Модели и алгоритмы распределения нагрузки

Алгоритмы на основе сетей СМО

А.С. Хританков

Аннотация. В работе представлен краткий обзор методов теории сетей массового обслуживания и некоторых алгоритмов балансировки нагрузки. Вторая часть работы посвящена классификации алгоритмов балансировки и анализу на примере нескольких систем.

Ключевые слова: балансировка вычислительной нагрузки, распределенные и параллельные вычисления, теория массового обслуживания, классификация алгоритмов балансировки.

Введение

Данная работа является продолжением обзора моделей и алгоритмов балансировки нагрузки [23]. В первой части была представлена модель коллектива вычислителей, модели систем с соперником и диффузионная модель. В данной части речь идет о сетях массового обслуживания (СМО), представлен ряд алгоритмов, основанных на теории сетей СМО, приводится вариант классификации алгоритмов балансировки и пример анализа нескольких параллельных вычислительных систем.

В разделе 1 представлен ряд простых моделей сетей массового обслуживания. Модели СМО позволяют рассчитать характеристики системы при решении последовательности задач одного или нескольких классов. Обзор теоретических методов исследования моделей систем массового обслуживания дан в работе [1]. Для модели сети массового обслуживания в разделе 1.3 рассматривается задача о минимизации времени ответа системы из нескольких серверов, объединенных вычислительной сетью и обрабатывающих поток заявок, поступающих в систему. В качестве примера системы, описываемой данной моделью, можно привести вычислительный кластер, используемый для веб-хостинга.

В разделе 2 приводится одна из наиболее удачных, на наш взгляд, классификаций алгоритмов балансировки нагрузки и примеры анализа алгоритмов, реализованных в ряде вычислительных систем.

Среди других подходов к моделированию и анализу вычислительных систем следует отметить стохастические сети Петри. Сети Петри ориентированы на моделирование низкоуровневой реализации вычислительных систем и анализ параллельных программ. Обзор качественных результатов в данной области можно найти, например, в работе [3]. Моделированию параллельных и распределенных систем при помощи сетей Петри посвящен специальный выпуск журнала по параллельным и распределенным вычислениям (J. of Parallel and Distributed Computing, Vol. 15, No. 3, 1992), подробный обзор количественных результатов приведен в статье [2] в указанном выпуске.

1. Моделирование вычислительных систем с помощью сетей массового обслуживания

2.1. Введение в сети массового обслуживания

Сетью массового обслуживания [4] называется совокупность систем массового обслуживания (СМО), называемых серверами, и заявок, представляющих запросы к данным серверам. Заявки обработанные одной СМО, могут поступать на вход другой. По аналогии с СМО, сеть СМО называется закрытой, если заявки в системе создаются конечным числом клиентов и каждый клиент ожидает получения обработанной заявки перед отправкой следующей. Количество заявок в закрытой сети остается постоянным. Сеть СМО называется открытой, если заявки поступают извне в виде входящего потока и обработанные заявки выводятся из сети. Сети СМО, в которых присутствуют оба типа заявок, называются смешанными. Закрытые модели используются для описания вычислительных систем, в которых число пользователей ограниченно, например систем, обслуживающих пользователей локальной сети. Сеть может содержать заявки разных классов, при этом заявки считаются неразличимыми в рамках одного класса по времени обработки.

Рассмотрим простой пример открытой сети, состоящей из одного сервера. Схема сети представлена на Рис. 1. Заявки поступают в систему, ожидают в очереди, пока сервер не освободится, обрабатываются на сервере и покидают систему как обработанные заявки.

Сети СМО, так же как и СМО, разделяют на классы по виду функции распределения, описывающей входящий поток заявок, и функции распределения времени обработки заявки на сервере, а также по другим признакам. Пусть для данной сети указанные распределения были зафиксированы. Для сети может быть задано два параметра: интенсивность входящего потока заявок и среднее время обслуживания заявки на сервере. Для данной системы в стационарном режиме, решив уравнения теории массового обслуживания в простейшем случае, можно получить следующие характеристики производительности системы:

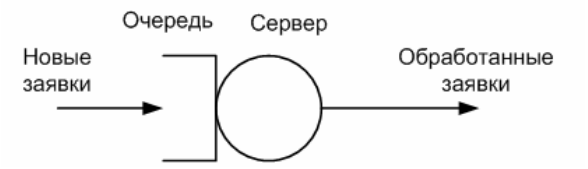

Рис.1. Графическая нотация, применяемая при моделировании сетей СМО

- загруженность системы – доля времени, которую система занята обработкой заявок;
- время пребывания – среднее время, в течение которого заявка находилась в системе, включая время ожидания и обработки на сервере;
- длина очереди – среднее количество заявок на сервере;
- пропускная способность – среднее количество заявок, обрабатываемых системой в единицу времени.

В приведенных характеристиках «среднее» понимается как значение характеристики в стационарном режиме работы сети.

В общем случае, сеть массового обслуживания можно определить как направленный граф $G =< N, E >$, где $N = \{C, Q, S, T\}$ - множество вершин, C - множество истоков, соответствующих классам заявок, Q - множество очередей, S - множество серверов, обрабатывающих заявки, T - множество стоков, соответствующих выходам заявок из системы. Кроме того, для всех вершин, кроме стоков, определяется правило распределения заявок по исходящим ребрам. Одним из способов задания правила является указание вероятности направления заявки вдоль каждого ребра при покидании вершины. Для очередей задается политика приоритизации (установления приоритета обслуживания) заявок, например FIFO: первым пришел – первым обработан. Для серверов задается функция распределения времен обработки заявок каждого класса. В случае открытой модели для истоков указывается функция распределения времени между созданием новых заявок. При использовании закрытой модели указывается функция распределения продолжительности ожидания источника заявок после получения ответа на предыдущую заявку перед созданием новой заявки.

Для классификации систем массового обслуживания часто используется известная клас-

сификация Кендалла [4]. Система характеризуется словом A/B/C/K/N/D, в котором:

A характеризует процесс поступления заявок в систему (M – Пуассоновский процесс, G – произвольный процесс и т.д.),

B – процесс обработки заявок (аналогично),

C – число серверов на одну очередь в системе, параметры,

K – максимальное число заявок в системе,

N – полное число заявок,

D – политика приоритетов очереди.

Параметры K, N и D обычно опускаются, при этом полагается, что длина очереди не ограниченна, система может содержать сколь угодно большое число заявок и подразумевается политика приоритетов FIFO.

Пример моделирования устройства компьютера [14] приведен на Рис.2. Серверами являются процессор и дисковая подсистема. С процессором связана очередь процессов, ожидающих квант времени на выполнение, с дисковой подсистемой связана очередь запросов на операции ввода-вывода.

На диаграмме для серверов заданы характеристики обслуживания, строка «M/2» обозначает показательное распределение времени обслуживания и указывает, что на сервере два обработчика связаны с одной очередью.

Приведем несколько простых соотношений, справедливых для произвольной системы и сети массового обслуживания, рассматриваемой как «черный ящик». Пусть измерены следующие параметры системы:

T – продолжительность времени наблюдения за системой;

A – число пришедших заявок;

C – число обработанных заявок.

Используя их, определим простые характеристики системы:

λ – интенсивность входящего потока заявок, $\lambda = \frac{A}{T}$;

X – пропускная способность системы, $X = \frac{C}{T}$.

Если система состоит из одного сервера и известно время B, в течение которого сервер был занят обработкой заявок, определяются еще две величины:

U – загруженность системы, $U = \frac{B}{T}$;

S – среднее время обработки заявки, $S = \frac{B}{C}$.

Закон загруженности выражается следующим соотношением:

$$U = XS .$$

Одним из важнейших и наиболее универсальных соотношений теории массового обслуживания является закон Литтла [5], утверждающий, что среднее число заявок в системе в установившемся режиме N равно интенсивности обслуживания, умноженной на среднее время пребывания заявки в системе:

$$N = XR,$$

где R - среднее время пребывания заявки в системе, равное сумме времен ожидания и обработки заявки.

Сетью Джексона называется такая открытая сеть массового обслуживания из m серверов, в которой:

- входящий поток заявок является пуассоновским;
- время обработки заявок на каждом сервере имеет показательное распределение;
- заявка, покидающая сервер, либо попадает на другой сервер, либо покидает систему с фиксированными вероятностями P_{ij}, при этом события перехода независимы и имеют одинаковое распределение (например, равномерное);

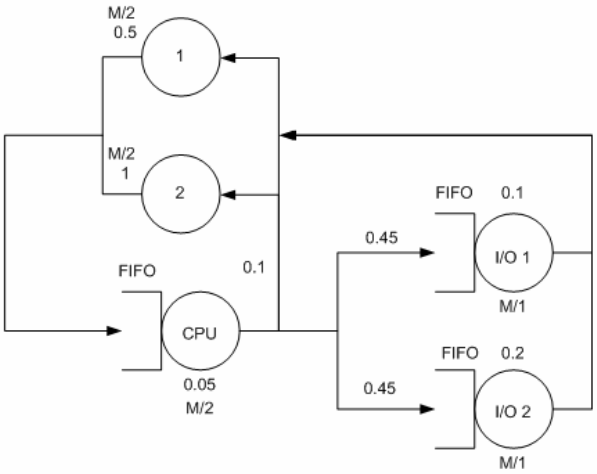

Рис. 2. Пример модели сети СМО для ЭВМ

• загруженность всех серверов системы меньше единицы.

Для сети Джексона можно показать [4], что серверы допустимо рассматривать независимо друг от друга и каждый из них создает пуассоновский поток обработанных заявок. Примером сети Джексона может служить сеть из двух последовательно соединенных серверов с показательными временами обслуживания заявок и входящим пуассоновским потоком. Требование независимости переходов заявок с одного сервера на другой подразумевает отсутствие петель в графе сети. В противном случае нарушается свойство марковости (возможности описания состояния системы некоторым марковским случайным процессом) системы, и потоки обработанных заявок не будут пуассоновскими. Однако для сетей с петлями справедлива теорема Джексона, утверждающая, что в такой сети серверы могут рассматриваться независимо и результирующее распределение количества заявок в очередях серверов представимо в виде произведения:

$$p(k_1,..,k_n) = p_1(k_1)\cdot..\cdot p_n(k_n),$$

где $p_i(k_i)$ - вероятность того, что в очереди i-го сервера находится k_i заявок. Данное представление также называется мультипликативной формой (product form).

Обобщением сетей Джексона являются BCMP сети, названные по именам авторов [6]. В BCMP сетях может присутствовать несколько классов заявок и серверы должны принадлежать одному из следующих типов:

• обработка заявок в порядке поступления (FCFS) и показательное распределение времени обработки с одинаковой интенсивностью для всех классов; интенсивность обработки может зависеть только от полного числа заявок в очереди сервера;

• режим разделения процессора (PS); каждый класс заявок обслуживается независимо с различными распределениями времени обработки;

• бесконечный сервер (IS), на котором заявки пребывают некоторое время, не зависящее от числа заявок;

• обработка последней полученной заявки с вытеснением обрабатывающейся в данный момент (LCFS-PR).

BCMP теорема утверждает, что в BCMP сетях серверы могут рассматриваться независимо друг от друга и вероятность того, что на серверах системы находится определенное количество заявок, может быть получена как произведение соответствующих вероятностей для отдельных серверов:

$$p(\bar{k}) = p_1(\bar{k}_1)\cdot..\cdot p_n(\bar{k}_n),$$

где $\bar{k} = (\bar{k}_1,..,\bar{k}_n)$, $\bar{k}_i = (k_{i,1},..k_{i,r},..,k_{i,R})$ и $k_{i,r}$ - число заявок класса r на сервере i.

При рассмотрении потоков в транспортных сетях был сформулирован парадокс типа Браэсса. Парадокс состоит в том, что при использовании неоптимального алгоритма распределения транспортных потоков добавление дополнительных путей в системе может приводить к увеличению времени прохождения заявки от источника к получателю. Подобные парадоксы встречаются и при распределении нагрузки в распределенных и параллельных системах. В работе [8] рассматривается система из двух различных серверов, которые могут передавать заявки друг другу для обработки. Показано, что если каждый из серверов стремится минимизировать среднее время пребывания заявок на нем (критерий для класса заявок), то при некоторых положительных значениях пропускной способности коммуникационного канала между серверами среднее время пребывания на каждом из них больше, чем в отсутствие обмена заявками. В то же время кажется очевидным, что время пребывания, по крайней мере, не должно увеличиться. В этом и состоит парадокс. Заметим, что если минимизируется среднее время пребывания для двух серверов (критерий для всех заявок) или для каждой заявки (индивидуальный критерий), то парадокса не возникает. Явление парадокса типа Браэсса связано со структурой равновесия, достигаемого в системе. В случае классового критерия возникает равновесие Нэша, которое может не соответствовать глобальному оптимуму, достигаемому при использовании общего критерия. Подробный анализ видов равновесия и возни-

кающих при балансировке нагрузки в распределенных системах парадоксов можно найти в работе [7] и ссылках в ней.

1.2. Оптимизация иерархической структуры Грид

В работе [9] исследована проблема построения оптимальных древовидных структур для распределенных сетей сервисов (Грид). Рассматривается система из N узлов, с помощью теории сетей СМО рассчитывается среднее время обработки заявки в открытой модели для различных конфигураций взаимодействий между узлами. При этом структура системы является деревом, для организации взаимодействия между узлами используется парадигма ведущий-ведомый (master-slave). Ведущие узлы распределяют нагрузку между ведомыми узлами и располагаются на один уровень выше в дереве структуры системы. Полагается, что время обработки одной заявки ведущими узлами пропорционально числу относящихся к нему ведомых узлов вследствие необходимости опроса состояния и принятия решения о направлении заявки. Также полагается, что задержка передачи заявок по сети не зависит от числа передаваемых заявок и друг от друга.

В предлагаемой модели имеется L уровней ведущих узлов, плоская структура соответствует L=1. Входящий поток заявок полагается пуассоновским с параметром λ. Серверы на ведомых узлах полагаются одинаковыми и пуассоновскими с интенсивностью обслуживания ν. Интенсивность обслуживания на ведущих узлах составляет

$$\mu_i = \frac{c_i}{n},$$

где n - число ведомых узлов, c_i - константа пропорциональности для уровня i. Время передачи заявки по сети составляет T_i. На основание результатов имитационного моделирования ведущий узел и его ведомые узлы на последнем уровне рассматриваются как система $M/M/n$ для больших n.

Ведущий узел на уровне i опрашивает ведомые узлы как система $M/M/1$, при этом интенсивность обслуживания полагается обратно пропорциональной количеству ведомых узлов. Среднее время ответа при этом определяется формулой

$$W_i = \frac{1}{\mu_i(1-\rho_i)},$$

где $\rho_i = \lambda_i / \mu_i$ - загруженность ведущего узла в предположении, что интенсивность опроса ведущим узлом ведомых равна интенсивности входящего потока. Предполагается, что входящий поток равномерно разделяется между ведомыми узлами: $\lambda_i = \lambda k^{-i}$. Заметим, что, вообще говоря, такой поток уже не будет пуассоновским, однако его можно приближенно считать таковым, как и поступили авторы работы. Среднее время ответа для ведущего узла на предпоследнем уровне, рассматриваемом как система $M/M/n$, составляет:

$$W_{final} = \frac{1}{\lambda}\left[\sum_{j=1}^{n-1}\frac{\rho^j}{(j-1)!} + \frac{\rho^n(n^2-n\rho+\rho)}{(n-1)!(n-\rho)^2}\right]\times \times\left[\sum_{j=1}^{n-1}\frac{\rho^j}{j!} + \frac{\rho^n}{(n-1)!(n-\rho)}\right]^{-1}.$$

Структура с одним ведущим узлом возможна, пока загруженность ведущего узла меньше единицы: $\rho_1 = \lambda/\mu_1 = \lambda N / c_1 < 1$. Среднее время ответа определяется формулой

$$W = W_1 + W_{final}.$$

Если же $c_1 > \lambda N$, то необходимо уменьшить число ведомых узлов и создать трехуровневую структуру с k узлами на втором уровне. Среднее время ответа может быть рассчитано по формуле:

$$W = W_1 + T_1 + W_2 + W_{final}.$$

Необходимо минимизировать правую часть последнего выражения по k. С учетом условия отсутствия перегрузки ведущих узлов допустимым диапазоном для k является

$$\sqrt{\frac{\lambda N}{c_2}} < k < \frac{c_1}{\lambda}.$$

В работе предложена простая эвристика для числа ведущих узлов в промежуточном слое. Для получения значения k производится минимизация выражения загруженности ведущих узлов первого и второго уровней

$$k = \sqrt[3]{\frac{2Nc_1}{c_2}}.$$

Данное значение, как указывается в работе, превышает оптимальное, однако время ответа системы в обоих случаях отличается незначительно. Принимая во внимание другие приближения, сделанные в работе, эвристически полученное значение *k* можно принять за начальное приближение в построении оптимальной иерархической структуры системы.

С нашей точки зрения, основная ценность работы состоит в использовании нескольких моделей СМО для описания одной вычислительной системы. Для описания процесса опроса состояния ведомых узлов и процесса обработки заявок ведомыми узлами используются различные модели, параметры которых связаны между собой.

1.3. Задача о минимальном времени ответа

В работе [12] ставится задача минимизации времени ответа распределенной системы. Система моделируется сетью массового обслуживания, состоящей из серверов $M/M/1$, объединенных в сеть с произвольной топологией. Характеристики производительности узлов могут быть различными, полагается только, что каждая заявка может быть обработана на любом сервере. Заявки извне поступают на сервер *i* в виде пуассоновского потока интенсивности ϕ_i. Заявка, поступающая на сервер *i* извне, может быть либо обработана на данном сервере, либо передана на другой сервер *j* с интенсивностью x_{ij}; полученная с другого сервера заявка далее не передается и обрабатывается на данном сервере, результат обработки отправляется обратно. Решение о передаче заявки на другой сервер не зависит от состояния системы, в этом смысле задача является статической. Обработанные заявки покидают систему. Интенсивность обработки заявок на *i*-ом сервере обозначается через β_i.

Время ответа для заявки складывается из времени ожидания в очереди, времени обработки и, возможно, задержки при передаче. Среднее время пребывания заявки на сервере *i* обозначается $F_i(\beta_i)$, при этом требуется, чтобы функция $F_i(\beta_i)$ была выпуклой, возрастающей и дифференцируемой. Коммуникационные задержки описываются матрицей $G_{ij}(x)$, $x = [x_{km}]$. Задержка, возникающая при обработке заявки другим сервером, складывается из задержки при передаче заявки другому серверу и передаче ответа обратно. Функции $G_{ij}(x)$ должны быть дифференцируемыми, неубывающими и выпуклыми.

Время пребывания заявки в системе (время ответа) обозначается $D(\beta, x)$, $\beta = [\beta_i]$. В работе предполагается, что построенная сеть является сепарабельной и результирующее распределение заявок в системе представимо в виде произведения распределений индивидуальных серверов [10]. Поэтому среднее время ответа представляется в виде суммы:

$$D(\beta, x) = \sum_{i=1}^{n} \frac{\beta_i}{\Phi} F_i(\beta_i) + \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{x_{ij}}{\Phi} G_{ij}(x), \quad \Phi = \sum_{i=1}^{n} \phi_i$$

Задача состоит в минимизации среднего времени ответа $D(\beta, x)$ по отношению к β и *x*.

Задача оптимизации ставится следующим образом.

$$\min_{\beta,\, x} D(\beta, x),$$

$$\sum_{i=1}^{n} \beta_i = \Phi,$$

(закон сохранения потока для системы)

$$\beta_i + \sum_{j=1}^{n} x_{ij} = \phi_i + \sum_{j=1}^{n} x_{ji}, \quad i = 1..n,$$

(закон сохранения потока для сервера)

$$\beta_i \geq 0, \quad i = 1..n,$$

$$x_{ij} \geq 0, \quad i, j = 1..n.$$

В работе подразумевается, что задержка передачи заявок зависит только от суммарной интенсивности передачи заявок по сети $G = G(\lambda), \quad \lambda = \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}$ и также выполняется неравенство треугольника:

$$G_{ij}(\lambda) \leq G_{ik}(\lambda) + G_{kj}(\lambda).$$

Исходная задача эквивалентна следующей

$$\min_{\beta,\ x} D(\beta,x),\quad D(\beta,x)=\sum_{i=1}^{n}\frac{\beta_i}{\Phi}F_i(\beta_i)+\frac{\lambda}{\Phi}G(\lambda),$$

$$\sum_{i=1}^{n}\beta_i=\Phi,$$

$$\beta_i\ge 0,\quad i=1..n,$$

$$\lambda=\frac{1}{2}\sum_{i=1}^{n}|\phi_i-\beta_i|.$$

Так как результат решения не зависит от конкретных путей передачи заявок между узлами, выбираются фиксированные значения для интенсивностей передачи заявок:

$$x_{ij}=\begin{cases}\dfrac{\beta_j-\phi_j}{\lambda}(\phi_i-\beta_i),\ \text{если } \phi_i>0,\beta_j>0\\ 0,\ \text{иначе}\end{cases}.$$

Для сетей серверов $M/M/1$ задержка передачи заявок полагается равной:

$$G(\lambda)=\frac{1}{1/t-\lambda/2},\quad \lambda<\frac{2}{t},$$

где *t* - среднее время передачи заявки и получения ответа. Данное положение соответствует моделированию канала обмена как системы $M/M/1$ с интенсивностью обслуживания $1/t$ и входящим потоком интенсивности $\lambda/2$ (половина, вследствие получения ответа). Производная среднего времени передачи $\lambda G(\lambda)$ по интенсивности равна:

$$g(\lambda)=\frac{d}{d\lambda}\big(\lambda G(\lambda)\big)=\frac{1/t}{(1/t-\lambda/2)^2},\quad \lambda<\frac{2}{t}.$$

Обозначим

$$f_i(\beta_i)=F(\beta_i)+\beta_i\frac{\partial}{\partial\beta_i}F(\beta_i).$$

Вводятся также следующие обозначения:

$$\beta=\sum_{i=1}^{n}\beta_i,$$

$R_d=\{i:\ \beta_i=0\}$ - простаивающие серверы,

$R_a=\{i:\ 0<\beta_i<\phi_i\}$ - загруженная серверы,

$N=\{i:\beta_i=\phi_i\}$ - нейтральные серверы,

$S=\{i:\beta_i>\phi_i\}$ - серверы-стоки,

где под $f_i^{-1}(\alpha)$ подразумевается обратная функция.

При введенных обозначениях оптимальное решение должно удовлетворять условиям:

$$f_i(\beta_i)\ge\alpha+\beta g(\lambda),\ \beta_i=0\ (i\in R_d),$$

$$f_i(\beta_i)=\alpha+\beta g(\lambda),\ 0<\beta_i<\phi_i\ (i\in R_a),$$

$$\alpha\le f_i(\beta_i)\le\alpha+\beta g(\lambda),\beta_i=\phi_i\ (i\in N),$$

$$f_i(\beta_i)=\alpha,\ \beta_i>\phi_i\ (i\in S).$$

Оно обязано также удовлетворять уравнению баланса потока

$$\sum_{i\in R_a}f_i^{-1}(\alpha+\Phi G'(\lambda))+\sum_{i\in N}\phi_i+\sum_{i\in N}f_i^{-1}(\alpha)=\Phi=\sum_{i=1}^{n}\phi_i$$

где Φ – суммарный поток заявок, попадающих в систему, R_a – множество активных, то есть обрабатывающих заявки, серверов. Из последнего уравнения численно, так как функции $f_i(\beta_i)$ в общем случае не определены, находится множитель Лагранжа α.

В работе приводится алгоритм для параметрического по *t* исследования оптимального распределения заявок по серверам сложности $O(n^2M)$ и алгоритм построения оптимального распределения для фиксированного *t* сложности $O(nM\log n)$, где *M*- сложность нахождения оптимального решения, как указано выше.

Достаточно понятная версия алгоритма приводится работе [11]:

1. Упорядочить узлы по возрастанию значений $f_i(\phi_i)$, $O(n\log n)$;

2. Определить множитель Лагранжа α, $O(nM)$. Для этого вычислить

$$S(\alpha)=\{i:\alpha>f_i(\phi_i)\},$$

$$\lambda_S(\alpha)=\sum_{i\in S(\alpha)}\left(f_i^{-1}(\alpha)-\phi_i\right),$$

$$R_d(\alpha)=\{i:f_i(0)\ge\alpha+g(\lambda_S(\alpha))\},$$

$$R_a(\alpha)=\{i:f_i(\phi_i)>\alpha+g(\lambda_S(\alpha))>f_i(0)\},$$

$$\lambda_R(\alpha)=\sum_{i\in R_d(\alpha)}\phi_i+\sum_{i\in R_d(\alpha)}\left(\phi_i-f_i^{-1}(\alpha+g(\lambda_S(\alpha)))\right);$$

3. Оптимальное решение после этого вычисляется по формулам $O(nM)$:

$$\beta_i = 0, \quad i \in R_d(\alpha),$$
$$\beta_i = f_i^{-1}(\alpha + g(\lambda_S(\alpha))), \quad i \in R_a(\alpha),$$
$$\beta_i = \phi_i, \quad i \in N(\alpha),$$
$$\beta_i = f_i^{-1}(\alpha), \quad i \in S(\alpha).$$

Для частного случая отсутствия задержек при передаче заявок между серверами в работе [13] было получено аналитическое решение:

$$T^*_{time} = \delta + \frac{R^I_\Sigma - |I|\left(\omega^I_\Sigma - \lambda\right)}{\lambda\left(\omega^I_\Sigma - \lambda\right)} =$$
$$= \delta + \frac{R^I_\Sigma - |I|\,\omega^I_\Sigma\left(1 - U^I\right)}{\lambda\omega^I_\Sigma\left(1 - U^I\right)}, \; U^I = \frac{\lambda}{\omega^I_\Sigma}$$

$$p^*_i = \frac{1}{\lambda\tau_i}\left(1 - \frac{\omega^I_\Sigma - \lambda}{R^I_i}\right) = \frac{\omega_i}{R^I_i} + \frac{\omega_i}{\lambda}\left(1 - \frac{\omega^I_\Sigma}{R^I_i}\right), \quad R^I_i > \omega^I_\Sigma - \lambda$$

$$p^*_i = 0, \; R^I_i \le \omega^I_\Sigma - \lambda$$

$$I = \left\{i : 1 \le i \le i^*(\lambda)\right\},$$

$$i^*(\lambda) = \min_{i=1..n}\; i : \tau_i > \frac{R^I_\Sigma}{\left(\omega^I_\Sigma - \lambda\right)^2} \Leftrightarrow \omega_i < \frac{\left(\omega^I_\Sigma - \lambda\right)^2}{R^I_\Sigma}$$

$$R^I_j = \sum_{i \in I}\sqrt{\omega_i\omega_j}, \; R^I_\Sigma = \sum_{j \in I} R^I_j$$

$$\omega^I_\Sigma = \sum_{i \in I}\omega_i, \; \omega_i = \frac{1}{\tau_i},$$

здесь τ_i - среднее время обработки заявки *i*-ым сервером, λ - интенсивность входящего потока заявок. В работе [13] также предложен алгоритм MinTime построения оптимального распределения заявок по серверам сложности $O(n \log n)$, который не требует численного нахождения множителя Лагранжа.

Алгоритм MinTime.

1. Рассчитать для всех $j = 1..n$ величину $1/\tau_j$ и упорядочить полученные значения по убыванию. $\omega_j = \frac{1}{\tau_j}$

2. Для всех $i = 1..n$ рассчитать

$$\omega^i_\Sigma = \sum_{j=1}^{i}\omega_j, \; r^i_\Sigma = \sum_{j=1}^{i}\sqrt{\omega_j}$$

3. Найти точку активации для каждого сервера по формуле, для всех $i = 1..n$

$$\lambda^i = \omega^{i-1}_\Sigma - \sqrt{\omega_i}\, r^i_\Sigma$$

4. Рассчитать для всех $i = 1..n$, $j = 1..n$

$$R^i_j = r^i_\Sigma\sqrt{\omega_j}, \; R^i_\Sigma = \sum_{j=1}^{i} R^i_j$$

5. При заданном значении интенсивности входящего потока λ

$$p^*_j = \frac{\omega_j}{R^I_j} + \frac{\omega_j}{\lambda}\left(1 - \frac{\omega^I_\Sigma}{R^I_j}\right), \; i = \min_{j=1..n}\left\{j : \lambda < \lambda^j\right\},$$

при этом ожидаемое время ответа системы

$$T^*_{time} = \delta + \frac{R^i_\Sigma - i\left(\omega^i_\Sigma - \lambda\right)}{\lambda\left(\omega^i_\Sigma - \lambda\right)}$$

6. При изменении значения λ повторить с шага 5

Для рассматриваемой системы [13] приведем анализ некоторых распространенных алгоритмов: Round Robin, Weighted Round Robin, Least Load, MinTime и Least Queue.

- Round Robin (RR) – распределение заявок происходит по порядку, все серверы получают в среднем одинаковое число заявок.
- Weighted Round Robin(WRR) – распределение заявок по порядку, при котором каждый сервер обладает некоторым «весом» или приоритетом - равномерное с весами.
- Least Queue – динамический алгоритм с обратной связью. Заявка будет направлена тому серверу, очереди к которому в данный момент ожидает наименьшее число заявок.
- Least Load – динамический алгоритм с обратной связью. Заявка направляется на тот сервер, который менее всего загружен. Величи-

на загруженности сервера может определяться, например, по времени соединения с сервером.

- MinTime – динамический алгоритм без обратной связи. Распределение запросов по серверам рассчитывается для текущей величины интенсивности входящего потока заявок.

Рассмотрение системы в стационарном или установившемся режиме, разумеется, не учитывает преимущества динамических алгоритмов с обратной связью, способных подстраиваться под систему. В установившемся режиме алгоритмы можно описать следующим образом.

Используются следующие обозначения:

ω_i - интенсивность обслуживания,

$p_i = \frac{\lambda_i}{\lambda}$ - вероятность направления заявки на *i*-й сервер,

$\lambda_i = \phi_i + \sum_{j=1}^{n} x_{ji}$ - интенсивность потока заявок, попадающего на сервер,

$\rho_i = \frac{\lambda_i}{\omega_i}$ - загруженность *i*-го сервера.

Каждый из них будет стремиться к тому, чтобы выполнялось соотношение для всех *i*:

- Round Robin, $p_i = 1/n = const$, $\lambda_i = \lambda / n$,

$$T = \sum_{i=1}^{n} \frac{1}{n\omega_i - \lambda}.$$

- WRR, величина $\frac{p_i}{w_i} = const$, w_i - вес *i* -го сервера,

$$T = \sum_{i=1}^{n} \frac{1}{\frac{w}{w_i}\omega_i - \lambda},\ w = \sum_{i=1}^{n} w_i.$$

- Least Queue, длина очереди на всех серверах одинакова:

$$EQ_i = \frac{\rho_i^2}{1-\rho_i} = const,$$

$$i = \min_{j=1..n}\left\{ j : \lambda < \lambda^j \right\},$$

$$p_j = \frac{i\cdot\omega_j - \omega_\Sigma^i}{i\cdot\lambda} + \frac{1}{i}, \text{ при } \lambda > \lambda^j = \omega_\Sigma^{j-1} - (j-1)\cdot\omega_j,$$

$$p_j = 0, \text{ при } \lambda < \lambda^j,$$

$$T = \frac{i}{\omega_\Sigma^i - \lambda}.$$

- LeastLoad, нагрузка на все серверы одинакова:

$$1 - U_i = \rho_i = const,\ p_i = \frac{\omega_i}{\omega_\Sigma},\ T = \frac{n}{\omega_\Sigma - \lambda}.$$

- MinTime, среднее время ответа системы минимально.

В работе далее указывается, что алгоритмы LeastQueue и LeastLoad приводят к одинаковому времени ответа системы несмотря на то, что распределения заявок различны. Для алгоритма WRR можно выбрать веса таким образом, чтобы в стационарном режиме он был идентичен LeastLoad в рассматриваемой модели. Результаты расчета среднего времени ответа системы в зависимости от интенсивности входящего потока приведены на Рис.3. Исследуемая система состоит из трех серверов:

Сервер 1, $\omega_1 = 10$, 100мс на обработку заявки

Сервер 2, $\omega_2 = 2$, 500мс на обработку заявки

Сервер 3, $\omega_3 = 1$, 1000мс на обработку заявки

1.4. Метод анализа по средним значениям (MVA)

Метод анализа по средним значениям (MVA) применяется для расчета характеристик закрытых сетей массового обслуживания. Существует точный метод MVA и приближенный. В данном разделе мы рассмотрим точный метод MVA для закрытых сетей с одним классом заявок [14].

Метод использует следующие соотношения:

Закон Литтла, примененный к сети в целом:

$$X(N) = \frac{N}{Z + \sum_{k=1}^{K} R_k(N)},$$

где $X(N)$ - пропускная способность сети, $R_k(N)$ - время пребывания заявки на сервере *k*, если в сети присутствует *N* заявок (пользовате-

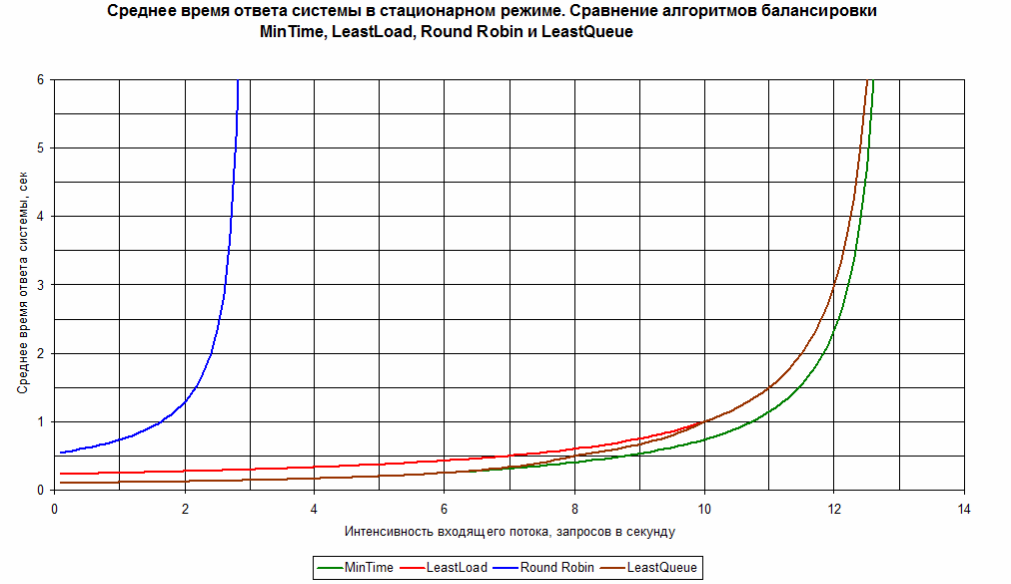

Рис.3. График зависимости времени ответа от интенсивности входящего потока заявок при использовании различных алгоритмов балансировки

лей), Z- среднее время ожидания заявки. По сути, Z выражает среднее время «раздумий» пользователя перед повторным обращением к системе.

Закон Литтла, примененный к каждому серверу в отдельности:

$$Q_k(N) = X(N)R_k(N).$$

В обоих случаях перед применением закона Литтла необходимо знать время пребывания заявок на всех серверах.

Уравнения для времени пребывания:

$$R_k(N) = \begin{cases} D_k & \text{– для серверов с задержкой,} \\ D_k\left(1 + A_k(N)\right) & \text{– для серверов с очередью,} \end{cases}$$

где $A_k(N)$ - среднее число заявок на сервере k при поступлении новой заявки на сервер.

Для расчета необходимо сначала найти $A_k(N)$ для всех серверов сети. Точный метод MVA применим к закрытым сепарабельным сетям. Можно считать, что сепарабельность позволяет рассматривать серверы сети независимо друг от друга. Условия сепарабельности сети будут приведены позднее. Для таких сетей справедливо соотношение [15]:

$$A_k(N) = Q_k(N-1).$$

Другими словами, средняя длина очереди, наблюдаемая при поступлении новой заявки, когда в сети имеется N заявок, равна среднему по времени значению длины очереди на данном сервере, когда в сети N-1 заявок.

Точный алгоритм MVA состоит в итеративном применении представленных уравнений с увеличением числа заявок в системе. В качестве начального условия принимается, что если в сети нет заявок, то все очереди серверов пусты.

for k =1 to K do $Q_k = 0$
for n=1 to N do
begin
for k =1 to K do $R_k(N)$=

$$\begin{cases} D_k & \text{– для серверов с задержкой,} \\ D_k\left(1 + A_k(N)\right) & \text{– для серверов с очередью,} \end{cases}$$

$$X = \frac{n}{Z + \sum_{k=1}^{K} R_k}$$

for k =1 to K do $Q_k = XR_k$
end

Известны достаточные условия сепарабельности сети:

- для каждого сервера выполняется условие сохранения потока, заявки не появляются и не исчезают на серверах

- ординарность сети, никакие две заявки не «изменяют своего состояния» одновременно, то есть не завершают обработку или прибывают в систему.

Следующие условия называются предположениями однородности:

- однородность маршрутизации - переходы заявок между серверами описываются постоянными значениями вероятностей передачи заявки от одного сервера другому;

- однородность серверов, время обработки заявки на сервере зависит только от числа заявок в его очереди;

- однородность входящего потока, поступление заявок в систему не зависит от числа заявок в системе и их распределения между серверами.

Сети, удовлетворяющие приведенным предположениям, называются сепарабельными или сетями Джексона [15]. Для приведенного алгоритма требуется дополнительное предположение:

- однородность времени обработки - время обработки заявок на сервере не зависит от числа заявок в очереди; время пребывания заявки на сервере определяется, таким образом, временем ожидания в очереди и постоянным временем обработки.

2. Анализ алгоритмов распределения нагрузки

2.1. Один из подходов к классификации алгоритмов балансировки

В данном разделе приводится одна из возможных классификаций алгоритмов балансировки и проводится анализ трех существующих систем [16].

Алгоритмы распределения нагрузки можно разделить на статические, динамические и адаптивные. Динамические алгоритмы распределения нагрузки используют информацию о состоянии системы, например, о загруженности узлов, для принятия решений о распределении нагрузки в процессе работы системы. Статические алгоритмы подобной информации не используют. В статических алгоритмах решения о распределении нагрузки реализованы заранее, до начала работы системы, и не изменяются в процессе работы. Примером статического алгоритма может служить алгоритм Round Robin, распределяющий задачи между узлами по очереди.

Динамические алгоритмы благодаря использованию информации о состоянии системы имеют возможность обогнать по эффективности статические алгоритмы, например, простой динамический алгоритм может быть идентичен по эффективности сложному статическому. Вследствие того, что динамически алгоритмы должны собирать, хранить и анализировать информацию о состоянии системы, им свойственны большие накладные расходы на балансировку по сравнению со статическими алгоритмами.

Адаптивные алгоритмы являются частным случаем динамических алгоритмов и в зависимости от условий и априорного знания о свойствах алгоритмов балансировки выбирают наиболее подходящий из них.

Информация о состоянии системы является важным элементом работы динамического алгоритма. Существуют различные методы сбора данных о загруженности системы и построения характеристик загруженности. Загруженность узла может определяться, например, как доля процессорного времени, затраченная на решение задачи, как количество задач в очереди узла, либо как вероятность обнаружить узел занятым решением задачи в какой-либо момент времени.

По отношению к моменту начала передачи задач различают вытесняющие и невытесняющие алгоритмы балансировки. Вытесняющие алгоритмы могут передавать задачи, решенные частично, в то время как не вытесняющие могут распределять только задачи, выполнение которых еще не начато. Примером вытесняющего алгоритма может служить алгоритм планирования процессов в системе Unix.

Динамические алгоритмы балансировки различаются степенью централизованности. Алгоритмы могут быть централизованными, иерархическими, полностью распределенными либо совмещать указанные подходы. Централизованные алгоритмы потенциально менее надежны,

чем полностью распределенные, так как сбой центрального компонента приведет к сбою всей системы. Решением данной проблемы может служить поддержание избыточного числа компонентов, дублирующих друг друга. Другая слабость централизованных алгоритмов состоит в том, что центральный компонент представляет собой возможное узкое место системы. В то время как иерархические алгоритмы позволяют частично решить обе проблемы, полное решение достигается только распределенными алгоритмами. С другой стороны, централизованные алгоритмы проще контролировать и для них проще производить отладку.

Обычно динамический алгоритм балансировки содержит четыре элемента:

- политику балансировки (transfer policy);
- алгоритм выбора партнера (location policy);
- алгоритм выбора задачи для передачи (selection policy);
- механизм сбора необходимой информации о состоянии системы (information policy).

Политика балансировки определяет, является ли узел объектом балансировки. Различные виды политик балансировки используют пороговые значение загруженности узлов, отклонение величины нагрузки узла от среднего значения по системе и другие методы.

Алгоритм выбора задачи определяет, какую задачу необходимо передать. При выборе задачи для отправки алгоритм может учитывать, что накладные расходы, связанные с пересылкой задачи, должны быть минимальны, сложность выполнения задачи должна быть большой, число связей в задаче с локальными ресурсами должно быть минимально.

Алгоритм выбора партнера отвечает за выбор подходящего узла для операции балансировки. Распространенной техникой в распределенных алгоритмах является опрос (polling) узлов. Опрос может быть последовательным или параллельным, использовать результаты предыдущих опросов. В централизованных алгоритмах узел обращается к специализированному координатору для определения подходящего партнера для балансировки. Координатор собирает и поддерживает в актуальном состоянии информацию о загруженности узлов системы, а алгоритм поиска партнера использует эти данные.

Механизм сбора информации о загруженности системы определяет, когда собираются данные о загруженности, что является источником информации и где информация хранится. Выделяют несколько классов механизмов сбора информации.

1. Сбор данных по необходимости. Распределенные алгоритмы данного класса собирают информацию о загруженности, когда узел нуждается в балансировке нагрузки. Они подразделяются на инициируемые отправителем, получателем и симметрично инициируемые. В алгоритмах, инициируемых отправителем, узел, передающий задачи, ищет получателей, которым он может передать часть задач. В алгоритмах, инициируемых получателем, узел, получающий задачи, заимствует задачи у отправителя. В симметричных алгоритмах используется комбинация упомянутых подходов.

2. Периодический сбор данных. Алгоритмы данного класса могут быть как централизованными, так и распределенными. В зависимости от собранных данных, алгоритм инициирует балансировку нагрузки.

3. Сбор данных по изменению состояния. В системах, реализующих алгоритмы данного класса, узлы сами распространяют информацию об изменении загруженности при изменении внутреннего состояния. В случае распределенных алгоритмов данные направляются соседним узлам, в случае централизованных алгоритмов данные направляются координатору.

Под стабильностью алгоритма балансировки или системы, его использующей, обычно понимают одну из двух характеристик:

- системная устойчивость, то есть недопустимость ситуации, когда отдельные узлы системы перегружены, в то время как остальные простаивают или недогружены;
- алгоритмическая стабильность, выраженная в том, что алгоритм не совершает бесполезных действий с ненулевой вероятностью, например, не передает задачу по кругу между узлами так, что задача не выполнится никогда.

2.2. Примеры анализа алгоритмов балансировки некоторых систем

Система Satin, описываемая в работе Ньюпорта [17], предназначена для выполнения программ,

решающих задачи по принципу «разделяй и властвуй» на системах с распределенной памятью. Satin реализована на Java и предоставляет простые примитивы для организации параллельной работы программ: аннотирование метода как пригодного для параллельного выполнения, асинхронный вызов метода и команда синхронизации. При асинхронном вызове метода создается соответствующая вызову подзадача, которая помещается в деку. Satin использует модифицированный компилятор Manta в родной (native) код целевой платформы и специальную среду выполнения для реализации представленных примитивов. Система Satin использует алгоритм случайного заимствования (Random Stealing, RS) подзадач для распределения нагрузки между локально распределенными узлами. Для случая распределения нагрузки между узлами разных кластеров, соединенных медленной WAN, в другой статье Ньюпорта [18] был предложен алгоритм случайного заимствования с учетом кластеров (Cluster-Aware Random Stealing, CRS). Между узлами одного кластера алгоритм выполняет синхронное локальное заимствование. Задачи из других кластеров заимствуются асинхронно и заранее, например, при превышении количества отказов в локальном заимствовании. Согласно экспериментальным данным, новый алгоритм CRS позволяет улучшить производительность системы из двух глобально распределенных кластеров до практически одного объединенного кластера, использующего алгоритм RS.

Алгоритм случайного заимствования является полностью распределенным алгоритмом балансировки. Работа алгоритма инициируется получателем нагрузки по окончании решения задачи, партнер по балансировке выбирается равномерно случайно из соседей данного узла, сбор информации о загруженности узлов осуществляется по требованию. Алгоритм рассматривает все задачи для передачи при балансировке. Алгоритм заимствования с учетом кластера для удаленных кластеров использует сбор информации по изменению состояния. Доказательство стабильности алгоритма случайного заимствования для систем с полным графом структуры приведено в работе [19].

С учетом места алгоритма в классификации на основании исследований, представленных в работе [18], можно сделать вывод, что алгоритм является устойчивым, будет малоэффективен для случая средне и слабо загруженной системы, а также для решения большого числа мелких задач, так как при балансировке передается только одна задача. Для сильной загрузки алгоритм будет высокоэффективен, как и показывают исследования, проведенные Ньюпортом.

Система PICO, описываемая Экштейном в работе [20] Рис.4, представляет собой среду (framework), написанную на C++ с использованием MPI, для реализации общих алгоритмов метода ветвей и границ. Среда PICO предоставляет механизм реализации различных алгоритмов благодаря применению принципа обращения зависимости (Inversion of Control, IoC) [21] для взаимодействия среды с кодом алгоритма ветвей и границ. Согласно описанию, предоставляемому авторами работы, система PICO разрабатывалась как результат обобщения опыта разработки других систем, решающих задачи оптимизации с помощью алгоритмов метода ветвей и границ в распределенных средах.

Суть алгоритма ветвей и границ состоит в разбиении области определения минимизируемой функции (ветвление), вычислении оценки значения функции на каждой из подобластей снизу, отбрасывании (ограничении) тех подобластей, для которых оценка больше наименьшего рассчитанного значения функции в какой-либо точке или оценки (рекорд).

В системе PICO подзадача имеет шесть возможных состояний: готова к оценке, оценка происходит, оценена, ветвление происходит, ветвление завершено и отброшена. Подзадачи всегда создаются в состоянии «готова к оценке». Это означает, что вычисления для оценки созданной задачи еще не производились. После завершения оценки подзадачи PICO может принять решение о ветвлении подзадачи, в результате ветвления могут быть созданы новые подзадачи. После того, как создана последняя дочерняя подзадача, состояние подзадачи становится «отброшена». Подзадачи могут также перейти в состояние «отброшена» в результате отсева из других состояний.

Каждый узел системы PICO содержит конфигурируемые пользователем локальный пул подзадач и обработчик, которые определяют

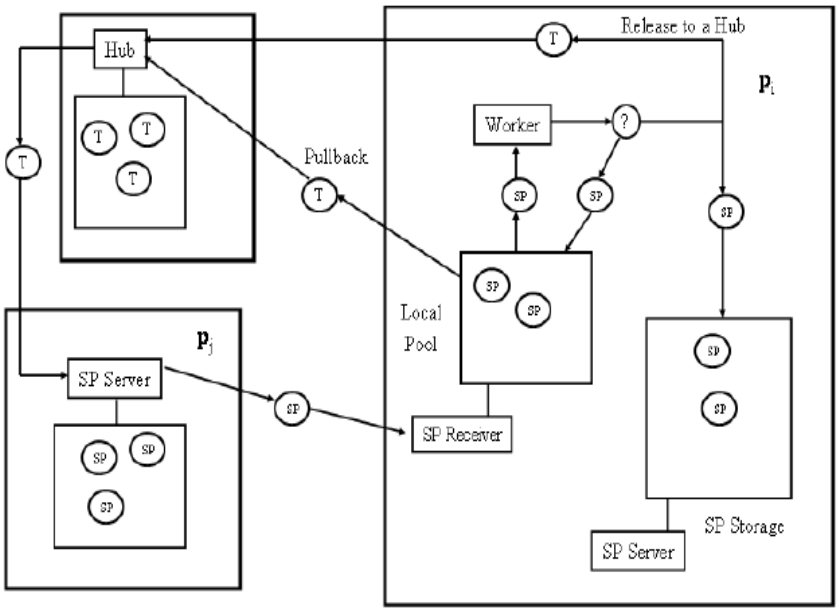

Рис.4. Структурная схема PICO

конкретный алгоритм ветвей и границ. PICO поддерживает локальный пул подзадач одного из трех типов: очередь с приоритетами по оценке подзадачи, стек и очереди (FIFO). Имеется возможность использовать пользовательскую реализацию. Соответствующие типам упорядочивания подзадач в пуле алгоритмы - эвристический поиск с возвратом (Best First Search), поиск в глубину (Depth First Search) и поиск в ширину (Breadth First Search). Заметим, что в пуле могут находиться подзадачи в разных состояниях.

С точки зрения организации параллельного решения задачи система PICO представляет собой иерархическую систему, использующую парадигму «ведущий-ведомый» (Master-Slave) с локальным пулом подзадач, балансировкой меток (идентификаторов-ссылок) подзадач и прямым взаимодействием узлов при передаче подзадач. Система состоит из трех компонентов: рабочего процесса, распределяющего процесса (распределитель) и хранилища подзадач. Рабочий процесс содержит локальный пул подзадач, принимающий задачи от распределителя. Обработчик извлекает задачи из локального пула для решения. Хранилище подзадач может быть использовано для кэширования подзадач из локального пула и передачи подзадач, определяемых полученными метками, по требованию распределителя. Задачи для балансировки выделяются рабочим процессом и с некоторой вероятностью перемещаются в хранилище подзадач, при этом метки перемещенных подзадач передаются распределителю. Вероятность передачи задачи выбирается в зависимости от загрузки узла по отношению к другим узлам кластера из указанных до старта значений минимальной, целевой и максимальной вероятности. Распределяющий процесс передает находящиеся в его распоряжении метки подзадач дочерним узлам так, чтобы в локальном пуле каждого узла содержалось достаточное количество подзадач. Кроме того, распределитель реализует алгоритм оценивания сложности отдельной подзадачи и стремится сделать распределение «эффективной» загрузки узлов равномерным в соответствии с конфигурируемым уровнем отклонения. При балансировке, в случае недостаточного количества меток, на распределителе предусмотрен механизм перераспределения нагрузки. Если узел определяет, что число подзадач в его локальном пуле превышает более чем в заданное число раз среднее количество подзадач в пуле на узлах кластера, он отсылает часть подзадач распределителю.

Для балансировки нагрузки между кластерами узлов, контролируемыми распределителями, PICO реализует вариант алгоритма ран-

деву (Rendezvous). Нагрузка узла полагается равной суммарной трудоемкости задач, имеющихся на узле. Для оценки трудоемкости задач, имеющихся на кластере, система PICO использует функцию L, которая в фиксированный момент времени выражается формулой:

$$L(c) = \sum_{P \in C(c)} |\bar{z}(c) - z(P,c)|^{\rho} \cdot$$

Данная функция представляет сумму разностей рекорда и оценки подзадачи по всем подзадачам в локальном пуле в некоторой степени ρ.

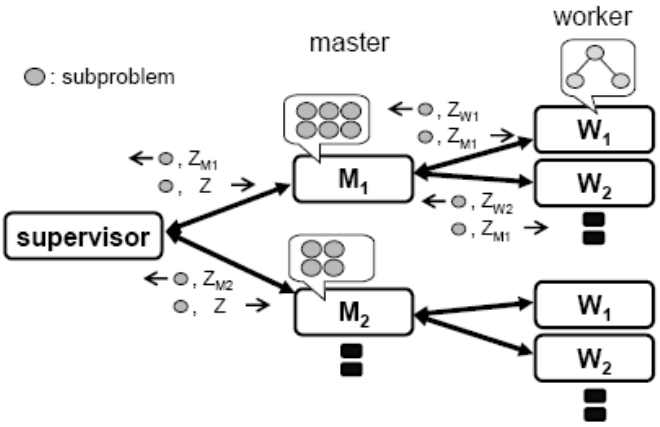

Рис.5. Схема иерархической системы «ведущий-ведомый»

Наиболее простой вариант алгоритма рандеву основан на разделении узлов относительно усредненного значения загрузки по кластеру. Недогруженные и перегруженные узлы нумеруются отдельно по возрастанию величины отклонения. После этого осуществляются парные обмены подзадачами между узлами с соответствующими номерами.

В системе PICO в балансировке участвуют не все узлы, а лишь те, загрузка которых отклоняется от средней более чем на заданную величину. Кроме того, алгоритм подразумевает наличие глобальной информации о загрузке узлов в системе. Информация о загрузке узлов в системе PICO распространяется через сложный механизм, основанный на построении дерева связей между узлами. Подробнее об алгоритме балансировке можно узнать в статье [20].

В статье Кенто Айда [22] предложен алгоритм распределения нагрузки для систем, построенных с использованием парадигмы «ведущий-ведомый» и имеющих иерархическую структуру (Рис.5). Система состоит из трех типов узлов:

- рабочих, которые содержат только обработчик подзадач;

- мастеров, содержащих локальный пул подзадач;

- координатора, обеспечивающего балансировку подзадач между мастерами.

Система работает следующим образом. Мастер определяет, какой из его дочерних узлов простаивает и направляет ему подзадачу из локального пула либо передает подзадачи координатору для балансировки. Рабочий процесс принимает задачи от мастера, решает их последовательно и отправляет результат обратно мастеру. Координатор опрашивает загрузку мастеров, которая учитывает загрузку их дочерних узлов, и производит обмен подзадачами, передавая их через себя.

Координатор вычисляет среднее количество подзадач по мастерам, составляет вектор отклонений от среднего, после этого рассылает мастерам оповещение о необходимости передать либо получить подзадачи от координатора. Алгоритм балансировки инициируется по истечении фиксированного периода времени.

Алгоритм балансировки можно охарактеризовать как централизованный, инициируемый координатором, использующий свободные находящиеся в пуле подзадачи для балансировки. Алгоритм собирает информацию о загрузке через фиксированные промежутки времени. Отметим, что алгоритм будет плохо масштабируемым по числу мастеров в системе, а также при решении большого числа простых подзадач, так как число узлов на мастере будет изменяться быстрее периода опроса состояния. В результате, координатор будет иметь неверные данные о загрузке. Кроме того, отсутствие локального пула подзадач на дочерних узлах приводит к большим накладным расходам по передаче подзадач.

Литература

1. Lauwereins, R., Peperstraete, J. Queuing theoretical analysis of processor utilization in parallel computers. // Comput. Syst. Sci. Eng. V. 8, N. 1, pp. 13-23, 1993
2. F. Bacelli, G. Balbo, R.J. Boncherie, J. Campos, G. Chiola. Annotated bibliography on stochastic Petri nets. //

In O.J. Boxma, G.M. Koole, eds., Performance Evaluation of Parallel and Distributed Systems – Solution Methods, CWI, Amsterdam, 1994
3. T. Murata. Petri nets: Properties, analysis and applications. / Proceedings of the IEEE, V. 77, pp. 541-580, 1989
4. Kleinrock L. Queuing Systems, Volume 1. Theory. Wiley, New York, 1975
5. Б.В. Гнеденко, И.Н. Коваленко "Введение в теорию массового обслуживания". М.: Наука.1966
6. F. Baskett, K. M. Chandy, R. R. Muntz and F.G. Palacios. Open, closed and mixed networks of queues with different classes of customers. / Journal of the ACM, V. 22: pp. 248–260, 1975
7. Kameda H., Pourtallier O. Paradoxes in distributed decisions on optimal load balancing for networks of homogeneous computers. / Journal of the ACM (JACM), v.49 n.3, pp.407-433, 2002
8. Kameda, H., Altman, E., Kozawa, T., and Hosokawa, Y. 2000. Braess-like paradoxes in distributed computer systems. IEEE Trans. Automatic Contr. 45, 1687—1691
9. Palmer, J., Mitrani, I. Optimal Tree Structures for Large Service Networks. / In 1st EuroNGI Conference on Next Generation Internet Networks (NGI 2005), IEEE Computer, pp. 91-98, 2005
10. Nelson, R. D. The mathematics of product form queuing networks. // ACM Comput. Surv. V. 25, N. 3, pp. 339-369, 1993
11. Chonggun Kim, Hisao Kameda, "An Algorithm for Optimal Static Load Balancing in Distributed Computer Systems". IEEE TRANSACTIONS ON COMPUTERS, VOL.41, NO.3, MARCH 1992.
12. A. A. Tantawi, D. Towsley. Optimal Static Load Balancing in distributed Computer Systems. / Journal of the ACM, 32(2), 445-465(1985).
13. Хританков А.С. Оптимальное распределение нагрузки открытого типа на совокупность разнородных узлов. / Труды конференции САИТ-2005, Т. 2, стр. 229-234., 2005
14. E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik. Quantitative Computer Performance: Computer System Analysis Using Queuing Network Models. – Prentice Hall, 1984.
15. J.R. Jackson, Jobshop-like Queuing Systems. // Management Science 10, pp. 131-142, 1963
16. N. G. Shivaratri, P. Krueger, and M. Singhal. Load Distributing for Locally Distributed Systems, IEEE Computer, December 1992, pp 33-44
17. R. V. van Nieuwpoort, Jason Maassen, Gosia Wrzesinska, Thilo Kielmann and Henri E. Bal. "Adaptive Load Balancing for Divide-and-Conquer Grid Applications". In Journal of Supercomputing, 2006.
18. R. V. van Nieuwpoort, T. Kielmann, H. E. Bal. Efficient load balancing for wide-area divide-and-conquer applications. In Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (Snowbird, Utah, United States). PPoPP '01. ACM Press, New York, NY, 34-43.
19. Blumofe, R. D. and C. E. Leiserson: 1994, "Scheduling Multithreaded Computations by Work Stealing". In: 35th Annual Symposium on Foundations of Computer Science (FOCS '94). Santa Fe, New Mexico, pp. 356-368
20. J. Eckstein, C. A. Phillips, and W. E. Hart, "PICO: An Object-Oriented Framework for Parallel Branch and Bound", RUTCOR Research Report 40-2000, Rutgers University, Piscataway, NJ (2000).
21. Robert C. Martin "Agile Software Development: Principles, Patterns and Practices". Pearson Education. 2002
22. Kento Aida, Wataru Natsume, Yoshiaki Futakata, "Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm," Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), pp.156-163, May. 2003.
23. А.С. Хританков. «Модели и алгоритмы распределения нагрузки. Модель коллектива вычислителей. Модели с соперником». Информационные технологии и вычислительные системы, №2 2009, с.65-80.

Хританков Антон Сергеевич. Аспирант. Окончил Московский физико-технический институт в 2007 году. Имеет 12 печатных работ. Область научных интересов: оценка производительности вычислительных систем, распределенные вычисления, проектирование программных систем. E-mail: anton.khritankov@gmail.com.