Содержание

Аннотация

Добровольные вычисления позволяют любому желающему предоставить свои вычислительные ресурсы для выполнения задач, важных для науки. С помощью них научные проекты с небольшим финансированием могут получить доступ к вычислительным мощностям, сравнимым с современными суперкомпьютерами. BOINC – основная программная платформа для добровольных вычислений, получившая популярность среди проектов в различных областях: математика, медицина, молекулярная биология, астрофизика и другие.

Эффективное использование ресурсов, определение стратегии распределения заданий и параметров проекта в среде добровольных вычислений осложняется ее особенностями, среди которых высокая гетерогенность, непредсказуемость и ненадежность среды, возможность использовать только часть ресурсов, зачастую в фоновом режиме. При этом тестирование различных гипотез по улучшению производительности добровольных вычислений внутри реальных проектов практически невозможно: эксперименты с изменением параметров могут занимать длительное время и при этом чаще всего не воспроизводимы, а также могут мешать выполнению задач проектов. По этой причине наличие симулятора добровольных вычислений, учитывающих специфику добровольной среды и архитектуру BOINC, очень важно для их развития.

Целью этой работы является выбор инструмента для моделирования добровольных вычислений путем анализа, проведение экспериментов для изучения специфики добровольных вычислений, доработка выбранного инструмента для более точной симуляции.

Ключевые слова

Добровольные вычисления, BOINC, симуляция

1 Введение

Распределенная система – это набор независимых компьютеров, которые выполняют общую задачу и синхронизируются путем передачи сообщений по сети.

Распределенные системы обладают несколькими преимуществами по сравнению с использованием одного компьютера. А именно, с их помощью можно добиться масштабируемости и производительности: появляется возможность обрабатывать большие объемы данных и повышать производительность, как с помощью добавления дополнительных узлов, так и путем параллельного выполнения задач на разных компьютерах. Также распределенные системы обеспечивают отказоустойчивость за счет реплицирования данных на разных вычислительных узлах. Благодаря такой избыточности распределенная система устойчива к отказам ее частей.

Для распределенных систем характерны следующие особенности, которые зачастую усложняют их проектирование:

1 *Отсутствие единого времени.* Нет гарантии на совпадение времени на различных узлах распределенной системы, что усложняет синхронизацию.

2 *Отсутствие общей памяти.* Распределенная система состоит из нескольких независимых функционирующих узлов, из-за чего взаимодействие и синхронизация между ними должны происходить с помощью сети.

3 *Большие накладные расходы для коммуникации.* Как упомянуто выше, синхронизация в распределенной системе происходит по сети, что увеличивает время доставки информации между компонентами по сравнению с локальной синхронизацией.

4 *Гетерогенность.* Вычислительные узлы, входящие в распределенную систему, могут иметь различные характеристики(объем оперативной памяти, скорость процессора, установленная операционная система и т.д.), из-за чего время выполнения одной и той же задачи на них могут различаться.

5 *Географическая распределенность.* Узлы распределенной системы физически могут находиться в удаленных друг от друга точках, из-за чего может ухудшаться надежность сети и увеличиваться задержки в ней, что усложняет синхронизацию в распределенной системе.

6 *Возможность отказов.* Распределенные системы подвержены отказам различных компонент. При этом отказывать могут как вычислительные узлы, так и сеть между ними.

1.1 Добровольные вычисления

Добровольные вычисления(Volunteer computing, VC) – это тип распределенных вычислений, в которых используются вычислительные ресурсы, принадлежащие добровольцам и соединенные по сети Интернет.

Добровольцем может стать любой человек, подключив свое устройство(компьютер, ноутбук, телефон) к VC проекту. При этом одно и то же устройство может быть одновременно подключено сразу к нескольким проектам, и хозяин этих ресурсов может задавать их распределение между проектами.

Традиционно, VC проекты создаются для решения вычислительно трудоемких задач науки. Одновременно к VC проекту могут быть подключены тысячи устройств, предоставляя огромную вычислительную мощность. Это позволяет развивать общественно значимые исследовательские проекты, даже если у них небольшое финансирование.

У систем добровольных вычислений есть несколько особенностей, которые отличают их от обычных распределенных систем:

1 Как правило, для VC систем характерна *высокая гетерогенность*. К проекту подключаются абсолютно разные устройства, которые могут очень сильно различаться по скорости процессора, объему оперативной памяти, количеству пространства на диске и скорости сети.

2 *Непредсказуемая среда.* Добровольцы не связаны какими-либо обязательствами, поэтому могут отключать и подключать свои ресурсы в любой момент, даже во время исполнения задачи. Так, например, один участник может подключаться на несколько часов каждый месяц, а у другого ресурсы подключены к системе постоянно.

3 *Возможность использования только части ресурсов.* В большинстве случаев проекты могут использовать предоставленные ресурсы только в фоновом режиме, не препятствуя исполнению основных программ. То, как VC проекты могут использовать ресурсы участников, обычно задается ими в параметрах(например, ограничение использования CPU и диска или возможность запуска вычислений только при отсутствии активности мышки, использование только ночью и так далее).

4 *Ненадежность результатов.* Среда добровольных вычислений не является доверенной, поэтому получение результатов выполнения задачи еще не означает, что эти результаты корректны. На надежность результатов могут влиять аппаратные сбои, а так-

же возможное наличие злоумышленников, цель которых – как-либо намеренно навредить системе, поэтому необходимо валидировать результаты.

Таким образом, системы добровольных вычислений предоставляют огромные вычислительные мощности и при этом не требуют высокого финансирования, как например, аренда суперкомпьютера. Однако таким системам свойственны высокая гетерогенность, непредсказуемость и возможное наличие злоумышленников, которые усложняют проектирование таких систем.

2 Обзор инструментов для моделирования

SimGrid [10] – фреймворк для моделирования различных распределенных систем (распределенные вычислительные системы, грид-вычисления, одноранговые системы и так далее). В модели SimGrid и сеть, и вычислительные устройства рассматриваются как ресурсы, на которых могут исполняться задания. У каждого ресурса есть некоторые характеристики производительности, которые могут быть заданы константой или с помощью traces(набор пар [время, значение характеристики]), аналогично у каждого задания есть стоимость. Такая модель является достаточно гибкой и позволяет моделировать большое количество распределенных систем. Также этот фреймворк позволяет полноценно моделировать сеть, включая задержки, пропускную способность, фоновый трафик. При этом SimGrid хорошо масштабируем и демонстрирует высокую точность симуляции. SimGrid предоставляет API на языке C, с помощью которого возможно моделирование более специализированных систем с произвольной топологией сети, динамичными ресурсами и возможными отказами ресурсов.

GridSim [7] – инструмент для симуляции грид-вычислений. В GridSim ресурсами выступают выступают вычислительные устройства, у каждого из которых есть стоимость за определенное количество выполненных операций. Клиент, которому необходимо выполнить задачу, подключается к Resource Broker, который ответственен за нахождение ресурсов, распределение задач по ресурсам согласно политике, указанной клиентом, отправку данных для задачи и сбор результатов. Такая архитектура позволяет учитывать интересы каждого пользователя отдельно, однако усложняет модель и не является необходимой для симуляции BOINC, в котором есть один централизованный клиент. GridSim позволяет симулировать гетерогенные многопроцессорные ресурсы, которые могут являться как time-shared, так и space-shared. Однако GridSim не позволяет в полной мере моделировать непредсказуемость среды добровольных вычислений, так как параметры ресурсов и сети задаются константа-

ми. Также GridSim значительно медленнее SimGrid и плохо масштабируется из-за накладные расходов на синхронизацию, так как в GridSim для каждой сущности создается отдельный поток [9].

Одним из примеров дискретно-событийных симуляторов BOINC является SimBA [13]. SimBA моделирует генерацию, распределение, сбор результатов и валидацию задач, которые выполняются в нестабильной гетерогенной среде. Использование traces, полученных из реальных проектов на BOINC, для определения поведения и характеристик сети и вычислительных узлов позволяет достаточно реалистично имитировать среду добровольных вычислений. Более того, дополнительно добавляется случайность в результат выполнения задачи и задержки выполнения(в том числе задержки в сети) с помощью равномерного и гауссовского распределения соответственно. Таким образом, SimBA учитывает основные особенности выполнения задач на BOINC, позволяет тестировать различные политики распределения задач, масштабируется до 50000 вычислительных узлов и имеет погрешность несколько процентов. Однако SimBA подразумевает наличие только одного проекта BOINC; не моделирует выполнение заданий, в том числе многопроцессорные вычислительные узлы; не берет во внимание наличие файлов, которые необходимо скачивать для выполнения задач.

EmBOINC [11] – эмулятор системы BOINC. По своей архитектуре он во многом похож на SimBA, однако он эмулирует BOINC scheduler, используя реальное программное обеспечение BOINC и его базу данных, но взаимодействуя напрямую с демонами BOINC, запускающими генерацию, распределение, сбор и проверку заданий. Такой подход позволяет получать более точные результаты и сохраняет совместимость эмулятора с BOINC, позволяет проводить симуляции с количеством хостов до 100 000 и допускает возможность наличия сразу нескольких проектов BOINC в симуляции. Тем не менее, EmBOINC во многом имеет те же минусы, что и SimBA – отсутствие симуляции вычислительных узлов и необходимости загрузки файлов для задач.

Combos [1]– симулятор, использующий SimGrid API, который в наиболее полном объеме учитывает архитектуру BOINC. Авторы Combos добавили возможность участия добровольца в нескольких проектах одновременно и симуляцию вычислительных узлов, выполняющих задачи, включая стратегию исполнения задач и запроса новых задач. Также была добавлена возможность создания кластеров добровольцев, так как в реальности зачастую университеты или другие научные организации могут предоставлять свои сервера для проектов добровольных вычислений. Более того, симулятор учитывает время доступа к диску, размер передаваемых файлов, время исполнения задания. Combos масштабируется лучше упомянутых ранее симуляторов BOINC: эксперименты включали до миллиона вычис-

лительных узлов. Таким образом, Combos является наиболее подходящим симулятором для получения результатов, близких к реальным.

3 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [2] – платформа для быстрой и удобной организации добровольных вычислений. Изначально она была создана для проекта SETI@HOME, однако впоследствии стала доступна и для других проектов. На данный момент BOINC используется с целью задействия добровольных ресурсов в проектах разных областей науки и является одним из самых популярных программных комплексов своего рода.

BOINC отвечает за регистрацию добровольцев, распределение заданий между ними, получение и обработку результатов вычислений, управление базами данных проектов, сохраняя при этом безопасность и надежность. BOINC позволяет клиентам подключаться к нескольким проектам и указывать, как ресурсы распределяются между этими проектами.

Для проекта создается список задач(workunit),каждый из которых соответствует определенной подзадаче. Затем для каждого workunit путем репликации создается несколько workunit instances, которые впоследствии будут отправлены на разные устройства. Репликация заданий используется для устранения нестабильности системы, а также других проблем, таких как вредоносные атаки, аппаратные сбои, которые в конечном итоге влияют на надежность результатов.

Важной частью BOINC является валидация результатов. При получении результата вычисления какого-то workunit instance этот результат сравнивается с остальными результатами соответствующего workunit. Если на результатах большинством достигнут консенсус, то выбирается канонический результат. Иначе нужно ожидать последующих результатов для этого workunit. Такая стратегия позволяет предотвратить атаки злоумышленников.

3.1 Архитектура BOINC

Желающие предоставить свои ресурсы регистрируются на сайте проекта, скачивают и запускают BOINC клиент. После присоединения к проекту клиент периодически посылает запросы серверу проекта, в которых содержится информация о текущей нагрузке, результаты выполненных задач и запрос новых задач. Клиент скачивает необходимые входные данные и файлы приложения, исполняет задачи и загружает полученные результаты.

Сервер BOINC проекта состоит из одного или нескольких серверов, на которых исполняется программное обеспечение BOINC. Основным хранилищем является база данных, в которой содержится информация о приложениях, workunits, результатах, аккаунтах и так далее. В зависимости от функционала BOINC сервер можно разделить на две компоненты: Scheduling servers – серверы, которые отвечают прием соединений от клиентов, назначение workunit instances и сбор результатов; Data servers, с которых клиенты скачивают входные и исполняемые файлы, в них также загружаются выходные файлы.

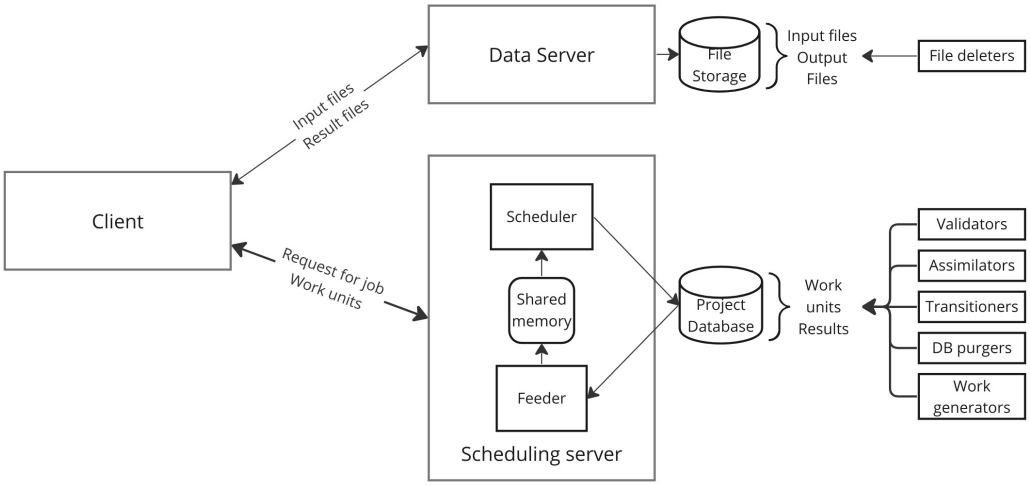

Рис. 3.1: Архитектура BOINC

Остальная функциональность сервера выполняется другими процессами-демонами:

1 Feeder – добавляет в пустые ячейки кэша еще не выполненные задачи.

2 Validator – ответственен за валидацию результатов: сравнивает результаты исполнения work unit instances и проверяет достижение кворума и в случае успеха выбирает канонический результат. Результаты могут сравниваться побитово либо предоставленной проектом функцией.

3 Assimilator – обрабатывает успешно выполненные результаты с помощью функции, специфичной для проекта. Например, может отправить результаты на компьютер авторов проекта.

4 Transitioner – отвечает за переход workunit instances между состояними.

5 File deleter – удаляет входные файлы и результаты выполненных workunit instances.

6 Database purger – удаляет записи о выполненных workunit и workunit instances.

На каждый запрос клиента создается новый экземпляр scheduler, задача которого – назначить workunit instance. Для этого необходимо сканирование базы данных, что является достаточно трудоемкой и долгой операцией, поэтому вместо этого был создан кэш еще не выполненных workunit instances. Он находится в разделяемой памяти и доступен всем экземплярам scheduler. После отправки некоторой задачи из кэша исполнителю, scheduler удаляет ее из кэша. За обновление кэша ответственен процесс-демон под названием feeder, который сканирует базу данных и добавляет невыполненные workunit instances в освободившиеся места в кэше.

Разделение функций сервера между несколькими независимыми процессами-демонами дает сразу несколько преимуществ: во-первых, такая система повышает устойчивость к отказам. Если один из демонов перестает работать, это никак не влияет на остальных. Данные накапливаются в базе данных и в конце концов будут обработаны после восстановления отказавшей компоненты. Во-вторых, архитектура BOINC легко масштабируется: все вспомогательные процессы могут исполняться на разных машинах. Более того, каждый демон может исполняться в нескольких потоках, что позволяет BOINC серверу проекта обрабатывать миллионы workunit instances в день.

4 Эксперименты

Для проведения экспериментов был выбран симулятор Combos, который в наиболее полном объеме учитывает особенности добровольных вычислений, моделируя сеть, сторону добровольцев, необходимость скачивать файлы и обращаться к диску, и при этом хорошо масштабируется. Код экспериментов находится в репозитории.

4.1 Параметры

Параметры симуляции задаются в файле parameters.xml. Этот файл содержит характеристику проектов и кластеров вычислительных узлов. Основой для этих параметров стала статистика проекта Asteroids@home [4] .

Доступность и недоступность вычислительных узлов в кластере генерируется с помощью случайного распределения. Согласно [12], распределение Вейбулла($Weibull(k = 0.393, \lambda = 2.964)$ – наиболее подходящее для генерации периода доступности сервера, в то время как для периодов недоступности больше подходит логнормальное распределение ($Lognormal(\mu = -0.586, \sigma = 2.844)$.

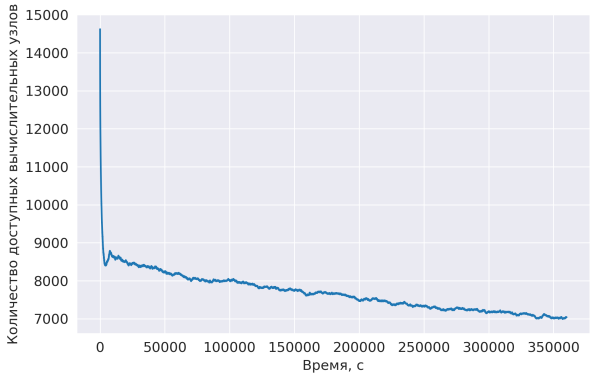

Рис. 4.1: Количество доступных вычислительных узлов

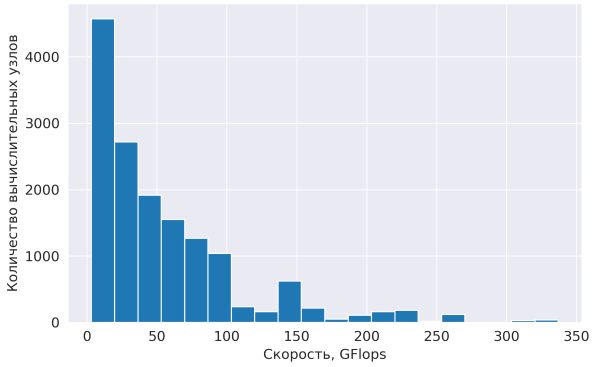

Рис. 4.2: Распределение мощности вычислительных узлов

Мощность вычислительных узлов может как генерироваться с помощью случайного распределения, так и задаваться с помощью traces. Asteroids@home предоставляет соответствующую статистику, поэтому в этой работе для определения мощности вычислительных узлов используются traces.

Выполнение одного workunit instance в проекте Asteroids@home в среднем требует 108,000 GigaFLOPS, а значит на вычислительном устройстве с процессором Intel Core i5-8400 с 6 ядрами такие вычисления заняли бы около часа. Файлы с входными данными и результаты имеют размер 5 Мегабайт.

Для валидации workunit необходимо получение 2 корректных результатов соответствующих instances. Общее количество выданных instances не более 20, а ошибочных – не более 7.

4.2 Полученные результаты

На графике 4.3 изображена зависимость количество выполненных и провалидированных workunits от времени.

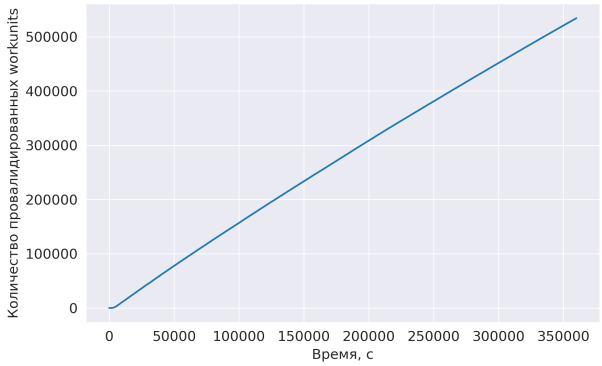

Рис. 4.3: Количество провалидированных workunits

Судя по этому графику, количество провалидированных workunits зависит от времени линейно. Это объясняется тем, что вне зависимости от нестабильности системы, отключения и подключения вычислительных узлов, гетерогенности, средняя вычислительная мощность системы почти не изменяется.

5 Модификация Combos

5.1 Возможность присоединения новых узлов

Непредсказуемость среды в симуляции моделируется с помощью двух случайных распределений: распределение доступности и распределение недоступности. Для каждого хоста после подключения с помощью первого распределения генерируется промежуток времени, в течение которого происходят вычисления. После окончания этого периода с помощью второго распределения генерируется промежуток времени, в течение которого вычислительный узел бездействует, и этот цикл повторяется до окончания симуляции.

На графике 4.1 заметно, что в такой модели количество активных хостов со временем уменьшается, так как количество вычислительных узлов в эксперименте ограничено. Для устранения этого эффекта в параметры было добавлено распределение количества новых узлов в сутки. С помощью него генерируется количество новых вычислительных узлов для каждых суток в симуляции. Также для каждого нового хоста генерируется время суток, в которое он подключается для того, чтобы хосты подключались равномерно в течение суток.

5.2 Разделение вычислительных узлов на 4 типа в зависимости от доступности

Изначально в симуляторе Combos доступность и недоступность узлов генерировалась с помощью случайного распределения, указанного в параметрах симуляции. Такой способ не является достаточно гибким, так как в системе могут присутствовать как те узлы, которые считают постоянно(например, узлы, подключенные создателями проекта), так и те, которые отключаются навсегда, не посчитав ни одного задания. Количество тех или иных узлов может меняться от проекта к проекту, что очень сложно отразить с помощью одного случайного распределения.

Для возможности более точного моделирования доступности узлов в этой работе их разделили на 4 типа:

1 **Reliable** – доверенные узлы, которые сразу подключаются к проекту и выполняют задачи, не отключаясь. В качестве Reliable узлов могут выступать вычислительные мощности, подключенные авторами проекта или какой-либо компанией/образовательным учреждением, которое заинтересовано в результатах проекта. Reliable узлы подключаются в течение первых суток симуляции и доступны до ее конца.

2 **Periodic** – узлы, которые моделируют подключение в течение рабочего дня. Добровольцы зачастую подключают компьютер в течение дня на период 7-9 часов и отключают его ночью. Также для таких добровольцев характерны редкие отключения на более длительный период(2-3 недели), например, если владелец компьютера уезжает в отпуск. Для генерации доступности Periodic узлов используется нормальное распределение с пиком в 8 часах($N(\mu = 8, \sigma = 1)$), а для генерации недоступности – распределение с двумя пиками, которое с вероятностью 0.95 соответствует $N(\mu = 16, \sigma = 1)$, иначе соответствует $N(\mu = 168, \sigma = 48)$.

3 **Usual** – узлы, которые обычно производят вычисления несколько часов, а затем отключаются на несколько часов, а также редко могут отключаться на более длительный период. Этот тип узлов соответствует обычным добровольцам, для которых нельзя выделить какую-либо специфику поведения. Случайное распределение для Usual хостов можно указать в параметрах симуляции, что сохраняет обратную совместимость в случае, если необходимости в разделении хостов на типы нет. В этой работе для таких хостов используется $N(\mu = 3, \sigma = 1)$ для генерации доступности и нормальное распределение с двумя пиками, аналогичное распределению из предыдущего пункта, для генерации недоступности.

4 **Unreliable** – ненадежные вычислительные узлы, соответствующие добровольцам, которые решили попробовать участвовать в проекте добровольных вычислений, однако по каким-то причинам почти сразу отключились, возможно, не посчитав ни одного задания. В симуляторе такие узлы подключаются на несколько часов(их количество определяется с помощью $Exp(1.5)$) и затем отключаются навсегда.

В параметрах эксперимента указывается количество каждого из типов узлов в процентах, что позволяет варьировать надежность среды. Следует заметить, что новые Reliable хосты не генерируются, исходя из их определения. Также с целью моделирования географической распределенности, а именно различных часов поясов, для каждого хоста генерируется не только день подключения, но и время в сутках, что особенно важно Periodic хостов.

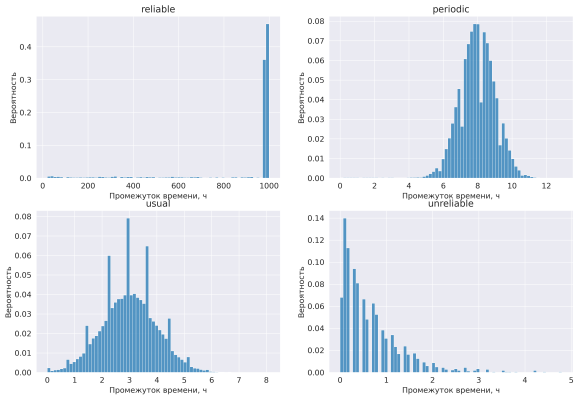

Рис. 5.1: Распределение доступности вычислительных узлов в зависимости от их типа

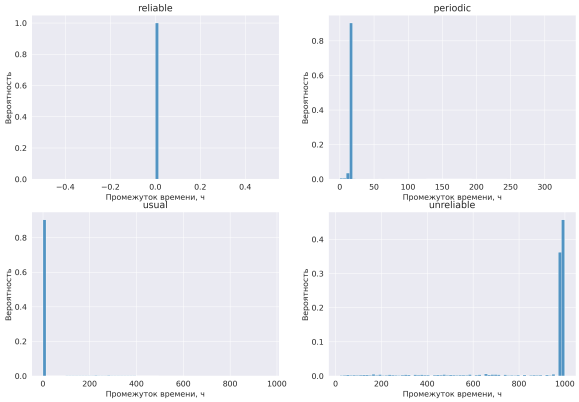

Рис. 5.2: Распределение недоступности вычислительных узлов в зависимости от их типа

Графики 5.1 и 5.2 показывают распределение доступности и недоступности вычислительных узлов в зависимости от их типа. Как было упомянуто выше, для генерации недоступности Periodic и Usual хостов используется нормальное распределение с двумя пиками, поэтому из-за разницы в масштабах графики выглядят именно так.

Количество активных хостов каждого из типов изображено на графике 5.3. Для него использовалось соотношение в количестве хостов 10/40/40/10, которое далее использовалось в экспериментах. Видно, что в течение первых суток симуляции количество вычислительных узлов каждого из типов увеличивается, после чего количество Reliable хостов остается постоянным. Unreliable хосты генерируются в течение всей симуляции, но из-за того, что их длительность подключения зачастую меньше, чем период, соответствующий одному делению гистограммы, далеко не все они отображены на диаграмме.

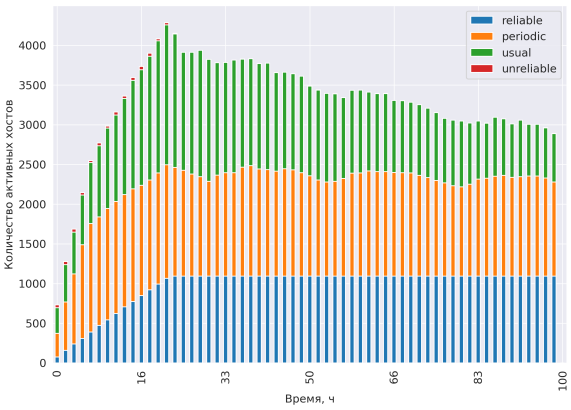

Рис. 5.3: Количество активных вычислительных узлов

Таким образом, разделение вычислительных узлов на разные типы в зависимости от их доступности и указания их соотношения в системе помогают более точно описать среду, в которой производятся вычисления проекта.

5.3 Моделирование нескольких ядер

В реальной системе BOINC существует специальная стратегия CPU scheduling, согласно которой задания распределяются по ядрам. С целью упрощения симуляция не учитывает эту деталь, моделируя несколько ядер через одно с пропорциональным увеличением мощности.

Так как проект Asteroids@home предоставляет статистику мощности вычислительных узлов, включая количество ядер на них, было принято решении о проведении эксперимента с моделированием каждого из ядер как отдельного вычислительного узла. Для этого были скорректированы traces для мощности вычислительных узлов. Хостам, соответствующим ядрам одного и того же вычислительного узла, назначается один и тот же тип.

Однако в таком сценарии периоды доступности и недоступности у ядер не синхронизированы, т.е. хост, соответствующий одному ядру, может выполнять вычисления, в то время как хост, соответствующий другому ядру, может "спать". Такое поведение более приближено к реальной системе, где зачастую задачи проекта можно выполнять только в фоновом режиме. Тогда на некоторых ядрах исполняются основные вычисления(в этот момент соответствующие хосты в проекте бездействуют), а на некоторых ядрах вычисляются задачи проекта. Тем не менее, такой способ не избавляет от всех проблем, ведь все еще отсутствует общая scheduling политика между ядрами одного вычислительного узла, а также увеличивается(примерно в 10 раз) количество хостов в симуляции, из-за чего симуляция длится дольше(см. график 5.4).

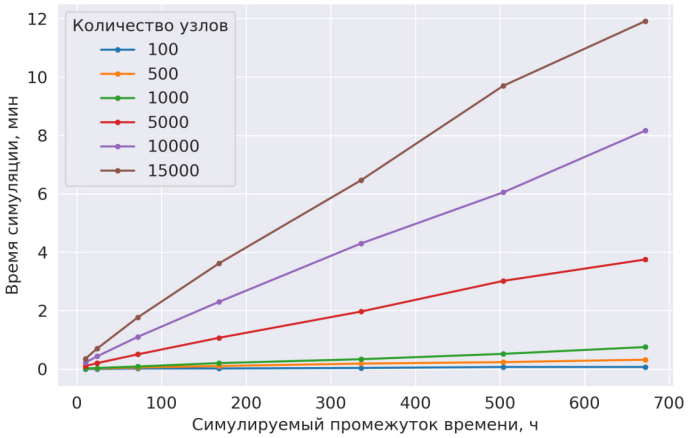

Рис. 5.4: Зависимость длительности симуляции от симулируемого промежутка времени и количества вычислительных узлов

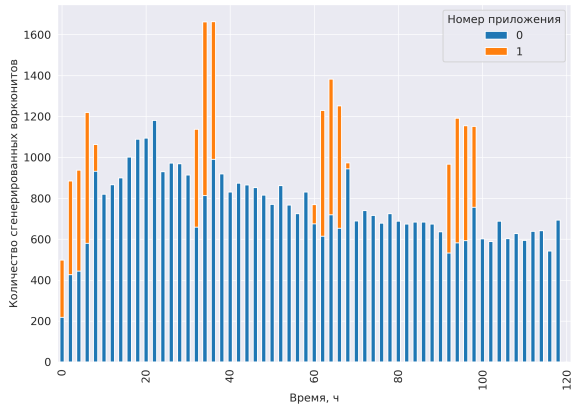

Рис. 5.5: Количество сгенерированных workunits в единицу времени

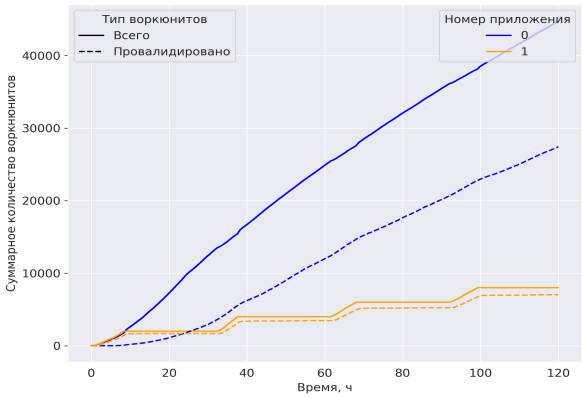

Рис. 5.6: Соотношение общего количества и провалидированных workunits

5.4 Несколько приложений

В BOINC каждому проекту соответствуют одно или несколько приложений. Они могут отвечать за разные подзадачи проекта или предназначены для разных операционных систем. То есть сложность задач, параметры репликации, размеры и количество входных и выходных файлов могут отличаться в зависимости от приложения. До этого в Combos нельзя было указать несколько приложений для одного проекта, поэтому такая возможность была добавлена в рамках этой работы.

В параметрах для каждого из приложений указывается число GigaFLOPS, которое необходимо выполнить для вычисления задачи, параметры входных и выходных файлов, параметры репликации, размер кворума, процент валидных результатов среди всех результатов. Также для каждого из приложений указывается процентное отношение количества workunits, которое оно генерирует, к общему количеству workunits проекта. Таким образом можно регулировать важность каждого из приложений, повышая этот параметр для приложения.

Кроме того, в доработанном симуляторе также возможно смоделировать сценарий, когда есть основное приложение, в котором постоянно генерируются workunits, и приложения, которое появляется периодически, генерирует определенное количество workunits и исчезает на некоторый период. На графиках 5.5 и 5.6 показано количество сгенерированных workunits в единицу времени и соотношение провалидированных workunits к их общему числу в зависимости от приложения для такого сценария. Оба приложения генерировали workunits в равном объеме, однако задачи второго приложения выполняются быстрее в 10 раз. Второе приложение генерирует 2000 workunits и затем исчезает на 24 часа.

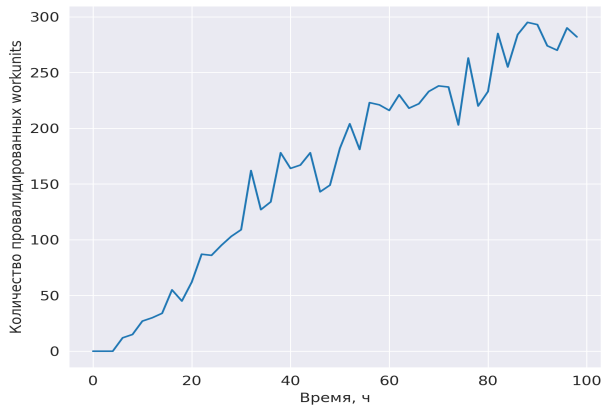

Рис. 6.1: Количество провалидированных workunits в единицу времени

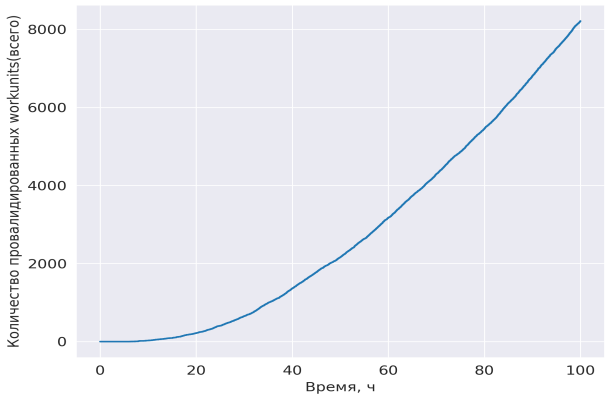

Рис. 6.2: Количество провалидированных workunits всего

6 Исследование отдельных сценариев

6.1 Начало проекта

Отдельным сценарием, заслуживающим исследования, является начало проекта. А именно, как быстро скорость валидации workunits выйдет на желаемый порог.

Для этого эксперимента код симулятора был изменен так, чтобы все хосты, за исключением Reliable, присоединялись равномерно в течение симуляции. Reliable вычислительные узлы присоединяются с равномерной скоростью в течение первых суток симуляции. Количество активных хостов для такого сценария изображено на графике 6.3.

Графики 6.1 и 6.2 демонстрируют количество провалидированных workunits в единицу времени и общее количество провалидированных workunits соответственно. Так как вычислительные узлы присоединяются к проекту равномерно в течение суток, количество провалидированных workunits в единицу времени растет окололинейно. Как следствие, общее количество провалидированных workunits имеет квадратичную зависимость от времени, в то время как в обычных условиях (график 4.3) эта зависимость линейна.

6.2 Окончание проекта

Окончание проекта отличается тем, что в этом случае количество задач меньше, чем количество активных вычислительных узлов. Узлы, которые долго не получают задач от проекта, могут отключаться от него, из-за чего общая вычислительная мощность падает и завершение проекта может потребовать намного больше времени, чем хотелось бы его авторам.

Для эксперимента work_generator генерировал 1000 workunits, для каждого из кото-

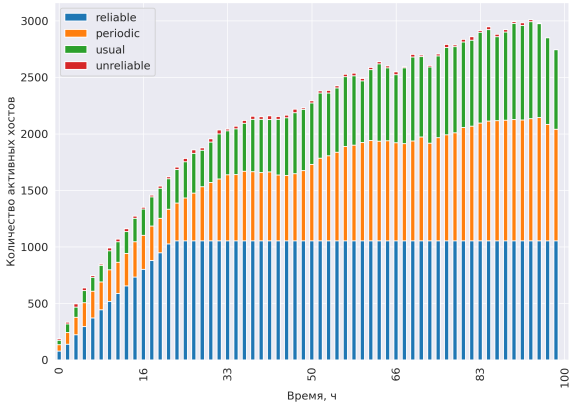

Рис. 6.3: Количество активных вычислительных узлов

рых создавалось 2 workunit instances, после чего генерация новых workunits останавливалась и генерировались только workunit instances, заменяющие те, которые завершились с какой-либо ошибкой. Количество вычислительных узлов – около 10000.

Изначально в симуляторе при отсутствии задач в ответе от scheduler клиент сразу же отключался от проекта. Это приводило к тому, что после генерации 2000 workunit instances и отправки их хостам, остальные хосты(которые составляют большинство), отключались от проекта, из-за чего вычислительной мощности не хватало для досчета компенсирующих workunit instances. Поэтому стратегия поведения клиента в таком случае была изменена на ту, которая более приближена к BOINC: collision avoidance с exponential back off. А именно, если scheduler присылает пустой ответ, то клиент увеличивает промежуток времени до следующего запроса в x раз, где $x \in [1.9, 2.1]$ и выбирается случайно из этого промежутка. Такое поведение позволяет снизить нагрузку на scheduler и избежать эффекта, когда почти все запросы от вычислительных узлов приходят одновременно.

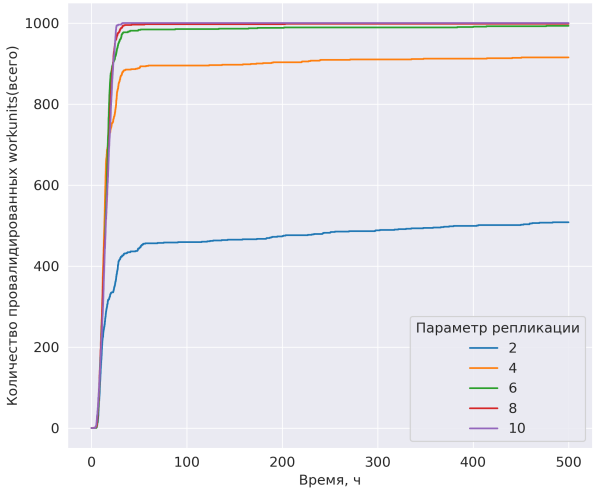

Рис. 6.4: Количество провалидированных workunits в зависимости от параметра репликации

На графике 6.4 изображена зависимость количества провалидированных workunits в зависимости от параметра репликации(количества workunit instances, генерируемых изначально для каждого workunit). Размер кворума при этом остается постоянным и равен 2.

Можно заметить, что проект завершается быстрее при увеличении параметра репликации. Действительно, в условиях дефицита задач и наличия вычислительных узлов, которые простаивают, увеличение избыточности вычислений позволяет быстрее достигнуть кворума. Ведь если для workunit создается, например, 4 workunit instances, а не 2, то кворум размера 2 скорее всего достигнется быстрее.

7 Заключение

В рамках данной работы были выявлены особенности среды добровольных вычислений, которые важно учитывать при ее моделировании. Также проведен анализ существующих инструментов для симуляции добровольных вычислений, на основе которого был выбран симулятор Combos, так как он в наиболее полном объеме учитывает специфику добровольных вычислений: моделирует сеть, сторону добровольцев, необходимость скачивать файлы и обращаться к диску, и при этом хорошо масштабируется.

В качестве основы для входных параметров симуляции(traces мощности вычислительных узлов, параметры задач) была взята статистика проекта Asteroids@home. Доступность узлов генерировалась с помощью подходящего случайного распределения.

С целью повышения точности моделирования код симулятора Combos был доработан: добавлена возможность присоединения новых узлов, разделения вычислительных узлов на 4 типа в зависимости от их доступности, моделирования нескольких ядер в каждом вычислительном узле и нескольких приложений для каждого проекта. Кроме того, проведены эксперименты с симуляцией начала и конца проекта.

Список литературы

[1] Saúl Alonso-Monsalve, Félix García-Carballeira и Alejandro Calderón. “Combos: A complete simulator of volunteer computing and desktop grids”. В: *Simulation Modelling Practice and Theory* 77 (2017), с. 197—211.

[2] David P Anderson. “BOINC: a platform for volunteer computing”. В: *Journal of Grid Computing* 18.1 (2020), с. 99—122.

[3] David P Anderson. “Boinc: A system for public-resource computing and storage”. В: *Fifth IEEE/ACM international workshop on grid computing*. IEEE. 2004, с. 4—10.

[4] *Asteroids@HOME project*. URL: https://asteroidsathome.net/boinc (дата обр. 04.02.2024).

[5] *BOINC source code*. URL: https://github.com/BOINC/boinc (дата обр. 15.03.2024).

[6] *BOINC statistics*. URL: https://www.boincstats.com/ (дата обр. 04.02.2024).

[7] Rajkumar Buyya и Manzur Murshed. “Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing”. В: *Concurrency and computation: practice and experience* 14.13-15 (2002), с. 1175—1220.

[8] Henri Casanova. “Simgrid: A toolkit for the simulation of application scheduling”. В: *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE. 2001, с. 430—437.

[9] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson и Frédéric Suter. “Versatile, scalable, and accurate simulation of distributed applications and platforms”. В: *Journal of Parallel and Distributed Computing* 74.10 (2014), с. 2899—2917.

[10] Henri Casanova, Arnaud Legrand и Martin Quinson. “Simgrid: A generic framework for large-scale distributed experiments”. В: *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*. IEEE. 2008, с. 126—131.

[11] Trilce Estrada, Michela Taufer, Kevin Reed и David P Anderson. “EmBOINC: An emulator for performance analysis of BOINC projects”. В: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE. 2009, с. 1—8.

[12] Bahman Javadi, Derrick Kondo, Jean-Marc Vincent и David P Anderson. “Discovering statistical models of availability in large distributed systems: An empirical study of seti@ home”. В: *IEEE Transactions on Parallel and Distributed Systems* 22.11 (2011), с. 1896—1903.

[13] Michela Taufer, Andre Kerstens, Trilce Estrada, David Flores и Patricia J Teller. “SimBA: a discrete event simulator for performance prediction of volunteer computing projects”. B: *21st International Workshop on Principles of Advanced and Distributed Simulation (PADS'07)*. IEEE. 2007, с. 189—197.