УДК 519.622

Организация матричных и символьных вычислений для исследования поведения решений обыкновенных дифференциальных уравнений

Безгин С.В., Пчелинцев А.Н.

Тамбовский государственный технический университет

Аннотация: В работе описывается эффективный алгоритм вычисления матричной экспоненты. Рассмотрен также комплекс программ для ЭВМ, позволяющий проводить одновременно символьные вычисления для различного типа задач (например, расчет производных и интегралов) в распределенной компьютерной среде.

Ключевые слова: матричная экспонента, символьные вычисления, пакет Maxima, среда ICE.

Structure of matrix and symbolic calculi for research of the behavior of solutions of ordinary differential equations

Bezgin S.V., Pchelintsev A.N.

Tambov State Technical University

Annotation: An efficient method for computing matrix exponential is described in this work. It's also presented a computer framework for simultaneous symbolic calculations in distributed computing environment.

Keywords: matrix exponential, symbolic calculi, Maxima, ZeroC ICE.

Введение

При анализе поведения решений систем обыкновенных дифференциальных уравнений часто требуется вычислять матричную экспоненту [1]. Известные подходы на сегодняшний день связаны с тем, что приходится рассчитывать большие степени матриц. В данной работе рассматривается алгоритм приближенного вычисления матричной экспоненты, который за фиксированное число матричных операций дает результат с заданной точностью. Проведен вычислительный эксперимент с целью анализа эффективности разработанного алгоритма.

Для того чтобы проводить построение решений систем обыкновенных дифференциальных уравнений, необходимо программное обеспечение, позволяющее эффективно производить символьные вычисления [2,3]. Поскольку объем таких вычислений достаточно велик, в данной работе описывается комплекс программ для ЭВМ (система MaximaLib), позволяющий повысить эффективность численного анализа движений динамических систем за счет распределения вычислительного процесса в компьютерной среде. Заметим, что при построении движений данных систем символьные вычисления могут одновременно использоваться как для дифференцирования или интегрирования, так и для вычисления значений алгебраических выражений. Предлагаемый подход позволяет проводить одновременно символьные вычисления для различного типа задач.

Вычисление матричной экспоненты

Вычисление экспоненты

$$e^{At} = E + \sum_{i=1}^{\infty} \frac{(At)^i}{i!} = \sum_{i=0}^{\infty} \frac{(At)^i}{i!}, \tag{1}$$

где E – единичная матрица, t – время; связано с необходимостью расчета высоких степеней матрица A. Получим формулу, позволяющую вычислить матричную экспоненту с помощью n степеней матрицы A, где n – ее порядок.

Пусть характеристическое уравнение матрицы A имеет вид

$$\lambda^n - p_1\lambda^{n-1} - p_2\lambda^{n-2} - \ldots - p_n = 0. \tag{2}$$

По теореме Гамильтона-Кэли [4] матрица A удовлетворяет матричному уравнению, аналогичному (2):

$$A^n - p_1 A^{n-1} - p_2 A^{n-2} - \ldots - p_n E = 0,$$

откуда

$$A^n = p_1 A^{n-1} + p_2 A^{n-2} + \ldots + p_n E. \tag{3}$$

Следуя методу Д.К. Фаддеева [4], коэффициенты характеристического уравнения определяются по рекуррентному соотношению

$$p_k = \frac{s_k - p_1 s_{k-1} - \ldots - p_{k-1} s_1}{k},$$

где $s_k = \mathrm{Sp}\,A^k$ – след матрицы A^k (сумма элементов, стоящих на главной диагонали), $p_1 = \mathrm{Sp}\,A$, $k = \overline{2,n}$.

Далее введем обозначение: если $m = 0$, то $q_{0,k} = p_k$; иначе (при натуральном m) –

$$q_{m,k} = p_k q_{m-1,1} + q_{m-1,k+1}, \; q_{m-1,n+1} = 0. \tag{4}$$

Умножим обе части соотношения (3) на матрицу A с учетом введенных обозначений. Получим

$$\begin{gathered}A^{n+1}=q_{0,1}A^{n}+q_{0,2}A^{n-1}+\ldots+q_{0,n}A=\\=(p_1q_{0,1}+q_{0,2})A^{n-1}+(p_2q_{0,1}+q_{0,3})A^{n-2}+\\+(p_3q_{0,1}+q_{0,4})A^{n-3}+\ldots+\\+(p_{n-1}q_{0,1}+q_{0,n})A+p_nq_{0,1}E.\end{gathered}\tag{5}$$

Выражение (5) можно переписать как

$$A^{n}=q_{1,1}A^{n-1}+q_{1,2}A^{n-2}+\ldots+q_{1,n}E\,.\tag{6}$$

Теперь умножим обе части равенства (6) на матрицу A, подставив при этом в полученное соотношение формулу (3):

$$\begin{gathered}A^{n+2}=q_{1,1}A^{n}+q_{1,2}A^{n-1}+\ldots+q_{1,n}A=\\=(p_1q_{1,1}+q_{1,2})A^{n-1}+(p_2q_{1,1}+q_{1,3})A^{n-2}+\\+(p_3q_{1,1}+q_{1,4})A^{n-3}+\ldots+\\+(p_{n-1}q_{1,1}+q_{1,n})A+p_nq_{1,1}E.\end{gathered}\tag{7}$$

Тогда из выражения (7) с помощью последовательного умножения на матрицу A обеих его частей следует, что

$$A^{n+m}=q_{m,1}A^{n-1}+q_{m,2}A^{n-2}+\ldots+q_{m,n}E=\sum_{k=0}^{n-1}A^{k}q_{m,n-k}\,.$$

Теперь представим матричную экспоненту как

$$\begin{gathered}e^{At}=\sum_{k=0}^{n-1}A^{k}\frac{t^{k}}{k!}+\sum_{m=0}^{\infty}\frac{t^{n+m}}{(n+m)!}\sum_{k=0}^{n-1}A^{k}q_{m,n-k}=\\=\sum_{k=0}^{n-1}A^{k}\frac{t^{k}}{k!}+\sum_{k=0}^{n-1}A^{k}\sum_{m=0}^{\infty}q_{m,n-k}\frac{t^{n+m}}{(n+m)!}.\end{gathered}$$

Откуда имеем

$$\begin{gathered}e^{At}=\sum_{k=0}^{n-1}A^{k}\left[\frac{t^{k}}{k!}+\sum_{m=0}^{\infty}\frac{q_{m,n-k}}{(m+n)!}t^{m+n}\right]\equiv\\\equiv\sum_{k=0}^{n-1}A^{k}\left[\frac{t^{k}}{k!}+\sum_{m=0}^{\infty}r_{m,k}t^{m+n}\right].\end{gathered}\tag{8}$$

Описание алгоритма

Для реализации вычисления матричной экспоненты, согласно (8), был предложен следующий алгоритм. Сначала инициировать результат значением нулевой матрицы. Вычислить значения A^{k} для k от 0 до n. Далее выполнить для k от 0 до n–1 следующую последовательность операций:

1. Вычислить сумму $\sum_{m=0}^{\infty} r_{m,k} t^{m+n}$. В качестве критерия прекращения суммирования использовать условие $\left| r_{m,k} t^{m+n} \right| < \varepsilon$, где ε – положительное число, характеризующее точность вычисления суммы.

2. Используя значения, полученные ранее, получить произведение $A^k \left[\frac{t^k}{k!} + \sum_{m=0}^{\infty} r_{m,k} t^{m+n} \right]$ и прибавить его к текущему значению результата.

При вычислении матричной экспоненты с помощью данного алгоритма используется рекуррентная последовательность (4). При больших значениях m и k большинство значений q будут рассчитываться повторно много раз. Поскольку q(m,k) является чистой функцией (зависит только от входных аргументов), то будет разумно применить стратегию мемоизации.

Мемоизация – оптимизационная техника, заключающаяся в запоминании результатов вычисления функции для предотвращения множественного расчета значения функции от одних и тех же аргументов. Данная оптимизация позволяет улучшить временные характеристики алгоритма за счет увеличения затрат памяти.

Сравнение с классическим алгоритмом

Разработанный алгоритм обеспечивает вычисление матричной экспоненты, используя только n степеней матрицы, в то время как классический алгоритм (1) не детерминирован, и вычисления степеней матрицы продолжаются, пока не выполнится условие останова алгоритма. Обычно таким условием является

$$\left\| \frac{(At)^i}{i!} \right\| < \varepsilon .$$

Вычисление нормы матрицы является само по себе достаточно затратной операцией и имеет сложность $O(n^2)$. В разработанном алгоритме не используются вычисления нормы матрицы, что благоприятно сказывается на его производительности.

Заметим, что классический алгоритм имеет потребление памяти $O(n^2)$. Предлагаемый же нами алгоритм, в виду необходимости хранить $n-1$ степень матрицы A, имеет потребление памяти $O(n^3)$.

Нами был проведен вычислительный эксперимент с целью сравнить быстродействие алгоритмов. Для этого была разработана программа [5] на языке C++, реализующая оба алгоритма. С помощью данной программы были произведены расчеты матричной экспоненты для матриц различного размера. Порядок матрицы изменялся от 2 до 100. Матрица инициализировалась случайными числами в диапазоне [0;1]. Экспонента вычислялась для $t = 1$. Результаты сравнительного эксперимента представлены на рис. 1. По оси абсцисс отложен порядок матрицы A, по оси ординат – время счета. Полученные точки соединены сплайнами для наглядности.

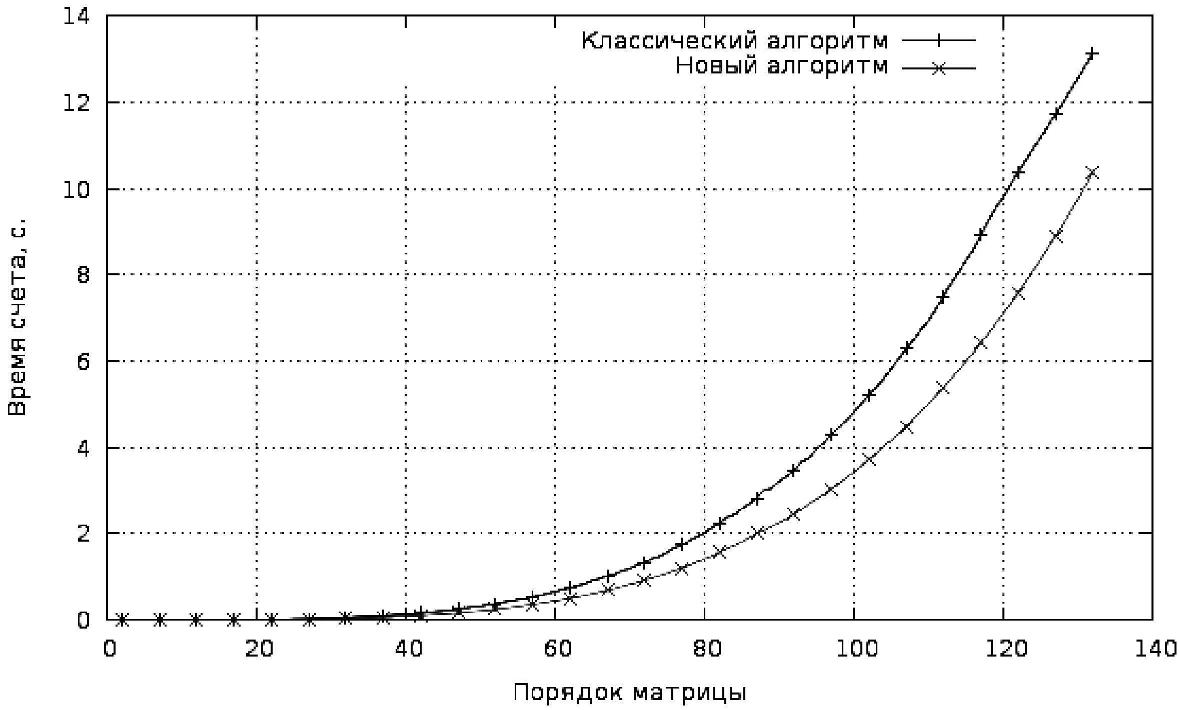

Рис. 1. Сравнение временных характеристик классического алгоритма (верхняя кривая) вычисления матричной экспоненты и разработанного в данной работе (нижняя кривая).

Для матриц порядка 100 наблюдается 40% улучшение быстродействия алгоритма по сравнению с классической схемой. Таким образом, представленный алгоритм предлагает значительное ускорение вычисления матричной экспоненты за счет увеличения потребления оперативной памяти. При этом точность $\varepsilon = 10^{-4}$ выбиралась одинаковой для обоих алгоритмов, максимальное расхождение матриц, получаемых в результате их работы, составляет по норме их разности не более 10^{-2}.

Повышение эффективности использования пакета символьных вычислений Maxima

На сегодняшний день пакет Maxima является стандартом де-факто для символьных вычислений в свободных операционных системах. Как правило, в вычислительных задачах этот пакет используется для проведения символьного дифференцирования или интегрирования различный выражений. Простейший способ обращения к пакету заключается в подготовке входных данных в виде текстового файла, вызове пакета с помощью стандартной функции system() языка C и последующей обработке выходного файла. Пример такого подхода описан в работе [2].

Однако такая схема работы с пакетом Maxima, несмотря на всю простоту, не лишена недостатков. Самый главный из них заключается в довольно длительном старте пакета по сравнению с полезным временем обработки задания. Получается, что в некоторых случаях большую часть времени вычислительный алгоритм может проводить в ожидании старта и завершения процесса.

Очевидное решение этой проблемы – обеспечить старт Maxima в начале расчетов, а в дальнейшем работать с ним в on-line режиме. Авторами данной работы была разработана объектно-ориентированная библиотека maxima_comm [5], которая обеспечивает такой режим работы. Для запуска внешней программы и последующей коммуникации с ней используется известный подход fork-exec [6].

Суть подхода заключается в следующем:

1. Родительский процесс подготавливает каналы связи с дочерним процессом. Обычно для этих целей используется пара конвейеров (pipe). Один из них служит для приема данных, а другой – для передачи.

2. Родительский процесс с помощью вызова функции fork() порождает дочерний процесс, который содержит точную копию родительского, в том числе и подготовленные на шаге 1 каналы передачи данных.

3. Дочерний процесс подменяет дескрипторы стандартного ввода-вывода дескрипторами конвейеров, созданных на шаге 1. Для этого используется функция dup2().

4. Дочерний процесс подменяет в памяти свой образ на образ запускаемого процесса с помощью одной из функций семейства exec*.

После выполнения всех этих шагов запускается новый процесс, связанный с родительским процессом каналами ввода-вывода.

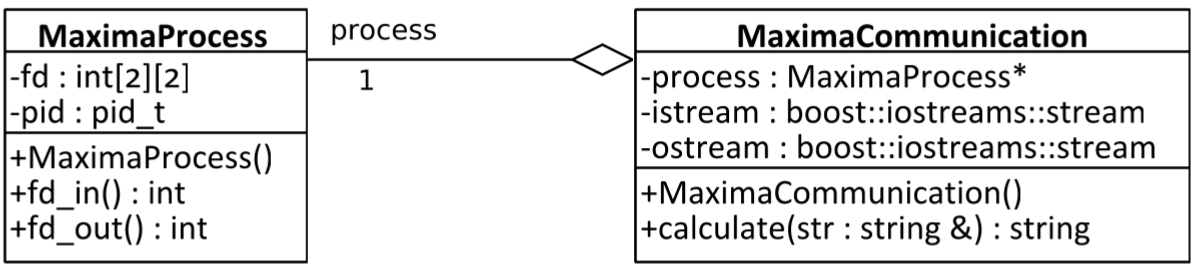

Рис. 2. Диаграмма классов библиотеки maxima_comm.

Библиотека maxima_comm состоит из 2 классов (рис. 2). Класс MaximaProcess инкапсулирует подход fork-exec для запуска процесса Maxima. Публичный интерфейс этого класса предоставляет пользователю метод-конструктор запуска процесса и два файловых дескриптора для организации связи с ним. Класс MaximaCommunication предоставляет высокоуровневый интерфейс для работы с пакетом Maxima. Его публичный интерфейс состоит из двух методов: конструктора, обеспечивающего запуск пакета и первоначальную его настройку, и метода calculate(), предназначенного для проведения собственно символьного вычисления. Этот метод принимает в качестве своего аргумента корректное Maxima-выражение, а возвращает его вычисленное значение.

Работа с библиотекой maxima_comm заключается в добавлении в исходный текст директивы препроцессора #include "maxima_comm.h" и использовании в программе класса MaximaCommunication, а также компоновке исполняемого файла с разделяемой библиотекой libmaxima_comm.so. На рис. 3 приведен небольшой пример исходного кода программы на языке C++, осуществляющей символьное интегрирование выражения

$$\int \frac{x}{x^3+1} dx .$$

```
#include <iostream>
#include <string>
#include "maxima_comm.h"
using namespace std;
using namespace maxima_comm;

int main()
{
        MaximaCommunication comm;
        string s = "integrate(x/(x^3 + 1), x)";
        cout << comm.calculate(s) << endl;
        return 0;
}
```

Рис. 3. Пример использования библиотеки maxima_comm.

Скомпилировав программу и запустив ее на выполнение, получим результат интегрирования, представленный на рис. 4.

```
$ ./maxima_comm_test
log(x^2-x+1)/6+atan((2*x-1)/sqrt(3))/sqrt(3)-log(x+1)/3
```

Рис. 4. Результат работы программы символьного интегрирования в операционной системе Linux.

Использование пакета Maxima в распределенной компьютерной среде

Для обеспечения возможности использования пакета Maxima в распределенной компьютерной среде была разработана распределенная система символьных вычислений MaximaLib [4]. Диаграмма развертывания системы приведена на рис. 5.

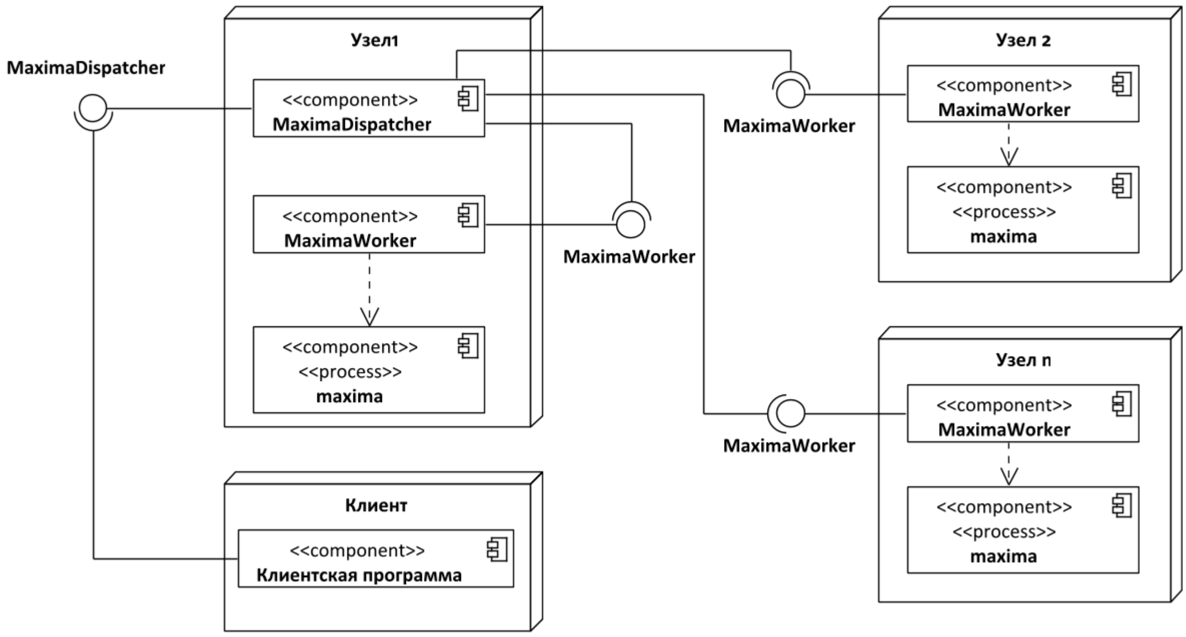

Рис. 5. Диаграмма развертывания системы MaximaLib в распределенной компьютерной среде.

Система состоит из двух компонентов – MaximaWorker и MaximaDispatcher. Каждый из этих компонентов является исполняемым файлом, предоставляющим некие интерфейсы в среде ZeroC ICE [7]. ICE поддерживает очень большое количество платформ программирования, включая C++, Java, .NET, VisualBasic, Python, Ruby и PHP.

На выделенном сервере запускается процесс MaximaDispatcher, являющийся диспетчером запросов на проведение символьных вычислений. На других узлах сети запускаются процессы-работники MaximaWorker, предоставляющие доступ к процессам пакета Maxima. Процессы-работники используют описанную ранее библиотеку maxima_comm для работы с пакетом Maxima. Компоненты MaximaWorker после запуска регистрируют себя в качестве вычислительного ресурса в компоненте MaximaDispatcher. Компонент MaximaDispatcher предоставляет клиенту системы MaximaLib общедоступный интерфейс для проведения вычислений. Все сетевое взаимодействие внутри системы происходит совершенно прозрачно для клиента.

Система состоит из трех модулей: MaximaLib, MaximaDispatcher и MaximaWorker. Модуль MaximaLib содержит в себе объявления общедоступных интерфейсов на языке slice, а так же сгенерированные файлы Maxima.h и Maxima.cpp, которые должны быть подключены в каждой программе, использующей систему. В этих файлах находятся объявления и реализации интерфейсов и классов, необходимых для корректного функционирования программы в среде ICE. Так, например, в них объявлен класс MaximaWorkerPrx (прокси-класс), обеспечивающий доступ к удаленному объекту, реализующему интерфейс MaximaWorker.

Модуль MaximaWorker – исполняемый файл, предоставляющий реализацию интерфейса MaximaWorker. Состоит из двух классов. Класс MaximaWorkerApp обеспечивает корректный старт модуля, регистрацию его в среде ICE и корректное завершение. Класс MaximaWorkerImpl представляет собой реализацию класса-работника, выполняющего символьные вычисления с помощью пакета Maxima.

В начале работы модуля создается экземпляр класса MaximaWorkerImpl, и этот объект регистрируется в диспетчере с помощью вызова метода addWorker(). При завершении работы модуля объект удаляется из списка работников диспетчера путем вызова метода removeWorker().

Модуль MaximaDispatcher – исполняемый файл, предоставляющий реализацию интерфейса MaximaDispatcher. Он обеспечивает управление коллекцией исполнителей и маршрутизацию запросов пользователей. В начале работы создается экземпляр класса MaximaDispatcherApp, инициирующий среду ICE и выполняющий регистрацию в ней экземпляра класса MaximaDispatcherImpl. Последний содержит в себе коллекцию исполнителей, реализованную в виде класса WorkerQueue. В качестве структуры данных для упорядоченного доступа к исполнителям используется очередь. При запросе исполнителя из коллекции с помощью метода getWorker() возвращается объект из головы очереди, и тот час же ставится в ее конец. Таким образом, реализуется стратегия использования вычислительных ресурсов, известная как round-robin.

На рис. 6 приведена диаграмма последовательности взаимодействия объектов при получении запроса от клиента. Клиент вызывает метод calculate() объекта, реализующего интерфейс MaximaDispatcher. Тот в свою очередь путем вызова метода getWorker() объекта queue получает исполнителя из очереди ресурсов. Задание отправляется на обработку исполнителю, результат возвращается клиенту.

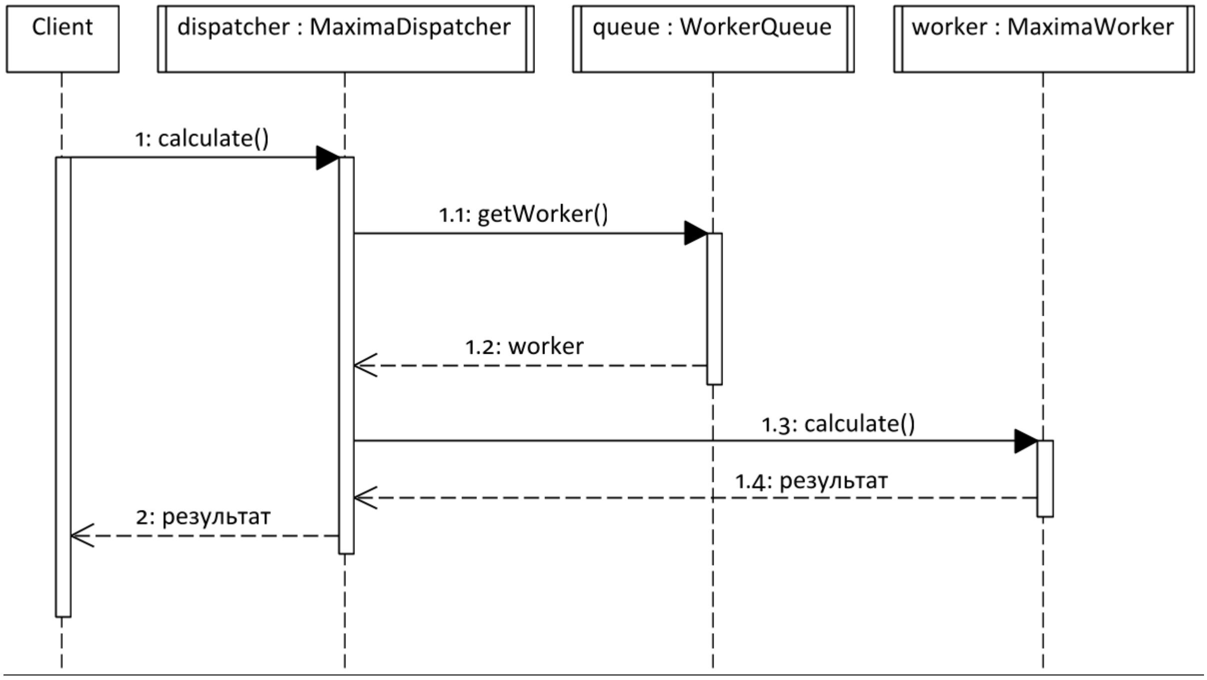

Рис. 6. Диаграмма последовательности взаимодействия объектов при получении запроса от клиента.

Список литературы

1. Демидович Б.П. Лекции по математической теории устойчивости. – М.: Наука, 1967. – 399 с.

2. Пчелинцев А.Н. Численные методы построения обобщенно-периодических решений дифференциальных уравнений при моделировании динамических процессов : дис. на соиск. уч. степ. к-та физ.-мат. наук : 05.13.18 : защищена 12.11.2009 : утв. 12.02.2010. – Воронеж, 2009. – 114 с.

3. Пчелинцев А.Н., Поветьев А.Ю., Подольский В.Е. О построении периодических решений одного класса неавтономных систем дифференциальных уравнений в распределенной компьютерной среде // Вестник ТГТУ. – 2011. – Т. 17, №2. – С. 502-512.

4. Гантмахер Ф.Р. Теория матриц. – М.: Наука, 1967. – 576 с.

5. http://cluster.tstu.ru/tiki-download_file.php?fileId=30

6. Стивенс У.Р., Раго С.А. UNIX. Профессиональное программирование. – СПб: Символ-Плюс, 2007. – 1040 с.

7. http://www.zeroc.com/ice.html