НИТУ МИСИС

Институт ИТАСУ

Кафедра инженерной кибернетики

Квалификация (степень): **магистр**

Направление подготовки: 09.04.03 «Прикладная информатика»

Группа: МПИ-20-4-2

Курсовая работа по курсу

«Облачные технологии»

III семестр 2021–2022 у. г.

Распределённое асинхронное обучение нейронных сетей. Проверка эффективности обучения на эталонных и реальных датасетах

Преподаватель _____ / доцент, к.т.н. Курочкин И. И. /

Подпись должность, уч. степ., Фамилия И.О.

Учащиеся:

Андреев Савелий

Дзюба Ирина

Ефремова Марьяна

Новицкий Дмитрий

Оглавление

Введение 3

1. Аналитический обзор литературы 4

1.1. Распределённое обучение сети 4

1.1.1. Подходы к распределенной реализации SGD 4

1.1.2. Параллелизм данных и модели 4

1.1.3. Централизованные и децентрализованные архитектуры 6

1.1.4. Обновление глобальной модели 7

1.1.4.1. Синхронное обновление 8

1.1.4.2. Асинхронное обновление 8

1.1.4.3. Периодичность обновления 9

1.2. Архитектура BOINC 10

1.3. Globus Toolkit 12

1.4. Компоненты BOINC сервера 12

1.4.1. Web-сервер 12

1.4.2. База данных 13

1.4.3. Служба обработки состояния подзадач (Transitioner) 13

1.4.4. Служба проверки результатов (Validator) 14

1.4.5. Служба освоения (Assimilator) 15

1.4.6. Служба удаления файлов (File deleter) 15

1.4.7. Служба подачи (Feeder) 15

1.4.8. Планировщик (Scheduler) 15

1.4.9. Мост (Bridge) 16

1.4.10. Приложения BOINC 16

1.5. Жизненный цикл задания из проекта. 18

Список использованных источников 20

Введение

В исследованиях в математики, физики и многих других науках, требуется обработка информации или вычисление расчетных операций. С этой целью создавались системы, способные сократить время обработки в десятки раз.

В 1990-х годах возникла идея GRID, когда с развитием компьютерной техники и коммуникаций соединение между собой географически удаленных компьютеров стало выгоднее, удобнее и более мощным средством повышения вычислительных мощностей, чем наращивание производительности одного суперкомпьютера.

GRID – это объединение нескольких компьютеров для решения единой вычислительно сложной задачи, разбитой на подзадачи. Каждый компьютер при этом решает несколько подзадач, а результаты решения объединяются. Основное преимущество GRID в том, что система может состоять из удаленных друг-от-друга на тысячи километров и совершенно различные по характеристикам компьютеры.

Одной из форм реализаций GRID, является Volunteer Computing (Добровольные вычисления), её специфика заключается в использовании времени простоя компьютеров обычных пользователей.

BOINC (Berkeley Open Infrastructure for Network Computing) – это открытая программная платформа университета Беркли, для реализации GRID сетей. Причиной создания стала нехватка свободных вычислительных мощностей для обработки данных, поступающих с радиотелескопов. И, в связи с этим, разработчики решили объединить несколько проектов и сообществ, для решения масштабной задачи.

На данный момент именно платформа BOINC является самой популярной для разворачивания своих проектов и привлечения участников для решения поставленной задачи. В связи с этим возникает вопрос: «Как развернуть свой проект на платформе BOINC?» В данной курсовой работе рассматривается не только ответ на данный вопрос, но решение более узкой задачи, заключающейся в том, как настроить проект, а именно, нейронную сеть, таким образом, чтобы она обучалась асинхронно. И, наконец, будет проведено обучение нейронной сети на эталонном и реальном датасетах с последующим сравнением и анализом полученных результатов.

1. Аналитический обзор литературы

1.1. Распределённое обучение сети

1.1.1. Подходы к распределенной реализации SGD

С развитием глубокого обучения и увеличения тренировочных данных перед разработчиками моделей нейронных сетей встала проблема обучаемости нейронных сетей. Обучение таких сетей требовало огромных ресурсов хранения данных, а также занимало большое количество времени.

Для того, чтобы преодолеть это «узкое место» глубоких нейронных сетей и добиться значительного снижения времени обучения, сам процесс обучения должен быть распределён между множеством процессоров или графических ускорителей с целью строгого масштабирования, то есть с увеличением количества вычислительных узлов время обучения сети в теории должно уменьшаться пропорционально.

Существуют два основных подхода к распределенному стохастическому градиентному спуску (SGD) для обучения глубоких нейронных сетей: синхронный all-reduce SGD, который основывается на быстром коллективном общении узлов системы [1][2], и асинхронный SGD, который использует параметрический сервер [3][4].

Оба метода имеют свои недостатки при масштабировании. Синхронный SGD теряет производительность при замедлении работы хотя бы одного узла, использует вычислительные ресурсы не в полном объёме и не толерантен к выходу из строя вычислительных процессоров или целых узлов. В свою очередь асинхронный подход использует параметрические сервера, создавая тем самым коммуникационную проблему «узкого горлышка» и неиспользуемых сетевых ресурсов, замедляя тем самым сходимость градиента.

Так как для нашей задачи необходимо решить задачу обучения глубокой нейронной сети на грид-системе, то рассмотрим более детально асинхронные методы распределения обучения сети, так как в грид-системах используются узлы различной вычислительной мощности, и к тому же получение результата со всех узлов системы не гарантировано.

1.1.2. Параллелизм данных и модели

При распределении обучения модели глубокой нейронной сети может быть применено два подхода: разделение обучающего датасета на разные части для различных

устройств (параллелизм данных) или разделение самой модели между различными устройствами (параллелизм модели).

При параллелизме данных всё исходное обучающее множество делится на N частей, которые отправляются на N узлов распределенной системы (рисунок 1). Каждый из этих узлов содержит копию модели нейронной сети и параметры этой модели Wn. Процесс обучения проходит следующим образом:

- Каждый узел считывает небольшую выборку из тренировочных данных и вычисляет на её основе локальные градиенты ∇W_n.
- Каждый узел посылает вычисленные значения локальных градиентов «мастер» - узлу. После получения градиентов со всех узлов «мастер» - узел агрегирует все градиенты и обновляет модель новым вычисленным значением.
- «Мастер» - узел распространяет модель на все другие узлы.
- Операция повторяется до завершения обучения модели.

Кроме параллелизма данных также существует параллелизм самой модели нейронной сети, при котором модель распределяется между разными узлами. Данный подход применяется в том случае, если модель слишком велика для расположения на одном узле.

Схемы параллелизма распределенного обучения представлены на рисунке 1.

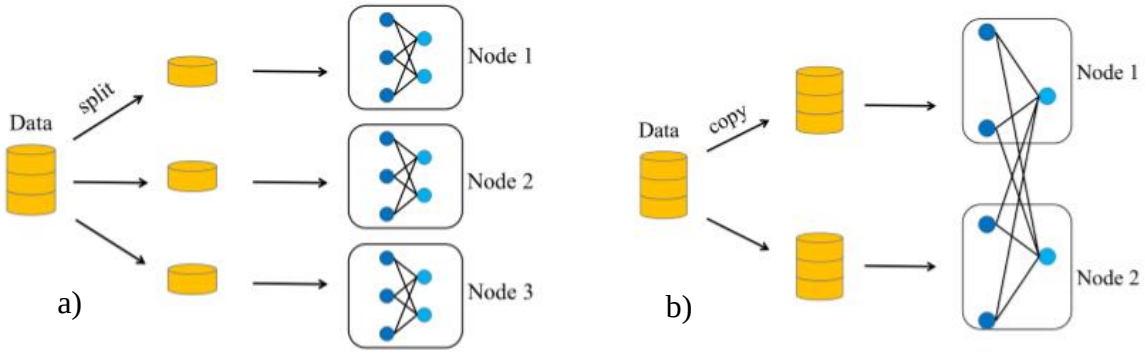

Рисунок 1 – Схемы параллелизма нейронных сетей (a – параллелизм данных, b – параллелизм модели)

1.1.3. Централизованные и децентрализованные архитектуры

Логическое построение архитектуры распределенной системы влияет на режимы коммуникации узлов и производительность сети. Использование параметрического сервера [5][6] является наиболее популярной централизованной архитектурой в распределенном обучении глубоких сетей. Такая архитектура обычно включает в себя серверный узел и «рабочие» узлы. На сервере находятся глобальные параметры модели, а на «рабочих» узлах находятся локальные копии глобальной модели. Если серверный узел представлен несколькими вычислительными машинами, то каждая такая машина содержит свою часть всех параметров модели.

Что касается «рабочих» узлов, то каждый из них содержит полную копию глобальной модели при параллелизме данных или часть модели при параллелизме модели. «Рабочие» узлы общаются с сервером посредством push/pull операций, тогда как между «рабочими» узлами нет никакой связи вообще.

На этапе операции push рабочие узлы сообщают параметрическому серверу градиенты ΔW, вычисленные локально. Далее, в зависимости от алгоритма обновления глобальной модели, параметрический сервер «с ходу» обновляет глобальную модель полученным значением градиента по формуле 1.

$$W' = W - \alpha * \Delta W, \quad (1)$$

где W' – обновленная модель;

W – значение параметров глобальной модели до обновления;

a – коэффициент обновления;

ΔW – градиенты, вычисленные локально.

Такая формула подходит для асинхронного обновления, которое будет рассмотрено далее. Затем на операции pull «рабочий» узел получает уже обновленную модель.

Для синхронного обновления характерно то, что операции pull и push совершаются одновременно всеми «рабочими» узлами. В связи с этим необходимо первоначально агрегировать результаты вычислений всех узлов и только затем обновить модель. В общем виде это представлено в формуле 2.

$$W' = W - \alpha * \sum \Delta W, \quad (2)$$

где W' – обновленная модель;

W – значение параметров глобальной модели до обновления;

a – коэффициент обновления;

$\sum \Delta W$ – агрегированные градиенты, вычисленные локально.

На рисунке 2 показана традиционная архитектура параметрического сервера, в которой сервер отвечает за хранение и обновление параметров глобальной модели. Она подвержена узкому месту в виде полосы пропускания сети со стороны сервера, особенно при наличии большого количества «рабочих» узлов. Древовидные параметрические серверы [7][8][9] в некоторой степени смягчают такие узкие места.

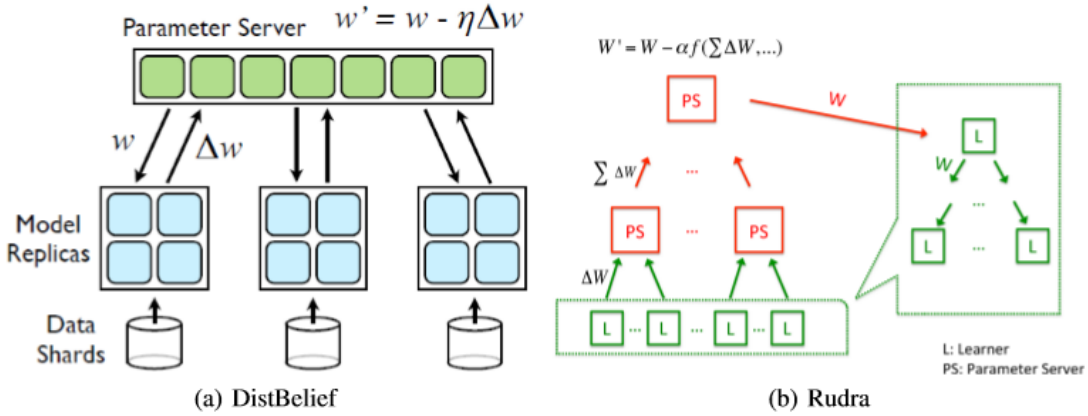

Рисунок 2 – Архитектуры параметрических серверов. (a – традиционная архитектура, b – древовидая архитектура)

Децентрализованная архитектура лишена этого недостатка. Также, как и в архитектуре параметрического сервера с параллелизмом данных, каждый узел содержит копию полной модели сети, но отличие заключается в том, что общение происходит между всеми узлами с помощью операции all-reduce, вместо push/pull. В ходе этого общения происходит обмен градиентами и различными параметрами. Тем не менее, операция all-reduce имеет множество реализаций с существенно отличной производительностью, что означает, что она может влиять на накладные расходы на коммуникацию.

1.1.4. Обновление глобальной модели

Исходя из различий в ширине каналов связи и вычислительных мощностей, некоторые узлы могут вычислять градиенты быстрее, чем другие. Основная проблема данного обстоятельства заключается в вопросе синхронизации градиентов на различных вычислительных узлах. Существует три различных метода обновления градиентов: синхронное, асинхронное и обновление с задержкой.

1.1.4.1. Синхронное обновление

При синхронном обновлении сервер не обновляет глобальную модель сети до тех пор, пока не получит градиенты от всех «рабочих» узлов на каждой итерации. Из этого вытекает описанная выше проблема – быстрые узлы ждут медленных. Одной из известных реализаций синхронного обновления является bulk synchronous parallel (BSP) [10]. Характерной особенностью синхронного режима является то, что сервер всегда будет получать последние градиенты всех узлов, которые не влияют на сходимость модели. Однако, быстрые узлы ничего не выполняют при ожидании медленных узлов, что приводит к пустой трате ресурсов. Кроме того, это также замедлит общее время обучения.

1.1.4.2. Асинхронное обновление

Асинхронные алгоритмы, такие как Hogwild [11], преодолевают вышеперечисленные проблемы. При асинхронных обновлениях быстрые «рабочие» узлы не ждут медленных. Один «рабочий» узел может посылать свои локальные градиенты на сервер, в то время как другие вычисляют свои градиенты, как показано на рисунке 3.

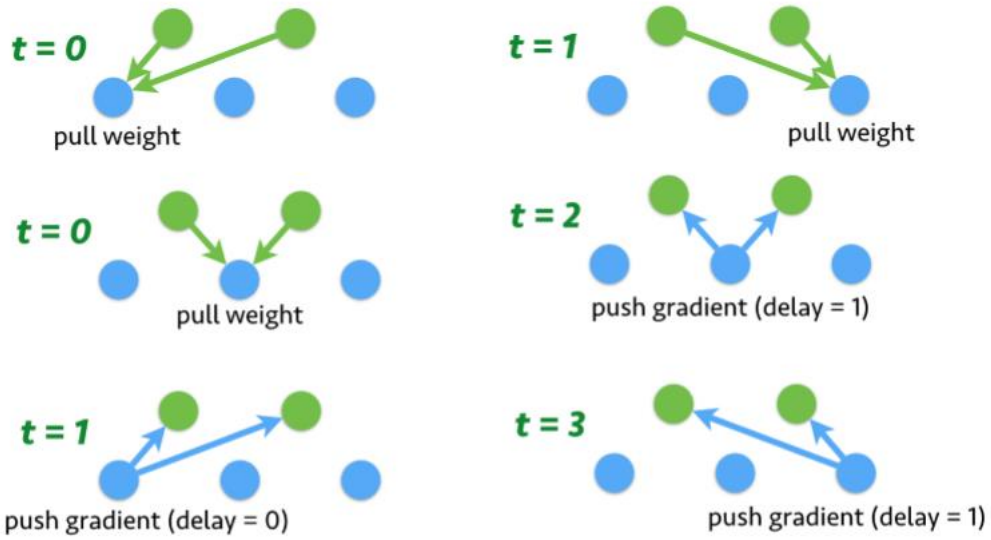

Рисунок 3 – Асинхронное обновление параметров модели. (зеленые – серверы, синие – «рабочие» узлы)

Основная проблема, возникающая при асинхронных обновлениях – это неактуальность данных, потому что быстрые «рабочие» узлы всегда могут использовать устаревшие параметры, которые ставят под угрозу сходимость модели. Кроме того, самый

быстрый «рабочий» узел обновляет свои локальные параметры только по своему обучающему подмножеству, что заставляет локальную модель отклоняться от глобальной модели.

Для преодоления недостатков асинхронных обновлений исследователи попытались ограничить неактуальность параметров. При таком обновлении быстрые «рабочие» узлы будут использовать неактуальные параметры, но эта неактуальность (как и задержка на рисунке 3) будет ограничена [12][13]. Такое ограничение в некоторой степени смягчает проблему неактуальности и увеличивает скорость обучение модели.

Однако, выбор ограничения неактуальности тоже является вопросом, ведь если оно будет слишком большим, то это означает полную асинхронность, а если слишком маленьким, то это соответствует синхронному обновлению модели.

1.1.4.3. Периодичность обновления

При обучении ГНС с использованием традиционного распределенного SGD, каждый работник на каждой итерации выполняет один шаг SGD и вычисляет локальные градиенты, затем обменивается градиентами и параметрами с другими узлами. Связь между узлами происходит в конце каждой итерации.

Предположим, что мы распараллеливаем задачу обучения ГНС на K узлах с помощью T итераций, сложность раундов идеального параллельного SGD равна O(T). В связи с такой высокой сложностью раундов связи, многие работы в распределении обучения моделей предлагают уменьшить частоту обмена градиентами и параметрами между узлами.

При усреднении модели отдельные параметры модели, обученные на локальных узлах, усредняются периодически. Она происходит в большинстве τ итераций, где τ является фактором периода. Усреднение происходит в каждой итерации, т. е. $\tau = 1$ будет идентична идеальному параллельному SGD, а если усреднение происходит только в конце тренировки, т. е. $\tau = T$, то это равнозначно усреднению в конце обучения («one-shot averaging»). Также есть другой вариант усреднения, при $1 < \tau < T$. На рисунке 4 изображены все 3 варианта, упоминавшихся ранее.

Рисунок 4 – Сравнение периодичности общения в разных режимах (зеленый – вычислительные операции, желтый – операции общения)

Экспериментальные работы [14][15][16][17] подтвердили, что усреднение модели может снизить накладные расходы на связь во время обучения до тех пор, пока этот период является приемлемым. Кроме того, некоторые теоретические исследования [18][19][20][21] дали анализ того, почему усреднение модели может обеспечить хорошую скорость сходимости.

Однократное усреднение («one-shot averaging»), которое требует коммуникации только в конце обучения, является экстремальным случаем усреднения модели. Однократное усреднение имеет достаточную производительность как по выпуклым [14], так и по некоторым невыпуклым [15] задачам оптимизации. Однако согласно работе Чжана [16] некоторые задачи невыпуклой оптимизации не могут быть решены с помощью однократного усреднения. Решением данной проблемы является более частое усреднение.

1.2. Архитектура BOINC

Рассмотрим архитектуру платформы BOINC (рис. 5).

Архитектура системы BOINC клиент-сервер, состоит из программы-клиента и составного сервера BOINC, что подразумевает возможность использования нескольких компьютеров в качестве сервера.

В качестве бонуса система «вознаграждает» пользователя за выполнение определенных подзадач «зарабатывая кредиты». Кредиты не представляют никакой финансовой ценности, а нужны лишь для поощрения и создания соревновательного духа,

так как подсчитываются рейтинги по наибольшему количеству кредитов и выставляются на главной странице проекта.

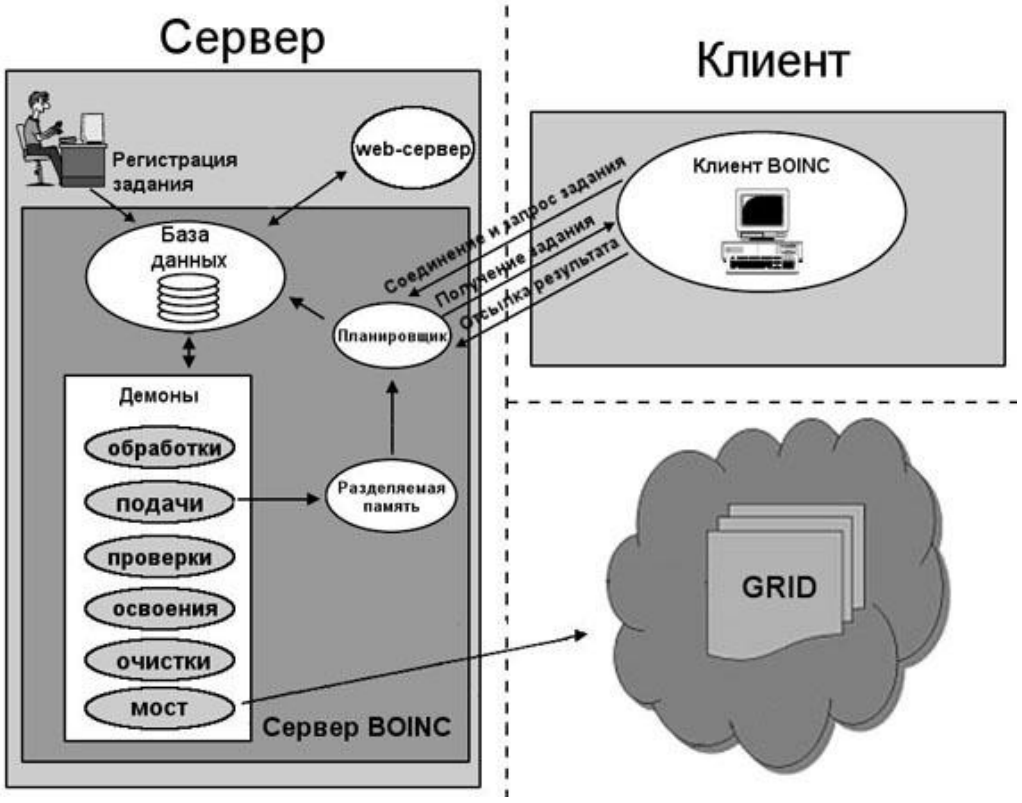

Рис. 5. Архитектура системы BOINC

Составной сервер, раздает задания и собирает результаты, в то время как множество подключаемых к серверу программ-клиентов выполняют основную вычислительную работу по получению результата.

Все программы-клиенты подключаются к серверу и получают задание на обработку. После выполнения задания ответ высылается на обработку сервером, после чего получает новое задание. За выполнение задачи зарегистрированным участникам проекта насчитываются кредиты, характеризующие количество процессорных вычислений, потраченных на решение задания.

В качестве основы архитектуры BOINC использована идея конечного автомата – сервер состоит из набора отдельных подсистем; каждая из подсистем отвечает за свою определенную задачу, к примеру, арифметику, передачу данных и т.д. Каждая из подсистем работая в бесконечном цикле проверяя состояние подзадачи, производя какие-то действия и изменяя состояние подзадачи.

В основном, система состоит из сервера BOINC (есть возможность распределения на несколько физических серверов),

групп клиентов, вычисляющих задачи сервера и нескольких дополнительных компонентов, представляющих собой присоединенные GRID-сети, к примеру, инструментарий Globus Toolkit.

1.3. Globus Toolkit

Это набор инструментов для создания вычислительных Grid-сетей предоставленный в общем доступе (Open Source). В его составе набор сервисных программ и библиотек, которые, ведут учет ресурсов, обнаруживают и управляют вычислительными узлами, обеспечивают безопасность и управление файлами. Разработкой и поддержкой занимается организация Globus Alliance.

Сервер BOINC состоит из следующих частей:

- Web-сервер (один или несколько), обрабатывающий входные и выходные сообщения;
- сервер баз данных, хранящий данные клиентов о набранных ими баллах, а также отслеживает состояние всех выполненных задач и результаты к ним;
- пять различных демонов, проверяющие с заданным интервалом состояние базы данных и выполняющие необходимые работы по распределению подзадач и обслуживанию системы.

Далее более подробно рассмотрим компоненты, составляющие сервер BOINC.

1.4. Компоненты BOINC сервера

1.4.1. Web-сервер

Web-сервер является второстепенным звеном структуры сервера BOINC. Причиной наличия сервера является сама сущность «volunteer computing» – необходимость привлечения внимания участников к проектам. С этой целью создан сайт, рассказывающий насколько важную задачу для всего человечества, решаете вы. Значит и те кто присоединился к вам, добровольцы – выполняют: ищут внеземные цивилизации (SETI@HOME), разрабатывают новые лекарственные препараты (Docking@Home), предсказывают погоду (ClimatePrediction.net) или же решение сложных математических задач... Рассказав о своем проекте, вы сможете привлечь добровольцев, потенциальные вычислительные мощности. Все проекты по распределенным вычислениям на базе BOINC

дают своим участникам возможность объединения в команды, и следить за изменениями набранных ими баллов...

В этом случае, Web-сервер может быть перенесен на другой физический сервер. Для актуальности данных статистики, необходима связь с базой данных сервера BOINC.

1.4.2. База данных

Демон

Демоном называют компьютерную программу, работающую в фоновом режиме не нуждающуюся во взаимодействиях с пользователями, термин используется в UNIX-подобных системах. Чаще всего, в виде демонов реализуются серверные программы, к примеру, sftp-сервер, почтовый-сервер и т. д.

База данных – это основная часть всего проекта BOINC,

В базе данных расположено:

- все данные, которые относятся к BOINC серверу;
- данные и версии приложений;
- данные приложений-клиентов BOINC включая их версии;
- сведения зарегистрированных участниках и связанных с ними хостами;
- данные подзадач с результатами их вычислений.

С информацией взаимодействуют специальные служебные демоны. Система BOINC первоначально разработана для работы с СУБД MySQL, развитие этой идеи привело к нынешней системе.

Необходимо учесть, что вся нагрузка, связанна с активной передачей данных внутри проекта BOINC, вся она приходится на базу данных, и как правило, является тем самым «узким местом» производительности сервера.

1.4.3. Служба обработки состояния подзадач (Transitioner)

Эта служба является обработчиком статуса вычисляемых подзадач и результатов их решения. Служба не зависима от приложений и едина для всех проектов, таких как поиск решения математической задачи или предсказание погоды. Задачей службы обработки является проверка текущего статуса подзадачи в базе данных и обновление соответствующих полей, в момент готовности подзадачи, переход в новый статус. Основной сложностью является то, что подзадачи имеют множество различных статусов.

Эти статусы содержат в себе состоянии результатов вычислений. К примеру, результаты готовые к проверке, и данных хватает для осуществления проверки кворумом, то статус подзадачи изменяется на “готова к проверке”. Служба обработки дает высокую нагрузку на процессоры, в связи с этим ее можно разделить на несколько демонов, все отвечают за определенную подзадачу или несколько подзадач. Исходя из этого, демоны могут функционировать не только на одном физическом сервере, их можно разделить на несколько.

1.4.4. Служба проверки результатов (Validator)

Грид является набором вычислительных узлов, соединенных для вычисления общей ресурсоемкой задачи. Вычисления на базе grid используются для проведения астрономических исследований, для создания новых материалов или жидкостей и т. д.

Создание Grid – задача нетривиальная, в частности, требующая решения проблем по взаимодействию, управлению и обнаружению вычислительных узлов.

Задачей службы является организация проверки входящих результатов. В целях обеспечения верности решения каждая из подзадач решается на нескольких различных машинах-клиентах. Получив результат, его нужно проверить, сверив между собой полученные от других машин-клиентов результаты и определив «конечное» решение – результат, получается кворумом клиентов. Для каждого типа задачи необходим свой алгоритм проверки. Реализацией алгоритма проверки является служба. Помимо того, программа проверки следит за правдоподобностью результатов. К примеру, при моделировании физических процессов, есть возможность проверки, не является ли конечный результат ниже или выше предельного возможного значения. Если результат выходит за рамки, то он отбрасывается, как заведомо ложный. Таким образом, можно отследить ошибки в результатах.

При включении служба посылает запрос в базу данных на принятие информации о требующих проверки новых результатах. Далее при нахождении, служба проверки запускает функцию для сравнения полученных результатов. Для всех глобальных задач, решаемых системой BOINC, необходимо создать две функции находящихся в службе проверки: первая функция сравнивает два результата, так же используется для начисления очков, когда приложение клиент передал новый результат и найдено верное решение. Вторая, наборы результатов, используется для определения наиболее верного результата из множества результатов, которые передали несколько клиентов. Число результатов, необходимое для принятия эталонного решения, определяется в начале создания подзадачи.

Это значение можно задать для всего приложения в целом, а также возможно указать различные значения для разных клиентов.

1.4.5. Служба освоения (Assimilator)

Задачей службы освоения является периодическая проверка наличия решенных задач. Разработчику проекта необходимо сделать функцию, определяющую необходимые действия с эталонными результатами. К примеру, ответы можно заархивировать и отослать по электронной почте или автоматически запустить дальнейшую обработку данных, выделив интересующие фрагменты и записывая их в документы. Подзадачу пометят как завершенную в случае обработки службой освоения.

1.4.6. Служба удаления файлов (File deleter)

Служба удаления файлов – это «чистильщик мусора» проекта BOINC, она, проверяя статусы задач, ищет завершенные и освоенные подзадачи, после чего удаляет с сервера связанные с ними входные и выходные данные. Выходные файлы, содержащие эталонный результат, обрабатываются на фазе освоения. Есть возможность удалять только файлы, оставляя, при этом, записи в базах данных, в этом случае всегда будет возможность посмотреть по базе данных и найти необходимую информацию о подзадаче, участниках и т.д. включая случай решения подзадач (и удаления файла).

1.4.7. Служба подачи (Feeder)

Задачей службы является загрузка еще не решенных данных на сегмент разделяемой памяти, для которых еще не получен эталонный результат и не занесен в базу данных. Такую предварительную работу сервер выполняет с целью повышения производительности системы BOINC в целом путем ограничения количества запросов в базу данных.

1.4.8. Планировщик (Scheduler)

Планировщик – это CGI-программа, запускаемая в момент, когда к серверу проекта подключается клиент и запрашивает часть задания и входных данных. Вместо прямого взаимодействия с базой-данных, планировщик получает задания и входные данные из сегмента разделяемой памяти, в который данные загружаются службой подачи.

Планировщик может возможность самостоятельно назначить подзадачи клиентам, так как не все машины-клиенты одинаковы, как в настройках, так и аппаратных конфигурациях. К примеру, один клиент может использовать Mac-версию клиента и выделить 126 МБ дискового пространства и 300 МБ оперативной памяти, а другой клиент может запустить только Android-версию и выделить использование не более 16 МБ дискового пространства и 16 МБ оперативной памяти. В этом планировщик решает дать более вычислительно-емкое задание Mac -клиенту, а наиболее простые подзадачи Android -клиенту.

В момент сессии, работая с планировщиком, клиент также отсылает отчет завершении работ, которые уже загрузили на сервер с момента последней сессии планировки. В итоге клиенту передается список с заданиями на обработку и списком адресов, откуда получает нужные файлы, т. е. входящие файлы и файлы приложения, если они отсутствуют на компьютере-клиенте.

1.4.9. Мост (Bridge)

Задача этой службы, обеспечивать связь и работу над проектом в инфраструктуре BOINC и GRID, к примеру, на базе технологии Globus Toolkit.

Приложения BOINC вызывают функции BOINC через систему интерфейсов, реализованных в клиенте и выполняющих такие специфические работы как, к примеру, передача файлов. Исходя из этого, запуск подзадачи проекта BOINC не может быть напрямую (необходимы дополнительные модификации) расчет в инфраструктуре Grid. Помимо этого, Grid не способна, как клиент BOINC, на прямое соединение с планировщиком проекта и запрос подзадач для расчета. Для решения подобных проблем, во взаимодействии разных архитектур распределенных вычислений требуются реализации дополнительных механизмов, делающих возможным соединение BOINC-Grid. Для этого создан программный мост, при этом реализация моста зависит от особенности подключаемого Grid и проектов, в рамках которого проводятся вычисления.

1.4.10. Приложения BOINC

Для передачи задания клиенту должно быть разработано и запущено как минимум одно приложение BOINC. После создания проекта, исполняемый файл регистрируется на сервере BOINC, и администратор может начать создавать подзадачи для этого приложения. Подзадачами изначально являются не более чем описанием файлов входа. В том случае,

если проект необходимо запустить для вычислений на разных платформах, то надо реализовать и зарегистрировать версии для каждых из платформ.

Преобразование программы в приложение BOINC происходит с помощью добавления библиотек и функций BOINC API, реализованные на языке Си.

Необходимо чтобы, все приложения BOINC вызвали специальные функции в начале и конце программы: функция инициализации, функция завершения. В тексте программы BOINC также должны присутствовать функциями, передающие программе-клиенту о стадиях выполнения подзадания (в процентах), с целью информирования пользователя о прогрессе вычисления. Так же возможно для того, чтобы привлечь внимание к проекту, можно внести графическую составляющую (картинки, анимация, видео), приложение BOINC будет демонстрировать, поэтому необходимо вызывать функции рисования, если клиентское приложение запросит отображения графики.

Функции BOINC API перед открытием преобразуют все имена файлов что проходят через приложение. Есть необходимость в этом для тех приложений, которые запускают большое количество различных входных файлов и сгенерируют большое количество различных выходных файлов. Возможен случай, когда при запуске имена файлов не меняются, тогда при новом запуске необходимо создавать отдельные каталоги на сервере и на машине-клиенте для предотвращения конфликтных ситуаций. Приложение после нескольких запусков не меняется, нет возможности в смене имен внутри приложения и не логично каждый раз перекомпилировать все приложение. Для решения этих проблем, приложение использует логические имена файлов, эти имена переводятся приложением-клиентом в физические имена при запуске. Имена файлов физические определяются только при создании подзадачи.

Подсчет очков это большое преимущество проекта, но для его выполнения приложению надо вызвать специальные функции BOINC API, одна из которых говорит приложению о необходимости произведения подсчета очков. Важность этого в том, что пользователю необходимо настроить программу-клиент как использовать жесткий диск только в определенные интервалы времени для предотвращения частого раскручивания диска, которое приводит к быстрому износу.

Если подсчет очков клиента разрешен клиентом, то приложение автоматически подсчитает очки, и передаст в другую функцию BOINC API информацию о завершении подсчета.

При этом нужно учитывать, что клиентская программа учитывает затраченное время CPU при вычислении количества кредитов. При перезапуске подзадачи (к примеру, когда

компьютер перезагрузили), то подсчет времени CPU начнется с момента последнего подсчета очков.

1.5. Жизненный цикл задания из проекта.

Жизненный цикл заданий выглядят для всех проектов системы BOINC примерно одинаково (см. рисунок 6):

1. Генератор заданий (разрабатывается для каждого проекта отдельно) создает и дублирует входные данные;
2. Планировщик BOINC распределяет данные между клиентскими программами и отсылает их;
3. Клиентская программа делает вычисления и передает выходные данные на сервер;
4. Служба проверки результатов (разрабатывается для каждого проекта) занимается проверкой выходных данных, полученных от клиентов, сравнивает выходные данные разных клиентских программ с одинаковыми входными данными;
5. Служба освоения (разрабатывается для каждого проекта отдельно) занимается обработкой результатов;
6. Служба удаления занимается очищением базы данных от ненужных файлов и информации после завершения работы службы освоения.

Необходимо обратить внимание на то, есть службы стандартные и не зависимые от конкретного проекта и его реализации. Несмотря на это множество служб необходимо разработать отдельно для каждого проекта – в этом и заключается дополнительная сложность, которую необходимо преодолеть для возможности проведения распределенных вычислений.

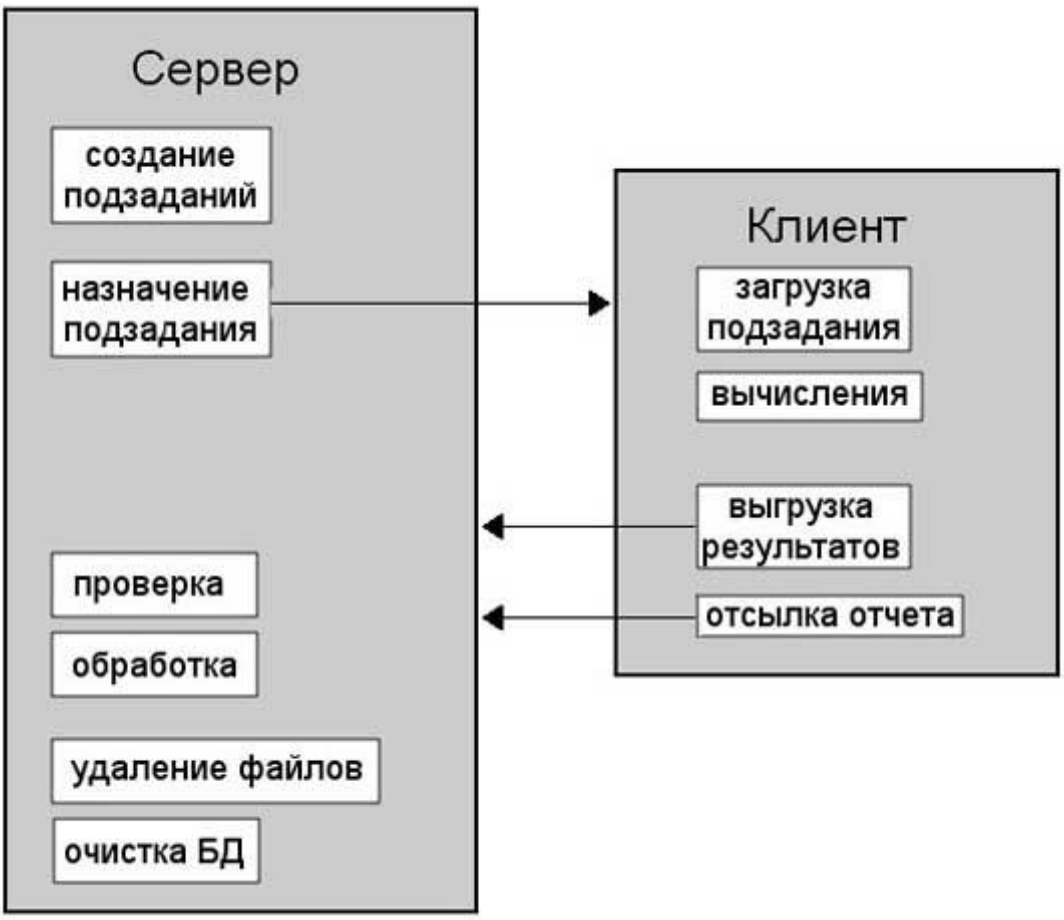

Рисунок 6. Жизненный цикл задания

Список использованных источников

1. Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul, and Pradeep Dubey. Distributed Deep Learning Using Synchronous Stochastic Gradient Descent.– [Электронный ресурс]: arXiv preprint arXiv:1602.06709.– 2016.– URL: https://arxiv.org/pdf/1602.06709.pdf (дата обращения: 17.04.2019).
2. Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting Distributed Synchronous SGD.– [Электронный ресурс]: arXiv preprint arXiv:1604.00981.– 2016.– URL: https://arxiv.org/pdf/1604.00981.pdf (дата обращения: 17.04.2019).
3. Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large Scale Distributed Deep Networks. // In Advances in Neural Information Processing Systems 25.– 2012.– С. 1223–1231.
4. Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project Adam: Building an Efficient and Scalable Deep Learning Training System. // In 11th USENIX Symposium on Operating Systems Design and Implementation.– 2014.– С. 571–582.
5. Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. // In 11th Symposium on Operating Systems Design and Implementation.– 2014.– С. 583–598.
6. Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. // In Advances in Neural Information Processing Systems.– 2014.– С. 19–27.
7. Luo Mai, Chuntao Hong, and Paolo Costa. Optimizing network performance in distributed machine learning. // In 7th Workshop on Hot Topics in Cloud Computing (HotCloud 15).– 2015.
8. Suyog Gupta, Wei Zhang, and Fei Wang. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. // In 2016 IEEE 16th International Conference on Data Mining (ICDM) .–IEEE.– 2016.– С. 171–180.
9. Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching LAN speeds. // In 14th Symposium on Networked Systems Design and Implementation.– 2017.– С. 629–647.

10. Leslie G Valiant. A bridging model for parallel computation. // Communications of the ACM.– 33(8).– 1990.– С. 103–111.
11. BenjaminRecht, ChristopherRe, StephenWright, and FengNiu. Hogwild: A lock-freeapproach toparallelizing stochastic gradient descent. // In Advances in neural information processing systems.– 2011.– С. 693–701.
12. James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R Ganger, Garth Gibson, Kimberly Keeton, and Eric Xing. Solving the straggler problem with bounded staleness. // In Presented as part of the 14th Workshop on Hot Topics in Operating Systems.– 2013.
13. Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. // In Advances in neural information processing systems.– 2013.– С. 1223–1231.
14. Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. // In Advances in neural information processing systems.– 2010.– С. 2595–2603.
15. Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. // In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics.– Association for Computational Linguistics.– 2010.– С. 456–464.
16. Jian Zhang, Christopher De Sa, Ioannis Mitliagkas, and Christopher Ré. Parallel sgd: When does averaging help?.– [Электронный ресурс]: arXiv preprint arXiv:1606.07365.– 2016.– URL: https://arxiv.org/pdf/1606.07365.pdf (дата обращения: 20.04.2019).
17. Hang Su and Haoyu Chen. Experiments on parallel training of deep neural network using model averaging.– [Электронный ресурс]: arXiv preprint arXiv:1507.01239.– 2015.– URL: https://arxiv.org/pdf/1507.01239.pdf (дата обращения: 20.04.2019).
18. Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. // In Proceedings of the AAAI Conference on Artificial Intelligence.– volume 33.– 2019.– С. 5693–5700.
19. Sebastian Urban Stich. Local sgd converges fast and communicates little. // In ICLR 2019 International Conference on Learning Representations.– 2019.
20. Yossi Arjevani and Ohad Shamir. Communication complexity of distributed convex learning and optimization. // In Advances in neural information processing systems.– 2015.– С. 1756–1764.

21. Fan Zhou and Guojing Cong. On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization. // In Proceedings of the 27th International Joint Conference on Artificial Intelligence.– AAAI Press.– 2018.– C. 3219–3227.