Математические методы оптимизации распределенных вычислений в гетерогенной среде

В. А. Новоженов, Т. Н. Романова

***Аннотация* — В данной статье рассмотрена возможная организация и архитектура гетерогенной кластерной среды с различной архитектурой, производительностью и характеристиками для распределенных вычислений. Также рассмотрены математические методы оптимизации времени вычисления очереди задач для распределенных вычислений в ней.**

Преследовалась цель выявления оптимального алгоритма планирования задач на кластере, для заданных условий и ограничений математической модели. В рамках поставленной задачи был спроектирован программный комплекс, а также был проведен ряд прикладных экспериментов на нем. В результате была показана эффективность алгоритма BlackFill, модифицированного под требования задачи. Результаты численного моделирования демонстрируют эффективность предложенных методов по сравнению с традиционными подходами. Применение разработанных методов позволяет значительно сократить время выполнения вычислений, повысить степень использования доступных ресурсов и обеспечить устойчивую работу системы в условиях изменяющейся нагрузки. Результаты могут быть полезны для разработки высокопроизводительных вычислительных систем и алгоритмов управления в гетерогенных средах.

***Ключевые слова* — распределенные вычисления, математическая оптимизация, алгоритмы планирования задач на кластере, гетерогенный кластер, очередь задач, кластерные вычисления.**

I. Введение

Распределенные вычисления представляют собой вычисления, которые могут одновременно использовать несколько серверных узлов, объединенных в одну систему для более быстрого решения задачи, для которой требуется большое количество времени вычисления. Такие вычисления могут осуществлять кластеры (о которых пойдет речь в этой работе), суперкомпьютеры, а также ГРИД-системы. Распределенные вычисления требуются в большом классе задач, например можно выделить следующие актуальные направления:

- численное решение систем дифференциальных уравнений с большим количеством переменных;
- мультиагентные системы управления технологическими процессами;
- моделирование физических, химических и других объектов и процессов;
- обработка больших объемов данных;
- обучение тяжеловесных нейронных сетей для задач естественного языка, построение больших языковых моделей, решение задач компьютерного зрения [1-4].

Из актуальных проблем можно выделить сложность построения распределенных вычислений. Разработчик должен понимать производительность каждого узла, а также грамотно организовать обмен данными между этими узлами после каждой итерации вычислений [5-6].

Одним из наиболее часто используемых стандартов распределенных вычислений является MPI (Message Passing Interface). Этот стандарт предоставляет программный интерфейс для обеспечения связи между отдельными процессами, в том числе выполняющимися в физически распределенной среде на разных физических серверах, независимо от архитектуры, взаимного расположения процессов в распределенной системе. Принцип организации параллельных вычислений представлен на Рис.1.

Рис. 1 Принцип организации параллельных вычислений

Система программирования на основе стандарта MPI относится к классу вычислительных устройств с индивидуальной памятью, то есть к многопроцессорным системам с обменом сообщениями. В состав MPI входит библиотека программирования. Она определяет процесс работы распределенной задачи. Программы, откомпилированные обычными компиляторами и связанные с MPI-библиотекой (вызовами процедур,

Статья получена 27 сентября 2024 г.
В.А. Новоженов – МГТУ им. Н.Э. Баумана, Москва, Россия. (email: gail64917@yandex.ru)
Т.Н. Романова – доцент, кандидат физико-математических наук, МГТУ им. Н.Э. Баумана, Москва, Россия (email: rtn@bmstu.ru)

обеспечивающих обмен сообщениями между узлами), считаются параллельными. MPI включает большой набор коллективных операций коммуникации, виртуальных топологий и различных способов коммуникации. MPI имеет достаточно много реализаций. Эти реализации обеспечивают асинхронную коммуникацию, эффективное управление буфером сообщения, эффективные группы, и другие функциональные возможности. Использование интерфейса MPI и одной из его реализаций MPICH позволяет эффективно организовать распределённые вычисления для локальных сетей.

II. Гетерогенные и гомогенные вычислительные системы

Во множестве всех кластерных систем можно выделить две группы: гомогенные вычислительные среды, которые состоят из однородных вычислительных узлов с одинаковыми ресурсами и конфигурациями, а также гетерогенные кластерные системы, которые состоят из узлов с отличающимися друг от друга конфигурациями, количеством ресурсов и производительностью.

Чаще всего для построения кластерной вычислительной системы используется подход с применением однородных узлов.

Однако со временем, при необходимости расширения кластера, в силу разных причин, могут быть применены процессоры, отличные от тех, что применялись в кластере изначально. Таким образом, вычислительный кластер может стать неоднородным [7].

Неоднородность кластера может создать следующие проблемы [8-10]:

- возрастает сложность в распределении задач между узлами из-за различий в производительности;
- узлы могут иметь разное соотношение вычислительных ресурсов по отношению друг к другу;
- различие в архитектуре процессоров может потребовать подготовки различающихся исполняемых задач для разных узлов;
- у разных узлов может быть доступ к разным хранилищам данных, необходимых для вычислений.

Для обеспечения надлежащего качества гетерогенной системы к вычислительной системе могут предъявляться следующие требования:

1. возможность постановки задач в очередь с дальнейшим запуском из очереди на наиболее подходящих для запуска (по тем или иным требованиям) наборах узлов;
2. минимизация времени вычисления задач в очереди для обеспечения максимальной производительности кластера (или же возможна минимизация времени выполнения задач с первым приоритетом, минимизация утилизируемых ресурсов в кластере и так далее);
3. обработка ошибок и отказов в кластере без прекращения работы всего кластера;
4. распределение задач в соответствии с приоритетами и текущей загрузки узлов и кластера в целом;
5. гарантированный объем доступных и/или свободных вычислительных ресурсов на кластере (возможна очередь из задач, которые можно прервать для запуска более приоритетных задач);
6. валидация результатов вычислений узла/задачи для обеспечения надежности результатов;
7. возможность задать ограничение для задачи по времени исполнения (после которого задача принудительно завершается) или по объему утилизируемых ресурсов. Это требование может носить как желательный, так и обязательный характер [11]. Для обеспечения надежности результатов вычислений, в случае отсутствия полного контроля над вычислительными узлами целесообразно использование алгоритмов консенсуса, например, такого как Raft. Результат вычисления задачи независимо проверяет n других узлов. Этот алгоритм требует выбора узла лидера, а также наличие коммуникации между узлами, выполняющими задачи и проверяющими их [12].

Ограничение времени выполнения задач и обеспечение контролируемого расходования средств на вычисления характерно для сред с платными ресурсами и подразумевает наличие механизмов, минимизирующих расход средств на оплату ресурсов при заданных пользователем ограничениях на время выполнения (или минимизирующих время выполнения при заданном ограничении на бюджет).

Обеспечение защиты системы от внешнего и внутреннего воздействия подразумевает изоляцию как вычислений от внешнего доступа (то есть невозможность получить доступ в исполняемый контекст задачи или обрабатываемые на узле данные) так и от внутреннего (то есть невозможность при помощи запускаемой задачи получить доступ к данным, которые использует другой пользователь в другой задаче, но на этом же сервере) [13].

Обеспечение конфиденциальности соединения и безопасности данных – подразумевает шифрование передаваемых данных, а также невозможность для внешнего участника получить и расшифровать трафик, с которым работают узлы кластера [14].

Для этих целей может использоваться виртуализация и аппаратное шифрование на вычислительных узлах. Виртуализация обеспечивает безопасную и изолированную операционную среду для всех типов традиционного программного обеспечения. У каждой виртуальной машины имеется свой набор доступных ресурсов (таких как ядра процессоров, объем памяти и другие). В рамках этих выделенных ресурсов запускаются отдельные операционная система и приложения. Операционная система использует набор аппаратных средств и не знает о совместном использовании с другими гостевыми операционными системами, работающими на одной и той же физической платформе [15].

Также целесообразно использование VPN-тоннелей и шифрование передаваемого трафика.

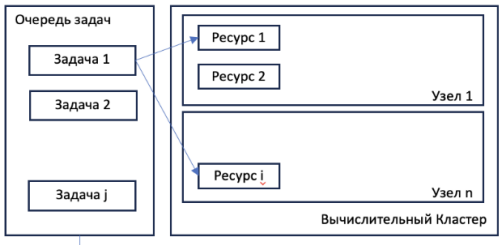
Рассмотрим возможные к применению в задаче методы оптимизации вычислений.

Задачу оптимального планирования выполнения

вычислительных задач на кластере можно свести к разным параметрам. Например, минимизации времени выполнения всех задач в очереди или минимизации утилизируемых узлов/ресурсов или же для приоритезации выполнения, более важных (или ресурсоемких/дешевых) задач.

Оптимизация времени выполнения задач на кластере может быть рассмотрена как задача динамической оптимизации, поскольку она включает в себя поиск оптимальных решений в реальном времени в зависимости от текущей загрузки кластера, доступных ресурсов и других факторов. Динамическая оптимизация позволяет адаптировать выполнение задач к изменяющимся условиям, чтобы достичь максимальной производительности и эффективности кластера.

С широким распространением кластеров стала актуальна задача распределения параллельных задач по процессорам. Задача оптимизации очереди вычислений на кластере в простейшем виде (все узлы однородны, нет зависимости от сетевой связанности и других факторов) может выглядеть следующим образом: примем известную конфигурацию кластера для распределенных вычислений. Будем считать, что нам известно, сколько узлов входит в него, и какими ресурсами располагает каждый узел. В рамках поставленного эксперимента ресурсами узла будем считать количество GPU-ускорителей (также параметрами может являться количество CPU-ядер на сервере, который является узлом кластера, или объем RAM, SSD и т. д.). В кластере существует очередь задач для распределенных вычислений (Рис. 2).



Очередь задач		
Номер задачи	вычислительная емкость (кол-во гпу минут)	минимально необходимое количество ресурсов задействованных одновременно (гпу)
1	4	4
2	16	16
3	8	4
4	8	8
5	4	4

Рис. 2. Описание системы

У каждой вычислительной задачи в очереди имеется такой ряд параметров, как минимальный/максимальный объем ресурсов, необходимых на ее вычисление, присутствуют требования к конкретным параметрам узла (например, соответствующая архитектура процессора) и оценочное время выполнения задачи на кванте ресурса.

Для исполнения задач на кластере планировщик должен подобрать такое расписание, которое минимизирует суммарное время вычисления всех задач в очереди. То есть, планировщику необходимо назначить для каждой задачи узел и ресурс, на котором она будет исполняться в конкретный момент времени. Большинство подобных задач являются NP-полными и недетерминированно-разрешимыми за полиноминальное время. А значит планировщик довольно затруднительно организовать таким образом, чтобы он для любой очереди эффективно распределял и планировал поток поступающих задач (поток из количества поступающих задач в момент времени - случайная величина количества задач в момент времени. Ее можно рассматривать, по теории массового обслуживания, как случайную величину с распределением Пуассона).

Проблему планирования задач решают эвристические алгоритмы, которые находят субоптимальные, но не самые эффективные решения из всех возможных, при этом за полиноминальное время.

В зависимости от работы с очередью происходит варьирование работы планировщика. Возможны варианты назначения задачи на узел.

Для выбора узла для выполнения задачи используются следующие подходы:

- назначение задачи на первый доступный узел,
- назначение на самый незагруженный узел,
- назначение на наиболее подходящий узел, при максимизации утилизации узла,
- назначение на наиболее подходящий по параметрам узел.

Для работы планировщика с очередью возможны стратегии:

- первая помещенная в очередь задача запускается первой (FIFO),
- последняя помещенная в очередь задача запускается первой (LIFO),
- наименее вычислительно емкая задача (требующая наименьшее количество времени для вычисления) запускается первой,
- наиболее вычислительно емкая задача (требующая наибольшее количество времени для вычисления) запускается первой,
- наиболее ресурсоемкая задача (требующая наибольшее количество одновременно используемых ресурсов) запускается первой,
- наименее ресурсоемкая (требующая наименьшее количество одновременно используемых ресурсов) задача запускается первой,
- задача с определенными параметрами запускается первой,
- случайно выбранная задача запускается первой,
- первыми запускаются задачи с наивысшим приоритетом.

Довольно частая проблема списочных алгоритмов (которые работают с отдельными задачами в очереди по порядку) – это низкая утилизация каждого отдельного узла кластера и неэффективное распределение задач.

С вышеописанной проблемой связана задача двумерной упаковки набора прямоугольников (в нашем

случае, задач в очереди) заданной ширины и высоты в горизонтальную, ограниченную, с одной стороны, полосу (характеризующей время работы кластера). В этой задаче необходимо минимизировать длину горизонтальной полосы, распределив прямоугольники как можно более эффективно. На прямоугольники накладываются ограничения: они не могут пересекаться, а также их нельзя вращать.

Современные управляющие системы вычислительного кластера и внешние планировщики для них применяют в основном различные модификации алгоритма Backfill.

В ходе работы алгоритма Backfill обрабатывается очередь задач, ожидающих запуска на выполнение. Задачи в очереди упорядочены по времени их регистрации в системе. При каждой регистрации и каждом завершении задачи происходит выполнение шагов алгоритма:

1) при наличии необходимого количества свободных процессоров для запуска первой в очереди задачи эта задача удаляется из очереди и запускается на выполнение;

2) если для запуска первой в очереди задачи свободных процессоров не хватает, то вычисляется момент времени, когда их станет достаточно, и производится резервирование для данной задачи;

3) продолжается движение по очереди с запуском на выполнение задач, не нарушающих резервирование первой в очереди задачи.

Этот алгоритм преследует две конфликтующие цели – повышение эффективности использования вычислительных ресурсов при помощи заполнения пустых пространств в расписании (полосе) и предотвращение длительного нахождения задач в очереди без исполнения за счет резервирования.

Ожидающие задачи хранятся в очереди и упорядочены согласно приоритетам. В каждом цикле планирования ресурсы выделяются задачам в порядке их приоритетов, причем задача может получить вычислительные ресурсы, если они не были отведены k другим самым приоритетным заданиям [16-20].

III. Параметры, ограничения и математическая модель задачи оптимизации времени выполнения задач на кластере

Рассмотрим однородный вычислительный кластер, узлы которого имеют одинаковое количество ресурсов. Пусть j (j ∈ Z) это количество задач, стоящих в очереди на выполнение на данном кластере. Необходимо назначить каждую задачу из очереди на исполнение на соответствующих вычислительных ресурсах таким образом, чтобы минимизировать наибольшее время утилизации среди всех вычислительных ресурсов (так как считаем ресурсы независимыми друг от друга). На одном ресурсе одновременно может выполняться не более одной задачи. (ресурс или квант ресурса в однородном кластере – некоторая часть узла однородного кластера)

Как правило, квант узла в кластере равен (где m ∈ Z) (то есть, например, половине узла: если в узле содержится 16 ядер CPU, 640 Gb RAM, 2 GPU-ускорителя, то квант может быть равен 8 CPU, 320 Gb RAM, 1 GPU-ускоритель)

Математическая постановка задачи оптимизации времени выполнения задач в очереди может выглядеть следующим образом:

- j – некоторый индекс задачи в очереди;
- t – некоторый момент времени работы кластера (для простоты будем считать t дискретным, t ∈N);
- i – некоторый номер ресурса в кластере;
- оценочная трудоемкость задачи $Complexity(j) = C(J) = C_j$ сколько необходимо времени ресурса на ее вычисление, то есть, например, CPU-минут или GPU-минут;
- запущена задача j на ресурсе i в момент времени t: $Running(j,i,t) = R_{j,i,t}$

$$R_{j,i,t} = \begin{cases} 0, & \text{если } j \text{ исполняется на } i \text{ в момент } t \\ 1 & \text{если } j \text{ не исполняется на } i \text{ в момент } t \end{cases} \quad (1)$$

- номер запущенной задачи в момент времени t, на ресурсе i: $Job(i,t) = J_{i,t}$;
- выражение $x(t) = \sum_j \sum_i R_{j,i,t} > 0$ или $x(t) = \sum_i J_{i,t} > 0$ - показывает, что для некоторого фиксированного t на каком-то узле еще производятся вычисления (то есть еще не все задачи в очереди исполнились);

Задачу минимизации времени исполнения всей очереди задач на кластерной вычислительной системе можно свести к максимизации количества моментов времени, когда на кластере ничего не выполняется.

Функция, показывающая, что в момент времени t что-либо исполняется:

$$f(\sum_j \sum_i R_{j.i,t}) = \begin{cases} 0, & \text{если } x(t) = \sum_j \sum_i R_{j.i,t} = 0 \\ 1 & \text{если } x(t) = \sum_j \sum_i R_{j.i,t} > 0 \end{cases} \quad (2)$$

Задача оптимизации очереди задач для сокращения суммарного времени вычисления задач сводится к:

$$f(\sum_j \sum_i R_{j.i,t}) \to \min \quad (3)$$

Так как, $x(t) = \sum_j \sum_i R_{j,i,t} \geq 0, \leq \forall t \in N$ а минимальный квант времени $\Delta t = 1$ то выражение примет вид:

$$f(x(t)-1) = \begin{cases} 0, & \text{если } x(t)-1 = -1 \\ 1, & \text{если } x(t)-1 \geq 0 \end{cases} \quad (4)$$

На выполнение задач также могут накладываться следующие ограничения:

конкретные вычислительные задачи в очереди имеют минимально и/или максимально необходимое количество ресурсов для запуска задачи (например, у

третьей задачи, то есть j = 3, задаче необходимо минимум 4 GPU для выполнения – так как для размещения весов нейронной сети и обучения с надлежащей скоростью требуется объем GPU-памяти эквивалентный 4 квантам ресурсов – например, происходит обучение модели Mixtral-8x7B. Возможны и другие конфигурации по количеству GPU и соответствующему времени обучения какой-либо задачи, их устанавливают при постановке задачи в очередь).

Для задачи j существует ограничение на количество утилизируемых в момент времени работы задачи ресурсов не менее q_i и не более Q_i ($q_i \le Q_i$) что (в любой момент времени выполняется условие: либо 0, либо больше q) минимальное отличное от 0 в любой момент времени количество ресурсов, утилизируемых под задачу, должно быть больше q_i то есть $\forall t$ и $\forall j$ выполняются условия:

$$\begin{cases} \sum_i R_{j.i.t} = 0, & \text{если задача } j \text{ нигде не выполняется в момент } t \\ q_i \le \sum_i R_{j.i.t} \le Q_{i,} & \text{если задача } j \text{ выполняется в момент } t \end{cases}$$

(5)

Это ограничение при q_i >1 говорит о том, что задача имеет возможности к параллелизму и должна выполняться более чем на одном ресурсе. Если же q_i =1, то задача обязательно должна выполняться без применения параллельных вычислений.

Конкретные вычислительные задачи в очереди могут запускаться на одном или же на n нескольких узлах (например, они поддерживают распределенные вычисления по стандарту MPI). Данное ограничение является расширением предыдущего. Однако задача усложняется расширением условия на количество квантов ресурсов в рамках одного узла кластера.

То есть $\forall t$ и $i \in$ множеству ресурсов на узлах, количество которых не превышает n

$$\begin{cases} \sum_i R_{j.i.t} = 0, & \text{если задача } j \text{ нигде не выполняется в момент } t \\ q_i \le \sum_i R_{j.i.t} \le Q_{i,} & \text{если задача } j \text{ выполняется в момент } t \end{cases}$$

(6)

конкретные вычислительные задачи в очереди могут масштабироваться во время исполнения по количеству исполнителей задачи вверх (то есть утилизировать для более быстрого выполнения больше ресурсов при их наличии и доступности), масштабироваться вниз (высвобождать ресурсы по мере выполнения), или не иметь возможности к масштабированию во время выполнения. Такие возможности могут быть предоставлены и специализированными фреймворками (например, для распределенного обучения нейронных сетей популярен Pytorch Elastic Learning, позволяющий задаче обучения нейронной сети масштабироваться во время вычислений).

В рамках данных ограничений целесообразно вычислять какая оценочная длительность задачи осталась. Также алгоритм планирования можно расширить до оценки вероятности размещения задачи j в момент времени t: $P_{j,t} \le 1$ Поступающие задачи в момент времени t могут рассматриваться как случайная величина, имеющая Пуассоновское распределение [21].

Конкретные вычислительные задачи в очереди могут быть приостановлены и возобновлены вновь, или же не имеют возможности к паузе, а только лишь к перезапуску

Ограничение на отсутствие возможности приостановить конкретную задачу можно выразить следующим образом: для конкретной задачи все интервалы времени на каждом ресурсе идут последовательно (то есть временной ряд первых разностей имеет не более двух отличий от нуля для конкретного номера задачи).

Конкретные вычислительные задачи в очереди могут изменять конфигурацию по требуемым квантам и количеству экземпляров исполнителей задачи (то есть для запуска задачи j = 3 может потребоваться 4 GPU на одном исполнителе, 2 GPU на 2-ух исполнителях или же по 1 GPU на 4-ех исполнителях. Данное свойство задач в очереди показывает возможна ли такая переконфигурация). Если же переконфигурация нужна в определенном моменте времени – целесообразно будет, например, (при наличии возможности остановки задачи) приостановить, поместить ее в начало очереди (или с более высоким приоритетом).

конкретные вычислительные задачи в очереди могут изменять суммарное оценочное время выполнения в зависимости от соотношения исполнителей и ресурсов, выделяемых на них). Для соблюдения ограничения можно выполнять оценку времени вычисления для возможных соотношений при планировании.

При наличии возможности к масштабированию задача становится более устойчивой к сбоям кластерной системы или к случаям ошибок на конкретных исполнителях.

Конкретные вычислительные задачи имеют требования к вычислительным узлам (например, соответствующую архитектуру процессора или аппаратную поддержку той или иной технологии), доступ в необходимую сеть, доступ к необходимым данным на внешних по отношению к выполняющему узлу источниках и др. Для этого целесообразно вводить новые переменные для учета соответствия требований задачи и параметров узла.

Все задачи в очереди имеют (или им были присвоены) некоторые приоритеты выполнения (то есть задачу n более целесообразно запустить, чем задачу m при соответствующем состоянии загрузки кластера или же всегда).

Конкретные вычислительные задачи имеют оценочную длительность или же их длительность заранее неизвестна.

Прогноз времени выполнения задачи необходимая в рамках задачи оптимизации величина. Необходимо оценивать длительность задачи какими-либо дополнительными эвристиками или методами.

Конкретные вычислительные задачи могут не иметь времени выполнения как такового и резервировать ресурсы для непрерывных вычислений. Такие задачи целесообразно помещать в отдельную очередь, не участвующую в планировании и запускать подобную задачу тогда, когда количество свободных ресурсов на кластере после осуществления планирования больше, чем емкость такой задачи. (или учитывать приоритеты).

Конкретные вычислительные задачи могут иметь требования к сетевым задержкам на узлах, на которых предполагается выполнение (например, при запуске задачи j = 3 на 4 квантах GPU в распределенном режиме необходимо, чтобы коммуникация между квантами, выполняющими задачу или между серверами составляла менее 50 мс, иначе выполнение задачи будет неэффективно.

Необходимо введение нового параметра узла (помимо количества ресурсов на нем), который показывает его связь с другим конкретным узлом.

Конкретные вычислительные задачи могут иметь требования к окончанию вычислений других задач, необходимых для запуска задачи (например, для запуска задачи j = 3 необходимо окончание вычислений задач j =1 и j = 2).

Возможно введение отношения предшествования задачи j от задачи l: Depends(task j, task l) = 1, если перед задачей j нужно выполнить задачу l.

Существует такое множество задач, перед которыми не нужно выполнять другие задачи. Также существует такое множество задач, которые не являются предшественниками других задач.

конкретные вычислительные задачи могут иметь количество попыток перезапуска в случае неудачного запуска/сбоя во время вычислений. В случае сбоя они возвращаются в очередь, проверяется количество оставшихся возможных перезапусков, и они участвуют в планировании загрузки кластера вновь (если не предусмотрено других условий).

конкретные вычислительные задачи могут иметь требование к валидации результатов или резервировании вычислений на случай сбоя на других узлах кластера (то есть повторения вычисления задачи для ее подтверждения). Например, возможен запуск задачи в трех экземплярах в целях последующего сопоставления результатов вычислений или же в целях обеспечения надежности вычислений в случае сбоя узла (или при других неполадках).

Конкретные задачи должны быть завершены не позднее срока τ. То есть, для соответствующей задачи

$$\sum_i R_{j.i.t} = 0, \quad \forall t > \tau \quad (7)$$

В гетерогенных кластерных системах, в отличие от гомогенных – объем ресурсов, доступных изначально на каждом узле, их соотношение (CPU, RAM, GPU), а также их отдельные характеристики (например, архитектура процессора) может отличаться. Поэтому, как уже было сказано ранее, возрастает сложность в планировании и распределении задач

Возможен сценарий, когда вычислительный кластер имеет возможности к расширению (в том числе динамическому, посредством подхода Infrastructure as a Code). Это условие может значительно усложнить задачу оптимизации.

В случае гетерогенного кластера из-за различия конфигураций узлов использование одного кванта ресурса неприменимо (так как на разных серверах разное соотношение характеристик сервера), поэтому имеет смысл выделять собственные параметры квантов для каждой характеристики. [22-26]

Для оптимизации вычислений на каждом конкретном узле возможно применение аппаратных возможностей для ускорения самого узла. Например, процессорная архитектура VLIW и ее поддержка в вычислительной задаче позволяет ускорить выполнение этой задачи.

Архитектура VLIW (Very Long Instruction Word), отличается от прочих архитектур процессоров широким машинным словом. Широкое машинное слово позволяет выполнять несколько инструкций за один такт процессора. Команды, которые необходимо выполнить в такт, укладываются в пакет, который в свою очередь вмещает в себя n-процессорных инструкции. Для оптимизации разработчику системы необходимо указать (на уровне ассемблерного кода) какие инструкции должны укладываться в пакет для достижения наибольшей производительности [26].

Например, конструкция {r1 = r0; r2 = add(r3,r4)} указывает процессору, что в пакет должны быть уложены две команды: первая – по копированию значения из r0 в r1, а вторая – по присвоению регистру r2 суммы значений из регистров r3 и r4. Исполнение команд из пакета будет произведено параллельно.

IV. Архитектура системы

Спроектированная и реализованная для эксперимента система представлена на Рис.3.

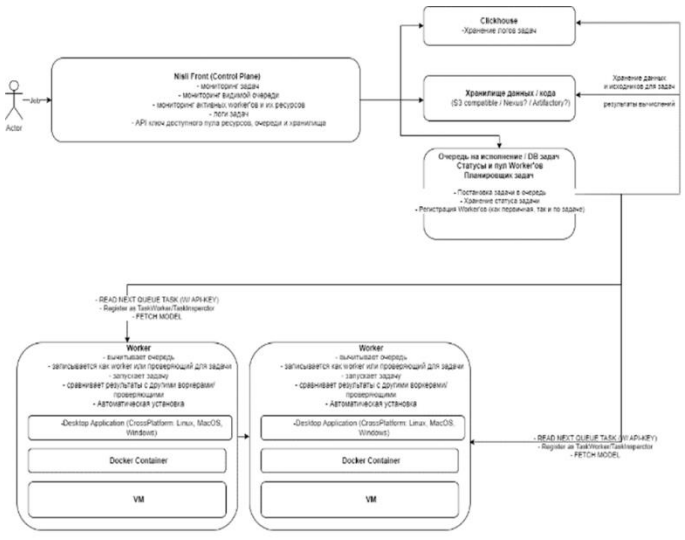


Рис. 3. Архитектура системы

Система обладает очередью задач, интерфейсом для отправки задач в очередь (в виде Docker-образов), а также алгоритмом получения и валидации результатов и резервирования вычислений на случай сбоя на других

узлах кластера (то есть повторения вычисления задачи для ее подтверждения на других узлах).

Система позволяет запускать как распределенные задачи, так и задачи с 1 исполнителем. Предусмотрена очередь задач и инструменты логирования и мониторинга состояния узлов кластера и хода выполнения задач.

Для целей эксперимента запускались задачи с выполнением задержки Sleep() на время соответствующее оценочному времени ожидания выполнения задачи.

V. Описание эксперимента

Для сравнения были выбраны и использованы модифицированный алгоритм BackFill и распределение задач по порядку их поступления на первые доступные для вычисления узлы.

Модификация алгоритма BlackFill заключается в упорядочивании задач не только по порядку поступления в очередь, но и упорядочивание в порядке убывания по требуем для запуска ресурсам. Таким образом сначала запускаются и планируются задачи, которые имеют наибольшие требования по ресурсам (в рамках выбранной задачи приоритет постановки задачи в очередь рассчитывается на основе взвешенного приоритета по порядку очереди и взвешенного приоритета по длительности задачи).

j – номер задачи в очереди

1. Приоритет задачи в очереди на исполнение по порядку задачи:

$$priority(j) = 1 - \frac{(j - \min(j))}{(\max(j) - \min(j))} \quad (8)$$

2. Приоритет задачи в очереди на исполнение по длительности задачи

$$priority(C_j) = \frac{C_j}{(\max(C_j))} \quad (9)$$

3. Общий приоритет задачи будет выражаться как

$$P(j) = 0{,}1 * priority(j) + 0{,}9 * priority(C_j) \quad (10)$$

Задача с наибольшим итоговым приоритетом запускается в первую очередь. При этом задача запускается на тех узлах, на которых она не увеличит общее суммарное время вычислений – то есть запускается на таком узле, чтобы минимизировать суммарное время выполнения всех задач.

Проведенный эксперимент на описанной кластерной системе с очередью из шести задач выглядит следующим образом:

Таблица 1. Имеется очередь из 6 задач с известной оценочной длительностью:

№ задачи	трудоемкость (GPU-часов)
1	2
2	4
3	4
4	1
5	4
6	3

Имеется кластер, состоящий из 5 квантов ресурсов (в реализованной модели квант – один узел).

Необходимо распределить задачи по квантам ресурсов так, чтобы минимизировать время, затраченное на вычисление всей очереди задач.

При использовании алгоритма распределения задач по порядку их поступления на первые доступные для вычисления узлы в начальный момент времени на ресурсе получены следующие результаты, представленные в Таблице 2.

Таблица 2. Планирование задач по порядку поступления

	t, время	1	2	3	4	5
i, ресурс						
1		1	1	6	6	6
2		2	2	2	2	
3		3	3	3	3	
4		4				
5		5	5	5	5	

Здесь в ячейках матрицы хранится номер исполняемой задачи.

Для модифицированного алгоритма BackFill получились следующие значения:

Таблица 3. Планирование задач модифицированным алгоритмом BackFill

	t, время	1	2	3	4
i, ресурс					
1		2	2	2	2
2		3	3	3	3
3		5	5	5	5
4		6	6	6	
5		1	1	4	

В поставленном эксперименте эффективнее оказался модифицированный алгоритм BackFill, при нем выполнение задач удалось уложить всего лишь в 4

кванта времени (часа).

Однако, применение резервирования (или механизма проверки) для конкретной задачи изменяло результаты алгоритма, показавшего минимальное время. Для более оптимального решения предлагается использование параметров резервирования планировщиком при составлении расписания.

VI. Выводы

В работе были проанализированы кластерные системы, их возможные алгоритмы планирования, а также их различия при той или иной организации кластера. Произведено сравнение алгоритмов планирования задач на кластере с резервированием задач и без. Полученные результаты показывают эффективность модифицированного алгоритма Blackfill для планирования задач в рамках системы, реализованной по предложенной архитектуре.

Литература

1. В.А. Новоженов, А.П. Ковтушенко. Среда обработки заданий, посылаемых на кластер из внешнего приложения для ускорения вычислений, Научно-образовательный журнал для студентов и преподавателей «StudNet», №4, с. 244–251, 2020.
2. Г.И. Радченко. 2012. Распределенные вычислительные системы.
3. Kwok Y.-K., Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv., vol. 31, № 4, pp. 406–471, 1999.
4. Singh K., Alam M., Sharma S.K. A survey of static scheduling algorithm for distributed computing system, Int. J. Comput. Appl. Foundation of Computer Science, vol. 129, № 2, pp. 25–30, 2015.
5. Alam M., Khan A., Varshney A.K. A review of dynamic scheduling algorithms for homogeneous and heterogeneous systems, Syst. Archit., pp. 73–83, 2018.
6. Chawla Y., Bhonsle M. A. Study on scheduling methods in cloud computing, Int. J. Emerg. Trends Technol. Comput. Sci., vol. 1, № 3, pp. 12–17, 2012.
7. Сухих А.В., Васяева Н.С. Исследование классификаций кластерных систем, Кибернетика и программирование, № 2. с. 20–27, 2016.
8. Карпенко А.П. Многопроцессорные системы (MIMD-системы). Вычислительные кластеры. МГТУ им. Н.Э.Баумана [Электронный ресурс] http://bigor.bmstu.ru/?cnt/?doc=Parallel/ch010104.mod/?cou=Parallel/base.cou
9. Салибекян С.М., Аминев Д.А., Семин В.Г., 2012, Результаты моделирования объектно-атрибутной вычислительной системы. Тезисы конференции «Инфо-2012».
10. Салибекян С.М., Панфилов П.Б., Хакимуллин Е.Р., Гетерогенные вычислительные системы на основе объектно-атрибутной модели программирования. https://www.hse.ru/pubs/share/direct/document/77867177
11. Сухорослов О.В., Организация вычислений в гетерогенных распределенных средах Известия ЮФУ. Технические науки. Раздел IV. Облачные вычисления, с. 115–130.
12. Кондратьев А.А., Скрытые проблемы распределенных вычислений, Electrical and data processing facilities and systems, № 3, т. 11, с. 61–65, 2015.
13. Чжо За. 2020. Использование виртуализации для увеличения эффективности вычислении. Диссертация на соискание ученой степени кандидата технических наук.
14. Грушин Д.А., Кузюрин Н.Н. О задаче эффективного управления вычислительной инфраструктурой, Труды ИСП РАН, т. 30, № 6, с. 123–142, 2018.
15. Кузюрин Н.Н., Грушин Д.А., Фомин С.А. Проблемы двумерной упаковки и задачи оптимизации в распределенных вычислительных системах. Труды Института системного программирования РАН. т. 26, № 1, с. 483–502, 2014.
16. Топорков, В.В. 2004. Модели распределенных вычислений. ФИЗМАТЛИТ, 320 с.
17. Полежаев П.Н. Современные алгоритмы планирования задач на кластерных вычислительных системах. Методология и проектирование экспертных систем, секция № 2 «Актуальные проблемы фундаментальной и прикладной математики», с. 87–95.
18. Коваленко В.Н., Коваленко Е.И., Корягин Д.А., Семячкин Д.А., 2007. Управление параллельными заданиями в гриде с неотчуждаемыми ресурсами. Удаленный ресурс Института прикладной математики им. М.В. Келдыша Российской академии наук.
19. Сальников, А.Н. 2006. Система разработки и поддержки исполнения параллельных программ. Диссертация на соискание ученой степени кандидата физико-математических наук.
20. Мазалов В.В., Никитина Н.Н. Оценка характеристик алгоритма Backfill при управлении потоком задач на вычислительном кластере. Вычислительные технологии, т. 17, № 5, 2012.
21. Шевченко С.В. Оптимизация распределенных вычислений в системах параллельной обработки данных, Интеллектуальные ит в управлении, № 3, с. 25–28, 2018.
22. Коваленко Н.С. Модель организации выполнения синхронных процессов в многопроцессорных системах и задачи их оптимизации. Математические структуры и моделирование т. 1, № 49, с. 80–88, 2019.
23. Е.Е. Перепелкин. 2013. Комплексное моделирование и оптимизация ускорительных систем на графическом процессоре (GPU). Диссертация на соискание ученой степени доктора физико-математических наук.
24. Макошенко Д.В. Математичекая модель времени вычислений для оптимизации программ, Известия вузов. Северо-Кавказский регион. Естественные науки, № 2, с. 10–13, 2009.
25. Пименов А.В., Федоров И.Р., Беззатеев С.В. Построение архитектуры туманных вычислений с использованием технологии блокчейн. Информационно-управляющие системы, № 5, с. 40–48, 2022.
26. Горин Д.И., Романова Т.Н. Метод оптимизации кода для процессора Qualcomm Hexagon, поддерживающий параллелизм на уровне команд и построенный по архитектуре VLIW (Very Long Instruction Word). Информационные технологии в науке, образовании и управлении, № 1, с. 105–116, 2021.

Mathematical Methods for Optimization of Distributed Computing in a Heterogeneous Environment

V. A. Novozhenov, T. N. Romanova

Abstract - In this article, we examined the possible organization and architecture of a heterogeneous cluster environment with varying architectures, performance, and characteristics for distributed computing. Mathematical methods for optimizing task queue computation time in such an environment were also considered. The goal was to identify the optimal task scheduling algorithm for the cluster under given conditions and constraints of the mathematical model. For this purpose, were designed and applied a new method of task queue computing time optimization. As a result, the efficiency of the BlackFill algorithm, modified to meet the task requirements, was demonstrated. Numerical simulation results show the effectiveness of the proposed methods compared to traditional approaches. The application of the developed methods significantly reduces computation time, improves resource utilization, and ensures stable system performance under changing workloads. The results may be useful for the development of high-performance computing systems and management algorithms in heterogeneous environments

Keywords **— distributed computing, mathematical optimization, task scheduling algorithms on a cluster, heterogeneous cluster, job queue, cluster computing.**

REFERENCES

1. V.A. Novozhenov, A.P. Kovtushenko. Creda obrabotki zadanij, posylaemyh na klaster iz vneshnego prilozhenija dlja uskorenija vychislenij, Nauchno-obrazovatel'nyj zhurnal dlja studentov i prepodavatelej «StudNet», #4, c. 244–251, 2020.
2. G.I. Radchenko. 2012. Raspredelennye vychislitel'nye sistemy.
3. Kwok Y.-K., Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv., vol. 31, # 4, pp. 406–471, 1999.
4. Singh K., Alam M., Sharma S.K. A survey of static scheduling algorithm for distributed computing system, Int. J. Comput. Appl. Foundation of Computer Science, vol. 129, # 2, pp. 25–30, 2015.
5. Alam M., Khan A., Varshney A.K. A review of dynamic scheduling algorithms for homogeneous and heterogeneous systems, Syst. Archit., pp. 73–83, 2018.
6. Chawla Y., Bhonsle M. A. Study on scheduling methods in cloud computing, Int. J. Emerg. Trends Technol. Comput. Sci., vol. 1, # 3, pp. 12–17, 2012.
7. Suhih A.V., Vasjaeva N.S. Issledovanie klassifikacij klasternyh sistem, Kibernetika i programmirovanie, # 2. c. 20–27, 2016.
8. Karpenko A.P. Mnogoprocessornye sistemy (MIMD-sistemy). Vychislitel'nye klastery. MGTU im. N.Je.Baumana [Jelektronnyj resurs] http://bigor.bmstu.ru/?cnt/?doc=Parallel/ch010104.mod/?cou=Parallel/base.cou
9. Salibekjan S.M., Aminev D.A., Semin V.G., 2012, Rezul'taty modelirovanija ob"ektno-atributnoj vychislitel'noj sistemy. Tezisy konferencii «Info-2012».
10.Salibekjan S.M., Panfilov P.B., Hakimullin E.R., Geterogennye vychislitel'nye sistemy na osnove ob"ektno-atributnoj modeli programmirovanija.
https://www.hse.ru/pubs/share/direct/document/77867177
11.Suhoroslov O.V., Organizacija vychislenij v geterogennyh raspredelennyh sredah Izvestija JuFU. Tehnicheskie nauki. Razdel IV. Oblachnye vychislenija, s. 115–130.
12.Kondrat'ev A.A., Skrytye problemy raspredelennyh vychislenij, Electrical and data processing facilities and systems, # 3, t. 11, s. 61–65, 2015.
13.Chzho Za. 2020. Ispol'zovanie virtualizacii dlja uvelichenija jeffektivnosti vychislenii. Dissertacija na soiskanie uchenoj stepeni kandidata tehnicheskih nauk.
14.Grushin D.A., Kuzjurin N.N. O zadache jeffektivnogo upravlenija vychislitel'noj infrastrukturoj, Trudy ISP RAN, t. 30, # 6, s. 123–142, 2018.
15.Kuzjurin N.N., Grushin D.A., Fomin C.A. Problemy dvumernoj upakovki i zadachi optimizacii v raspredelennyh vychislitel'nyh sistemah. Trudy Instituta sistemnogo programmirovanija RAN. t. 26, # 1, s. 483–502, 2014.
16.Toporkov, V.V. 2004. Modeli raspredelennyh vychislenij. FIZMATLIT, 320 c.
17.Polezhaev P.N. Sovremennye algoritmy planirovanija zadach na klasternyh vychislitel'nyh sistemah. Metodologija i proektirovanie jekspertnyh sistem, sekcija # 2 «Aktual'nye problemy fundamental'noj i prikladnoj matematiki», s. 87–95.
18.Kovalenko V.N., Kovalenko E.I., Korjagin D.A., Semjachkin D.A., 2007. Upravlenie parallel'nymi zadanijami v gride s neotchuzhdaemymi resursami. Udalennyj resurs Instituta prikladnoj matematiki im. M.V. Keldysha Rossijskoj akademii nauk.
19.Sal'nikov, A.N. 2006. Sistema razrabotki i podderzhki ispolnenija parallel'nyh programm. Dissertacija na soiskanie uchenoj stepeni kandidata fiziko-matematicheskih nauk.
20.Mazalov V.V., Nikitina N.N. Ocenka harakteristik algoritma Backfill pri upravlenii potokom zadach na vychislitel'nom klastere. Vychislitel'nye tehnologii, t. 17, # 5, 2012.
21.Shevchenko S.V. Optimizacija raspredelennyh vychislenij v sistemah parallel'noj obrabotki dannyh, Intellektual'nye it v upravlenii, # 3, s. 25–28, 2018.
22.Kovalenko N.S. Model' organizacii vypolnenija sinhronnyh processov v mnogoprocessornyh sistemah i zadachi ih optimizacii. Matematicheskie struktury i modelirovanie t. 1, # 49, s. 80–88, 2019.
23.E.E. Perepelkin. 2013. Kompleksnoe modelirovanie i optimizacija uskoritel'nyh sistem na graficheskom processore (GPU). Dissertacija na soiskanie uchenoj stepeni doktora fiziko-matematicheskih nauk.
24.Makoshenko D.V. Matematichekaja model' vremeni vychislenij dlja optimizacii programm, Izvestija vuzov. Severo-Kavkazskij region. Estestvennye nauki, # 2, s. 10–13, 2009.
25.Pimenov A.V., Fedorov I.R., Bezzateev S.V. Postroenie arhitektury tumannyh vychislenij s ispol'zovaniem tehnologii blokchejn. Informacionno-upravljajushhie sistemy, # 5, s. 40–48, 2022.
26.Gorin D.I., Romanova T.N. Metod optimizacii koda dlja processora Qualcomm Hexagon, podderzhivajushhiĭ parallelizm na urovne komand i postroennyĭ po arhitekture VLIW (Very Long Instruction Word). Informacionnye tehnologii v nauke, obrazovanii i upravlenii, # 1, s. 105–116, 2021.