Аннотация

Выпускная квалификационная работа содержит 37 страниц и 24 рисунка.

Выпускная квалификационная работа посвещена иследованию возможности организации эластичных вычислений на космическом аппарате оснащенным несколькими процессорами. В ходе работы использованы наработки в сфере технологии интернета вещей.

В частности, были рассмотрены возможности использования облачных вычислений и предложенно использования модели туманных вычислений для обеспечения автономности космического аппарата, а также для достижения низкой и предсказуемой задержки передачи информации с конечными устройствами. Для реализации доступа к эластичной сети используется модель обслуживания облачных вычислений PaaS (Platform as a Service - Платформа как Сервис).

В модели PaaS пользователю предоставляется возможность использования уже развернутой на физических и виртульных машинах системы для развёртывания на них своих собственных приложений и бизнес процессов. Для реализции PaaS также понадобиться создать инфрастурктуру, которая будет состоять из множества узлов(процессор с какими-то ресурсами), которые могут подключаться и отключаться из сети, и каждый из которых обладает определённой вычислительной мощностью, которую можно задействовать при выполении программы. Для составления модели организации инфраструктуры эластичных вычислений используются сети Петри.

В практической части была реализована программа на языке python с использованием вебфреймворка flask, которая позволяет создавать узлы для последующего подключения и отключения из эластичной сети, а также позволяет выдавать сети задания, для последующего выполнения, с распределением заданий между узлами. Также если в ходе выполнения заданий какой-либо узел отключиться из сети, его задания будут перераспределены между остальными узлами.

Для добавления ассинхроности в коде программы также были использованы модули asyncio и threading, Был произведён анализ эффективности полученого решения.

Содержание

Введение

В наше время всё больше космических аппаратов выводится в космос. Например, одна только Starlink вывела на орбиту земли более тысячи спутников, имея лицензию на вывод ещё 11-и тысяч и при этом планирует получить лицензию на дополнительные 18 тысяч. Это хоть и добавляет новые проблемы, такие как увеличение космического мусора на околоземной орбите, но при этом открывает гораздо больше новых возможностей, ведь теперь в космосе есть гораздо больше вычислительных ресурсов.

При этом потребность в больших вычислительных ресурсах явна есть, например, вести мониторинг обордования в реальном времени или произвести посадку космического аппарата на сложный участок марсианской поверхности, ведь по оценкам экспертов из НАСА, для подобного потребуются производить от 10, до 50 гигаопераций в секунду(GOPS). И это не предел, ведь есть ещё более сложные задачи.

Однако, из-за специфики условий космоса, таких как космическая радиация (которая может вывести систему из строя), вакуум (в котором единственным доступным методом для отвода тепла является излучение) и ограниченное количество энергии (которая востанавливается только за счёт солнечных батарей), а также использования устаревших компонентов из-за их надежности,процессоры космического класса имеют вычислительную мощность в десятки раз меньшую, чем у процессоров, установленных в современные мобильные телефоны.

Рис. 1: Процессор PowerPC 750FX разработоный 18 лет назад, установленый в космическом аппарате Orion

По этим причинам, хоть у выведенных спутников и есть несколько процессоров(различные датчики и прочее), каждый из которых занимается решением своей задачи, и которые он может использовать для решении поставленных перед ним задач. При этом может возникнуть ситуация, когда мощности какого-то процессора не хватает для выполнения нужных вычислений в срок. Если на спутнике просто установлены слабые процессоры, то решением проблемы может быть делегирование задачи другому спутнику, который имеет в своём распоряжении больше вычислительных мощностей, которых хватит для проведения всех нужных вычислений в срок, однако даже тогда не всегда такой спутник будет в зоне досягаемости, а также он уже может быть занят выполнением какой-либо другой более важной задачи.

Можно ли решить проблему, используя при этом только те ресурсы, которые есть на самом спутнике? Да, и решением в подобной ситуации может стать объединение нескольких слабых процессоров в одну, более мощную, вычислительную сеть.

В связи с этим возникает новая проблема: как будет будет происходить образование вычислительной сети? Самым очевидным решением будет организация вычислительной сети в ручном режиме, то есть прямо с земли, какой-то человек должен будет включать по необходимости выбранные процессоры в единую вычислительную сеть. Однако это создает лишь больше проблем: потребуется обученный персонал, которому придётся тратить время на объединение процессоров, вместо того чтобы решать других задачи. Также можно написать ПО, которое будет в автоматическом режиме объединять процессоры, когда необходимо, однако в этом случае сами спутники могут быть не в зоне досягаемости сигналов из командного центра (например, когда они находятся на другой стороне земли), что сделает невозможным объединение процессоров с земли. Эти проблемы делают эти решения не подходящими для использования в условиях реальной жизни.

Отсюда следует необходимость найти способ, с помощью которого будет происходить автономное (без вмешательства человека или команд с земли) объединение нескольких процессоров космического аппарата в единую вычислительную сеть, для последующего выполнения какой-либо задачи. При этом стоит учитывать то, что такая сеть должна уметь меняться со временем, то есть к ней должны уметь подключаться новые процессоры, а также отключаться старые, ведь оборудование всегда может выйти из строя, при этом в условиях космоса не будет возможности для его быстрой замены.

Такая концепция, когда пул ресурсов не статичен, а может меняться со временем, называется эластичными вычислениями. На эту концепию хорошо ложится такая технология как интернет вещей, где тоже постояно появляются новые устройства и отключаются старые.

В данной работе мне нужно будет исследовать методы организации эластичных вычислений. Провести анализ вариантов реализации с использованием интернета вещей. Изучить математический аппарат сетей

Петри для построения модели. Реализовать полученные знания для создания модели эластичных вычислений на космическом аппарате. Написать программу для моделирования эластичных вычислений на космическом корабле. Сделать вывод об эффективности изученных методов на основе полученных результатов моделирования.

1 Теоретическая часть

1.1 Постановка задачи и анализ

В данной дипломной работе будет решаться следующая задача: нужно найти способ, которым можно объединить множество процессоров в единую сеть,которая может проводить параллельные вычисления на всех спутниках входящих в данную сеть. При этом процессоры могут как добавляться в вычислительную сеть, так и выходить из неё, т.к. для поддержки всех процессоров в активном состоянии требуется значительное количество энергии. То есть нужно организовать эластичные вычисления - вычисления в которых ресурсы могут добавляться и освобождаться по необходимости.

Для начала рассмотрим как функционируют современные космические аппараты на примере спутника. Если абстрагироваться от всех технических деталей управления спутником, то можно сказать, что есть спутники, которые отсылают на землю данные с датчиков, и получают с земли команды, которые нужно выполнить(например сделать снимок земли).

1.2 Интернет вещей

Данная абстракция похожа на то, как работает современный интернет вещей(Internet of Things) - есть Internet of Things(IoT) устройство, которое отсылает данные (обычно полученные с датчиков) в облако(Cloud),

на котором происходит из обработка, с (необязательно) последующей отправкой команд обратно на IoT устройство. Поэтому можно воспользоваться наработками из этой области, для решения поставленной задачи, а также для прогнозирования будущих проблем, которые могут возникнуть при увеличении числа спутников на околоземной орбите.

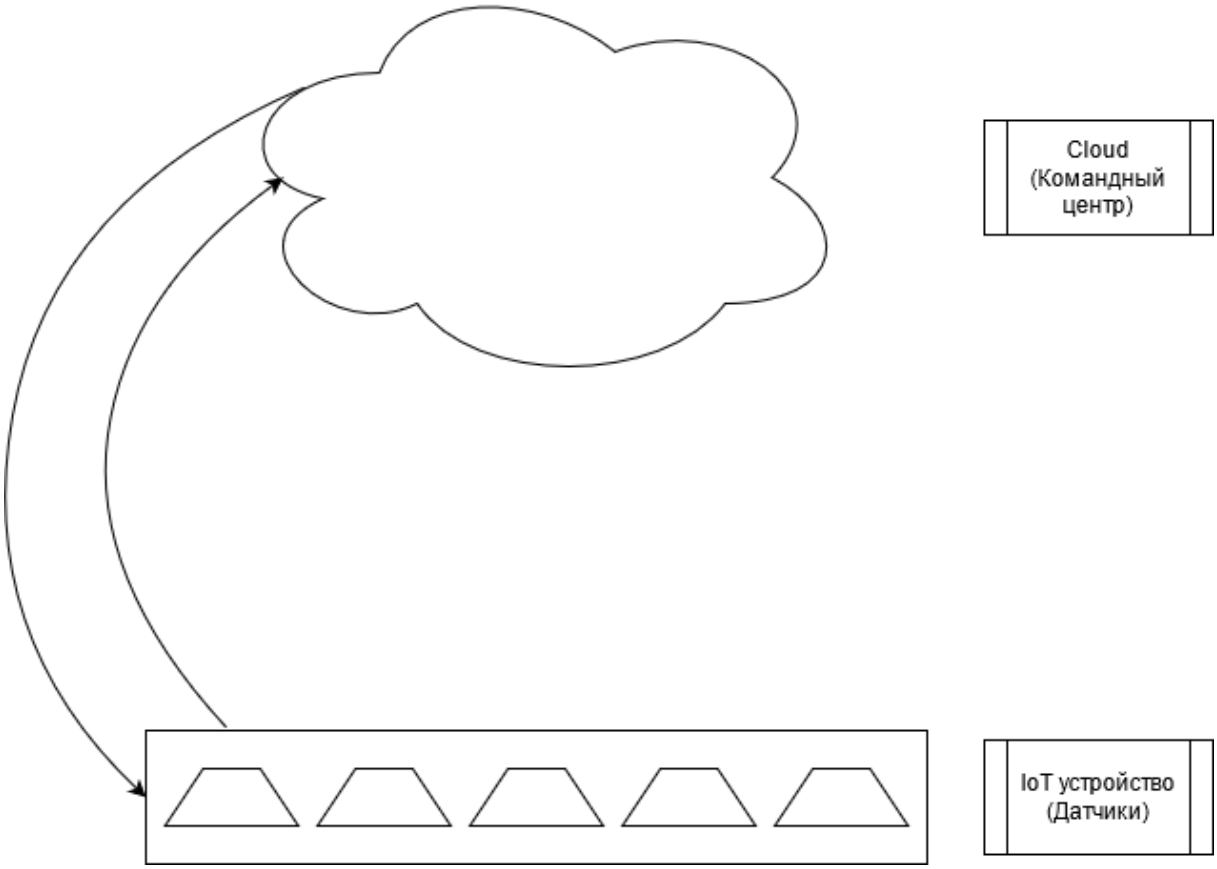

Рис. 2: Модель облачных вычислений, в которой все вычисления происходят в 'облаке'

Например, только на данный момент, порядка 40% всех данных в мире генерируется сенсорами, при том что всего генерируется 2,5 квантилиона байт данных в день. Современная облачная модель не в состоянии справится с таким количеством данных, поэтому потребовалась новая модель, которая в состоянии справиться с таким объёмом данных. Данной моделью стала модель туманных вычислений.

1.3 Туманные вычисления

Туманные вычисления - это одна из концепций Интернета вещей и является альтернативой облачных вычислений. Туманные вычисления предполагает обработку части данных на конечных устройствах сети (компьютерах, мобильных устройствах, датчиках, смарт-узлах и т.п.), а не в облаке(в данном случае центре управлении), решая таким образом основные проблемы, возникающие при организации интернета вещей.

Основное преимущество туманных вычислений заключается в том, что

они обладают низкой и предсказуемой задержкой передачи информации по сети, что крайне критично в системах требующих локальной обработки данных в реальном времени(такие как спутники). Также в туманных вычислениях не требуется связь с облаком. Отсюда получается, что если облако не доступно, но нет задач, решить которые можно только на облаке, то все вычисления можно провести в тумане, то есть туман самостоятельно разберётся с поставленными задачами и продолжит функционировать в стандартном режиме без ожидания появления доступа к облаку.

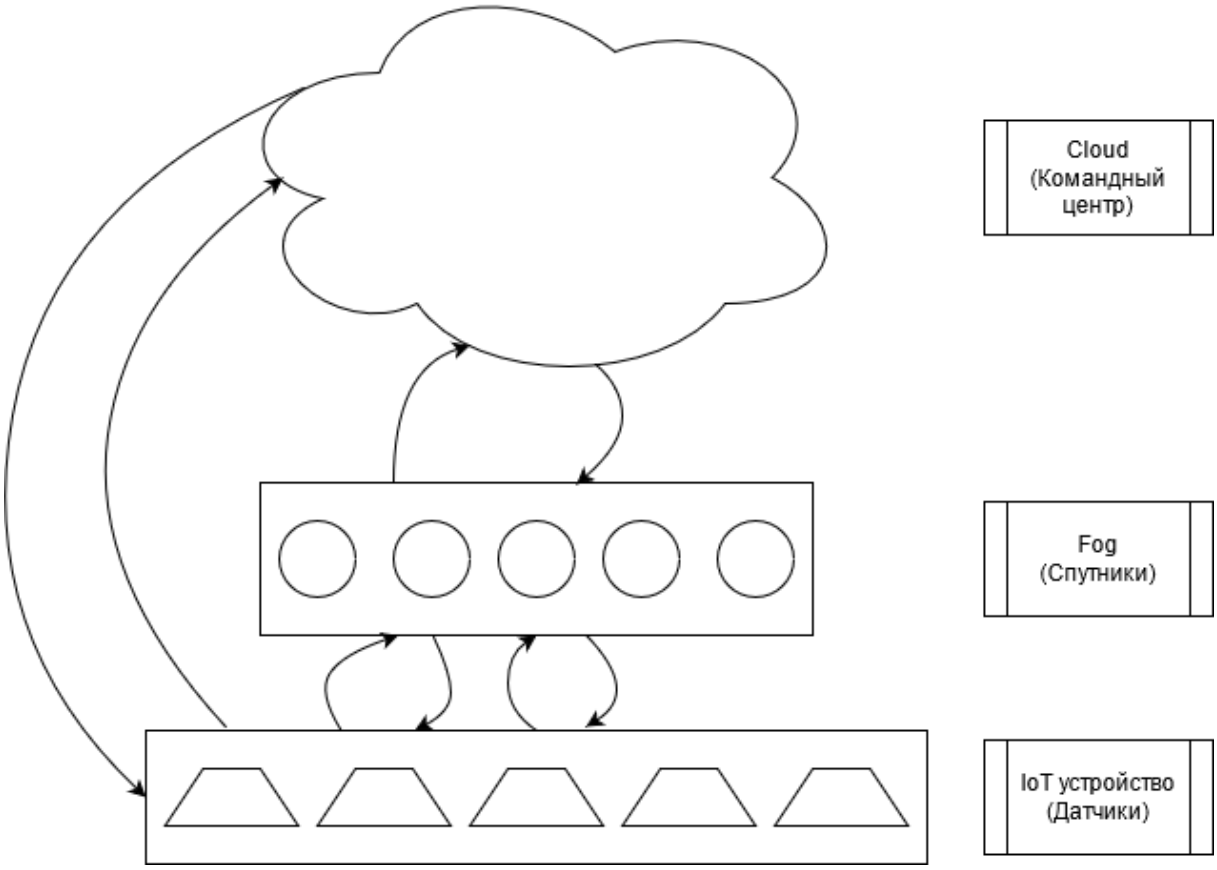

Рис. 3: Модель туманных вычислений, в которой вычисления могут происходить в 'тумане'

Таким образом, при объединении процессоров доступных спутнику в единую группу(Fog), за счёт увеличения вычислительной мощности, можно будет выполнять задачи, которые раньше требовали обращения в командный центр из-за нехватки ресурсов, прямо на спутнике, без обращения в командный центр, таким образом получая все преимущества туманных вычислений. Однако также нужно реализовать метод, который позволит адаптивно добавлять и удалять процессоры из вычислительной сети, т.к. постоянно поддерживать в активном состоянии все доступные процессоры будет требовать значительно больше энергии, которая ограничена на спутнике, таким образом нужно подключать в сеть минимально необходимое количество процессоров.

1.4 Сети Петри

При реализации было решено воспользоваться сетями Петри, из-за их возможности моделировать динамические дискретные системы (в частности асинхронные параллельные процессы). Введём все нужные понятия и определения.

Сети Петри — инструмент исследования систем. Теория сетей Петри делает возможным моделирование системы математическим представлением ее в виде сети Петри. Предполагается, что анализ сетей Петри поможет получить важную информацию о структуре и динамическом поведении моделируемой системы. Эта информация будет полезна для оценки моделируемой системы и выработки предложений по ее усовершенствованию и изменению.

Сеть Петри состоит из четырех элементов: множество позиций P, множество переходов T, входная функция I и выходная функция O. Входная и выходная функции связаны с переходами и позициями. Входная функция I отображает переход t_j в множество позиций $I(t_j)$ называемых входными позициями перехода. Выходная функция O отображает переход t_j в множество позиций $O(t_j)$ называемых выходными позициями перехода.

Структура сети Петри определяется ее позициями, переходами, входной и выходной функциями.

Определение. Сеть Петри C является четверкой, $C = (P, T, I, O)$. $P = (p_1, p_2, \ldots, p_n)$ - конечное множество позиций, $n \geq 0$. $T = (t_1, t_2, \ldots, t_m)$ - конечное множество переходов, , $m \geq 0$. Множество позиций и множество переходов не пересекаются, $P \cap T = \emptyset$. $I : T \to P$ является входной функцией, $O : T \to P$ есть выходная функция.

$$C = (P, T, I, O)$$

$$P = \{p_1, p_2, p_3, p_4, p_5\}$$

$$T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$$

$$I(t_1) = \{p_1\},\ I(t_2) = \{p_2\},\ I(t_3) = \{p_3\},$$
$$I(t_4) = \{p_4\},\ I(t_5) = \{p_2\},\ I(t_6) = \{p_5\},$$
$$O(t_1) = \{p_2\},\ O(t_2) = \{p_3\},\ O(t_3) = \{p_4\},$$
$$O(t_4) = \{p_1\},\ O(t_5) = \{p_5\},\ O(t_6) = \{p_4\},$$

Рис. 4: Структура сети Петри представлена в виде четверки, которая состоит из множества P позиций множества T переходов входной функции $I : T \to P$ и выходной функции $O : T \to P$

Определение. Мощность множества P есть число n, а мощность множества T есть число m. Произвольный элемент P обозначается символом p_i, $i = 1, \ldots, n$, а произвольный элемент T символом t_j, $j = 1, \ldots, m$.

Определение. Позиция p_i является входной позицией перехода t_j в том случае, если $p_i \in I(t_j)$. p_i является выходной позицией t_j, если $p_i \in O(t_j)$.

В значительной степени теоретическая работа по сетям Петри основана на формальном определении сетей Петри, изложенном выше. Тем не

менее для иллюстрации понятий теории сетей Петри гораздо более удобно графическое представление сети Петри. Теоретико-графовым представлением сети Петри является двудольный ориентированный мультиграф.

Определение. Граф G сети Петри есть двудольный ориентированный мультиграф $G = (V, A)$, где $V = (v_1, v_2, \ldots, v_n)$ - множество вершин, а $A = (a_1, a_2, \ldots, a_m)$ - направленные дуги, $a_i = (v_j, v_k)$ где $v_j, v_k \in V$. Множество V может быть разбито на два непересекающихся подмножества P и T таких, что $P \cup T = V$, $P \cap T = \emptyset$, и для любой направленной дуги $a_i \in A$, если $a_i = (v_j, v_k)$ тогда либо $v_j \in P$ и $v_k \in T$, либо $v_k \in P$ и $v_j \in T$.

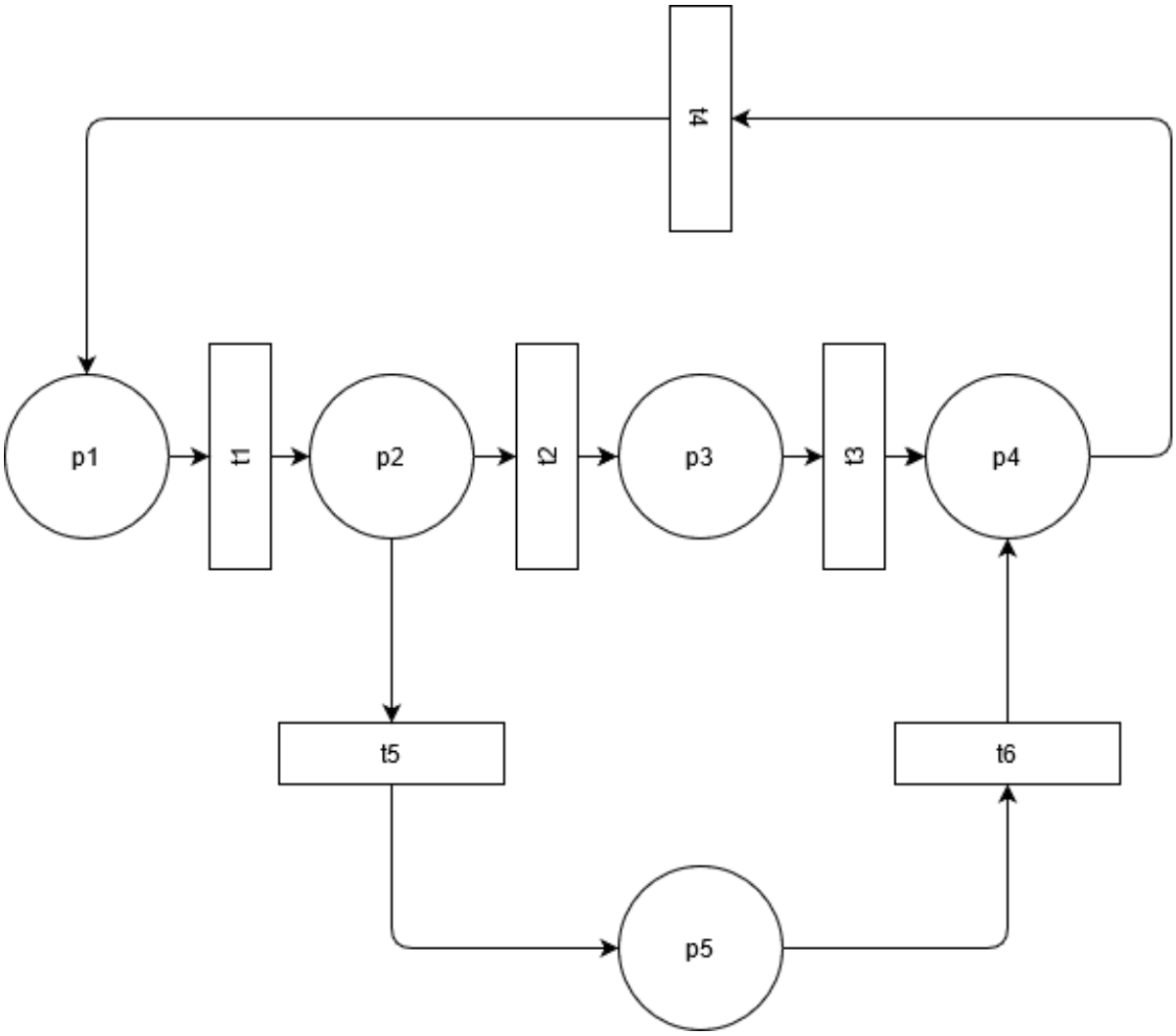

Рис. 5: Граф сети Петри, эквивалентный сети Петри на рисунке 4

При преобразовании сети Петри в граф, определим $V = P \cup T$. А определим как множество направленных дуг, таких, что для всех $p_i \in P$ и $t_j \in T$

$$\#((p_i, t_j), A) = \#(p_i, I(t_j))$$
$$\#((t_j, p_i), A) = \#(p_i, O(t_j))$$

Тогда $G = (V, A)$ есть граф сети Петри, эквивалентный структуре сети

Петри. Преобразование из графа в сеть Петри происходит аналогично.

1.5 Модели обслуживания в облачных вычислениях

Среди услуг облачных вычислений выделяют три основных типа

- Инфрастуруктура как сервис(IaaS)
- Платформа как сервис(PaaS)
- Программное обеспечение как сервис(SaaS)

1.5.1 IaaS

В IaaS(рис. 6) потребитель платит за получения доступа физическому и виртуальному аппаратному обеспечению(например сервера, диски для хранения информации), то есть платит за налаженную инфраструктуру, на которой может развернуть нужные ему операционые системы и ПО.

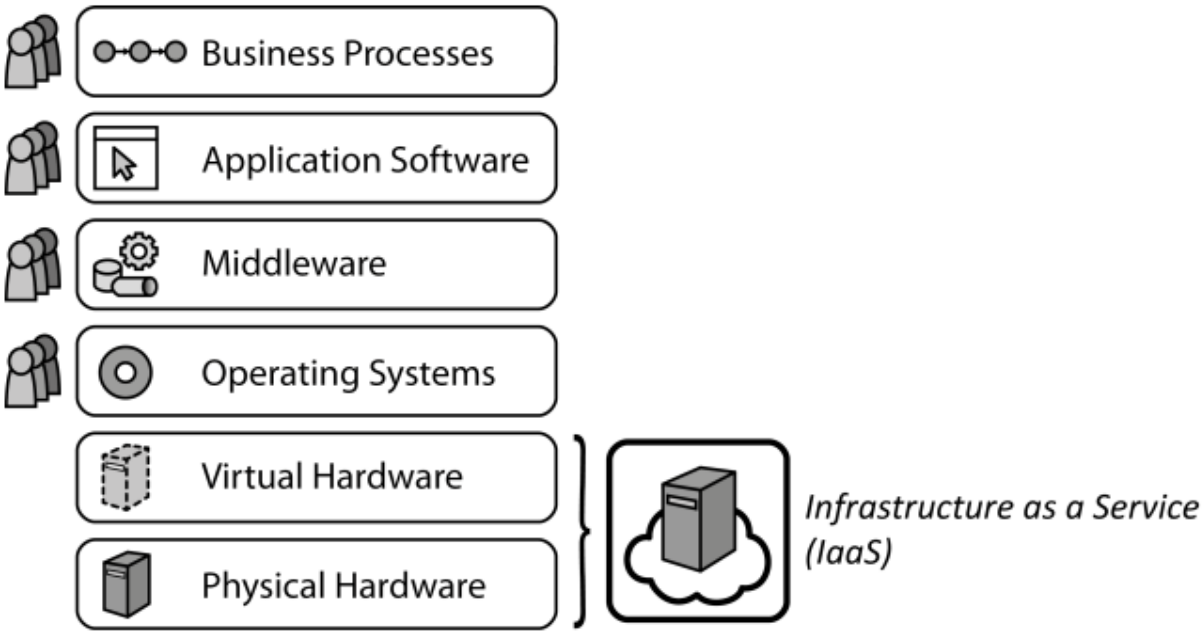

Рис. 6: Инфраструктура как сервис(IaaS)

1.5.2 PaaS

В PaaS(рис. 7) потребитель платит за получения доступа к операционной системе и связующему програмному обеспечению, на которых он может развернуть нужное ему ПО.

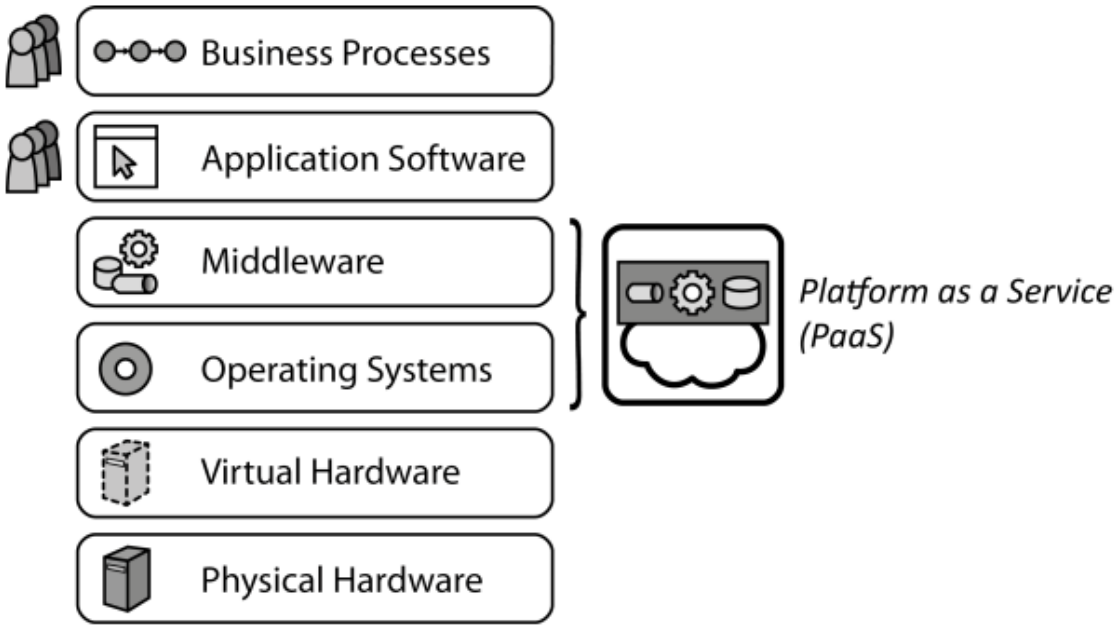

Рис. 7: Платформа как сервис(PaaS)

1.5.3 SaaS

В SaaS(рис. 8) потребитель платит за получения доступа к уже готовому и настроенному ПО, которое он может использовать для реализации нужных ему задач и бизнес процессов.

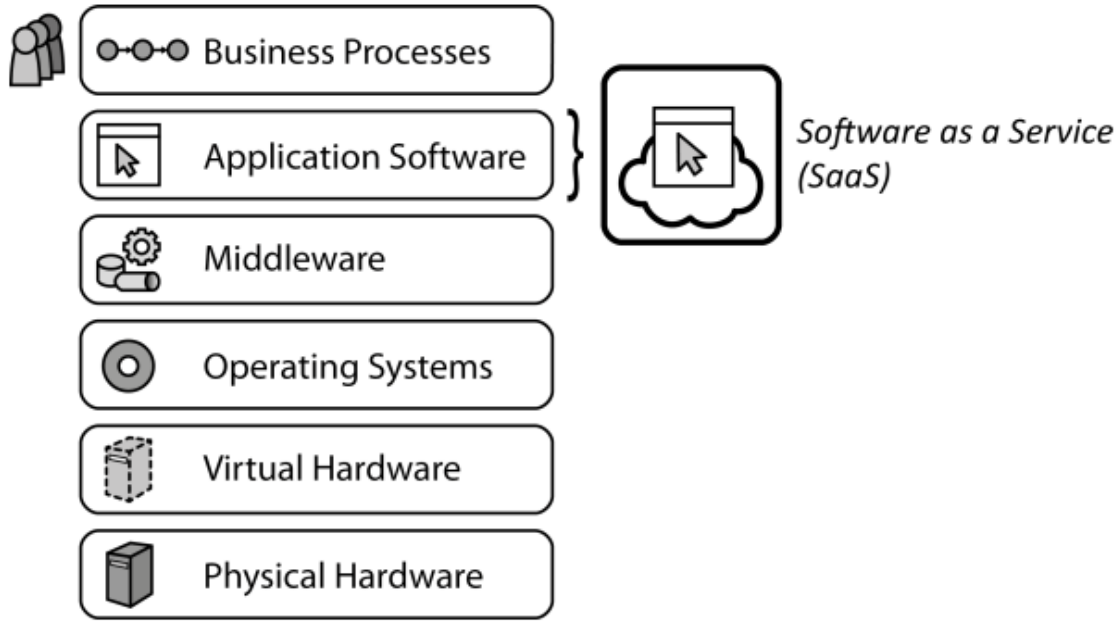

Рис. 8: Программное обеспечение как сервис(SaaS)

1.6 Виды нагрузок

Далее возникла необходимость рассмотреть существующие виды нагрузок, и рассмотреть какие характерны для космических аппаратов.

1.6.1 Статичная нагрузка

Статичная нагрузка. Эластичные облачные вычисления для статичной нагрузки чаще всего простому статичному выделению, из-за наличия оверхеда на организацию вычислений. Данный вид нагрузки не характерно для космических аппаратов(в частности спутников).

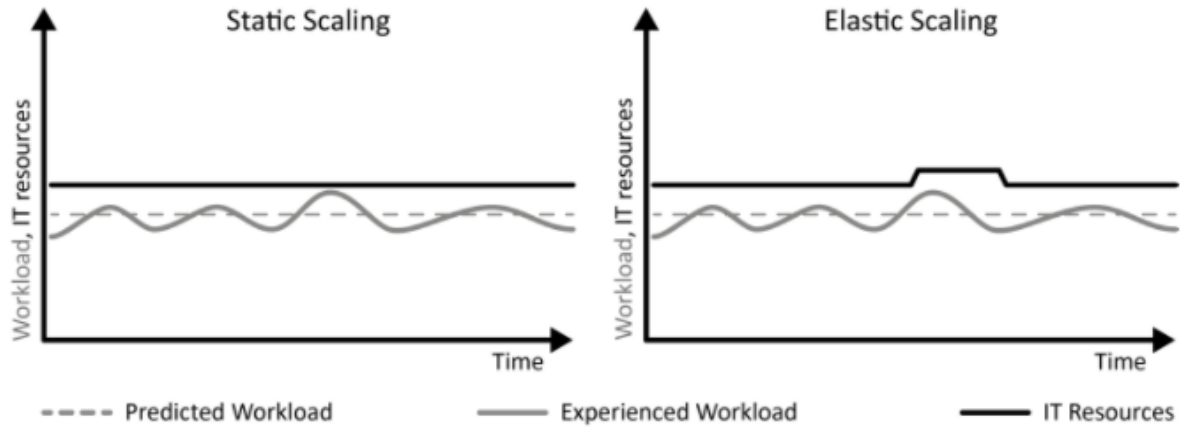

Рис. 9: Статичная нагрузка

1.6.2 Периодическая нагрузка

Периодическая нагрузка. В этом случае облачные вычисления выйгрывают за счёт освобождения ресурсов в период спадов нагрузки. Характерно для космических аппаратов в своей более общей форме - Once-in-a-lifetime.

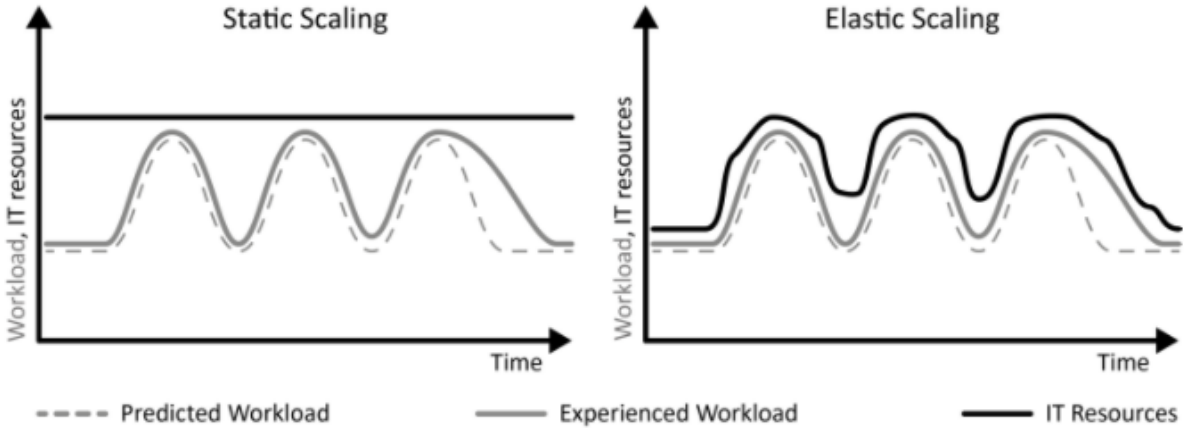

Рис. 10: Периодическая нагрузка

1.6.3 Once-in-lifetime нагрузка

Once-in-a-lifetime нагрузка. Отличие Once-in-a-lifetime нагрузки от переиодической заключается в том, что пик нагрузки происходит лишь раз в большой промежуток времени. Один из самых распространеных типов нагрузки для космических аппаратов т.к. оборудование часто получает какой-то набор команд, после выполнения которых уходит в длительный спящий режим до получения следующих команд.

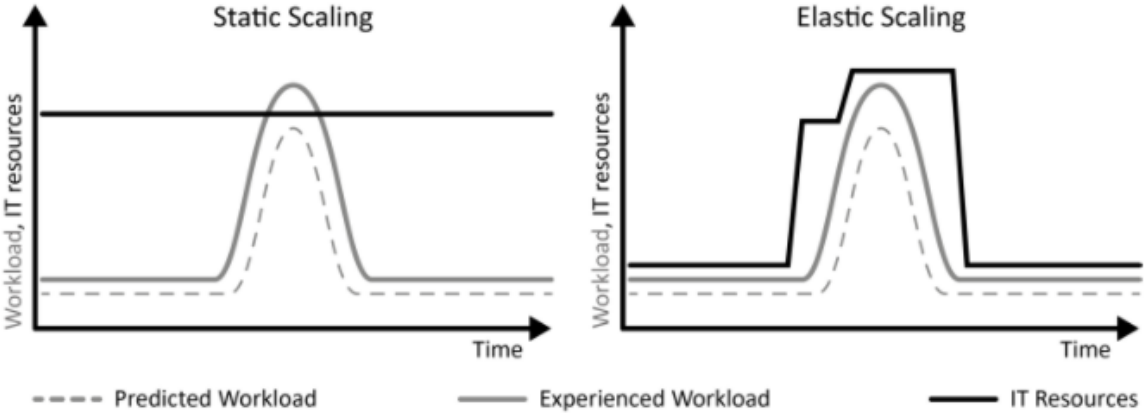

Рис. 11: Once-in-a-lifetime нагрузка

1.6.4 Непредсказуемая нагрузка

Непредсказуемая нагрузка. В случае такой нагрузки(довольно распространенной на практике) требуется постоянное выделение и освобождение ресурсов, для обеспечения выполнения нагрузки. Не характерна для спутников.

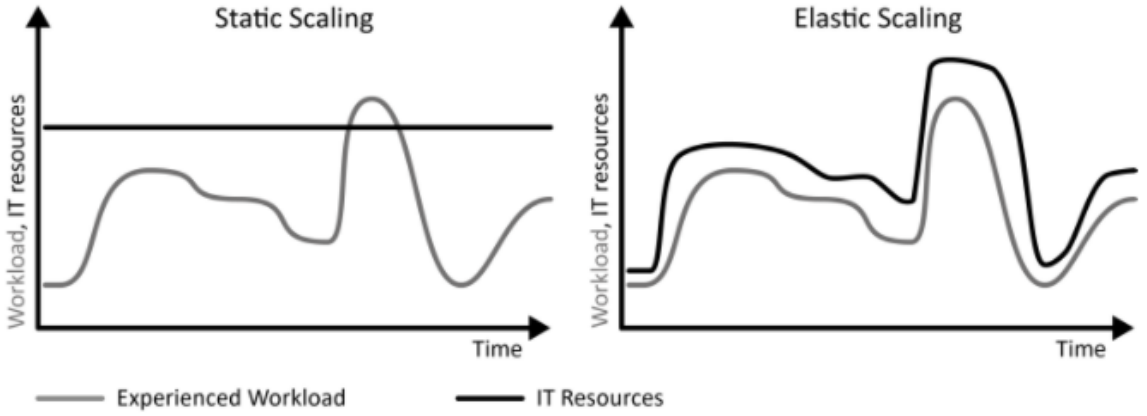

Рис. 12: Непредсказуемая нагрузка

1.7 Организация эластичных вычислений

Эластичные вычисления требуют реализации модели IaaS(нужно динамически добавлять новые и удалять старые ресурсы из инфраструктуры) и Paas(нужно динамически выделять и освобождать ресурсы в системе). Для этого разберем основные принципы реализации модели облачных вычислений.

1.7.1 Модель облачных вычислений

Есть некоторая работа(Job), которую можно разбить на набор заданий(Task), которые необходимо выполнить, причем выполнение некоторых из них могут идти параллельно на разных работниках(worker) - единицы вычислений, при этом некоторые задания могут быть выполнены только на некоторых работниках, таким образом нужен менеджр задач, который будет распределять задания между доступными работниками, при этом задания нужно распредилить так, чтобы успеть выполнить их до наступления их дедлайна. Таким образом достигается эластичность вычислений, то есть ресурсы выделяются если с имеющимися выделеными ресурсами задача не укладывается в дедлайны и освобождаются, если они имеются в излишке.

1.7.2 Модель дедлайнов

Отсюда следует необходимость оценки времени выполнения, для того чтобы уложится в дедлайны заданий которые необходимо выполнить. Данную оценку можно сделать статической, то есть найти максимальное время необходимое для выполнения задания, однако при таком подходе получается, что на выполнения задания всегда выделяется максимум ресурсов, независимо от того за сколько времени на самом деле выполняется работа на конкретном работнике, что приводит к неоптимальному использованию их ресурсов.

Поэтому стоит сделать оценку адаптивной, то есть вычислять сколько времени уходит на выполнение задания на каждом работнике, и из этих данных прогнозировать сколько задание займет времени в следующий раз. Этот подход более оптимален в плане использовании ресурсов, однако более сложен в реализации, а также вводит дополнительный оверхед, необходимый для вычисления времени выполнения. Однако выйгрыш ресурсов в результате оптимального их использования перевешивает оверхед, поэтому на практике используется именно адаптивная оценка времени.

Рассмотрим этот метод более подробно. Для адаптивной оценки понадобится построить модель знаний, руководствуесь которой, и будут выделяться дополнительные работники в случае если текущего количества нехватает для укладывания в дедлайн либо наоборот убираться лишние, если их излишек. При построении этой модели используются данные о времени выполнении для реальных работников, после анализа

которых существующая модель будет дополнена. Также эти данные будут использованы для симуляции вычислительной мощности работника в будущем.

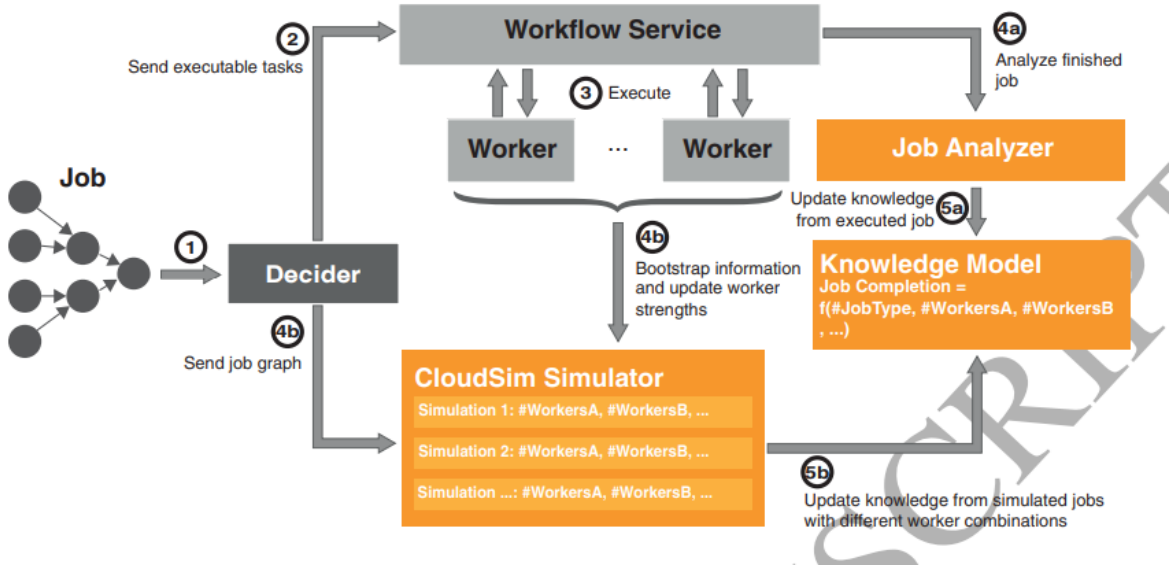

Рис. 13: Модель облачных вычислений с базой знаний

Однако данная система учитывает только время выполнения и пытается его минимизировать, то есть система нацелена только на то, чтобы обеспечивать лучшую скорость, игнорируя все остальные параметры, Однако в космическом аппарате есть много ограничений, например у него крайне ограниченный запас энергии, который медленно восстонавливается за счёт солнечных батарей. Таким образом при организации эластичных вычислений, целью ставиться не только минимизировать время выполнения, но также минимизировать энергозатраты, при этом в зависимости от выполняемой работы мы можем как ставить в приоритет скорость(например при уклонении от резко появившигося мусора), так и энергию(например при проверке состоянии аппарата).

Из этого следует необходимость использование эвристической функции, учитывающей как время выполнения, так и энергитические затраты, при этом влияние первого и второго зависит от типа выполняемой работы.

Для этого понадобиться знать энергопотребление при выполнении работы, это опять можно сделать либо статичекси, задав энергопотребление для каждой работы, либо адаптивно, с реалзицией модели знаний. Соответсвенно опять нужен профилировщик, который будет показывать потребление энергии для каждой задачи у каждого работника.

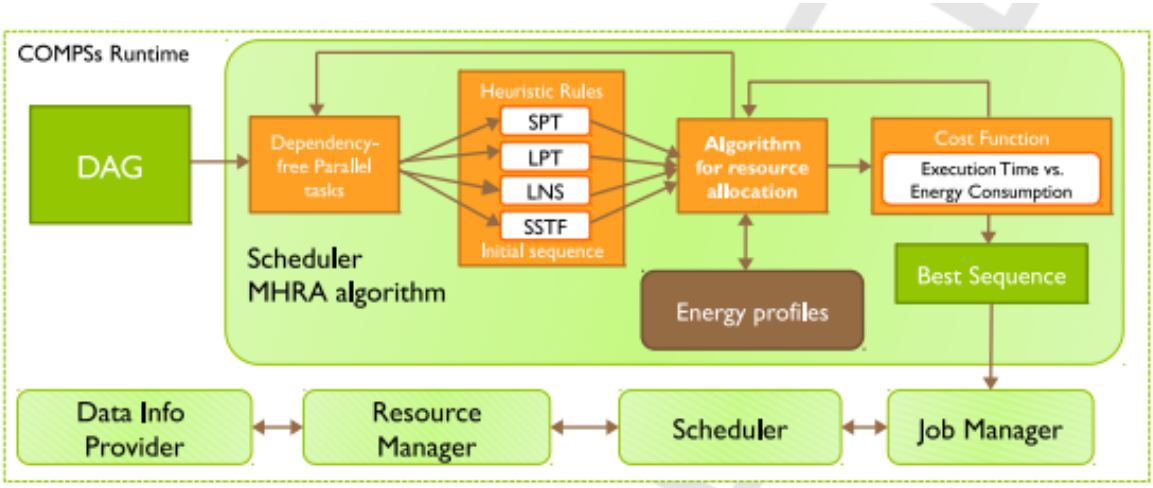

Рис. 14: Модель учитывающая энергитическое потребление

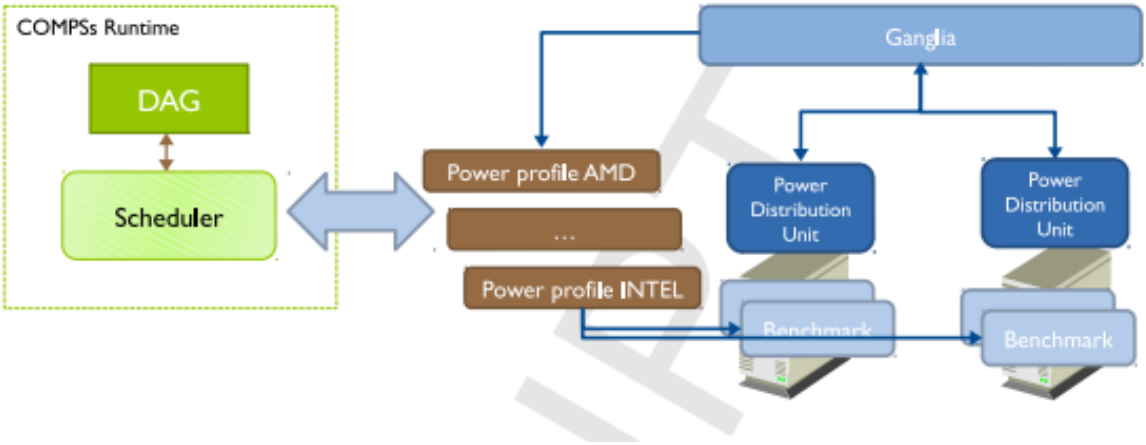

Рис. 15: Профилирование потребления энергитического

1.7.3 Модель ресурсов

Остановимся подробнее на том, на модели выделения ресурсов для выполнения задач. В самом простом виде выполнение задачи можно представить с помощью сети Петри, как показано на рисунке 16.

В данной схеме не учитывается необходимость выделения ресурсов для выполнения программы (рисунок 17), а также возможность наличия приоритета у задач (рисунок 18), в сотвествии с которым задачи с более низким приоритетом должны получать ресурсы для выполнения только после того, когда их получили задачи с более высоким приоритетом. Для выполения задачи могут требоваться разные типы ресурсов(память, процессоры), но для простоты представления они будут объедены в один тип.

Рис. 16: Простое представление выполнения задачи

Также задачи могут быть разных типов, и требовать для выполнения не пересекающиеся типы ресурсов, как на рисунке 19

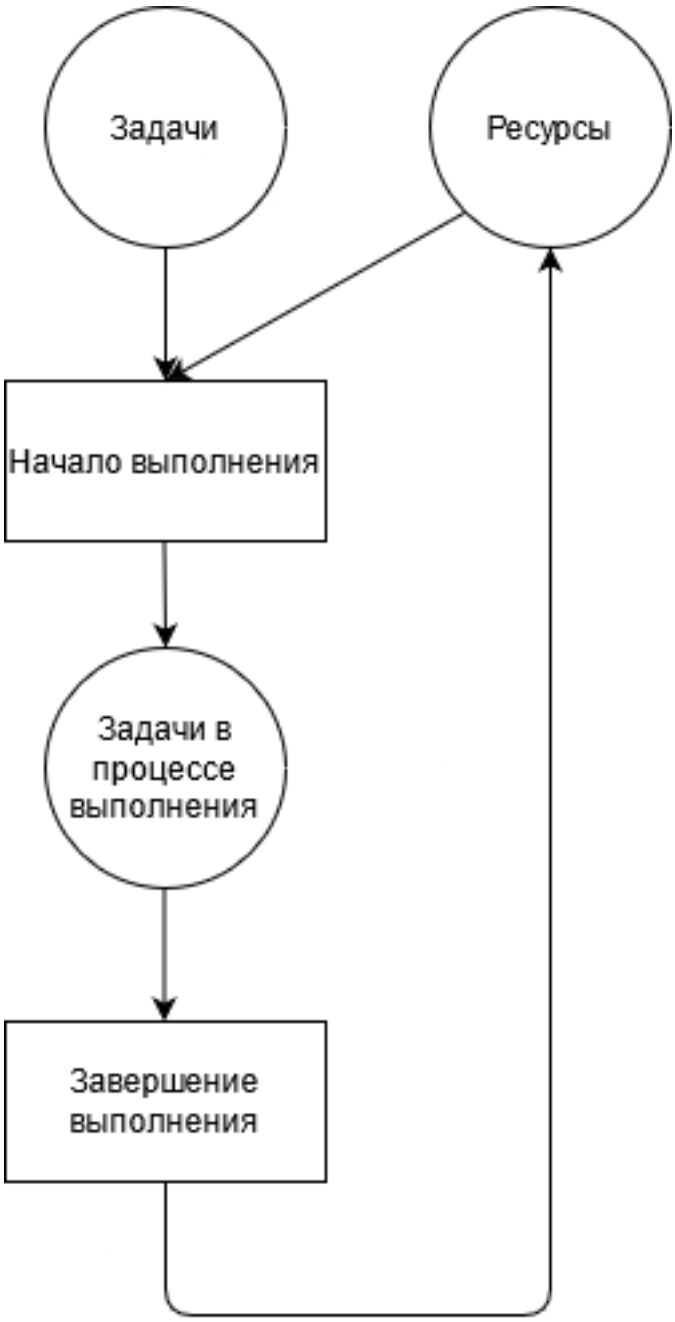

Рис. 17: Представление, учитывающие ресурсы

Рассмотрим простую представление с ресурсами. В данном представлении не учитывается что:

- ресурсы могут входить и выходить из вычислительной сети
- ресурс выделенный на выполнение задачи, может выйти из строя во время выполнеия задачи
- возможность добавление новых и выхода старых ресурсов из вычислительной сети
- уход ресурсов в спящий режим при их избытке
- переход ресурсов из спящего режима в активный при их нехватке

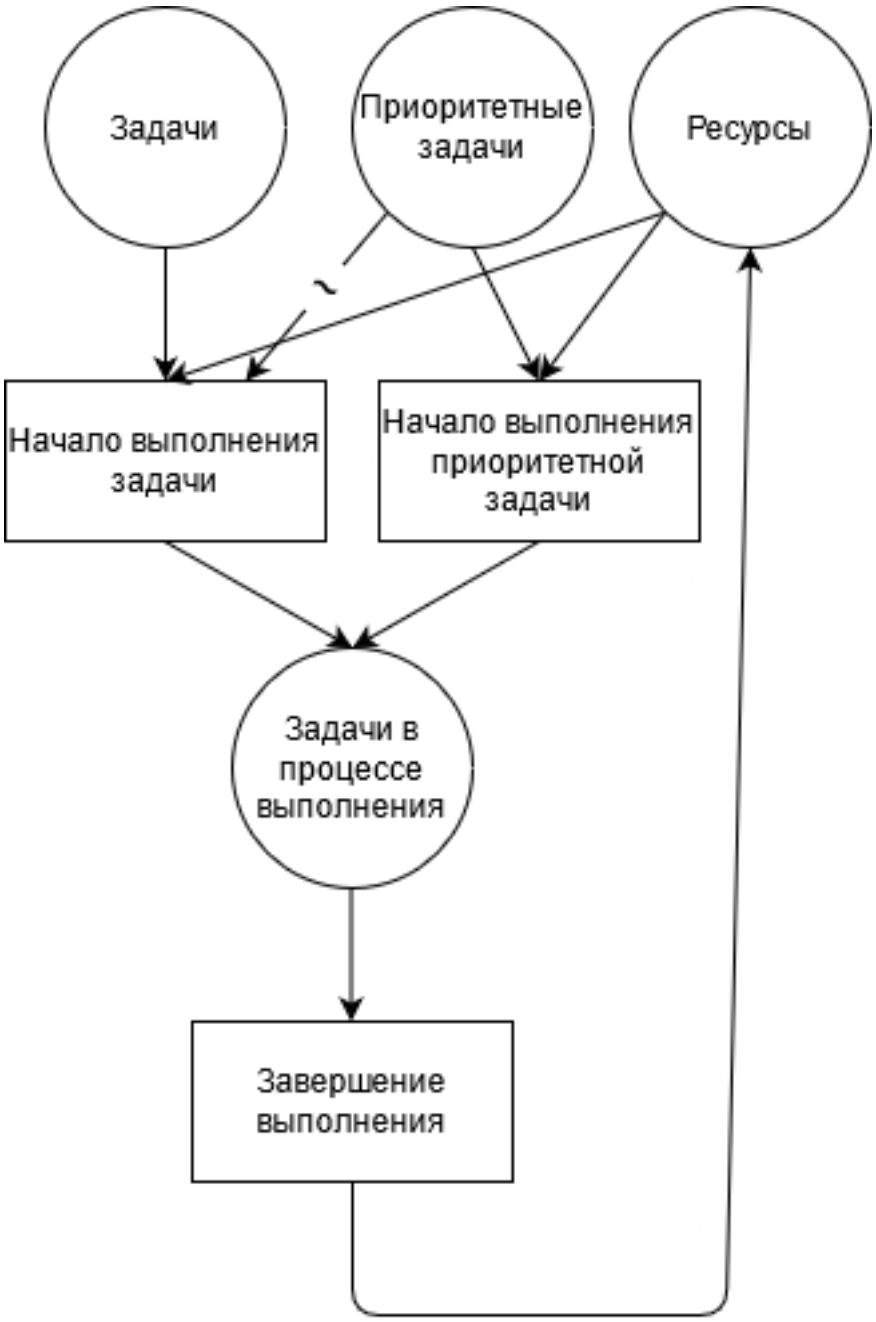

Рис. 18: Представление, учитывающие приоритет задач

Все эти нюансы учитываются на сети Петри представленной на рисунке 20. Приоритетные задачи и разные типы задач не были представлены чтобы сохранить читаемость схемы.

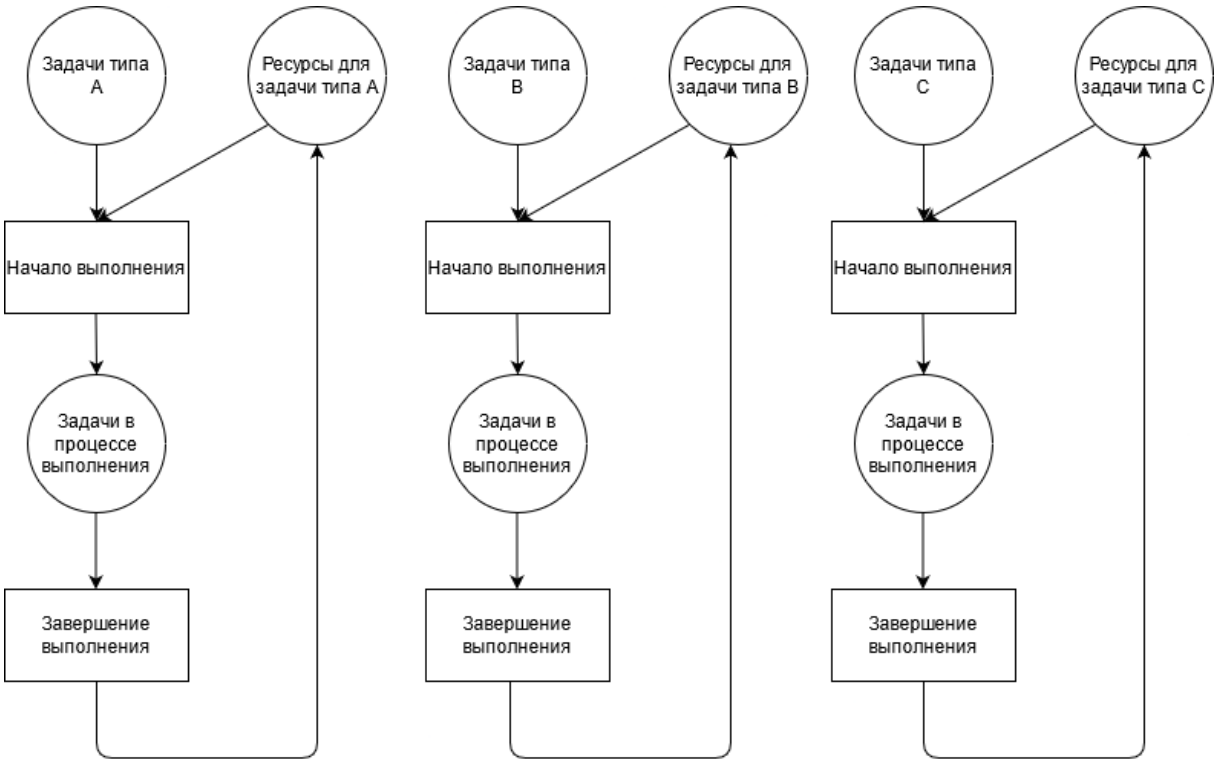

Рис. 19: Представление, с разными типами задач

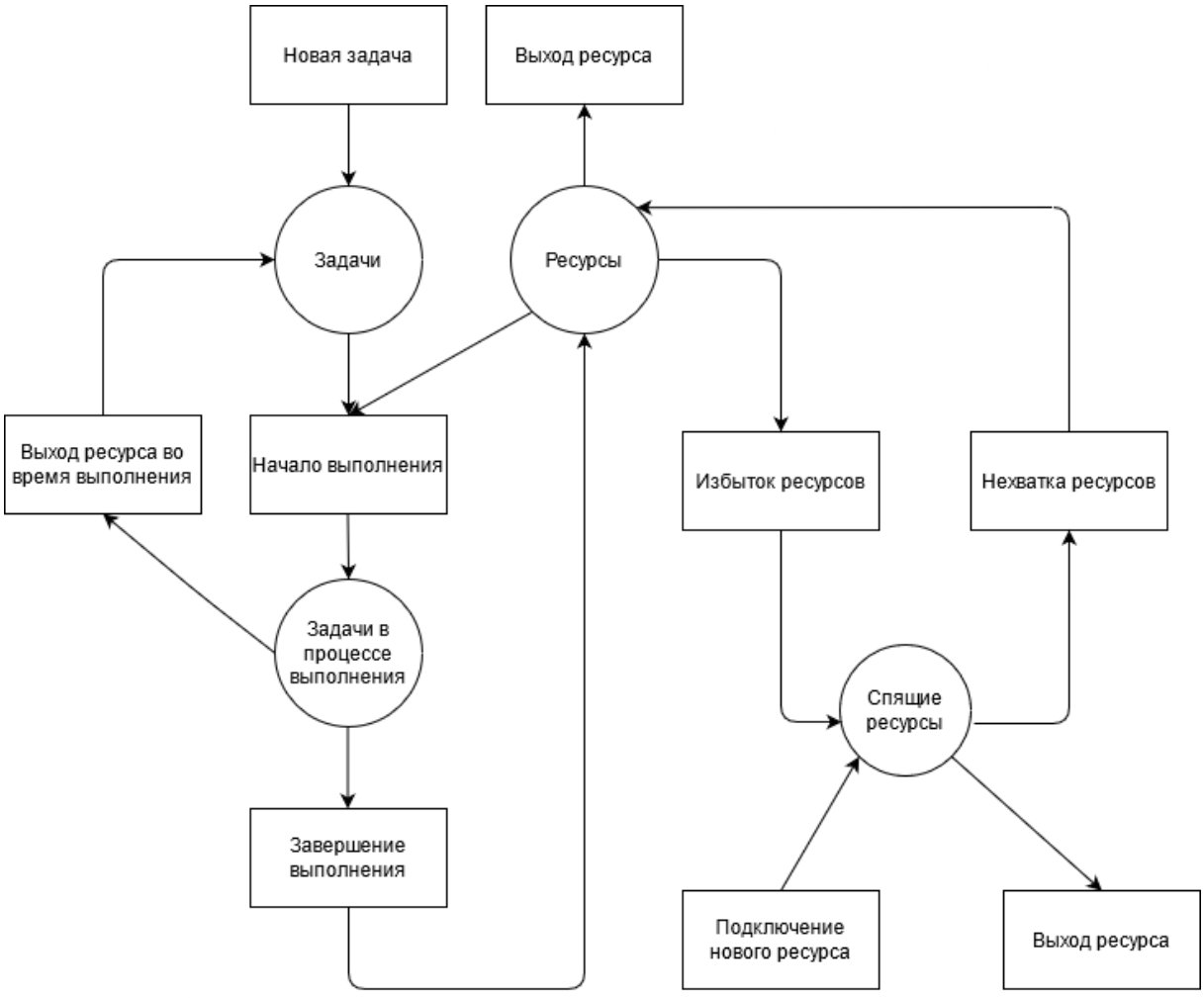

Рис. 20: Продвинутое представление

2 Практическая часть

Для реализации эластичных вычислений было решено воспользоват Написать программу на языке python с использованием библиотеки flask, для организации эластичных вычислений в рамках одной NAT сети. Для проверки работы в NAT сеть были подключены 2 устройства(ноутбук и телефон), после чего им на выполнение начали подаваться задачи. Полный код программы можно посмотреть на github.

Были получены данные выполнения и по ним построены графики количества невыполненых задач, количества узлов сети и количество активных узлов в сети в течении 10 минут

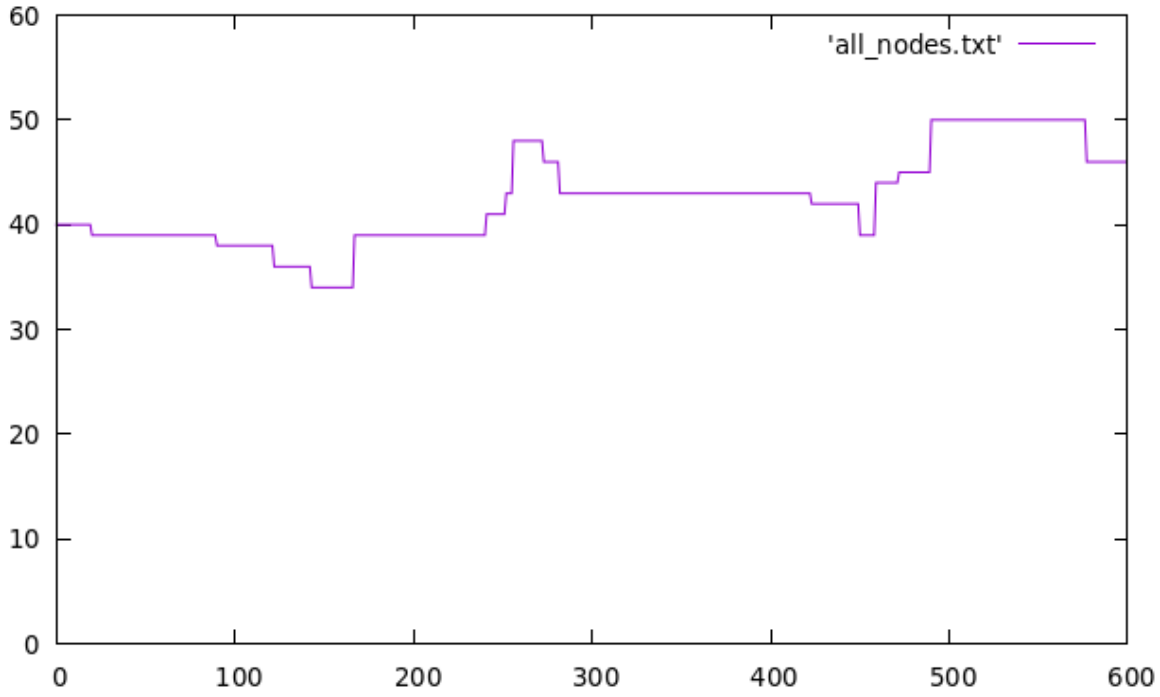

Рис. 21: Количество узлов в сети

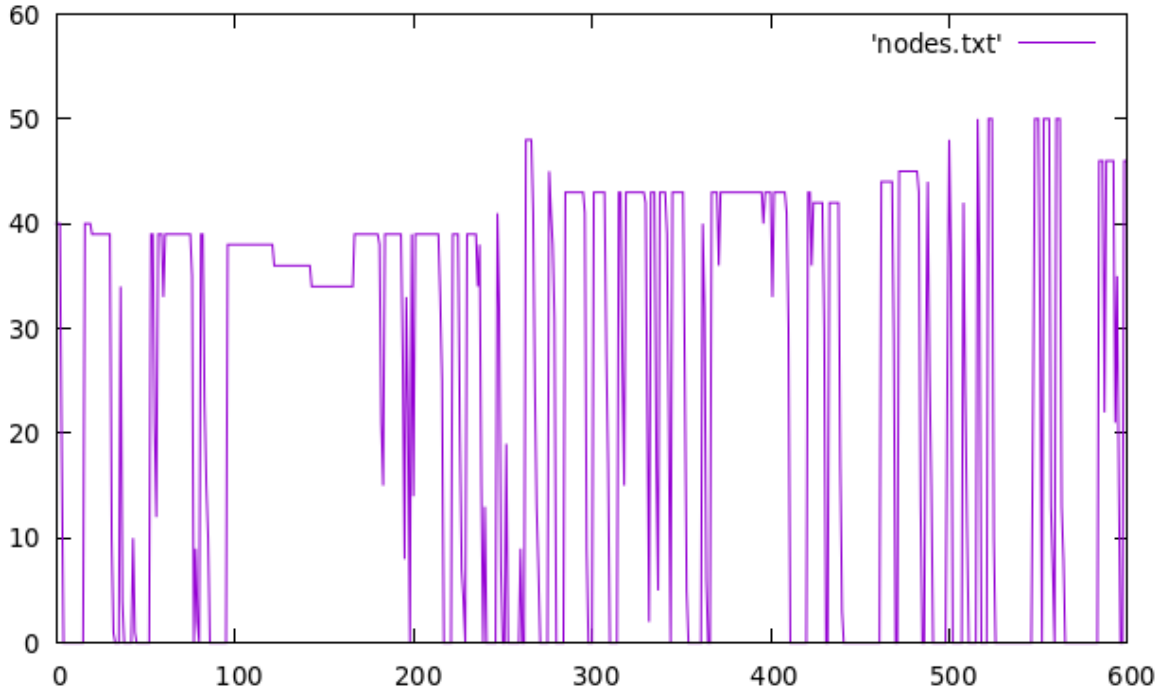

Рис. 22: Количество активных узлов

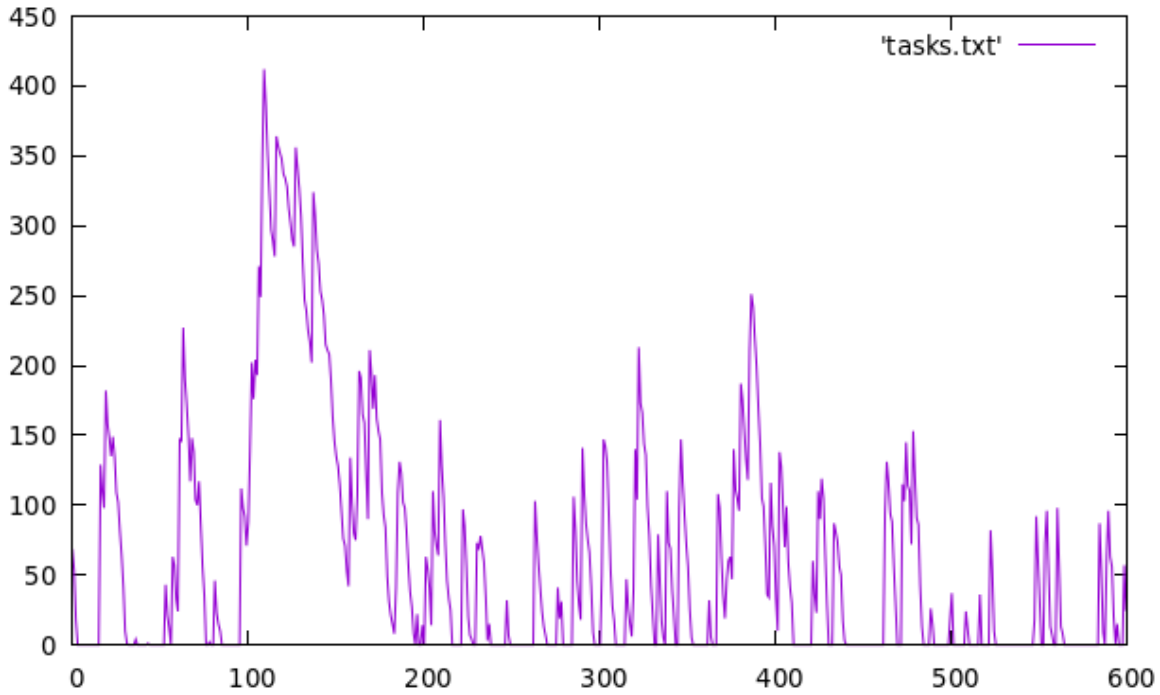

Рис. 23: Количество не выполненых задач

Каждое устройство имеет очередь задач, в которые могут добавляться новые задачи, которые будут ассинхроно выполнены процессами на устройстве.

Реализованны разные типы задач, которые могут выполняться только соотвествующими процессами на узлах.

Листинги

Листинг 1: Код отправки задания на узел

```
async def send_work(work):
    fails =0
    pinfo('Send')
    while fails<100:
        try:
            piece,value, shift ,type=work['piece'],work['val' ], work['shift '
                ], work['type']
            pinfo('Type {}'.format(type))
            pinfo('Dict {}'.format(jobs_types))
            node=jobs_types[type].get()
            pinfo('Node {}'.format(node))
            if node==None:
                raise NoFreeNodes("Don't get free nodes in timeout")
            url='http://' + node +'/upload'
            with open(work['input'],'rb') as f:
                response = requests.post(url,params=work,files={'file':(
                    work['input_filename'],f)})
            response.raise_for_status()
            print(response.content)
            jobs_types[type].put_to(node)

            return True
        except HTTPError as ex:
            perror("Error occured on HTTP request:",piece)
```

```
            template = "An exception of type {0} occurred. Arguments:\n
                {1!r}"
            message = template.format(type(ex).__name__, ex.args)
            perror(message)
            if ex.status_code==404:
                faulty_nodes.put(node)
            else:
                f_count= node_fail_count.get(node)
                if f_count==None:
                    faulty_nodes.put(node)
                if f_count>3:
                    faulty_nodes.put(node)
                node_fail_count[node].add(1)
        except Exception as ex:
            perror("Error occured on thread: {}".format(piece))
            perror(ex)
            template = "An exception of type {0} occurred. Arguments:\n
                {1!r}"
            message = template.format(type(ex).__name__, ex.args)
            perror(message)
            jobs_types[type].put_to(node)
            fails =fails+1
    return False
```

Листинг 2: Код ждущий получения выполнененого задания с узела

```
async def recv_work(work):
    fails =0
    global tasks_done
    while fails<100:
        try:
            job=tasks_done.get(str(work['job']))
            piece=str(work['piece'])
            if job==None:
                break
            with update:
                while True:
                    update.wait()
                    filename=job.get(piece)
                    if filename!=None:
                        #trace_count[work['type']].add(-1)
                        tasks_to_be_done.add(-1)
                        return filename
        except Exception as ex:
            print("Error occured on recv work:{}".format(work))
            template = "An exception of type {0} occurred. Arguments:\n
                {1!r}"
```

```
            message = template.format(type(ex).__name__, ex.args)
            print(message)
             fails =fails+1
    return None
```

Листинг 3: Код класса Worker, выполняющий задания из очереди queue

```
class Worker(threading.Thread):
    def __init__(self, q, other_arg, *args, **kwargs):
        self.q = q
        self.other_arg = other_arg
        super().__init__(*args, **kwargs)
    def run(self):
        global tasks_done
        while True:
            try:
                work = self.q.get()    # 3s timeout
                print("Got work",work)
                if work['fail']>3:
                    pwarning('Discard work:',work)
                    adr=CENTRAL_NODE#''+work['ip'] + ':' + work['
                        socket']
                    url='http://' + adr + '/discard'#socket to 5000?
                    response = requests.post(url,params={'job':work['job'
                        ],'piece':work['piece']})
                elif work['type'] in self.other_arg:
                    type=work['type']
                    trace_count[type].add(1)
                    pinfo(work)
                    adr=CENTRAL_NODE
                    pinfo(CENTRAL_NODE)
                    url='http://' + adr + '/result'#socket to 5000?
                    not_done=False
                    pinfo(tasks_done)
                    with tasks_done_lock:
                        if not (work['job'] in tasks_done):
                            tasks_done[work['job']]={}
                        elif not work['piece'] in tasks_done[work['job'
                            ]]:
                            not_done=True
                    if not_done:
                        main_work(work['input_file'],work['result_file'])
                        tasks_done[work['job']][work['piece']]=work['
                            result_filename']
                    with open(work['result_file'], 'r+') as f:#f=work['
                        file']
                        response = requests.post(url,params={'offset':
```

```
                        work['offset'], 'job':work['job'], 'piece':work[
                        'piece']}, files ={'file': (work['
                        result_filename'], f)})
                    response.raise_for_status()
                    pinfo(response.content)
                trace_count[type].add(-1)
                # If the response was successful, no Exception will
                    be raised
            else:
                logging.error("Not mine type")
                work['fail']=work['fail']+1
                self.q.put(work)
        except Exception as ex:
            print("Error occured on work:",work)
            template = "An exception of type {0} occurred.
                Arguments:\n{1!r}"
            message = template.format(type(ex).__name__, ex.
                args)
            print(message)
            work['fail']=work['fail']+1
            self.q.put(work)
            if not (type is None):
                trace_count[type].add(-1)
        finally:
            if close_workers:
                break
            #response = jsonify(error.to_dict())
            #response.status_code = error.status_code
            #return response
        # do whatever work you have to do on work
        self.q.task_done()
    pinfo("Worker down")
    down_workers.add(1)
```

Листинг 4: Код высылающий задание и ждущий ответ от узла

```
async def job_manage(work,timeout_send=60.0,timeout_recv=120.0):#

    done=False
    filename=None
    try:
        pinfo(work['input_filename'])
        response = send_work(work)#await asyncio.wait_for(send_work(
            work), timeout=timeout_send)
        filename = recv_work(work)#await asyncio.wait_for(recv_work(
            work), timeout=timeout_recv)
        response=await response
```

```
        pinfo(response)
        filename=await filename
        pinfo("Done recv")
        if filename=='failed':
            done = False
        else:
            done=True
    except asyncio.TimeoutError as ex:
        print("Task take too long. Arguments:\n{}".format(ex.args))
    except Exception as ex:
        print("Error occured on thread:",piece)
        template = "An exception of type {0} occurred. Arguments:\n{1!r
            }"
        message = template.format(type(ex).__name__, ex.args)
        print(message)
    finally:
        return done,filename
```

Листинг 5: Код который собирает результат выполнения всех заданий вместе

```
async def wait_job_done(work,shift=0,close_loop=True):
    pinfo("Waiting for job to be done")
    done=True
    messed_tasks=[]
    work_directory='./jobs/j{}/'.format(work['job'])
    input_dir=work_directory+'inputs/'
    done_job_dir=work_directory+'results/'
    pinfo(done_job_dir)
    try:
        job_filename=done_job_dir+'job{}.txt'.format(work['job'])
        results=list()
        work_amount=work['amount']
        for i in range(work_amount):
            work['piece']=i
            work['input_filename']='i_job{}_{}.txt'.format(work['job'],i)
            work['result_filename']='r_job{}_{}.txt'.format(work['job'],i
                )
            #results.append(job_manage(work))
            pinfo(i)
            results.append(asyncio.create_task(job_manage(copy.copy(
                work))))
        for i in range(work_amount):
            try:
                result = await results[i]
                #result=results[i]
                if result[0]:
```

```
                    task_filename=result[1]
                    concatenate_files(job_filename,task_filename)#i,
                        offset
                else:
                    print("Task {} not complete".format(i))
                    messed_tasks.append(i)
                    done = False
            except Exception as ex:
                print("Error occured on job piece:",i)
                template = "An exception of type {0} occurred.
                    Arguments:\n{1!r}"
                message = template.format(type(ex).__name__, ex.
                    args)
                print(message)
                done = False
                messed_tasks.append(i)
        print("Job done")
        return done,messed_tasks
    except Exception as ex:
        print("Error occured on job: {}",work['job'])
        template = "An exception of type {0} occurred. Arguments:\n{1!r
            }"
        message = template.format(type(ex).__name__, ex.args)
        print(message)
        done = False
    finally:
        return done,messed_tasks
```

Листинг 6: Код треда в котором начинается выполнение новой работы

```
def do_job(job):
    global tasks_done
    job_input[job['job']]=job
    asyncio.run(wait_job_done(job))
```

Листинг 7: Код который получает новую работу для последующего выполнения

```
@app.route('/start_job', methods=['GET', 'POST'])
def start_job():
    global tte
    global cur_node
    global job_num
    global tasks_done
    if request.method == 'POST':
        #job_loop = asyncio.get_event_loop()
        cur_job=job_num.add(1)
        value = request.args.get('value')
        dir_list=request.args.get('dir_list')
```

```
        if dir_list==None:
            dir_list=STD_JOB_DIR_LIST
        tasks_done[str(cur_job)]={}
        work_amount=request.args.get('amount')
        type=request.args.get('type')
        work={}
        work['val']=value
        work['amount']=work_amount
        work['shift']='2'
        work['job']=cur_job
        work['type']=type
        work['adr']=''+self_adr()
        work['offset']=0
        work['socket']=SOCKET
        work['input']='random.txt'
        create_job_dir(cur_job,dir_list)
        tasks_to_be_done.add(work_amount)
        threading.Thread(target=do_job,args=[work]).start()
        return redirect(url_for('result', name=value))
    return '''
    <!doctype html>
    <title>Start Job</title>
    <h1>Start Job</h1>
    <form method=post>
      <label for="fname">Value:</label><input type="text" id="value"
          name="fname"><br><br>
      <label for="amount">Amount:</label><input type="text" id="
          amount" name="fname"><br><br>
      <label for="type">Type:</label><input type="text" id="type"
          name="fname"><br><br>
      <input type=submit value=Submit>
    </form>
    '''
```

Листинг 8: Код, с помощью которого к главному узлу присоединяются новые узлы

```
@app.route('/new_node', methods=['GET','POST'])
def new_node():
    global node_num
    global node_dict
    global jobs_types
    temp=dict(request.args)
    print(temp)
    adr=request.remote_addr+':'+request.args.get('socket')
    task_types=request.args.get('task_types')
    if task_types is None:
        raise InvalidUsage("Task types not provided")
```

```
    task_types=json.loads(task_types)
    pinfo(task_types)
    for i in task_types:
        jobs_types[i].set_node(adr,my_queue(task_types[i]))
    pinfo(jobs_types)
    node_dict[adr]=node_num.add(1)
    return json.dumps(node_dict)
```

Листинг 9: Код с помощью которого узлы присоединяются к главному узлу

```
@app.route('/connect', methods=['GET', 'POST'])
def connect():
    global cur_node
    global jobs_nums
    global node_dict
    if request.method == 'GET':
        fails=0
        while fails<10:
            try:
                adr=CENTRAL_NODE
                url=r'http://' + adr + r'/new_node'
                response = requests.post(url,params={'task_types':json.
                    dumps(jobs_nums),'socket':SOCKET})
                response.raise_for_status()
                pinfo(response.text)
                node_dict=json.loads(response.text)#TODO
                pinfo(node_dict)
                cur_node=node_dict[self_adr()]
                pinfo(cur_node)
                return str(cur_node)
            except HTTPError as http_err:
                print(f'HTTP error occurred: {http_err}')
            except Exception as ex:
                perror("Error occured on connection. Number of failes:{}"
                    .format(fails))
                template = "An exception of type {0} occurred.
                    Arguments:\n{1!r}"
                message = template.format(type(ex).__name__, ex.
                    args)
                print(message)
                time.sleep(1)
            finally:
                fails=fails+1
        raise InvalidUsage("Failed To Connect",status_code=500)
    elif request.method == 'POST':
        cur_node=request.form['node']
    return str(cur_node)
```

Листинг 10: Код с помощью которого узел выходит из вычислительной сети

```
@app.route('/disconnect', methods=['GET', 'POST'])
def disconnect():
    global CENTRAL_NODE
    global jobs_types
    try:
        if request.method == 'GET':
            #global node_available
            #node_available=False
            node=self_adr()
            node_dict.pop(node, None)
            if node==CENTRAL_NODE:
                CENTRAL_NODE=next(iter(node_dict))
                min=node_dict[CENTRAL_NODE]
                for i,it in node_dict.items():
                    if it<min:
                        CENTRAL_NODE=i
                        min=it
            pinfo(CENTRAL_NODE)
            #TODO: Cloud storing central node
            pinfo(node_dict)
            for i in node_dict:
                fails=0
                while fails<MAX_FAILS:
                    try:
                        response=requests.post('http://'+i+'/disconnect',
                            params={'node':node,'c_node':
                            CENTRAL_NODE,'j_types':json.dumps(
                            jobs_types,default=lambda o: o.val())})
                        response.raise_for_status()
                        break
                    except Exception as ex:
                        perror("Disconect Exception: {}. Exception type:
                            {}. Arguments: {!r}".format(fails, type(ex).
                            __name__, ex.args))
                        fails=fails+1
        elif request.method == 'POST':
            node = request.args.get('node')
            c_node = request.args.get('c_node')
            j_types=request.args.get('j_types')
            jobs_types=dict_to_j_types(json.loads(j_types),node)
            pinfo("Disc: node {}".format(node))
            if node:
                try:
```

```
                    node_dict.pop(node, None)
                    pinfo(node_dict)
                except:
                    perror("Disconect: delete node exception. Exception
                        type: {}. Arguments: {!r}".format( type(ex).
                        __name__, ex.args))

            change_central_node(c_node)
    except Exception as ex:
        perror("Unexpected exception occured on disconnect. Exception
            type: {0}. Arguments: {1!r}".format(type(ex).__name__, ex
            .args))
        raise InvalidUsage("Unexpected exception occured on disconnect.
            Exception type: {0}. Arguments: {1!r}".format(type(ex).
            __name__, ex.args))
    pinfo(CENTRAL_NODE)
    pinfo(node_dict)
    return 'Disconected {}'.format(node)
```

Листинг 11: Код с помощью которого новое задание загружается на узел

```
@app.route('/upload', methods=['GET', 'POST'])
def upload():
    global tte
    global cur_node

    if request.method == 'POST':
        work={}
        for i in request.args:
            work[i]=request.args.get(i)
        work_directory='./jobs/j{}/'.format(work['job'])
        input_dir=work_directory+'inputs/'
        done_job_dir=work_directory+'results/'
        pinfo(work['result_filename'])
        if tasktype_queues[work['type']].full():
            raise InvalidUsage('Queue is full', status_code=503)
        file = request.files['file']
        if file.filename == '':
            raise InvalidUsage('No file found')
        if not file:
            raise InvalidUsage("Can't open file",status_code=506)
        work['input_file'] = os.path.join(input_dir,    work['
            input_filename'])
        work['result_file']= os.path.join(done_job_dir,work['
            result_filename'])
        file.save(work['input_file'])
        work['fail']=0
```

```
            cur_job=work['job']
            dir_list=STD_JOB_DIR_LIST
            create_job_dir(cur_job,dir_list)
            tasktype_queues[work['type']].put(work)
            return str(cur_node)
        return '''
        <!doctype html>
        <title>Upload new Task</title>
        <h1>Upload new Task</h1>
        <form method=post enctype=multipart/form-data>
          <input type=file name=file>
          <input type=submit value=Upload>
        </form>
        '''
```

Листинг 12: Код который получает результат выполнения задания

```
@app.route('/result', methods=['GET', 'POST'])
def result():
    global dataUpdate
    global tasks_done
    if request.method == 'POST':
        # check if the post request has the file part
        if 'file' not in request.files:
            raise InvalidUsage('No file part', status_code=500)
        file = request.files['file']
        # If the user does not select a file, the browser submits an
        # empty file without a filename.
        if file.filename == '':
            raise InvalidUsage('No file attached', status_code=500)
        if file and allowed_file(file.filename):
            print("I save")
            filename=file.filename
            work={}
            #work['offset']=request.args.get('offset')
            #work['filename']=file
            #work['data']=file.read()
            work['job']=request.args.get('job')
            work['piece']=request.args.get('piece')
            print(tasks_done)
            with update:
                td=tasks_done.get(work['job'])
                if td == None:
                    print("Got job:{}".format(work['job']))
                    return "Job already done"
                pinfo(td)
                piece_num=td.get(work['piece'])
```

```
                #if piece_num != None:
                #    return "Task already done"
                work_directory='./jobs/j{}/results/'.format(work['job'])
                filename=os.path.join(work_directory,secure_filename(
                    filename))

                file.save(filename)
                td[work['piece']]=filename#TODO
                pinfo(tasks_done)
                pinfo(filename)
                update.notify_all()
            return "Got it"#redirect(url_for('result', name=filename))
        pinfo("Empty post request")
    return '''
    <!doctype html>
    <title>Upload new Result</title>
    <h1>Upload new Result</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
    '''
```

---

3 Заключение

В ходе выполнения выпускной квалификационной работы были рассмотренны методы организации эластичных вычислений с использованием интернета вещей и была выбрана модель туманных вычислений. Была написана программа на языке python с использованием вебфреймворка flask, предоставляющая пользователю платформу(PaaS), которая может объединять несколько устройств в единную вычислительную сеть, с последующим добавлением и выходом из неё узлов(ресурсов). На этой платформе пользователь может запустить своё приложение, выполнение которого автоматически распределяется между доступными узлами сети, и в случае, если узел отключился из сети во время выполнения, переводит выполнение задания на другой узел сети. Были произведены замеры производительности и построен график использования ресурсов, на котором видно что ресурсы используются оптимально, то есть ресурсы выходят из спящего режима, когда уже работающих ресурсов нехватает для выполнения заданий.

У эластичных вычислений в космосе довольно большой потенциал в будущем, т.к. в космическое пространство выводится всё больше новых космических аппаратов и постепено в космосе начинает развиваться рынок услуг. Таким образом человечество всё активнее осваивает космос и для этого понадобится большие вычислительных мощности. И если организовать эластичные вычисления не только в рамках одного аппарата, но объединить несколько разных аппаратов в единную сеть, то можно будет получить значительно большую вычислительную мощность, чем есть на любом отдельно взятом космическом аппарате, при этом хоть вычислительные возможности на земле и будут выше, но благодаря локальности(близкой рассположенности узлов друг от друга), такая сеть будет обладать значительно меньшей задержкой перед непосредственной началом выполнения задачи(например сигнал с Марса в мдёт минимум 3 минуты и максимум 24 минуты, в среднем получается порядка 13 минут), позволяя ей обрабатывать задачу гораздо быстрее, что особенно важно в случаях в случаях уклонения от космического мусора, приземления на планету и других задач требующих быстрого ответа. На землю же можно будет отправлять уже обработанные данные или особенно сложные задачи, которые нельзя решить в рамках туманной сети.

4 Список литературы

1. Burger, A., Cichiwskyj, C., Schmeißer, S., & Schiele, G. (2020). The Elastic Internet of Things - A platform for self-integrating and self-adaptive IoT-systems with support for embedded adaptive hardware. Future Generation Computer Systems.
2. Juarez, F., Ejarque, J., & Badia, R. M. (2018). Dynamic energy-aware scheduling for parallel task-based application in cloud computing. Future Generation Computer Systems, 78, 257–271.
3. Питерсон Дж. Теория сетей Петри и моделирование систем / Дж. Питерсон; пер. с англ. под ред. В.А. Горбатова – М.: Мир, 1984. – 264 с.
4. De Coninck, E., Verbelen, T., Vankeirsbilck, B., Bohez, S., Simoens, P., & Dhoedt, B. (2016). Dynamic auto-scaling and scheduling of deadline constrained service workloads on IaaS clouds. Journal of Systems and Software, 118, 101–114.
5. https://dzone.com/articles/implementing-fog-computing-for-iot-ecosystem
6. https://habr.com/ru/post/477526/
7. https://m.habr.com/ru/company/selectel/blog/551460/
8. https://m.habr.com/ru/company/selectel/blog/553446/
9. https://hi-news.ru/technology/kak-rabotayut-sputniki.html#kakaya_raznica_-mezhdu_sputnikom_i_kosmicheskim_musorom