Использование функций мессенджера Telegram для обмена сообщениями между узлами распределенной вычислительной системы

В.А. Алексеев, П.А. Домашнев, Т.В. Лаврухина, О.А. Назаркин

***Аннотация*- В работе рассмотрено использование функций популярного мессенджера Telegram для организации одно- и двунаправленных каналов передачи информации в системах распределенных вычислений. Основная цель нашего подхода – реализация бесплатного обмена сообщениями в режиме реального времени для коммуникаций между узлами распределенной сети добровольных вычислений с клиентскими компонентами, работающими в контексте веб-браузера. Telegram Bot API предоставляет надежную бессерверную среду обмена сообщениями с поддержкой браузерных JavaScript-клиентов. Ключевой особенностью предлагаемой схемы является то, что в ней не требуется сервер приложений, поэтому решение подходит для децентрализованных распределенных алгоритмов, где взаимодействующие узлы (как вычислительные, так и управляющие) работают в сеансах браузера. Бот – это специальная учетная запись, управляемая пользовательской программой, которая предназначена для получения сообщений, их анализа и создания новых сообщений. Несмотря на утверждение документации Telegram о том, что ботам не разрешен обмен сообщениями друг с другом, фактический обмен данными между ними возможен через каналы (Telegram channels), в которых боты назначены администраторами. В работе предлагается схема двусторонней связи – эмуляция децентрализованного однорангового взаимодействия на прикладном уровне. Кроме того, возможно построение однонаправленных каналов для передачи сообщений от узлов-инициаторов к удаленным обработчикам или приемникам данных.**

***Ключевые слова*— grid-системы, мессенджер Telegram, боты Telegram, добровольные вычисления, обмен сообщениями в реальном времени.**

I. Введение

Будем рассматривать системы распределенных вычислений (grid-системы), в которых узлы осуществляют обмен информацией через Интернет. В настоящей работе ключевое внимание уделяется вопросам коммуникации между узлами распределенной системы. Существуют разные способы организации каналов обмена сообщениями для веб-систем:

- традиционная клиент-серверная модель, в которой узлы-клиенты обращаются с http-запросами к центральному серверу; серверное программное обеспечение самостоятельно реализует прикладную логику обработки поступающих от клиентов сообщений;
- бессерверная (serverless) модель, в которой центральная облачная платформа обеспечивает универсальную реакцию на http-запросы от узлов-клиентов посредством запуска в облачной среде предварительно загруженных и зарегистрированных пользовательских программных модулей; пользовательские модули получают в качестве аргументов поступающие от клиентов сообщения и должны содержать собственную реализацию их обработки на том языке, который поддерживается платформой;
- бессерверная модель, реализующая паттерн “Издатель-Подписчик” (Publish-Subscribe); в этом случае облачная платформа не обеспечивает запуск прикладных программных модулей обработки входящих сообщений (от узлов-издателей), а только осуществляет перенаправление сообщений узлам-подписчикам;
- децентрализованная p2p-схема (peer-to-peer), в которой сообщения, исходящие от узлов-инициаторов, маршрутизируются непосредственно к адресатам (узлам-акцепторам); акцептор обеспечивает прикладную обработку принятых сообщений и может, в свою очередь, выступать инициатором сообщений-ответов.

У каждого из этих решений есть свои достоинства и недостатки, а также условия применимости. Немаловажным фактором, влияющим на выбор конкретной коммуникационной модели, является стоимость ее реализации. В настоящее время затруднительно обеспечить надежный и быстрый обмен сообщениями в grid-системе глобального масштаба [1]

Статья получена 14 февраля 2019 г.

В.А. Алексеев, Липецкий государственный технический университет, Россия (e-mail: alexeev48@gmail.com).

П.А. Домашнев, Липецкий государственный технический университет, Россия (e-mail: pdomashnev@gmail.com).

Т.В. Лаврухина, Липецкий государственный технический университет, Россия (e-mail: lavrukhina_tv@mail.ru).

О.А. Назаркин, Липецкий государственный технический университет (e-mail: nazarkino@mail.ru).

без необходимости оплаты услуг провайдеров облачных ресурсов и сервисов. Можно найти бесплатные компромиссные решения (такие как [2]), в которых облачные сервисы используются в нестандартном режиме, при этом, в первую очередь, существенно снижается скорость передачи сообщений. Некоторые облачные сервисы (такие как [3, 5]) допускают возможность реализации бесплатного скоростного обмена сообщениями в соответствии с моделью Publish-Subscribe, но при этом существенно ограничивают количество подключенных узлов (10-20 узлов). Другие провайдеры ограничивают количество бесплатно передаваемых сообщений в единицу времени до неприемлемо низкого уровня, либо используют буферизацию сообщений во внутренних очередях, так что при обмене возникают существенные задержки [4].

В таких условиях в сферу внимания разработчиков grid-систем естественным образом попадают инновационные глобальные коммуникационные платформы, декларирующие поддержку быстрого, надежного, безопасного и при этом абсолютно бесплатного обмена сообщениями (“We believe in fast and secure messaging that is also 100% free.” [6]). Одним из наиболее перспективных продуктов в этом классе является мессенджер Telegram. Однако, как и любой интерактивный мессенджер, Telegram ориентирован на управление пользователем-человеком и напрямую не предназначен для программной организации обмена сообщениями между вычислительными процессами. Тем не менее, Telegram предоставляет для этого все необходимые средства через развитый интерфейс прикладного программирования (API).

Целью настоящей работы является исследование возможностей Bot API мессенджера Telegram для поддержки одно- и двунаправленной передачи информации между узлами распределенной вычислительной системы в реальном времени, а также разработка соответствующих структурных и алгоритмических решений.

II. Основные методы Telegram Bot API, необходимые для программного обмена сообщениями между ботами

Telegram предоставляет Bot API [7] – программный интерфейс для разработки ботов. Обращения к Bot API осуществляются посредством http-запросов. В запросе передаются индивидуальный токен бота, наименование и параметры вызываемого метода. Следовательно, программа-бот может выполняться на любом вычислительном узле, в контексте любого процесса, которому разрешено выдавать http-запросы.

Ключевым методом Bot API является getUpdates – получение обновлений, т.е. сведений о новых сообщениях, адресованных данному боту, либо направленных в каналы, на которые подписан бот. Серверная платформа поддерживает для этого метода механизм long polling, который позволяет:

1) без задержек получать в клиентском процессе информацию о новых сообщениях (аналог push-оповещений);
2) существенно снизить количество отправляемых с клиентской стороны избыточных запросов во время фактического отсутствия обновлений.

Схема алгоритма получения обновлений с помощью метода getUpdates в режиме long polling (здесь T – максимальная длительность опроса, рекомендуемое значение – несколько секунд):

```
api.getUpdates(T, 0, callback); // отправка первого запроса обновлений

function callback(updates) {
  updates.process(); // обработка принятых сообщений
  id = updates.id; // id – идентификатор подтверждаемого обновления
  api.getUpdates(T, id + 1, callback); // запуск новой итерации опроса
}
```

Обновления формируются в рамках некоторого “окна” в хронологической перспективе, и для сдвига этого окна в направлении новых обновлений следует передавать в качестве дополнительного параметра метода getUpdates идентификатор, на единицу больший идентификатора последних *подтвержденных* (confirmed) обновлений. Такая схема позволяет надежно осуществлять выборку сообщений даже в условиях обрыва связи или аварийного завершения бот-процесса: серверная сторона полагает, что вызов getUpdates с идентификатором (update_id + 1) означает, что бот *подтвердил* (успешно принял и локально сохранил) обновления с предыдущим идентификатором update_id. При этом боту достаточно не отправлять подтверждающий запрос с новым идентификатором, пока не окончено локальное сохранение информации о предыдущем пакете обновлений. Внезапный обрыв бот-сессии в момент локального сохранения не приведет к потере данных: возобновленная сессия получит все неподтвержденные сообщения, начиная с update_id, так как она просто не сможет сформировать подтверждающий идентификатор (update_id + 1), не имея в локальном хранилище сведений об обновлении с предыдущим идентификатором update_id. Исходный запрос getUpdates отправляется с параметром update_id = 0.

Вызов метода getUpdates соответствует действию subscribe в модели Publish-Subscribe, с той особенностью, что подписка прерывается по истечении T секунд и должна принудительно возобновляться клиентским процессом. Серверная сторона после получения запроса getUpdates ожидает в течение не более T секунд; в случае отсутствия новых сообщений клиенту возвращается пустой массив updates. Но при поступлении новых сообщений во время ожидания

текущая фаза опроса прекращается немедленно, и обновления сразу возвращаются клиенту. При этом, независимо от полученных результатов, функция обратного вызова callback должна вновь вызвать метод getUpdates для продолжения опроса.

Особенность политики Telegram состоит в том, что бот не имеет права самостоятельно начать диалог с другим пользователем. Такое ограничение оправдано в случае диалога с пользователем-человеком, но для автоматического программного обмена данными между вычислительными процессами следует разработать технологию, позволяющую боту-инициатору отправлять сообщения боту-акцептору. В настоящей работе для решения этой задачи предлагается использовать каналы Telegram. Канал в терминологии Telegram – это информационное пространство, в котором пользователи-администраторы имеют возможность размещать сообщения. Бота можно сделать администратором канала (вручную пользователем-человеком – создателем канала). Если у канала несколько ботов-администраторов, все они смогут публиковать в этом канале свои сообщения (и получать обновления о чужих публикациях).

Таким образом, для автоматического обмена данными между процессами A и B необходимо вручную создать как минимум двух ботов (условно, A_bot и B_bot), а также как минимум один канал (условно, AB_channel), в котором оба бота должны получить права администрирования. A_bot будет получать сообщения, инициированные B_bot, и наоборот.

Для записи нового сообщения в канал используется метод sendMessage. Этот метод соответствует действию publish в модели Publish-Subscribe. Его параметрами являются текст сообщения и идентификатор целевого канала. Присутствует ограничение: длина текста (полезной нагрузки сообщения) не должна превышать 4 Кб. Для отправки более длинных сообщений (до 20 Мб) следует использовать передачу файлов (метод sendDocument), но это приведет к дополнительным задержкам для принимающей стороны, поскольку ей потребуется сначала вызвать метод getFile, затем из его результата извлечь URL загрузки переданного файла, и, наконец, обратиться к этому URL с GET-запросом содержимого файла. Следовательно, для наиболее быстрого обмена между взаимодействующими процессами прикладной системе необходимо формировать короткие сообщения (до 4 Кб).

Для удаления сообщений из канала используется метод deleteMessage. Его параметром является идентификатор удаляемого сообщения

III. Однонаправленная передача сообщений по схеме “многие-к-одному”. Квазисерверы на основе ботов Telegram

Платформа Telegram не позволяет одному и тому же боту одновременно выдавать несколько запросов getUpdates (бот-программа должна выполняться только в одном глобальном экземпляре, идентифицируемом индивидуальным токеном). Это соответствует такому сценарию использования, когда бот является акцептором сообщений, приходящих от некоторого множества узлов-инициаторов. С точки зрения инициаторов бот-акцептор может считаться обслуживающим процессом (серверная роль). Поскольку сам бот-акцептор является *клиентом* сервиса Telegram, но выполняет типичную роль *сервера*, то в настоящей работе используется термин *квазисервер*.

Квазисервер qs в простейшем случае требует двух ботов:

- qs_reader_bot – бот-акцептор, вызывающий метод getUpdates; выполняется в одном экземпляре;
- qs_writer_bot – бот, используемый инициаторами для отправки сообщений акцептору; может работать во многих экземплярах (на многих узлах).

Кроме этого, с квазисервером qs должен быть ассоциирован некий канал (qs_channel), в котором боты являются администраторами. Все боты и каналы в Telegram используют одно и то же “плоское” пространство имен, поэтому для разных приложений следует вручную создавать разных ботов и разные каналы, например: <app_id>_reader_bot, <app_id>_writer_bot, <app_id>_channel. Владельцами разных квазисерверов могут являться разные пользователи.

Квазисервер способен реализовать только *однонаправленную* передачу информации (от инициаторов к акцептору). Это является существенным недостатком предлагаемой технологии. Тем не менее, в прикладных задачах распределенных вычислений существуют реалистичные сценарии использования, для которых однонаправленная схема является подходящей, особенно с учетом высокой скорости доставки сообщений, обеспечиваемой инфраструктурой Telegram. Среди таких сценариев можно выделить следующие:

- агрегация и анализ результатов, присылаемых с узлов-исполнителей, завершивших обработку вычислительных заданий;
- сбор информации в реальном времени с узлов-генераторов (например, мониторинг удаленных процессов, датчиков и т.п.);
- вычислительная обработка заданий от удаленных узлов с сохранением результатов в локальную или облачную базу данных;
- доверенный прокси-шлюз передачи данных к другому квазисерверу, применяемый с целью сокрытия идентификационной информации конечного получателя.

В практических реализациях квазисервера недостаточно применять единственный writer_bot. Дело в том, что платформа Telegram не разрешает слишком

частую отправку сообщений одним и тем же ботом; как указано в документации, ботам не рекомендуется отправлять в среднем более одного сообщения в секунду (хотя допускаются кратковременные всплески активности).

Это ограничение представляется достаточно существенным с точки зрения масштабирования прикладной системы, поэтому каждому квазисерверу требуется несколько различных ботов writer_bot. Процесс-инициатор должен обладать списком токенов всех ботов, которым разрешено посылать сообщения в канал квазисервера. Простая стратегия балансирования нагрузки заключается в том, что при каждой отправке сообщения в канал квазисервера процесс-инициатор обязан выбрать некоторого бота writer_bot случайным образом (рисунок 1).

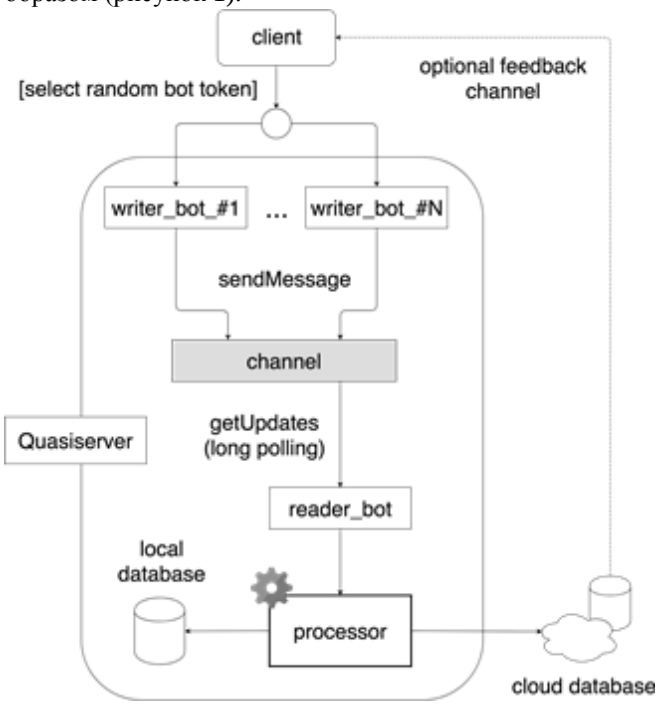

Рисунок 1 Структура сбора и обработки клиентской информации с помощью квазисервера на основе ботов Telegram.

Одноуровневое масштабирование с применением нескольких ботов writer_bot не позволит увеличить плотность отправки сообщений квазисерверу до величин порядка сотен или тысяч сообщений в секунду. Для систем с такой нагрузкой необходимо использовать уплотнение сообщений в пакеты на стороне-инициаторе (накопление в очереди с отложенной отсылкой) и доставку их целевому боту-акцептору в виде файлов. В любом случае, количество узлов, *одновременно* передающих отдельные сообщения или пакеты-файлы без угрозы превышения лимитов, определяется количеством ассоциированных с квазисервером ботов writer_bot.

IV. Двунаправленная передача сообщений. P2P-взаимодействие на прикладном уровне

Реализация двунаправленного обмена сообщениями между двумя узлами, каждый из которых может быть как инициатором, так и акцептором (рисунок 2) основана на тех же структурных решениях, которые применяются для передачи информации к квазисерверам. Фактически, в случае двунаправленного обмена каждый узел одновременно выполняет роли как клиента, так и квазисервера.

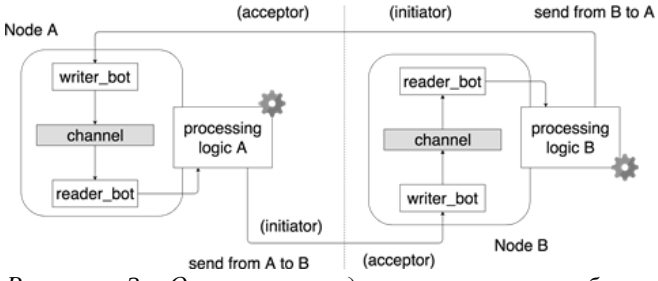

Рисунок 2 Организация двунаправленного обмена сообщениями между узлами распределенной системы.

Эту структуру можно расширить на N узлов, чтобы была возможность осуществлять обмен в соответствии со схемой “многие-ко-многим”. Если произвольный неориентированный граф представляет топологию дуплексных коммуникаций между узлами распределенной системы, то любое ребро AB этого графа имеют реализацию в соответствии с рис. 2. Такая сеть будет p2p-сетью на прикладном уровне. Каждый потенциальный инициатор обмена должен иметь список токенов всех целевых ботов $writer\_bot_i$, а также имена всех целевых каналов $channel_i$, $i \in [1, N]$, N – общее число узлов-акцепторов.

При динамическом характере идентификационной информации (например, если состав узлов системы подвержен частым и неконтролируемым изменениям) распространение токенов ботов и имен каналов удобно организовать через некоторую централизованную облачную БД. Эта БД должна закрытой, то есть предоставлять доступ только авторизованным участникам информационного обмена, чтобы идентификаторы ботов и каналов использовались только в границах прикладной вычислительной системы.

Другое решение может заключаться в выделении специального центрального квазисервера (диспетчера), идентификационная информация которого известна всем участникам обмена. Этот квазисервер имеет эксклюзивный доступ к закрытой БД, в которой содержатся идентификаторы ботов и каналов для всех узов. Для снижения нагрузки на диспетчер обращение к нему требуется только в начале информационного обмена между парой узлов (фаза *установки соединения*), а все последующие сообщения передаются уже в соответствии с p2p-схемой. При этом используется принцип, описанный в [1]: инициатор сначала посылает диспетчеру запрос, включающий уникальное имя (но не

токен бота и не имя канала) узла-акцептора, а также собственную информацию об инициаторе (токен бота и имя канала), необходимую для обратной связи. Получив такой запрос, диспетчер извлекает из БД токен бота и имя канала узла-акцептора и перенаправляет ему сообщение. Поскольку в сообщении присутствуют токен бота и имя канала узла-инициатора, отправка ответного сообщения уже не требует обращения к диспетчеру. Квазисервер, выполняющий роль диспетчера, может иметь относительно небольшое количество ботов qs_writer_bot, так как в типичном режиме установка *нового* (долговременного) соединения происходит гораздо реже, чем обмен данными в рамках уже установленного соединения; отсюда также следует, что один квазисервер-диспетчер может обслуживать достаточно большое число вычислительных узлов – участников информационного обмена.

Структура распределенной обработки информации с использованием квазисервера-диспетчера позволяет осуществлять миграцию узлов. Каждый узел имеет уникальное имя, но не имеет жесткой ассоциации с ним токенов ботов и имен каналов. Таким образом, система может организовать пул доступных токенов ботов и имен каналов, из которого узлы могут извлекать необходимые идентификаторы при подключении.

V. Заключение

В настоящей работе предложены структурные решения по организации одно- и двунаправленного обмена сообщениями между узлами распределенной вычислительной системы с использованием средств, предоставляемых инфраструктурой и прикладными программными интерфейсами мессенджера Telegram.

Реализованные прототипы программных модулей позволяют оценить основные достоинства предлагаемого подхода:

- возможность осуществлять бесплатный обмен сообщениями в реальном времени с относительно низкими задержками (около 200-300 мс) и приемлемыми лимитами (4 килобайт на одно сообщение, 1-2 сообщения в секунду от каждого из публикующих ботов);
- поддержка технологии long polling на стороне облачного сервиса, что снижает клиентский трафик и не требует компромиссов между частотой опроса и задержками получения оповещений об обновлениях;
- устойчивость к непредсказуемым отключениям узлов за счет специального протокола подтверждения полученных обновлений, поддерживаемого облачным сервисом;
- широкие возможности построения коммуникационных связей в grid-системах на прикладном уровне, как централизованных, так и децентрализованных (peer-to-peer);
- удобство разработки клиентских модулей (используется API на основе HTTP-запросов).

К недостаткам рассматриваемых решений следует отнести ограничения по масштабированию и пропускной способности, обусловленные тем, что предлагаемый подход ориентирован на упрощенное подмножество API Telegram (Bot API). Устранить эти ограничения позволяет использование полноценного клиентского API, а также рекомендуемого фреймворка Telegram Database Library.

Библиография

[1] Алексеев В.А., Домашнев П.А., Лаврухина Т.В., Назаркин О.А. Схема горизонтального масштабирования веб-сервисов на основе протокола WebSocket // Системы управления и информационные технологии №3(73), с. 52-55.

[2] Лаврухина Т.В., Назаркин О.А., Алексеев В.А., Домашнев П.А. Особенности использования распределенных вычислений при моделировании транспортных процессов // В сборнике: Инфокоммуникационные и интеллектуальные технологии на транспорте. Материалы I международной научно-практической конференции. – Издательство Липецкого государственного технического университета, 2018. – В 2 т. Т. 2. – С. 211-215.

[3] Назаркин О.А. Распределенная динамическая среда обработки данных на основе облачных сервисов Google Firebase // Материалы XII международной научно-практической конференции "Современные сложные системы управления (HTCS'2017)". – Издательство Липецкого государственного технического университета, 2017. – В 2 т. Т. 2. – С. 55-59.

[4] Назаркин О.А., Лаврухина Т.В., Алексеев В.А., Домашнев П.А. Использование облачного сервиса Firebase Cloud Messaging для обмена сообщениями в крупномасштабных распределенных вычислительных системах // Вести высших учебных заведений Черноземья. 2018. № 3 (53). С. 46-58.

[5] Redis Labs | Database for the Instant Experience [Электронный ресурс]. – Режим доступа: www. URL: https://redislabs.com/ – 18.01.2019.

[6] Telegram F.A.Q. [Электронный ресурс]. – Режим доступа: www. URL: https://telegram.org/faq/ – 18.01.2019.

[7] Telegram Bot API [Электронный ресурс]. – Режим доступа: www. URL: https://core.telegram.org/bots/api/ – 18.01.2019.

Usage of Telegram Bots for message exchange in distributed computing

V. Alexeev, P. Domashnev, T. Lavrukhina, O. Nazarkin

***Abstract*- In this paper, we consider the scheme of unidirectional and bidirectional messaging built over the popular Telegram service. The main purpose of our approach is to provide free real-time message exchange solution for inter-node communications in browser-based volunteer grid computing. Telegram Bot API provides reliable serverless implementation of messaging infrastructure, with support of browser JavaScript clients. The key feature of proposed scheme is that no application server is needed at all, so the solution is suitable for decentralized distributed algorithms, where interacting nodes (both compute and control) run in browser sessions. Bot is a special account managed by a user program that is designed to receive messages, analyze them and generate new messages. Despite the Telegram documentation statement that bots (as automatically running self-controlled routines) are not allowed to exchange messages with each other, the actual data transfer between bot contexts can be implemented through Telegram channels. We propose application level peer-to-peer duplex communications, along with simplex data channels for unidirectional messages dispatched from initiative sources to known remote processors, or "data sinks".**

Keywords: **grid computing, Telegram messenger, Telegram Bots API, volunteer computing, realtime message exchange.**

REFERENCES

[1] Alexeev V.A., Domashnev P.A., Lavrukhina T.V., Nazarkin O.A. Sxema gorizontal`nogo masshtabirovaniya veb-servisov na osnove protokola WebSocket // Sistemy` upravleniya i informacionny`e texnologii №3(73), s. 52-55.

[2] Lavrukhina T.V., Nazarkin O.A., Alexeev V.A., Domashnev P.A. Osobennosti ispol`zovaniya raspredelenny`x vy`chislenij pri modelirovanii transportny`x processov // V sbornike: Infokommunikacionny`e i intellektual`ny`e texnologii na transporte. Materialy` I mezhdunarodnoj nauchno-prakticheskoj konferencii. – Izdatel`stvo Lipeczkogo gosudarstvennogo texnicheskogo universiteta, 2018. – V 2 t. T. 2. – S. 211-215.

[3] Nazarkin O.A. Raspredelennaya dinamicheskaya sreda obrabotki danny`x na osnove oblachny`x servisov Google Firebase // Materialy` XII mezhdunarodnoj nauchno-prakticheskoj konferencii "Sovremenny`e slozhny`e sistemy` upravleniya (HTCS'2017)". – Izdatel`stvo Lipeczkogo gosudarstvennogo texnicheskogo universiteta, 2017. – V 2 t. T. 2. – S. 55-59.

[4] Nazarkin O.A., Lavrukhina T.V., Alexeev V.A., Domashnev P.A. Ispol`zovanie oblachnogo servisa Firebase Cloud Messaging dlya obmena soobshheniyami v krupnomasshtabny`x raspredelenny`x vy`chislitel`ny`x sistemax // Vesti vy`sshix uchebny`x zavedenij Chernozem`ya. 2018. № 3 (53). S. 46-58.

[5] Redis Labs – Database for the Instant Experience. https://redislabs.com/

[6] Telegram F.A.Q. https://telegram.org/faq/

[7] Telegram Bot API. https://core.telegram.org/bots/api/