МОДЕЛИРОВАНИЕ СВЯЗАННЫХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Ю. В. Тименков (Москва)

Введение

С развитием интернета все большую популярность приобретают такие понятия, как облачные вычисления, Software as a Service, IT-outsourcing и др. В самом деле, сейчас уже стала очевидна выгода от заказа IT-услуг у сторонних организаций, которые могут эффективно обслуживать тысячи клиентов.

Для эффективного управления такими центрами обработки данных необходим не только правильный мониторинг состояния систем, но и возможность предсказывать их поведение при возникновении различных ситуаций. В данной работе предлагается способ моделирования исполнения процессов, учитывающий не только конкуренцию за обладание системными ресурсами, но и возможные зависимости между процессами.

Обзор существующих работ

Проблема моделирования вычислительных систем далеко не нова. Но большинство работ в этой области посвящено моделированию отдельных компонент, таких как сеть, файловые системы, планировщики CPU и т.д.

Пожалуй, наиболее распространенным объектом для моделирования является сеть. Для нее очень хорошо подходят существующие математические методы: для описания потоков между узлами используется теория графов, а нагрузку можно описывать как случайный процесс [1].

Также большое количество работ посвящено анализу производительности файловых систем и диска. Для этого разработаны как специализированные тесты, позволяющие оценить работу в тех или иных условиях, так и методы математического моделирования [2].

Особняком стоят планировщики: они мало заметны обычному пользователю (так как человек работает только с одним приложением одновременно), но играют не последнюю роль в общей производительности компьютеров, особенно серверов. Существует большое количество приложений для решения различных задач, и многие из них разрабатывались на основе математических моделей.

Все описанные выше методы дают очень хорошие результаты для оценки производительности системы в целом. К сожалению, они не предназначены для прогнозирования хода исполнения отдельных приложений. Данная работа ставит перед собой задачу заполнить этот пробел и рассматривает вычислительную систему с позиции исполняющихся в ней процессов. Ценность ее и в том, что предлагается комплексный подход, учитывающий следующие факторы:

каждому приложению необходим набор разных видов компьютерных ресурсов;

в серверных системах одновременно работают несколько программ, которые вынуждены конкурировать между собой;

процессы могут находиться в сложной взаимосвязи, например один ожидает данных от другого. В этом случае предсказывать их работу еще сложнее, так как они не могут быть исполнены один без другого, но при этом конкурируют за ресурсы.

Математическая модель исполнения процесса

Как было сказано ранее, исполнение программы формально можно рассматривать как процесс потребления ресурсов (процессора, диска, сети и т.д.) [3]. В свою очередь, операционная система отвечает за их распределение.

Отметим сразу, что в данной работе мы не рассматриваем конкретные алгоритмы планировщиков и параметры их тонкой настройки. Прежде всего, нас интересует результат распределения ресурсов, наблюдаемый на макроуровне. Как показывает практика, он слабо зависит от выбранной стратегии планирования.

Пусть процессу для исполнения в течение некоторого отрезка времени необходим набор ресурсов $\vec{D} = \{D_i\}$. Это может быть количество байт, считанных или записанных на диск, тактов (секунд) исполнения на процессоре и т.д. В свою очередь, система готова ему выделить $\vec{R} \neq \vec{D}$ ресурсов. В этом случае внутреннее время исполнения процесса замедлится (или ускорится) на величину

$$\gamma = \min_i \frac{R_i}{D_i}. \quad (1)$$

Назовем ее коэффициентом недостаточности ресурса или коэффициентом голодания. Данное утверждение основывается на экспериментальном изучении скорости исполнения процесса, описанном в работе [4].

Минимум берется потому, что потребление ограничено наименее доступным ресурсом, остальные просто возвращаются системе. Данное утверждение становится очевидным, если провести следующий мысленный эксперимент. Пусть у нас есть приложение, требовательное к процессору, которое посылает немного данных другим компьютерам (типично для распределенных вычислений). Таким образом, сколько бы система не предоставляла ему сети, оно не будет работать быстрее и потребление сети будет постоянным, т. е. фактический уровень ресурсов, использованный процессом составит:

$$\vec{R}' = \gamma \vec{D}. \quad (2)$$

Пусть система может предоставить процессам суммарно $\vec{L}$ ресурсов. В данную величину заложены как возможности оборудования (мощность процессора, предельные скорости чтения с диска и т.д.), так и особенности программной реализации (эффективность планировщика, файловой системы и т.д.). Тогда каждому процессу будет предоставлено:

$$\vec{R}_j = \frac{1}{N} \vec{L}, \quad (3)$$

где j – номер процесса; N – их количество в системе.

В реальных системах за каждый тип отвечает свой планировщик, но мы рассматриваем все ресурсы одновременно. И если сначала разделить все ресурсы, а затем вычислить реальные уровни потребления (2), то суммарная величина будет гораздо ниже $\vec{L}$, поэтому при математическом описании распределения необходимо учитывать следующие ограничивающие факторы:

– Ресурсы должны быть распределены равномерно между процессами, в соответствии с их потребностями.

– Должно остаться минимальное количество нераспределенного ресурса.

Для описания работы операционной системы предлагается следующий итеративный алгоритм, учитывающий эти критерии:

1. Из уравнения (3) вычисляем векторы доступных ресурсов $\vec{R}_j$, где $j \in 1 \ldots N$ – номер потребителя.

2. Для каждого потребителя считаем коэффициент недостаточности по всем ресурсам из уравнения (1). Важно отметить, что из рассмотрения исключаются ресурсы, которые не требуются процессу в данный момент (т.е. такие, для которых $D_j^i = 0$).

3. Процесс с наименьшим коэффициентом исключается из дальнейшего рассмотрения:

$$j_{min} = arg\min_k \gamma_k. \tag{4}$$

При этом:

- из уравнения (2) вычисляем реальное количество ресурсов $\vec{R}'_{j_{min}}$, которые он потребит;
- уменьшаем мощность системы $\vec{L}$ на данную величину, так как эти ресурсы уже не доступны для потребления:

$$\vec{L} = \vec{L} - \vec{R}'_{j_{min}}. \tag{5}$$

4. Для оставшихся процессов проделываем эти же операции.

Таким образом, для текущего момента времени мы получили уровни потребления и коэффициенты недостаточности для каждой программы. Последние используются для вычисления нового внутреннего времени процесса и соответствующей ему потребности в ресурсах $\vec{D}$:

$$t_j^{n+1} = t_j + \gamma_j \Delta t. \tag{6}$$

Рассмотрим теперь моделирование зависимых процессов на примере наиболее распространенного способа взаимодействия поставщик–потребитель. Предположим, что за рассматриваемый интервал времени Δt один процесс произвел количество ресурса R_P. Будем называть его обменным. Это может быть количество байтов, пакетов и т.д. За это же время другой процесс потребил R_C (на макроуровне, даже на однопроцессорной системе программы исполняются параллельно). В результате в буфере останется ресурс, равный:

$$L' = L_n + R_P - R_C = L_n + D_P dt - R_C dt, \tag{7}$$

где L_n – ресурс, оставшийся с предыдущего момента времени. Причем для моделирования удобно полагать, что эта величина может быть и отрицательной, т.е. потребитель как бы работает в кредит.

Планировщик будет пытаться скомпенсировать образовавшийся избыток (или недостаток) ресурса, замедляя и ускоряя соответствующие процессы. Это можно выразить с помощью специальных коэффициентов недостаточности для обменных ресурсов:

$$\gamma_P = \begin{cases} 1 - \dfrac{L'}{L^{max}}, & L' \le L^{max} \\ 0 & L' > L^{max} \end{cases} \quad \gamma_C = \begin{cases} \dfrac{L'}{L^{max}} + 1, & L' \ge -L^{max} \\ 0 & L' < -L^{max} \end{cases} \tag{8}$$

Здесь L^{max} – нормировочный множитель, определяющий степень толерантности к дисбалансу ресурсов. Его физический смысл – размер буфера между процессами, выраженный в тех же единицах, что и обменный ресурс. Таким образом, уравнение (1) запишется в виде:

$$\gamma = \min\left(\frac{R_i}{D_i}, \gamma_I\right), \tag{9}$$

где γ_I – коэффициент недостаточности обменного ресурса. Стоит отметить, что γ_I зависит только от потребностей процессов. Поэтому он вычисляется в каждый момент времени один раз, в отличие от системных ресурсов, которые обновляются в цикле после каждого исключенного процесса. В постоянном пересчете нет необходимости, так как если процесс был готов произвести (или потребить) большое количество обменного ресурса, но ему не хватило компьютерных, то модель скорректирует это на следующем шаге. В результате на графиках могут наблюдаться колебания уровней потребления.

Результаты эксперимента

Для подтверждения данного метода было проведено сравнение с работой реальных программ. В качестве задачи был выбран пример конвейерной обработки данных через PIPE: создание данных (случайных чисел) с последующей запаковкой их в архив с помощью утилиты gzip. Выбранные программы позволяют продемонстрировать как конкуренцию (и генерация и архивирование требовательны к процессору), так и зависимость между скоростями исполнения. Даже при наличии достаточного количества ресурсов приложение не может исполняться, если у него нет входных данных для обработки, или буфера для размещения результатов.

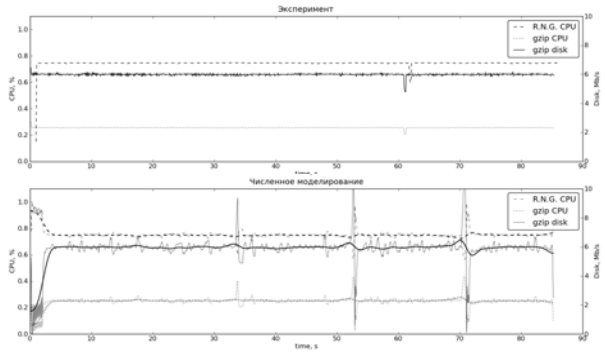

Рис. 1. Потребление ресурсов при исполнении на одном процессоре

В первом случае обе программы исполнялись на одном процессоре. Результаты приведены на рис. 1. Как видно из графиков, связь между скоростями исполнения не позволяет планировщику распределять нагрузку пополам. Процессор делится в соотношении затрат на производство и потребление одной единицы обменного ресурса.

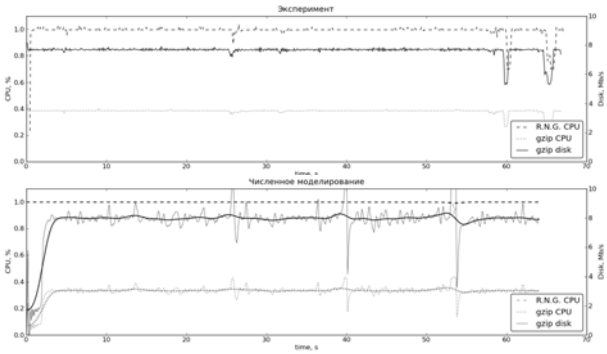

Рис. 2. Потребление ресурсов при исполнении на разных процессорах

Во втором эксперименте, результаты которого показаны на рис. 2, программы исполнялись на разных процессорах. Здесь мы можем наблюдать, как производитель из-за ограниченности ресурсов не дает потребителю исполняться с максимальной скоростью. В результате второе процессорное ядро остается недогруженным.

Для моделирования данной ситуации было введено два различных ресурса для каждого ядра, при этом каждая программа требовала свой тип. Таким образом каждое приложение получало в свое распоряжение ядро полностью и не испытывало конкуренции за процессор.

Время исполнения программ, с

Утилиты	Экспериментальное	Модельное
Random generator	64,3	-
Gzip	21,8	-
Последовательно	85,8	85,2
Параллельно	67,5	64,0

В таблице приведено среднее время завершения для каждого случая.

Заключение

Из сравнения результатов численного моделирования и эксперимента можно сделать вывод, что данный метод хорошо описывает рассматриваемое явление. Скорости монопольного и зависимого исполнения также показывают соответствие модели закону Амдала, т. е. при исполнении на одном процессоре время завершения равно сумме времен монопольного исполнения каждого процесса. А при параллельном запуске скорость ограничена наиболее медленной программой.

Предлагаемый метод можно использовать для прогнозирования поведения различных сервисных операций, где трудно предсказать влияние одних процессов на другие. К ним относятся миграция приложений и виртуальных машин, резервное копирование и другие.

Литература

1. Анализ и моделирование трафика в высокопроизводительных компьютерных сетях/ **С. Д. Белов, С. В. Ломакин, В. А. Огородников и др**. // Вестник НГУ. 2008. Т. 6. С. 41–48.
2. **Нижник Е. И., Тормасов А. Г.** Математическое моделирование производительности файловой системы NTFS при нагрузке типа «запись дисковых данных» // Вестник НГУ. 2007. Т. 5, № 2. С. 67–77.
3. **Тименков Ю. В., Тименкова Д. В.** Кинематическая модель исполнения процесса // Математические модели и задачи управления. 2011. С. 177–185.
4. **Тименков Ю. В., Тименкова Д. В.** Экспериментальное изучение скорости выполнения процессов и перераспределения ресурсов между ними // Труды 52-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук». ЧастьVII. Управление и прикладная математика. Т. 3 М.: МФТИ, 2009. С. 49–51.