электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 02, февраль 2016

УДК 004.738.2

Распределение задач между исполнителями в высоконагруженной системе

Якиль К.А., *бакалавр*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»

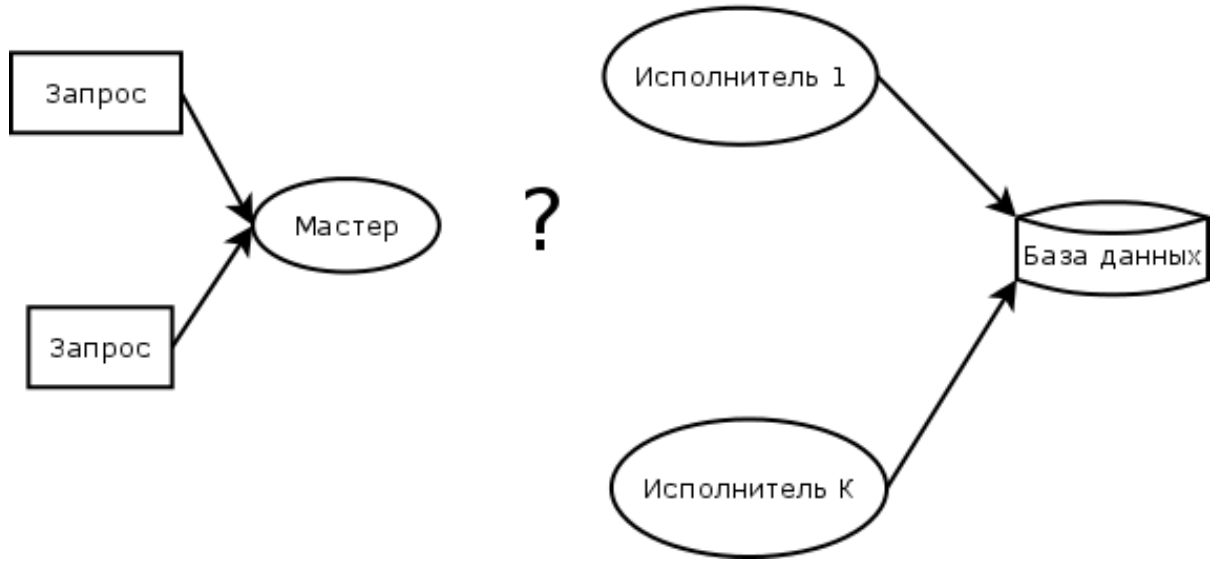
Научный руководитель: Рязанова Н.Ю., к.т.н. доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»
bauman@bmstu.ru

Введение. В настоящее время большинство создаваемых программных средств для широкого использования людьми является высоконагруженными системами. Наиболее часто используемым подходом к разработке программного обеспечения высоконагруженного сервиса является SOA (Service-oriented architecture) [1]. Данный подход рекомендует создавать распределенную систему, состоящую из самостоятельных сервисов, взаимодействующих между собой посредством заранее определенных протоколов и интерфейсов. Нормальное функционирование распределенной системы невозможно без балансировки нагрузки между бекендами (backends) и определения исполнителя ресурсоемкой задачи. В основном используют простые алгоритмы распределения задач (round robin [2], weighted round robin, привязка исполнителей к пользователям). Такие подходы не обеспечивают минимальное время обработки запросов в системе, а некоторые из них не создают и равномерного распределения нагрузки.

Постановка задачи. Рассмотрим систему, в которой запросы на обработку приходят в один компонент распределенной системы, который далее назначает исполнителей для каждого входного запроса. Такой компонент системы принято называть мастером. О запросах известно, какой пользователь его сформировал, каков тип запроса и дополнительные параметры. В системе функционируют исполнители, обозначим их как воркер. Каждый исполнитель характеризуется скоростью обработки и ограничением по используемой памяти. Известно также, что исполнители для обработки запросов

запрашивают дополнительную информацию, которая по пользователю запроса однозначно идентифицирует необходимые для исполнителя данные в хранилище. Обозначим его как DB (data base). Задача мастера определить какому из исполнителей перенаправить запрос так, чтобы время обработки запросов системой было минимально.

Схема взаимодействия компонентов представлена на рисунке.



Алгоритмы распределения задач между исполнителями. Предлагаются к рассмотрению следующие алгоритмы: без расчета времени обработки запроса и с оценкой времени исполнения запроса.

1) Алгоритмы распределения задач между исполнителями без расчета времени обработки запроса:

1. Round Robbin (RR)

Мастер получает запросы и передает их исполнителю согласно алгоритму Round Robin. Данный алгоритм прост в реализации, но может приводить к частой загрузке и выгрузке данных из кеша исполнителя. Может произойти неравномерная загрузка определенных исполнителей, если на их долю выпадут сложные запросы, требующие длительного исполнения. Данная проблема может быть решена введением счетчика ожидающих запросов к каждому исполнителю в мастере и ограничением на этот счетчик (при достижении лимита мастер передает запрос другому исполнителю согласно RR).

2. Общая очередь задач

Мастер формирует общую очередь задач. Исполнитель оповещает мастера, когда он готов обработать новый запрос. Мастер отдает из очереди первый запрос этому исполнителю. В случае наличия кешированных данных каждый исполнитель хранит их в списке и производит выгрузку по алгоритму LRU (least recently used [3]).

Данный подход прост в реализации, но может приводить к частой загрузке и выгрузки данных из кеша исполнителя.

3. Исполнитель обслуживает запросы заранее определенных пользователей

Каждому исполнителю назначается множество пользователей, запросы которых он обслуживает. Согласно этому соответствию мастер перенаправляет получаемые запросы нужному исполнителю. Данный подход достаточно прост в реализации, но не гарантирует равномерной загрузки исполнителей и минимального времени обработки запросов в системе

Данный подход может быть реализован общим списком кешей исполнителей, управляемым мастером по алгоритму LRU (удобно, если есть ограничение по памяти суммарно по всем исполнителям).

Подход может быть реализован и использованием списка кешей по алгоритму LRU каждого исполнителя (при ограничении по памяти каждого исполнителя).

2) Алгоритмы распределения задач между исполнителями с оценкой времени обработки запроса.

Для того, чтобы минимизировать время обработки запросов, необходимо определить временные затраты на обработку одного запроса. В формуле 1 представлен расчет времени пребывания запроса в системе.

$$Treq = T_{Mreq} + T_{m-w\ transfer} + T_{Wreq}, \quad (1)$$

где Treq – время обработки запроса в системе;

T_{Mreq} – время определения мастером исполнителя для запроса;

$T_{m-w\ transfer}$ – время на пересылку запроса от мастера к исполнителю;

T_{Wreq} – время, через которое исполнитель обработает запрос.

T_{Mreq} и $T_{m-w\ transfer}$ можно считать одинаковыми при обработке любого запроса

Поскольку в распределенной системе данные хранятся в отдельном компоненте, а для обработки запросов исполнителями необходимо обращаться за данными в хранилище, нужно учитывать время, затрачиваемое исполнителем на доступ к данным. В SOA зачастую уменьшают время доступа к данным засчет кеширования [4]. В общем случае, следует учитывать возможность кеширования исполнителем данных. Необходимо рассмотреть и ограничения по памяти каждого исполнителя. Таким образом могут возникнуть ситуации, когда необходимо выгрузить из памяти данные, находящиеся в кеше, и подгрузить новые из базы данных. В связи с этим $\mathrm{T_{Wreq}}$ можно представить как сумму времен по формуле 2.

$$\mathrm{T_{Wreq}} = \mathrm{T_{Wrest}} + (\mathrm{T_{Wunload}} + \mathrm{T_{DB_access}}) + \mathrm{T_{W_base_req}}, \qquad (2)$$

где T_{Wrest} – время, через которое исполнитель дообработает ранее поставленные задачи;
$T_{Wunload}$ – время на выгрузку данных из кэша исполнителя;
T_{DB_access} – время доступа к данным из базы данных;
$T_{W_base_req}$ – время непосредственной обработки запроса;
T_{DB_access} считается одинаковым для всех исполнителей.

В свою очередь

$$\mathrm{T_{W_base_req}} = \mathrm{f}(\mathrm{V}, \mathrm{req_{type}}, \mathrm{req_{params}}),$$

где V – скорость обработки единичной операции данным исполнителем,
req_{type} – тип запроса,
req_{params} - параметры запроса,

Функцию f можно определить эвристически, поэтому она считается определенной. T_{Wrest} вычисляется мастером, когда произошла отправка запроса исполнителю. Тогда

$$\mathrm{T_{Wrest}} = \mathrm{T_{Wrest(prev)}} + (\mathrm{T_{Wunload}} + \mathrm{T_{DB_access}}) + \mathrm{T_{W_base_req}}.$$

Для того, чтобы мастер мог вычислить ожидаемое время обработки запроса в системе, в общем случае ему необходимо знать о кешированных данных каждого исполнителя ($T_{Wunload}$).

Таким образом, мастер при поступлении запроса в систему может рассчитать через какое время полученный запрос будет обработан каждым исполнителем.

При накоплении очереди задач из m (m <= n) элементов мастер определяет, какому исполнителю какую задачу назначить при условии, что каждая задача решается ровно одним исполнителем и в текущий момент времени каждому исполнителю назначается ровно одна задача.

Обозначим i-ый запрос через req_i, j-ого исполнителя через w_j, причем j $\in \overline{1,n}$, где n – количество исполнителей.

Строится матрица T = $\{t_{ij}\}$, где t_{ij} – время, через которое i-ый запрос может быть обработан j-ым исполнителем (мастер может определить это время). Если m < n, то матрица T дополняется до квадратной строками, заполненными 0.

Вводятся значения $x_{ij} \in \{0; 1\}$, которые определяют, обрабатывается ли i-ый запрос j-ым исполнителем.

Строится целевая функция – время обработки m запросов в системе – $f = \sum_{j=1}^{n} \sum_{i=1}^{n} xij * tij$, которую необходимо минимизировать. Таким образом поставлена задача о назначениях [5]:

$$\begin{cases} \mathrm{f} = \sum_{\mathrm{j}=1}^{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{xij} * \mathrm{tij} \rightarrow \min \\ \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_{ij}} = 1, \mathrm{j} = \overline{1,\mathrm{n}} \\ \sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{x_{ij}} = 1\ , \mathrm{i} = \overline{1,\mathrm{n}} \\ \mathrm{x_{ij}} \in \{0; 1\} \end{cases} \quad (3)$$

В результате сведения задачи к системе (3) данную задачу можно решить венгерским методом [5].

Заключение. В результате проведенного анализа алгоритмов распределения запросов между исполнителями был предложен подход, минимизирующий время нахождения заявок в системе. Минимизацию времени нахождения предложено выполнять методом линейного программирования, а именно венгерским методом.

Список литературы

[1]. Бунин О. Разработка высоконагруженных систем. М.: Издательство Олега Бунина, 2011. 416с.

[2]. Round-robin scheduling. Режим доступа: http://en.wikipedia.org/wiki/Round-robin_scheduling (дата обращения 10.01.2015).

[3]. Least Recently Used. Режим доступа: http://en.wikipedia.org/wiki/Cache_algorithms#LRU (дата обращения 10.01.2015)

[4]. Cache (computing). Режим доступа: http://en.wikipedia.org/wiki/Cache_%28computing%29 (дата обращения 10.01.2015)

[5]. Волков И.К., Загоруйко Е.А. Исследование операций. М.: Издательство МГТУ им. Н.Э. Баумана, 2000. 539 с.