КОМП'ЮТЕРНІ ЗАСОБИ, МЕРЕЖІ ТА СИСТЕМИ

V.V. Zosimov, O.S. Bulgakova, A.V. Tyshchenko

WAYS AND METHODS OF INCREASING COMPLEX CALCULATIONS PERFORMANCE

This article describes the ways and methods of improving the performance of complex calculations by optimizing the interaction between the operating system and computer algorithms.
Key words: operating system, computer algorithm, the universal task schedulers.

У даній статті описані можливі шляхи і методи підвищення швидкодії складних обчислень за рахунок оптимізації взаємодії операційної системи і обчислювальних алгоритмів.
Ключові слова: операційна система, ком'ютерний алгоритм, універсальні планувальники задач.

В данной статье описаны возможные пути и методы повышения быстродействия сложных вычислений за счет оптимизации взаимодействия операционной системы и вычислительных алгоритмов.
Ключевые слова: операционная система, компьютерный алгоритм, универсальные планировщики задач.

УДК 004.832.3

В.В. ЗОСИМОВ, А.С. БУЛГАКОВА, А.В. ТИЩЕНКО

ПУТИ И МЕТОДЫ ПОВЫШЕНИЯ БЫСТРОДЕЙСТВИЯ СЛОЖНЫХ ВЫЧИСЛЕНИЙ

Введение. В настоящее время во многих сферах человеческой деятельности применяются ресурсоемкие научные расчеты для анализа данных и прогнозирования явлений и событий различного характера. Для более эффективного ведения исследований создаются международные исследовательские группы, в которых задействованы специалисты из различных областей знаний. Такое междисциплинарное взаимодействие позволяет участникам группы взглянуть на проблему под разными углами и помочь друг другу увидеть пути решения, не очевидные каждому отдельному специалисту. Не смотря на эффективность сотрудничества таких групп, в их работе могут возникнуть перебои связанные, в том числе, и с низкой скоростью обработки полученных в ходе исследования данных. В таких группах каждый ученый занимается своей частью общей проблемы. Сложность решаемых каждым из членов группы задач может быть различна и зачастую одним членам группы приходится ждать результаты расчетов других членов группы для перехода к следующему этапу исследования [1]. Частое возникновение таких задержек может отодвинуть выполнение проекта на месяцы. Кроме того, задержки и простои в работе могут негативно сказаться на слаженности и заинтересованности работы группы в проекте. Поэтому актуальной является задача повышения быстродействия ресурсоемких научных вычислений за счет оптимизации механизмов взаимодействия программного кода алгоритма и операционной системы.

Основная часть. Научные расчеты ведутся как на персональных компьютерах с многоядерными процессорами, так и на специализированных вычислительных кластерах с большим количеством ядер. Большая часть этих вычислительных систем работает под управлением операционной системы Linux. Одной из основных частей операционной системы является планировщик задач, который предназначен для распределения инициируемых запущенными приложениями потоков между процессорными ядрами, а также для обеспечения многозадачности, отсутствия зависаний системы и ускорения выполнения запущенных потоков [2–4]. В настоящее время в Linux используются универсальные планировщики задач, в которых наибольшим приоритетом обладают интерактивные задачи взаимодействия пользователя с интерфейсом операционной системы.

Поступающие в систему задачи по уровню приоритета подразделяются на K типов. Приоритет ξ_k каждого типа задачи может определяться номером k ее типа или величиной, пропорциональной k. Коэффициент пропорциональности удобно выбрать из условия, чтобы сумма приоритета задач всех типов была нормирована на 1:

$$\sum_{k=1}^{K} \xi_k = 1. \qquad (1)$$

Помимо того, что каждая задача относится к некоторому типу, она также принадлежит одному из 2-х классов.

В первый класс входят обычные задачи.

Второй класс составляют задачи, решение которых требует проведения значительно большего количества операций (так называемые «счетные задачи»). Процесс решения счетных задач занимает значительно большее время. Общее количество счетных задач примерно на порядок меньше общего количества обычных задач.

Таким образом, каждая задача характеризуется парой целых чисел (тип, класс).

Принимается, что потоки задач всех типов на входе системы – случайные и квазистационарные, а законы распределения количеств поступающих на вход задач – показательные. То есть функция распределения $F(t)$ величины промежутка времени t между поступлениями двух задач одного типа определяется как

$$F(t) = 1 - exp\{-t/T_k\tau\}, \qquad (2)$$

где T_k – среднее время между поступлениями двух задач одного типа.

В общем случае величины $T_k(\tau)$ различны для разных типов задач и зависят от системного времени τ (текущее время суток).

Свойство квазистационарности потоков означает, что интенсивность поступления задач в систему может слабо изменяться в течении времени порядка 1000 с, но в каждый момент τ законы распределения промежутков времени между поступлениями очередных задач показательные. В течении рабочего дня су-

ществует некоторый характерный профиль спроса на время процессора, имеющий несколько максимумов и минимумов.

Для решения задач требуется N типов ресурсов. Потребности задач в ресурсах определяются матрицами $R = R_{ij}$; $i = 1,N$; $j = 1,K$ и $\sigma = \sigma_{ij}$, $i = 1,N$; $j = 1,K$.

Элементы R определяют среднюю потребность (математическое ожидание) k-й задачи в n-м ресурсе [5].

Современные алгоритмы разработаны для обеспечения максимально комфортной работы пользователей и ориентированы в основном на использование в домашних и офисных компьютерах. То есть современные планировщики ориентированы на запуск в компьютере большого количества различных приложений, между которыми осуществляется регулярное переключение [6, 7]. Это делает их малоэффективными при работе с ресурсоемкими научными вычислениями, где основная часть потоков занята вычислительными задачами, а взаимодействие пользователя с интерфейсом операционной системы сведено к минимуму. Кроме того, есть ряд особенностей, снижающих эффективность существующих планировщиков при работе с ресурсоемкими научными вычислениями:

- существующие планировщики способны разделять задачи на интерактивные, вычислительные и т. д. Но идентифицируя задачу как вычислительную, планировщик не может получить дополнительную информацию о запущенном вычислении и этапе конкретного вычисления задачи;

- для обеспечения многозадачности и высокой скорости реакции интерфейса на действия пользователя существующие планировщики запускают на выполнение одновременно несколько потоков на одном ядре. Ядро в один момент времени выполняет только один поток. Поэтому для обеспечения параллельной работы запущенных приложений происходит регулярное переключение между потоками, работающими одновременно на одном ядре. Переключение между потоками является довольно трудоемкой задачей, а так как переключений осуществляется достаточно много, то большой процент процессорного времени расходуется именно на них.

В вычислительных алгоритмах существуют этапы, которые не могут быть распараллелены, т. е. для выполнения текущей операции необходим всего один поток. Например, считывание исходных данных, вывод результатов, передача результатов вычислений текущего этапа работы алгоритма на следующий. Также есть этапы непосредственно вычислений, которые могут быть распараллелены, так как каждое отдельное вычисление является отдельной задачей, а соответственно, для нее будет создан отдельный поток. Количество времени, необходимое на проведение каждого расчета, может быть различно, и задача планировщика задач операционной системы – максимально эффективно распределить вычислительные потоки между процессорными ядрами для минимизации времени ожидания каждого ядра и накладных расходов на переключение между задачами.

Для повышения скорости работы алгоритмов научных расчетов существует несколько путей:

- увеличение количества единиц в вычислительном кластере. Как правило, ресурсоемкие научные вычисления ведутся не на одном компьютере, а на специально подготовленных вычислительных кластерах. Наиболее простым и очевидным способом увеличения скорости расчетов является наращивание вычислительной мощности кластера за счет увеличения количества компьютеров. Не смотря на очевидность и простоту реализации, такой подход обладает рядом минусов. Основной минус – дополнительные материальные расходы на покупку и размещение дополнительных компьютеров. Но необходимо учитывать, что увеличение количества процессоров в вычислительном кластере далеко не всегда может способствовать повышению скорости ведения расчетов. Это происходит потому, что в алгоритмах расчетов не все части задачи могут быть распараллелены. А если задачи распараллелены на некоторое ограниченное количество ядер, то увеличение количества параллельных процессов не будет иметь смысла, так как приведет к ожиданию завершения работы одних процессов другими и простаиванию выделенных под них ядер. Таким образом высокие материальные затраты могут дать только незначительный прирост производительности, а в некоторых случаях не дать его вовсе. Кроме того, такой подход позволит получить прирост производительности на одном конкретном вычислительном кластере, но никак не повлияет на повышение производительности других;

- повышение быстродействия за счет усовершенствования и оптимизации работы алгоритма. Такой подход может помочь повысить быстродействие алгоритма, но потребует больших временных затрат на исследования в этом направлении без гарантии получения каких-либо результатов. В долгосрочной перспективе этот подход должен использоваться. Положительные результаты исследований могут дать значительный прирост скорости работы алгоритма и этот прирост будет качественно лучше, чем полученный первым способом. Он позволит ускорить работу алгоритма не зависимо от используемой вычислительной платформы. Однако такой подход абсолютно не применим к другим алгоритмам;

- оптимизация механизмов взаимодействия операционной системы, процессора и вычислительного алгоритма. Все алгоритмы разделяются на небольшие подзадачи, которые планировщик задач операционной системы передает на обработку процессору. Здесь необходимо разработать механизмы, которые позволят максимально эффективно использовать процессорное время при ведении сложных расчетов, что позволит увеличить вычислительную мощность системы независимо от ее конфигурации и используемых алгоритмов.

Очевидно, что третий способ в виду своей универсальности является наиболее перспективным. Его реализация подразумевает разработку информационной технологии, в которую будут входить несколько модулей:

- планировщик задач операционной системы, значительно отличающийся от нынешнего и ориентированный на ведение сложных расчетов при минимальном взаимодействии пользователя с интерфейсом операционной системы. Это позволит сократить количество процессорного времени, выделяемого для интерактивных задач, которые в существующих планировщиках являются приоритет-

ными. При проектировании планировщика необходимо учитывать возможность запуска нескольких расчетов одновременно. Они могут выполняться с выделением всех необходимых ресурсов системы для максимальной скорости работы запущенного первым расчета или в режиме, позволяющем выделять для расчетов только необходимое для обеспечения приемлемой работы количество ресурсов в соответствии с очередностью запуска. Это потребует разработки механизмов для определения необходимого количества ресурсов для приемлемой работы алгоритма. Под приемлемой работой следует понимать время расчета, меньше максимально возможного, но не превышающего предел ожидания пользователя. Необходимость разработки второго способа обусловлена наличием предела ведения параллельных вычислений, после достижения которого выделение под расчет дополнительных ядер все еще будет давать прирост производительности, но начнет быстро увеличиваться суммарное время простоя по всем процессорным ядрам. Поэтому дальнейшее выделение ядер под расчет становится нецелесообразным. Гораздо эффективнее выделить эти ядра для другого запущенного параллельно расчета, который еще не достиг этого предела. Однако если запустить только один расчет, ядра можно выделять до тех пор, пока растет производительность.

Описанные выше механизмы будут частью модуля обучения и адаптации планировщика под конкретные вычислительные алгоритмы.

Основной задачей модуля адаптации является минимизация количества переключений между задачами, так как эти операции не являются полезной работой планировщика и при неправильном планировании могут отнимать большой процент ресурсов процессора. Для этого необходим механизм обучения, анализирующий работу алгоритма после его завершения, что позволит:

- разработать механизмы оценки времени, необходимого для завершения всех переданных на выполнение ядру потоков. Это позволит минимизировать время простоя ядер за счет своевременного перераспределения между ними поставленных в очередь на выполнение потоков;
- разработать механизмы оценки предполагаемого времени выполнения передаваемого в очередь потока;
- разработать механизмы передачи информации о степени завершенности процессов и времени, необходимого для выполнения поставленных в очередь процессов, от приложения планировщику задач;
- усовершенствовать алгоритмы передачи данных по сети Ethernet с помощью стека протоколов TCP/IP между объединенными в кластер компьютерами.

Эти механизмы позволят настроить работу планировщика для каждого конкретного алгоритма и тем самым создать класс адаптивных планировщиков задач операционной системы Linux с возможностью автоматической оптимизации своих параметров на основе обучения.

На рис. 1 показана общая схема взаимодействия операционной системы с планировщиком задач.

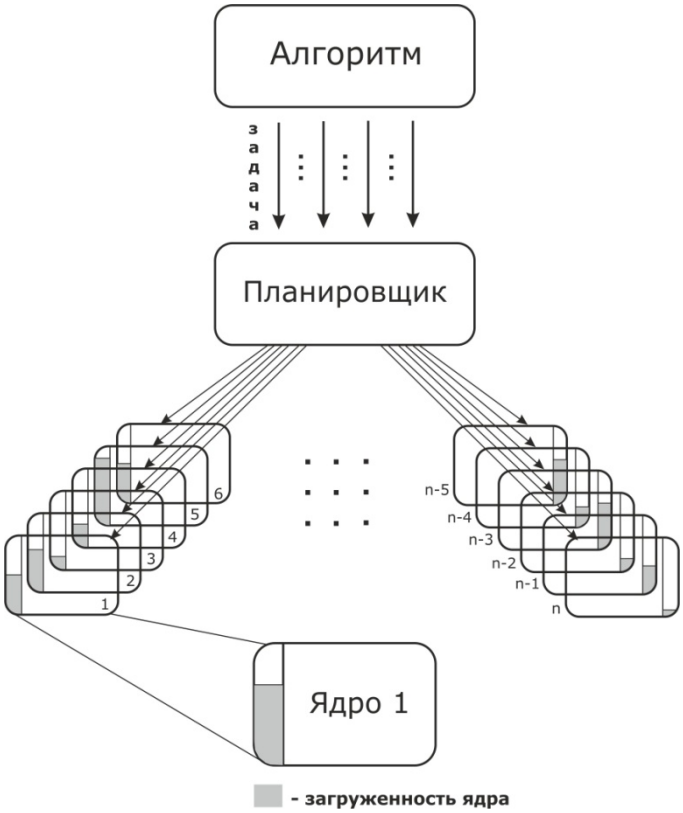

РИС. 1. Общая схема взаимодействия операционной системы с планировщиком задач

При использовании мощных вычислительных кластеров, где количество ядер превышает количество одновременно инициируемых системой потоков или незначительно ему уступает, необходимость в оптимизации планировщика отсутствует, так как стандартные планировщики обеспечивают эффективную работу алгоритма, направляя потоки на свободные ядра. Необходимость в оптимизации работы планировщика возникает в том случае, когда количество ядер значительно меньше числа одновременно инициируемых вычислительных потоков. Это происходит при ведении расчетов на одном многоядерном процессоре, на нескольких компьютерах, объединенных в небольшой вычислительный кластер, например, в пределах одного кабинета, или при запуске на кластере с большим количеством ядер нескольких десятков ресурсоемких вычислений. В последнем случае задача оптимизации работы планировщика усложняется необходимостью не просто ускорения выполнения всех запущенных одновременно потоков, но и учетом того к какому вычислению принадлежат потоки и распределением потоков таким образом, чтобы как можно быстрее завершался очередной этап каждого расчета и полученные результаты передавались на следующий этап.

Выводы. Предложенная информационная технология позволит увеличить производительность вычислительных систем независимо от их конфигурации и используемых алгоритмов. Наиболее эффективным применение этой технологии будет в условиях нехватки вычислительных ресурсов, когда количество инициируемых вычислительных потоков будет значительно превышать количество доступных ядер процессоров.

1. *Планировщики* процессов: http://ck.kolivas.org/patches/staircase-deadline/ rsdl_scheduler. readme
2. *Планировщи*к задач в Linux (proccess cpu kernel linux): http://www.opennet.ru
3. *Лав Р.* Разработка ядра Linux, 2-е издание / Р. Лав // Пер. с англ. – М.: ООО И.Д. Вильямс, 2006. – 448 с.
4. *Fujiyama R., Scherpelz J., Ferlo M.* Analyzing the Linux schedulers's tunables. – 2003. – 19 с.
5. *Белоусов С.М.* Имитационная модель и методы рационального распределения ресурсов операционной системы : Автореф. дис. канд. техн. наук. – Москва. – 2007. – 24 с.
6. *Kolivas C.* Linux Kernel CPU Scheduler Contributor, IRC conversations, no transcript, 2004.
7. *Jake* Moilanen's Linux Kernel Homepage: http://kernel.jakem.net

Получено 06.09.2013