Адаптивный метод управления потоком решения изолированных заданий в параллельной вычислительной среде[1]

М. А. Паньшенсков
Санкт-Петербургский государственный университет
mpansh@gmail.com

Модель решения изолированных заданий формулируется в расширенных терминах потока управления. Идентифицируются узкие места потока управления: пересылка данных по коммуникационным каналам, вычисление заданий на узлах. Для каждого из случаев предлагается алгоритм управления потоком решения заданий. Дается оценка времени исполнения предлагаемого алгоритма в зависимости от условий гетерогенной среды, в которой производятся вычисления.
Ключевые слова: распределенные вычисления, распределение заданий, адаптивные методы, коммуникация.

1. Введение

Поиск оптимального решения набора изолированных заданий является крайне интересной и актуальной задачей в сфере параллельных алгоритмов. В этой статье решение параллельных заданий рассматривается в виде потока параллельных действий, представимого в виде ориентированного графа. Вершины графа символизируют вычислительные узлы, а дуги — каналы. И дуги и вершины имеют некоторую стоимость, заранее неизвестную. Задачей является выбрать такой набор путей, который при соблюдении определенных ограничений является минимальным. Получается, что в терминах теории графов можно переформулировать задачу о распределении изолированных заданий. К сожалению, задача не сводится к задаче поиска кратчайшего пути в этом графе, или к другой известной автору задаче полиномиальной сложности. Но здесь возможно поставить другую задачу — минимизации неэффективного простоя ресурсов. Последняя на практике будет двойственна исходной. Ее решение должно удовлетворять следующим условиям:

- обеспечивать планирование, близкое к оптимальному,

[1]©М. А. Паньшенсков, 2008

- учитывать изменение параметров системы и корректировать распределение заданий.

Параметры системы могут довольно часто изменяться в условиях, когда вычислительная среда не отдана всецело для проведения параллельных вычислений. В этом случае может заметно меняться производительность вычислительных узлов. Например, пропускная способность каналов зачастую меняется в следствии внешних событий. Для того чтобы учитывать изменения состояния системы, необходимо постоянно анализировать значения текущих величин. Метод предлагаемый в статье использует принципы стохастической оптимизации. Основной причиной его использования является фундаментальная неопределенность состояния системы и необходимость быстрой адаптации к изменяемым условиям [1].

Рабочими примерами для запуска изолированных заданий являются:

- задачи визуализации (визуализация фрактала Жюлиа) [2],
- задачи математического моделирования (метод Монте-Карло для поверхностных реакций) [**?**],
- задачи поиска информации (SETI@Home — поиск сигналов внеземных цивилизаций [4]).

Статья организована следующим образом. Секция 2 описывает модель решения изолированных заданий в параллельной среде и содержит постановку задачи оптимального управления потоком решения изолированных заданий. В секции 3 разделяются состояния системы решения изолированных заданий с учетом загруженности различных частей. Для разных состояний системы предлагаются особые алгоритмы распределения заданий по узлам. Даются оценки скорости работ алгоритмов. В секции 4 производится классификация существующих алгоритмов распределения изолированных заданий в параллельной среде и выделяется место рассматриваемого алгоритма. В последней секции 5 подводятся итоги статьи и делается заключение.

2. Управление потоком решения заданий

Формализуется задача управления потоком решения изолированных заданий в параллельной среде.

Переменные. Для того, чтобы однозначно определить распределение заданий, достаточно для каждого задания назначить вычислительный узел и скорость передачи входных данных этого задания. Ниже представлены переменные определяющие ход решения исходного набора заданий.

Множество задач пусть проиндексировано. Для удобства ассоциируем задачу с ее натуральным номером (см. рис.1).

$$ID = \{n | n \in N\}$$

Определим переменную функцию, которая задает распределение задач по вычислительным узлам. Каждый узел определяется сложным индексом из множества K^*, представляющим собой набор конечных последовательностей натуральных чисел.

$$where : ID \to K^* \cup \{\emptyset\}$$

Случай $where(i) = \emptyset$ означает, что для задания i еще не выполнено распределения. В случае, когда распределение происходит динамически, количество нераспределенных заданий снижается по мере выполнения всех заданий.

Переменную функцию, которая задает скорость передачи входных данных с исходного узла на вычислительный узел i, обозначим:

$$how : ID \to R_+.$$

Параметры. Функция T номинального времени расчета задания:

$$T : ID \to R_+,$$

возвращает для каждого задания время его расчета на некотором (возможно, виртуальном) вычислительном узле. T считается известной априорно. На практике для каждой конкретной задачи T можно получить эмпирически, отправив предварительно подмножество заданий на однородную систему (получив значение T для

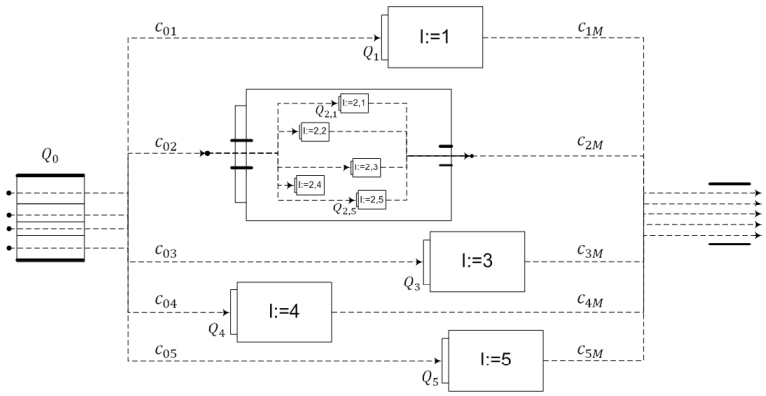

Рис. 1: Диаграмма потока данных.

подмножества всех заданий), сделать аппроксимацию на все множество.

Формулировка задачи в терминах потока управления. Задача — максимизировать эффективную загрузку доступных ресурсов.

Узел с которого осуществляется отправка заданий обозначим 0. Принимающий узел — M. Очередь на обработку заданий Q_k на узле k является вектором из $s+1$ элементов:

$$Q_k = (q_k^0, q_k^1, ..., q_k^s), \text{ где } q_k^i \in ID.$$

Очередь на отправку заданий Q_0 имеет такую же семантику как и Q_k. Переменную отражающую скорость передачи данных можно определить для исходящих каналов c_{0i} и для входящих каналов c_{0i} (см. рис. 1).

Ограничения Перечислим естественные ограничения, накладываемые на задачу.

Коммуникационный канал. Реальные коммуникационные каналы имеют ограниченную пропускную способность. В данной задаче есть два вида каналов: исходящие и входящие. Обозначим C_{0i} — ограничение скорости для входящих каналов c_{0i}, а C_{iM} — ограничение скорости исходящих каналов c_{iM}: $0 \le c_{0i} \le C_{0i},\ 0 \le c_{iM} \le C_{iM}$.

При одновременной передаче нескольких пакетов по одному каналу скорости передачи разных пакетов коррелируют между собой. Но также и при передаче пакетов по разным каналам, если в физическом представлении (соединительные провода, маршрутизаторы) имеют общую часть, скорости передачи c_{0i} и c_{iM} коррелируют между собой. В данной задаче все исходящие и входящие каналы имеют общую вершину в сетевом представлении. Поэтому есть общее ограничение на скорости передачи данных по входящим и исходящим каналам:

$$\begin{cases} \sum_i c_{0i} \leq C_i, \\ \sum_i c_{iM} \leq C_M. \end{cases}$$

Так же может существовать ограничение на входящие/исходящие каналы отдельно взятой подсистемы. Например, для системы $\{i = (2,k)|k \in N\}$ на рис. 1 может существовать дополнительное ограничение:

$$\sum_m c_{0(2,m)} \leq C_{(2,m)}.$$

На практике скорость передачи данных в подсистеме (локальной сети) выше скорости канала связи с входящим/исходящим узлом. Следовательно, в действительности ограничение на каналы подсистемы не является узким местом и потому не стоит его учитывать.

3. Общий алгоритм управления

Перед тем, как алгоритм управления начнет работу, необходимо иметь значения параметров: пропускная способность каналов и производительность вычислительных узлов. Если значение этих параметров не заданы априори, то их оценки можно получить в процессе работы.

В потоке управления на рис. 1 есть 3 потенциально узких места: исходящий канал передачи заданий, решение заданий на вычислительных узлах и входящий канал получения результатов заданий. В данном аспекте узкое место означает, что если моментально увеличить мощность соответствующего ресурса (входящий/исходящий коммуникационных канал или вычислительный узел), то сразу же уменьшится общее время решения задачи. Входящий канал довольно редко может быть узким местом, так как на практике довольно редко результат решения задания на порядки больше входных

данных, в то время как величины пропускных способностей входящего/исходящего каналов зачастую имеют одинаковые порядки. Кроме того, если даже исходящий канал является узким местом, это проявится лишь на последних шагах решения заданий и будет учтено переборным алгоритмом планирования для последних шагов. Для других двух случаев предложены алгоритмы.

Алгоритмы запускаются каждый раз перед отправкой заданий. Ниже описаны алгоритмы для случаев, когда узким местом является отправка заданий и для случая, когда узким местом является обработка заданий на вычислительных узлах.

- *Узкое место — обработка заданий на вычислительных узлах.* Это значит, что если отправить все задания на все вычислительные узлы, то процесс загрузки завершится до того, как первый из вычислительных узлов завершит обработку:

  $$\forall i : now + TimeToSendToAll <= momentOfAvailability_{K^*}.$$

  Алгоритм для этого случая — полная загрузка ресурсов.

- *Узкое место — отправка заданий.* Это означает, что существует такой вычислительный узел $i \in K^*$, для которого от рассматриваемого момента $MomentNow$ до времени, когда узел i освобождается, $momentOfAvailability$, проходит больше времени $TimeToAll$ чем необходимо на доставку входных данных всем узлам:

  $$\exists i : MomentNow + TimeToAll > MomentOfAvailability(K^*).$$

  Алгоритм для этого случая — максимизация эффективной загрузки ресурсов.

Алгоритм полной загрузки ресурсов состоит из четырех частей: обновление параметров, выбор задания для отправки, распределение задания на первый свободный узел, добавление задания в очереди:

```
Complete_Resourse_Load_Scheduling(ID)
begin
  Env ← estimateNewEnivronmentState()
  sort(ID*, orderDescendingBy(T))
  //Убираем из рассмотрения самые легковесные задачи
  ID^(active) ← range(ID*, λ * M + 1, I)

  t^(est) ← estimateTimeOfCompletion(ID^(active), Env)
  if (2 * t^(now) < t^(est))//Не прошло половины времени
      level ← 2t^(now) / t^(est) * max_{i∈ID^(active)} T(i)
      job ← max_{T(i)<level}(i) OR min_{T(i)≥level}(i)
  else //Прошла половина отведенного времени
      job ← max_{i∈ID^(active)}(i)
  end if
  where(job) ← chooseFirstSpareNode(job, Q, Env)
  add(Q_{where(job)}, job)
  end
```

Операции с префиксом *estimate* выполняют оценивание соответствующих параметров. Оценивание производится при помощи методов стохастической оптимизации, которые подробно описываются в [1,5]. Операция *chooseFirstSpareNode* выполняет выбор первого свободного узла, получая очереди исполняемых заданий в качестве входного параметра.

Алгоритм эффективной загрузки ресурсов состоит из тех же четырех частей: обновление параметров, выбор задания для отправки, распределение задания на первый свободный узел, добавление задания в очереди:

```
Effective_Resourse_Load_Scheduling(ID)
begin
  Env ← estimateNewEnivronmentState()
  sort(ID*, orderDescendingBy(T))
  //Убираем из рассмотрения самые легковесные задачи
  ID^(active) ← range(ID*, λ * M + 1, I)

  t^(est) ← estimateTimeOfCompletion(ID^(active), Env)
  if (2 * t^(now) < t^(est))//Не прошло половины времени
      level ← (2t^(now)) / t^(est) * max_{i∈ID^(active)} T(i)
      job ← max_{T(i)<level}(i) OR min_{T(i)≥level}(i)
  else //Прошла половина отведенного времени
      job ← max_{i∈ID^(active)}(i)
  end if
  where(job) ← chooseFirstSpareNode(job, Q, Q_0, Env)
  how(job) ← estimateTransferSpeed(where(job))
  add(Q_where(job), job)
  add(Q_0where(job), job)
  end
```

Этот алгоритм отличается от предыдущего только тем, что здесь учитывается время передачи входных данных (так как передача не может быть всецело осуществлена во время работы выч.узлов). Операция
chooseFirstSpareNode выполняет выбор первого свободного узла, получая очереди исполняемых заданий и очередь отправляемых заданий в качестве входных параметров.

4. Оценка времени работы

Первые два из четырех этапов у описанных алгоритмов совпадают. Оценим время выполнения операций поэтапно.

Время выполнения первого этапа: O(|K|) + O(|ID|*log|ID|),
второго — O(|K|)+O(|ID|),
третьего — O(|K|) для *Complete_ Resourse_ Load_ Scheduling*,
O(|K|*|K|) для алгоритма *Efficient_ Resourse_ Load_ Scheduling*.

Время четвертого этапа для обоих алгоритмов равно O(1).

Итоговые оценки времени выполнения алгоритмов:

$$T_{CRLS} = \mathcal{O}(|ID| * log|ID|) + \mathcal{O}(|K|) \quad (1)$$

$$T_{ERLS} = \mathcal{O}(|ID| * log|ID|) + \mathcal{O}(|K| * |K|) \quad (2)$$

Дополнение. Если окончание решения набора заданий не проходит оптимально, можно скорректировать план при помощи техники избыточного решения.

5. Обзор литературы

Результаты поиска оптимальных вычислений изолированных заданий только недавно начали появляться. Ранее использовались лишь сравнительно простые стратегии [6]. Позже были описаны различные проблемы в области адаптивных алгоритмов и сразу несколько авторов взялись за разработку этой проблематики. Некоторые авторы предлагают адаптироваться к неустойчивым состояниям системы путем изменения "относительной силы" на каждом шаге, когда неустойчивость системы больше некоторого порогового значения. По мнению авторов этот подход не всегда имеет оптимальное применение. Например, если система в среднем работает одинаково, то нет причин на каждом шаге переназначать задание другой машине. Однако авторы [6] утверждают, что их балансировка работает качественно.

В среде параллельных вычислений планирование всегда присутствует. Однако сами алгоритмы планирования не отличаются значительно и поддаются классификации. Если точная информация о трудоемкости заданий и производительности вычислительных узлов известна перед запуском, то *статическое планирование* [7] может быть выполнено. *Статическое планирование* является подклассом алгоритмов планирования, когда распределение заданий между процессорами определяется до запуска. Этот подход хорош, когда исходные параметры системы и отправляемых заданий не меняются в процессе выполнения. В противному случае используется *динамическое планированием.*

Динамическое планирование это подкласс алгоритмов планирования когда решения о распределении заданий между процессорами производятся во время исполнения, используя обновляемую информацию о выполняемых / выполненных заданиях. Такое планирование является более устойчивым к изменению производительности вычислительного узла. Динамические алгоритмы планирования могут быть разделены на распределенные и централизованные. Алгоритм *отдать задания* и алгоритм *украсть задания* являются единственными подклассами среди распределенных динамических алгоритмов [8, 9]. В данной статье рассматривается динамический централизованный планировщик. Более подробно о таксономии алгоритмов планирования параллельных вычислений можно прочитать в [10].

Если число целевых систем велико, то их можно уменьшить, введя иерархическую топологию [9]. В рассматриваемой модели такая топология применима без изменений, если считать, что параметры подсети доступны для наблюдения извне. Если такие параметры скрыты, а система является гомогенной, то необходимо немного изменить алгоритм отправки заданий, добавив новую переменную — количество вычислительных узлов в подсистеме.

6. Заключение

Обе временных оценки алгоритмов, как алгоритма полной загрузки вычислительных ресурсов (1), так и алгоритма эффективной загрузки вычислительных ресурсов (2), дают понять, что при хороших линейных коэффициентах задачу распределения заданий можно свободно масштабировать как по количеству заданий $|\mathrm{ID}|$ так и по числу вычислительных узлов $|\mathrm{K}|$.

Адаптивный метод распределения заданий в параллельной вычислительной среде будет использован при разработке нового сервиса-брокера для отправки заданий в систему распределенных вычислений ПУНК СПбГУ. Кроме этого метод может быть использован при создании прототипа Грида основанного на персональных вычислительных устройствах соединенных посредством Интернета. В последнем случае отношение *общего времени передачи данных* к *общему времени вычислений* может быть велико, что делает акту-

альной проблему оптимального распределения заданий в условиях ограниченности каналов.

Список литературы

[1] *Граничин О. Н., Поляк Б. Т.* Рандомизированные алгоритмы оценивания и оптимизации при почти произвольных помехах. — М. Наука. 2003. 291 с.

[2] *Estkover C. A. (Editor)* Chaos And Fractals: A Computer Graphical Journey. — Elsevier Science. 1998. P. 468.

[3] *Nakano A.* High Performance Computing And Simulations (Spring '07). Http://Cacs.Usc.Edu/Education/Cs653.Html

[4] *Anderson, Cobb etc.* SETI@Home: an experiment in public-resource computing // Comm. of the ACM. Vol. 45. No 11. 2002. PP. 56–61.

[5] *Kushner H., Yin G.* Stochastic Approximation and Recursive Algorithms and Applications, 2nd edition. — Springer. 2003.

[6] *Cermele M., Colajanni M, Necci G.* Dynamic load balancing of distributed SPMD computations with explicit message-passing // IEEE Workshop on Heterogeneous Computing. 1997. PP. 2–16.

[7] *Ichikawa S., Yamashita S.* Static load balancing of parallel PDE solver for distributed computing environment // ISCA 13th Int'l Conf. Parallel and Distributed Computing Systems. PDCS-2000. 2000. PP. 399–405.

[8] *Blumofe R., Leiserson C.* Scheduling multithreaded computations by work stealing // 35th Annual IEEE Conf. on Foundations of Computer Science. FOCS'94. Santa Fe, New Mexico.

[9] *Neill D., Wierman A.* On the benefits of work stealing in shared memory multiprocessors.
Отчет *http://www.cs.cmu.edu/ acw/15740/paper.pdf*

[10] *Casavant T., Kuhl J.* A taxonomy of scheduling in general-purpose distributed computing systems // IEEE Trans. on Software Engineering. Vol. 14. No 2. 1988. PP. 141–154.