=1}^{k} \sum_{s_i \in C_m} \mathrm{RTT}\left(s_i, \ell_m\right) + \beta\, \mathbb{E}\left[T_{\text{failover}}(C, L)\right] + \gamma\, T_{\text{setup}}(C, L),$$

где $\alpha, \beta, \gamma \geq 0$ – весовые коэффициенты; T_{failover} – время переизбрания лидера при отказах; T_{setup} – время включения нового узла.

Цель: минимизировать $J(C, L)$ при соблюдении согласованности данных в каждом C_m согласно Raft (безопасность, лог-матчинг, лидерство) и обеспечить динамическую реконфигурацию при изменении RTT и состава S без ручного вмешательства администратора.

Методология

Предлагаемая методология объединяет строгий консенсус протокола Raft и сетевую ориентированность, самоорганизацию подкластеров на основе метрики двусторонней задержки (RTT). Ключевая идея заключается в том, чтобы переместить фокус со статической, административно заданной конфигурации кластера на динамическое формирование топологии, в которой каждый сервер присоединяется к ближайшему лидеру по фактической сетевой близости, а устойчивость и согласованность обеспечиваются процедурами Raft. Благодаря этому узлы, расположенные в разных географических или сетевых доменах, перестают тянуть один общий кворум через высокие RTT, а формируют

локально быстрые подкластеры с независимым лидерством и контролируемыми межкластерными связями.

Самоорганизация основана на непрерывных активных замерах RTT на уровне приложения. Специализированный сервис Agent, запущенный на каждом сервере, периодически инициирует короткие обмены с соседями, накапливает статистику задержек и формирует динамический профиль сетевой близости. Для подавления случайных всплесков и шумов используются медианные и перцентильные оценки по скользящему окну, что позволяет принимать решения по факту устойчивой, а не мгновенной сетевой картины. На этой основе узел подбирает ближайшее окружение и пытается сформировать подкластер, в котором допустимая задержка не превосходит адаптивного порога. Минимальный размер подкластера привязывается к требованиям Raft по большинству (не менее трёх узлов), а при дефиците близких соседей порог постепенно расширяется, что гарантирует достижимость кворума без жёсткой привязки к заранее прописанным адресам. Алгоритм взаимодействия сервисов представлен на рисунке 1 в виде псевдокода.

Algorithm 1 Обработка запроса пользователя/агента

```
procedure ProcessUserRequest(client, auth, storage, database)
    ⟨token_work, token_refresh⟩ ← Authenticate(auth, client.credentials)
  ▷ шаги 1–2
    req ← BuildFormalRequest(client, token_work)                ▷ шаг 3
    ok, user_ctx ← VerifyToken(auth, token_work)                ▷ шаг 4
    if ok = true then
        UpdateContext(storage, user_ctx)                        ▷ шаг 5
        res ← Dispatch(storage, database, req)                  ▷ шаги 6–7
        Respond(client, res)                                    ▷ шаг 8
    else
        Respond(client, "unauthorized")
    end if
end procedure
```

Рисунок 1 – Обработка запроса пользователя/агента
Figure 1 – Processing user/agent request

Включение нового сервера в систему не требует ручной конфигурации и сводится к запуску агента. Далее узел самостоятельно обнаруживает доступных соседей, ранжирует их по устойчивой задержке и инициирует присоединение к лидеру, который оказывается ближе остальных по RTT. Лидер, получив запрос, проверяет готовность узла к репликации, актуальность его состояния и способность поддерживать заданную задержку; в течение короткого периода прогрева отслеживается конвергенция журнала Raft и стабильность канала. Если фактический профиль задержек узла после включения выходит за пределы окна допустимых значений, лидер инициирует его исключение, а сам узел повторяет процедуру выбора нового, более близкого лидера. Тем самым топология подкластера поддерживается компактной по сетевому расстоянию и не расползается за счёт случайных или деградировавших каналов связи.

Выбор лидера следует стандартной модели Raft с состояниями «Последователь», «Кандидат» и «Лидер» и тайм-аутами выборов, достаточными для предотвращения ложных переизбраний. Существенное отличие от изначальной реализации заключается в введении сетевой осведомлённости при голосовании. При конкурентных выборах преимущество получают кандидаты, для которых наблюдается минимальная устойчивая задержка до большинства узлов потенциального подкластера. Это решение не нарушает гарантии безопасности Raft, но стабилизирует время подтверждения записей и уменьшает хвостовые задержки при фиксации команд в журнале. Параметры тайм-аутов подбираются с учётом наблюдаемой медианы и дисперсии RTT. В лабораторной виртуализованной среде

достаточно около десятисекундного окна, тогда как в реальных сетях целесообразна адаптация на основе эмпирических перцентилей. На рисунке 2 приведен алгоритм самоорганизации и выбора с учетом RTT.

Присоединение нового сервера не требует ручной конфигурации. Агент измеряет задержки, выбирает ближайшего лидера и обеспечивает прогрев репликации журнала. Алгоритм присоединения нового сервера представлен ниже на рисунке 3.

Algorithm 2 RTT-ориентированная кластеризация и выбор лидера

procedure RTTClusterAndElection(node, θ_0, $m_{\min}$, T_e)
  $\theta \leftarrow \theta_0$; $role \leftarrow$ Последователь
  while true do
    $N \leftarrow \{ n : \mathrm{RTT}(node, n) \le \theta \}$ ▷ скользящее окно, медиана/перцентиль
    if $|N| < m_{\min} - 1$ **then**
      $\theta \leftarrow \theta + \delta$ ▷ расширение окна до достижимости кворума
    end if
    if no heartbeat в течение T_e **then**
      $role \leftarrow$ Кандидат; запросить голоса у N с приоритизацией по минимальному RTT
      if #голосов $> \frac{|N|+1}{2}$ **then**
        $role \leftarrow$ Лидер; рассылать heartbeat; принимать последователей
      else
        $role \leftarrow$ Последователь
      end if
    end if
    if $role =$ Лидер **then**
      исключать узлы с устойчивым RTT $> \theta$; поддерживать $|cluster| \ge m_{\min}$
    end if
  end while
end procedure

Рисунок 2 – Алгоритм кластеризации и выбора лидера
Figure 2 – Clustering and leader selection algorithm

Algorithm 3 Присоединение нового сервера к ближайшему лидеру по RTT

procedure JoinNearestLeader(new_node, leaders)
  измерить многократно RTT(new_node, ℓ) для всех $\ell \in leaders$
  $\ell^\star \leftarrow \arg\min_\ell \mathrm{medianRTT}(new\_node, \ell)$
  Start(new_node); Configure(new_node, join=$\ell^\star$.addr)
  JoinCluster(new_node, $\ell^\star$)
  while not IsSynced(new_node) **do**
    Sleep(1)
  end while
  if RTT($new\_node, \ell^\star$) устойчиво $> \theta$ **then**
    LeaveCluster(new_node, $\ell^\star$);
    goto JoinNearestLeader
  end if
end procedure

Рисунок 3 – Алгоритм присоединения нового сервера к ближайщему лидеру по RTT
Figure 3 – The algorithm for attaching a new server to the nearest RTT leader

Стабильность подкластера поддерживается непрерывным наблюдением за последователями. Лидер сопоставляет текущие метрики отклика и лаг репликации с целевыми пределами. При устойчивых выходах за допуски инициируется мягкое исключение узла с последующим самостоятельным переподключением к более близкому лидеру. Подобная политика предотвращает хроническое растягивание топологии, сохраняет низкую латентность внутри подкластера и одновременно оставляет системе свободу адаптации под динамику сети. При угрозе потери кворума активируется режим

резервирования. Агент подключает заранее подготовленный резервный сервер, восстанавливая большинство, а после нормализации состава возвращает конфигурацию к исходной, уведомляя сервисы верхнего уровня. Такой механизм обеспечивает отказоустойчивость без остановки пользовательских сервисов. На рисунке 4 приводится алгоритм включения резервного сервера, а на рисунке 5 алгоритм полного цикла агента.

Algorithm 4 Активация резервного узла при потере кворума

```
procedure ACTIVATERESERVENODE(cluster, reserve_node, storage_service)
    while true do
        alive ← MONITORCLUSTER(cluster)
        quorum ← ⌊|cluster| / 2⌋ + 1
        if |alive| < quorum then
            START(reserve_node)
            CONFIGURE(reserve_node, join=cluster.leader_address, credentials=cluster.auth)
            JOINCLUSTER(reserve_node)
            while not ISSYNCED(reserve_node) do
                SLEEP(1)
            end while
            NOTIFYNODEADDED(storage_service, reserve_node.ip)
            LOG("Reserve node " + reserve_node.ip + " activated and joined cluster.")
        end if
        SLEEP(5)                                  ▷ интервал мониторинга, сек
    end while
end procedure
```

Рисунок 4 – Алгоритм активации резервного узла
Figure 4 – Backup node activation algorithm

Algorithm 5 Полный цикл работы агента мониторинга/резервирования

```
procedure AGENTMAINLOOP(cluster, reserve_node, storage_service)
    reserve_active ← false
    while true do
        alive ← MONITORCLUSTER(cluster)
        quorum ← ⌊|cluster| / 2⌋ + 1
        if |alive| < quorum and reserve_active = false then
            ACTIVATERESERVENODE(cluster, reserve_node, storage_service)
            reserve_active ← true
        else if |alive| ≥ quorum and reserve_active = true then
            NOTIFYNODEREMOVED(storage_service, reserve_node.ip)
            GRACEFULLEAVE(reserve_node); STOP(reserve_node)
            reserve_active ← false
        end if
        SLEEP(5)
    end while
end procedure
```

Рисунок 5 – Полный цикл работы агента мониторинга
Figure 5 – Monitoring agent full cycle

Межкластерное взаимодействие организовано на уровне лидеров и носит характер кластера кластеров. Каждый лидер агрегирует метаданные о составе своего подкластера, доступных наборах данных и состоянии репликации и обменивается этой информацией с другими лидерами. Последователи работают исключительно со своим лидером, что локализует согласование и репликацию внутри зон низкой задержки. Глобальные операции, требующие единого порядка на уровне всей системы, допускаются, но используются умеренно, поскольку неизбежно увеличивают латентность. В типовом режиме межкластерные связи применяются для маршрутизации запросов, обмена справочной

информацией и координации размещения данных без вмешательства в путь подтверждения команд.

Безопасность и управление доступом реализованы через двухтокенную схему аутентификации. Пользователь или агент проходит проверку в сервисе auth и получает пару токенов: токен для многократных обращений к storage и одноразовый бессрочный токен для безопасного обновления пары. Storage валидирует каждый критичный запрос, обращаясь к auth, после чего маршрутизирует операцию в database. Такой подход снижает риски компрометации долгоживущих ключей, упрощает отзыв доступа и позволяет прозрачно управлять сессиями при реконфигурации кластера. Разделение плоскостей управления и данных повышает наблюдаемость и предсказуемость поведения. В плоскости управления проходят аутентификация, измерения RTT, голосования и реконфигурация, тогда как плоскость данных обслуживает операции чтения и записи.

Реализация прототипа опирается на проверенный стек. Сервисы auth и storage реализованы на Python. Клиентское программное обеспечение client реализует конечный автомат состояний для регистрации, входа и работы с данными. Для обмена сообщениями используется ZeroMQ, обеспечивающий низкую задержку и гибкую маршрутизацию. В качестве движка хранения и встроенного механизма консенсуса применена RethinkDB, чьи средства Raft позволяют прозрачно выбирать лидеров и поддерживать кворум. Внутри storage используется контекст обработчика, следящий за доступностью узлов базы данных и автоматически переключающийся на альтернативный сервер при сбоях, что делает отказоустойчивость прозрачной для клиентов. Сервис Agent инкапсулирует логику самоорганизации: непрерывно измеряет сетевую близость, инициирует выборы лидера при отсутствии отклика, контролирует стабильность подкластера и включает либо отключает резервный сервер в зависимости от состояния кворума, уведомляя storage о каждом изменении конфигурации.

Практические значения параметров подбираются из эмпирики наблюдений. Порог допустимого сетевого расстояния и окно его оценки устанавливаются на основе перцентильных характеристик RTT, а при нехватке доступных соседей порог увеличивается ступенчато до достижения устойчивого большинства. Порог исключения последователя формируется с гистерезисом и временной выдержкой, чтобы предотвратить разброс решений на фоне кратковременных всплесков задержек. Тайм-ауты выборов и отклики адаптируются к наблюдаемой дисперсии канала, что в совокупности уменьшает вероятность ложных переизбраний и избыточных реконфигураций. Такой параметрический подход делает поведение системы воспроизводимым и настраиваемым под различные сетевые условия – от локальных кластеров до географически распределённых развертываний.

С точки зрения пользовательских потоков взаимодействие выглядит единообразно независимо от текущей конфигурации подкластеров. Клиент или агент аутентифицируется и получает пару токенов, после чего обращается к storage за целевой информацией или для загрузки данных. Storage проверяет валидность токена через auth, уточняет права доступа, трансформирует пользовательский запрос в формализованную операцию над данными и направляет его в database. В зависимости от контекста database возвращает результат запроса либо принимает неструктурированные данные для сохранения. Далее storage формирует ответ клиенту. Вся эта цепочка продолжается без участия администратора даже в условиях реконфигураций, переизбраний лидера и временных отказов отдельных узлов, поскольку функции обнаружения, переключения и восстановления разнесены по уровням и автоматизированы.

Описанный набор решений непосредственно устраняет ограничения статических кластеров и централизованных оркестраторов, не учитывающих сетевую близость. RTT-ориентированная самоорганизация с локальным лидерством даёт выигрыш по задержке фиксации команд и ускоряет потоковую загрузку данных, а автоматическое резервирование и межлидерская координация поддерживают высокий уровень доступности без ручных

вмешательств. При этом архитектура остаётся совместимой с принятыми практиками ИТС на уровне обработки данных (лямбда- и каппа-архитектуры), что упрощает интеграцию аналитических сервисов поверх надёжного слоя хранения и координации.

Обсуждение результатов

Прототип развёрнут на 7 физических хостах на виртуальных машинах с операционной системой Ubuntu 22.04. Запущены: 7 серверов database (RethinkDB), сервисы storage и auth, 7 клиентов client (имитаторы агентов ИТС). Обмен сообщениями осуществляется через ZeroMQ. В качестве сценариев выбрана нормальная работа (одиночные загрузки файлов), нагрузочное тестирование (7 клиентов непрерывно записывают данные) и отказоустойчивость (отключение двух из 7 серверов database на пике нагрузки).

Сквозная производительность при отключении двух узлов не снизилась – кворум был сохранён, а переизбрание прошло прозрачно для клиентов. По логам storage разница между первым неуспешным и первым успешным записями составила около 730 мс. Время подготовки сервера уменьшилось примерно с 2 часов до 15 минут за счёт самоподключения агента, а загрузка 10 ГБ данных ускорилась с 46,2 до 24,2 минуты, что согласуется с уменьшением внутрикластерных задержек. В таблице 1 приведены результаты экспериментов.

Сокращение времени подготовки обусловлено отказом от ручной конфигурации и использованием механизма автоматического присоединения по RTT. Ускорение загрузки данных объясняется локализацией репликации и минимизацией пути подтверждения команд. Время переключения лидера порядка 100 миллисекунд соответствует типичным значениям для Raft при корректно выбранных тайм-аутах. Ограничениями исследования являются лабораторная среда и ограниченный профиль нагрузок. В том числе для географически распределённых сетей с переменным джиттером и потерями потребуется дополнительная настройка параметров и расширение сценариев тестирования.

Таблица1 – Результаты эксперимента
Table1 – Experimental results

Показатель	Базовый	Предложенный	Улучшение
Подготовка сервера	2 ч	15 мин	87,5 %
Загрузка 10 ГБ	46,2 мин	24,2 мин	47,6 %
Failover лидера	–	~0,73 мс	–

Заключение

В работе предложен и исследован метод построения децентрализованной ИТС, сочетающий строгий консенсус Raft и динамическую кластеризацию по метрике RTT. Сформулирована задача оптимизации топологии подкластеров с выбором лидеров, обеспечивающая минимизацию задержек, быстрое переключение при отказах и сокращение времени ввода новых узлов. Реализован прототип микросервисной системы (auth, storage, client, agent) на Python с брокером сообщений ZeroMQ и СУБД RethinkDB. Эксперименты показали 87,5 % сокращение времени подготовки сервера, 47,6 % ускорение загрузки данных объёмом 10 ГБ и 730 мс на переключение лидера при отказах.

Дальнейшие работы предполагают полевые испытания в мультицентричных сетях, адаптацию тайм-аутов Raft по онлайн оценкам RTT и джиттера. Исследование многорафтовых конфигураций и межкластерной консистентности для глобальных операций. Интеграцию с сервисными сетями для унификации операционных процессов, а также расширение модели безопасности (ротация ключей и поведенческая аналитика агента).

Библиографический список

1. **Агуреев И.Е., Ахромешин А. В., Пышный В.А.** К вопросу оптимизации маршрутной сети города с применением интеллектуальных транспортных систем // Современные технологии: актуальные вопросы, достижения и инновации. 2018. С. 45-50.

2. **Ванина М.Ф.** Распределенные информационные системы. Технологии реализации распределенных информационных систем: учебное пособие / Ванина М.Ф., Ерохин А.Г. Москва: Московский технический университет связи и информатики, 2020. 132 с.

3. **Дерюгина И.А., Берман Н.Д.** Интеллектуальные транспортные системы и технологии // Тогу-старт: фундаментальные и прикладные исследования молодых. 2021. С. 243-249.

4. **Ильяшенко О.Ю., Ильяшенко В.М.** Формирование требований к архитектурному решению интеллектуальной транспортной системы с использованием Больших Данных // Фундаментальные и прикладные исследования в области управления, экономики и торговли. 2018. С. 64-70.

5. **Козлов А.В.** Субсидиарные и децентрализованные системы и алгоритмы // Образовательные ресурсы и технологии. 2020. № 2 (31).

6. **Комбаров В.В., Полозов И.В., Соснина Е.Н.** Применение программного обеспечения принятия консенсуса в системе децентрализованного управления распределительными энергетическими сетями // Информационные системы и технологии ИСТ-2020. 2020. С. 349-355.

7. **Любимова Т.В.** Алгоритм консенсуса в распределенных системах // Университетская наука. 2019. № 1. С. 136-139.

8. **Amrani A., Pasini K., Khouadjia M.** Enhance Journey Planner with Predictive Travel Information for Smart City Routing Services // 2020 Forum on Integrated and Sustainable Transportation Systems (FISTS). IEEE, 2020. C. 304-308.

9. **Nakagawa T., Hayashibara N.** Resource management for raft consensus protocol // International Journal of Space-Based and Situated Computing. 2018. Т. 8. № 2. Pp. 80-87.

10. **Netto H.** et al. Incorporating the Raft consensus protocol in containers managed by Kubernetes: An evaluation // International Journal of Parallel, Emergent and Distributed Systems. 2020. Т. 35. № 4. Pp. 433-453.

11. **Walker C., Alrehamy H.** Personal data lake with data gravity pull // 2015 IEEE Fifth International Conference on Big Data and Cloud Computing. IEEE. 2015. Pp. 160-167.

UDC 004.75; 656.1

ON THE TASK OF BUILDING DECENTRALIZED INTELLIGENT TRANSPORT SYSTEM BASED ON RAFT PROTOCOL AND CLUSTERING BY NETWORK DISTANCE

M. G. Gorodnichev, Ph.D., Associate Professor, Dean of the Faculty of Information Technology, Head of the Department of Mathematical Cybernetics and Information Technology, MTUCI, Moscow, Russia; orcid.org/0000-0003-1739-9831, e-mail: m.g.gorodnichev@mtuci.ru

E. A. Saksonov, Doctor in Technical Sciences, Professor, Professor of the Department of Mathematical Cybernetics and Information Technology, MTUCI, Moscow, Russia;
orcid.org/0000-0002-2015-6956, e-mail: saksmiem@mail.ru

V. P. Kulagin, Doctor in Technical Sciences, Professor, Professor of the Department of Mathematical Cybernetics and Information Technology, MTUCI, Moscow, Russia;
orcid.org/0000-0001-8826-6724, e-mail: vpkulagin@mail.ru

A. F. Kaperko, Doctor in Technical Sciences, Professor, Professor, Department of Electronic Engineering, National Research University of Higher School of Economics, Moscow, Russia;
orcid.org/0000-0003-2253-1083, e-mail: akaperko@hse.ru

The article is devoted to the development and experimental evaluation of decentralized architecture for intelligent transport system (ITS) based on Raft consensus protocol and network distance metric (RTT) server clustering method. Existing solutions are shown either to require manual configuration and centralized coordination, or not to be optimized for latency with geographical distribution of nodes, which reduces fault tolerance and increases system response time. A method for dynamic formation of subclusters with automatic selection of leaders, in which each server joins the leader closest to RTT is proposed. The leader is re-elected and reconfigured without administrator participation. A prototype of microservice system implementing access control with tokens and automatic reconfiguration is presented. Experiments

demonstrate: acceleration of server preparation to be included in cluster by 87,5 %, reduction of loading time of 10 GB of data by 47,6 %, and leader switching time of about 730 ms in case of node failure.

Keywords: *intelligent transport system, Raft, RTT, clustering, fault tolerance, microservices, RethinkDB, ZeroMQ.*

DOI: 10.21667/1995-4565-2025-94-59-67

References

1. **Agureev I.E., Akhromeshin A.V., Pyshnyi V.A.** K voprosu optimizatsii marshrutnoi seti goroda s primeneniem intellektual'nykh transportnykh sistem [On the issue of optimizing the city route network using intelligent transport systems]. *Sovremennye tekhnologii: aktual'nye voprosy, dostizheniia i innovatsii*. 2018, pp. 45-50. (in Russian).

2. **Vanina M.F., Erokhin A.G.** Raspredelennye informatsionnye sistemy. Tekhnologii realizatsii raspredelennykh informatsionnykh sistem [*Distributed Information Systems. Technologies for the Implementation of Distributed Information Systems*]. Moscow, Moskovskii tekhnicheskii universitet sviazi i informatiki Publ. 2020. 132 p. (in Russian).

3. **Deriugina I.A., Berman N.D**. Intellektual'nye transportnye sistemy i tekhnologii [*Intelligent transport systems and technologies*]. Togu-start: fundamental'nye i prikladnye issledovaniia molodykh. 2021, pp. 243-249. (in Russian).

4. **Il'iashenko O.Iu., Il'iashenko V.M.** Formirovanie trebovanii k arkhitekturnomu resheniiu intellektual'noi transportnoi sistemy s ispol'zovaniem Bol'shikh Dannykh [*Forming requirements for the architectural solution of an intelligent transport system using Big Data*]. Fundamental'nye i prikladnye issledovaniia v oblasti upravleniia, ekonomiki i torgovli. 2018, pp. 64-70. (in Russian).

5. **Kozlov A.V.** Subsidiarnye i detsentralizovannye sistemy i algoritmy [Subsidiary and decentralized systems and algorithms]. *Obrazovatel'nye resursy i tekhnologii*. 2020, no. 2 (31). (in Russian).

6. **Kombarov V.V., Polozov I.V., Sosnina E.N.** Primenenie programmnogo obespecheniia priniatiia konsensusa v sisteme detsentralizovannogo upravleniia raspredelitel'nymi energeticheskimi setiami [The use of consensus decision-making software in a decentralized control system for distribution power networks]. **Informatsionnye sistemy i tekhnologii IST-2020**, 2020, pp. 349-355. (in Russian).

7. **Liubimova T.V.** Algoritm konsensusa v raspredelennykh sistemakh [Consensus algorithm in distributed systems]. *Universitetskaia nauka*. 2019, no. 1, pp. 136-139. (in Russian).

8. **Amrani A., Pasini K., Khouadjia M.** Enhance Journey Planner with Predictive Travel Information for Smart City Routing Services. *2020 Forum on Integrated and Sustainable Transportation Systems (FISTS). IEEE*. 2020, pp. 304-308.

9. **Nakagawa T., Hayashibara N.** Resource management for raft consensus protocol. *International Journal of Space-Based and Situated Computing*. 2018, vol. 8, no. 2, pp. 80-87.

10. **Netto H. et al.** Incorporating the Raft consensus protocol in containers managed by Kubernetes: An evaluation. *International Journal of Parallel, Emergent and Distributed Systems*. 2020, vol. 35, no. 4, pp. 433-453.

11. **Walker C., Alrehamy H.** Personal data lake with data gravity pull. *2015 IEEE Fifth International Conference on Big Data and Cloud Computing. IEEE*. 2015, pp. 160-167.