

Правительство Российской Федерации
Федеральное государственное автономное учреждение
высшего профессионального образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет Компьютерных наук
Магистерская программа «Науки о Данных»
Департамент больших данных и информационного поиска

КУРСОВАЯ РАБОТА МАГИСТРА
Реализация и сравнение алгоритмов планирования DAG
в распределенных системах

Выполнил студент группы АИД
Александров Роман Игоревич

Научный руководитель:
кандидат технических наук, доцент
Сухорослов Олег Викторович

Оглавление

Введение	2
Постановка задачи	3
1. Описание стандартных алгоритмов планирования вычисления композитных приложений в распределенной вычислительной среде	5
1.1 Алгоритм HEFT	5
1.2 Алгоритм DLS	7
2. Алгоритм генетического поиска для решения задачи планирования вычисления композитных приложений в распределенной вычислительной среде	9
3. Эксперименты	11
Заключение	12
Литература	13

Введение

В работе представлена задача планирования вычисления DAG в распределенных системах вычисления. Согласно статье [1], задача поиска оптимального расписания является NP-полной. Поэтому для наиболее оптимального и при этом быстрого решения данной задачи было придумано много различных приближенных методов, большое количество которых описано в статье-обзоре [2]. В главе 1 подробно рассмотрена постановка задачи. В главе 2 описаны классические эвристические алгоритмы планирования. В главе 3 представлено применение генетического алгоритма к данной задаче. В главе 4 приведены результаты сравнения алгоритмов затронутых в данной работе.

Постановка задачи

Для постановки задачи планирования необходимо ввести две модели - модель вычислительной среды и модель вычислительного приложения, которая как раз описывается с помощью DAG. Кроме того, неявно предполагается наличие управляющей системы или брокера выполнения, то есть программного комплекса, непосредственно запускающего задачи. Однако с точки зрения задачи планирования достаточно предположить что накладные расходы управляющей системы пренебрежимо малы.

Вычислительная среда состоит из ресурсов R_i ($1 \leq i \leq M$) с различной производительностью. Между каждой парой ресурсов есть сетевое соединение с заданной пропускной способностью. При этом эффектами взаимодействия потоков данных обычно пренебрегают. Возможен учет дополнительных факторов и характеристик: объем доступной памяти, количество ядер процессора или вероятность отказа ресурса. В дальнейшем будем предполагать что вероятность отказа за время работы отдельного приложения крайне мала и может игнорироваться.

Композитные приложения (КП) удобно описывать на языке потоков работ (workflow). Поток работ обычно представляется в виде направленного ациклического графа (DAG). Вершины графа T_a ($1 \leq a \leq N$) отражают выполнение отдельной задачи в КП, а ребра C_{ab} (количество ребер $|C_{ab}| = E$ задают зависимость или связь между задачами. Если присутствует связь C_{ab} значит задачу T_a называют (непосредственным) родителем задачи T_b , а задачу T_b дочерней T_a . Дочерние задачи не могут быть начаты пока не завершились все их родители. Для всех рассматри-

ваемых далее методов планирования необходима возможность оценить время выполнения задач на любом ресурсе вычислительной среды. Обозначим эту оценку как $EET(T_a, R_i)$ (EET = Estimated Execution Time) и будем считать её известной.

Для каждой связи C_{ab} определен объем данных c_{ab} передаваемых от родительской задачи к дочерней. Аналогично времени выполнения EET , обозначим время передачи данных от задачи T_a к T_b как $ECOMT(C_{ab}, R_i, R_j)$ ($ECOMT$ = Estimated Communication Time). Есть различные подходы к оценке $ECOMT$, но чаще всего используют простую линейную модель

$$ECOMT(c_{ab}, R_i, R_j) = t_{con} + \frac{c_{ab}}{L_{ij}}$$

,где t_{con} латентность соединения, L_{ij} - пропускная способность сети между ресурсами R_i, R_j .

Отметим, что при практической реализации планирования вопрос вычисления EET и оценки $ECOMT_{ab}$ является сложной независимой задачей.

В рамках введенной модели мы можем описать время завершения каждой задачи и всего приложения в целом. Пусть задача T_a выполняется на ресурсе R_j , а ее родительские задачи T_b - на ресурсах R_{j_b} . Тогда:

$$ECT(T_a) = EET(T_a, R_i) + \max_{b \in \text{родительские задачи}} (ECOMT(c_{ab}, R_i, R_{j_b}) + ECT(T_b, R_{j_b}))$$

Оценка времени завершения последней задачи в композитном приложении $ECT_{\max} = \max_a (ECT(T_a))$ определяет время выполнения всего композитного приложения.

В данной работе будут рассмотрены алгоритма планирования с наилучшим качеством, то есть минимизации времени выполнения.

Описание стандартных алгоритмов планирования вычисления композитных приложений в распределенной вычислительной среде

В данной главе рассматриваются классические методы приоритетного планирования. Данные алгоритмы стараются учитывать полную структуру композитного приложения. Их основной идеей является то, что лучшие ресурсы должны быть выделены под задачи, которые оказывают наибольшее влияние на общее время выполнения композитного приложения. Классическим примером таких алгоритмов является HEFT (Heterogeneous Earliest Finish Time, [4]). Также можно выделить алгоритм DLS (Dynamic Levels Scheduler, [3]), который также является удачным развитием данного подхода.

Алгоритм HEFT

Алгоритм HEFT относительно простой и при этом экспериментально эффективный. Его часто берут за эталонный при сравнении по эффективности и скорости составления расписания с новыми алгоритмами. Работу данного алгоритма можно разбить на два этапа — упорядочивание задач и размещение их на ресурсы.

На первом шаге производится ранжирование задач по следующему признаку:

$$rank(T_a) = \overline{EET}(T_a) = \max_{T_b \in \text{дочерние задачи } T_a} (\overline{ECOMT}(c_{ab}) + rank(T_b))$$

Здесь $\overline{EET}(T_a)$ — средняя оценка времени выполнения задачи T_a по всем ресурсам, $\overline{ECOMT}(c_{ab})$ — среднее время передачи объема данных c_{ab} между всевозможными парами ресурсов.

Затем все задачи в приложении планируются в неубывающем порядке значения функции $rank(T_a)$. Нельзя не отметить важную особенность функции ранжирования - родительская задача всегда имеет более высокий приоритет чем ее дочерние задачи. Благодаря такому свойству на момент планирования каждой задачи известно, где будут выполнены все её родительские задачи. Выбор ресурса на которой будет выполнена задача основан на минимизации времени завершения ECT . Таким образом, задачи с большим значением $rank$ получают более производительные ресурсы. Для повышения эффективности используется планирование со вставками (с резервированием временных окон). Благодаря этому низкоприоритетная задача может выполняться на ресурсе до высокоприоритетной, пока высокоприоритетная задача ожидает выполнения родительских задач.

Итого, алгоритм *HEFT* можно представить в следующем виде:

for all T_a **do**

$$rank(T_a) \leftarrow \overline{EET}(T_a) = \max_{T_b \in \text{дочерние задачи } T_a} (\overline{ECOMT}(c_{ab}) + rank(T_b))$$

$OrderedTasks \leftarrow$ задачи отсортированные по убыванию $rank_{T_a}$

for all $T \in OrderedTasks$ **do**

$$R \leftarrow \arg \min_{R_i} ECT(T, R_i)$$

Запланировать выполнение задачи T на R

Алгоритм DLS

Алгоритм DLS тоже основывается на идеи потенциалов. Однако, в алгоритме DLS потенциал считается для пары — ресурс, задача. В оригинальной статье эти потенциалы называются Dynamic Levels, отсюда и название алгоритма. На первом шаге алгоритма для каждой задачи считается SL (SL = Static Level):

$$SL(T_a) = \overline{EET}(T_a) + \max_{T_b \in \text{дочерние задачи } T_a} (SL(T_b))$$

Уже на основе посчитанного статического потенциала высчитывается Dinamic Level для каждой пары ресурс/задача:

$$DL(T_a, R_i, Timetable) = SL(T_a) + D(T_a, R_i) - EEST(T_a, R_i, Timetable)$$

Здесь SL — Static Level. $D(T_a, R_i)$ — разница между временем выполнения задачи T_a на ресурсе R_i и на усредненном ресурсе. $EEST(T_a, R_i, Timetable)$ (EEST = Earliest Execution Start Time) наименьшее время, когда задача T_a , может начать выполняться на ресурсе R_i . Чтобы посчитать $EEST(T_a, R_i, Timetable)$ нужно знать расписание на конкретном ресурсе R_i и знать время конца выполнения все родительских задач, отсюда и зависимость от $Timetable$.

Отличие данного алгоритма от HEFT в том, что потенциалы высчитываются по мере планирования задач на ресурсы. Об этом и говорит название которое выбрали для потенциала — Dinamic Level. Изначально DL посчитан для всех задач у которых нет зависимостей, они уже готовы к выполнению. Далее на каждом шагу выбирается пара задача/ресурс которая имеет максимальный DL . Данная задача планируется к выполнению на этом ресурсе на минимально возможное время. Потом

DL вычисляется для всех задач у которых стало известно время конца выполнения родительских задач.

Весь алгоритм выглядит так:

for all T_a **do**

$$SL(T_a) \leftarrow \overline{EET}(T_a) + \max_{T_b \in \text{дочерние задачи } T_a} (SL(T_b))$$

$ReadyToSchedule \leftarrow$ задачи готовые к выполнению

for all $T_a \in ReadyToSchedule$ **do**

for all R_i **do**

$$DL(T_a, R_i) \leftarrow SL(T_a) + D(T_a, R_i) - EEST(T_a, R_i)$$

while $ReadyToSchedule \neq \emptyset$ **do**

$$R, T \leftarrow \arg \max_{T_a \in ReadyToSchedule, R_i} (DL(T_a, R_i))$$

Запланировать выполнение задачи T на R

$$ReadyToSchedule \leftarrow ReadyToSchedule - \{R\}$$

$ReadyChilds \leftarrow$ дети задачи R готовые к выполнению

$$ReadyToSchedule \leftarrow ReadyToSchedule \cup ReadyChilds$$

for all $T_a \in ReadyChilds$ **do**

for all R_i **do**

$$DL(T_a, R_i) \leftarrow SL(T_a) + D(T_a, R_i) - EEST(T_a, R_i)$$

Алгоритм генетического поиска для решения задачи планирования вычисления композитных приложений в распределенной вычислительной среде

Генетический алгоритм поиска является универсальной концепцией для приближенного решения трудновычислимых задач. Его приложение к задаче планирования описано более подробно здесь [5]. Здесь же будет краткое описание работы алгоритма.

Общий вид генетического алгоритма одинаков для всех приложений:

1. *Инициализация начального поколения*

while Критерий остановки не выполнен **do**

2. *Кроссовер*

3. *Мутация*

4. *Оценка поколения*

5. *Выбор в новое поколение*

Основную роль в генетическом алгоритме играет элемент поколения — хромосома. В рамках задачи планирования хромосома выглядит как пара строк. Одна строка, будем называть его *mat* задает соответствие между задачей и ресурсом. Вторая строка — *ss* задает последовательность задач в которой они будут запланированы. Важно что *ss* является некоторой топологической сортировкой ациклического графа композитного приложения.

В начальное поколение генетического алгоритма нужно добавлять совсем случайные хромосомы и хромосомы которые соответствуют реше-

ниям построенным другими различными эвристиками, например с помощью DLS или HEFT. Тогда генетический алгоритмы может улучшить какое-то из решений с помощью случайных процессов которые происходят с хромосомами.

Кроссовер является одним из наиболее важных частей алгоритма. Он помогает обмениваться информацией между двумя хромосомами — создавать новые хромосомы отличные от родительских. Кроссовер между двумя *mat* строками просто режет их в одинаковых частях и меняет отрезанные части. Кроссовер между двумя *ss* строками также режет их в одинаковых местах и потом в отрезанной части строки задачи переставляются в порядке задаваемом строкой с которой происходит кроссовер.

Мутация позволяет создавать хромосомы с совсем новым решением. Процесс мутации для *mat* строки очень простой — выбирается случайная задача и ей в соответствии задается случайный ресурс. Мутация же *ss* строки происходит чуть сложнее. Выбирается случайная задача и переносится в случайное место такое, чтобы новая строка также будет являться топологической сортировкой.

Оценка хромосомы осуществляется с помощью симуляции или того расписания которое было построено (оно дает приблизительное время выполнения композитного приложения так как в составлении расписания не учитывается общее пользование сети). После того как все элементы поколения получили оценку, они сортируются на основе этой оценки и потом в зависимости от места в порядке сортировки им присуждается вероятность с которой они будут выбраны. Выбор производится с повторением, то есть некоторые хромосомы могут дублироваться, однако это будут скорее всего хромосомы с хорошим расписанием. Для i -го элемента будет такая вероятность:

$$P_i = R^{N-i-1}(1 - R)/(1 - R^N)$$

N — размер популяции, R - параметр алгоритма.

Эксперименты

В рамках этой работы были реализованы алгоритмы DLS и генетический алгоритм. Реализации алгоритмов добавлены в открытый сборник алгоритмов, который находится на сервисе [github](#). Данный репозиторий содержит алгоритмы, тестовые приложения и различные конфигурации вычислительных систем и самое важное там находится удобная обертка над программой моделирования вычисления приложения в распределенной системе.

Заключение

Генетический алгоритм достаточно сложен в настройке, однако результаты которые можно с помощью него достигнуть могут быть намного лучше простых эвристических подходов. DLS по сравнению с HEFT немного лучше адаптируется к специфике распределенной системы и композитного приложения. Однако и асимптотика и скорость работы DLS больше чем у HEFT.

Литература

1. R.L. Graham, E.L. Lawler, J.K. Lenstra и A.H.G. Rinnooy Kan. «Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey». В: Discrete Optimization 17 II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver. Под ред. E.L. Johnson P.L. Hammer и B.H. Korte. Т. 5. Annals of Discrete Mathematics. Elsevier, 1979, с. 287—326. doi: [http://dx.doi.org/10.1016/S0167-5060\(08\)70356-X](http://dx.doi.org/10.1016/S0167-5060(08)70356-X). url: <http://www.sciencedirect.com/science/article/pii/S016750600870356X>.
2. Назаренко Алексей, такой-то год такое-то издание.
3. G. C. Sih и E. A. Lee. «Dynamic-level scheduling for heterogeneous processor networks». В: Parallel and Distributed Processing, 1990. Proceedings of the Second IEEE Symposium on. Дек. 1990, с. 42—49. doi: [10.1109/SPDP.1990.143505](http://dx.doi.org/10.1109/SPDP.1990.143505)
4. H. Topcuoglu, S. Hariri и Min-You Wu. «Task scheduling algorithms for heterogeneous processors». В: Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth. 1999, с. 3—14. doi: [10.1109/HCW.1999.765092](http://dx.doi.org/10.1109/HCW.1999.765092)
5. Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhury и Anthony A. Maciejewski. «Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic- Algorithm-Based Approach». В: J. Parallel Distrib. Comput. 47.1 (нояб. 1997), с. 8—22. issn: 0743-7315. url: <http://dx.doi.org/10.1006/jpdc.1997.1392>