Правительство Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Ерофеева Виктория Александровна

Адаптивная балансировка загрузки системы Web-серверов на основе рандомизированных алгоритмов стохастической аппроксимации

Магистерская диссертация

Допущена к защите.

Зав. кафедрой:

д. ф.-м. н., проф. Терехов А.Н.

Научный руководитель:

д. ф.-м. н., проф. Граничин О.Н.

Рецензент:

аспирант каф. системного программирования,

Иванский Ю. В.

Санкт-Петербург

2015

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics&Mechanics Faculty

Chair of Software Engineering

Victoria Erofeeva

Adaptive Load Balancing of Web Server System Using Simultaneous Perturbation Stochastic Approximation

Master's Thesis

Admitted for defence.

Head of the chair:

Professor A.N. Terekhov

Scientific supervisor:

Professor O.N. Granichin

Reviewer:

Ph.D. student Y. V. Ivanskiy

Saint-Petersburg

2015

Оглавление

Введение 5

1. Анализ предметной области 7
   1.1. Балансировка нагрузки: общие сведения 7
      1.1.1. Задача балансировки нагрузки узлов вычислительной сети 7
      1.1.2. Стратегии балансировки 7
      1.1.3. Виды балансировок 8
      1.1.4. Оценка загрузки 9
   1.2. Актуальность проблемы балансировки Web-серверов 10
   1.3. Анализ компонентов существующих решений 12
      1.3.1. HAProxy 12
      1.3.2. Nginx Plus 12
      1.3.3. Amazon Web Services 13

2. Постановка задачи 15
   2.1. Балансировка нагрузки узлов вычислительной сети 15
   2.2. Оптимизация функционала среднего риска 16
   2.3. Адаптация шага алгоритма 18

3. Адаптивный алгоритм с использованием SPSA 20
   3.1. Исследование зависимости работы алгоритма от параметров . . . 20
   3.2. Оптимизация шага алгоритма 29

4. Реализация балансировщика 31
   4.1. Программная платформа «Akka» 31
   4.2. Архитектура прототипа системы 31
   4.3. Результаты 32

4.3.1. Сравнение адаптивной и неадаптивной версий алгоритма SPSA 32
4.3.2. Сравнение адаптивного алгоритма SPSA с другими алгоритмами 37

Заключение 40

Список литературы 41

Введение

В настоящее время веб-сервисы находят все более широкое применение. Быстрое увеличение их числа и пользователей этих сервисов требует высокопроизводительного программного и аппаратного обеспечения для реализации быстрых ответов на запросы, которые могут появиться в любой момент времени. Необходимо добиваться еще и того, чтобы веб-серверы обладали отказоустойчивостью и масштабируемостью.

Для реализации высокопроизводительных и надежных серверов используют распределенные решения. Распределенные веб-серверы представляют собой набор узлов – это продублированные ресурсы для одновременного предоставления сервисов многим клиентам. Входящие запросы могут быть распределены по серверам согласно определенным стратегиям распределения загрузки и поэтому эти запросы могут быть быстро обработаны. Распределенная архитектура веб-серверов может быть организована различным образом:

- интеграция в кластер;
- географическое распределение.

Распределенная система обладает высокой степенью масштабируемости. Количество серверов может быть увеличено простым добавлением нового узла в локальную сеть.

Использование высокопроизводительных распределенных систем серверов требует балансировки загрузки. Входящие клиентские запросы должны быть равномерно распределены между серверами с той целью, чтобы пользователь мог как можно быстрее получить ответ на запрос.

Существуют различные программные продукты, использующие алгоритмы балансировки (см. например [25-28]). Все они учитывают определенные параметры системы, такие как: среднее время отклика, количество активных соединений и т. д. Тем не менее, остается вопрос их адаптируемости к изменениям

состояния системы. В большинстве случаев, параметры, на которых основывается алгоритм балансировки, задаются администратором сети вручную и не меняются в ходе работы. Однако, с учетом развития информационных технологий следует стремиться к автоматизации этих процессов. Так, существуют алгоритмы, которые успешно можно применить для отслеживания параметров системы. Например, в [11, 15, 21] приводятся приложения и алгоритмы из различных областей применения.

В [4, 12-13] предлагается алгоритм оценивания параметров сети на основе радомизированных алгоритмов стохастической аппроксимации (англ. SPSA - Simultaneous perturbation stochastic approximation) при наличии внешних помех. При этом у авторов остается открытым вопрос о выборе размера шага алгоритма.

В диссертационной работе исследуется зависимость качества балансировки от выбора размера шага алгоритма и предлагается его адаптивная версия, основывающаяся на поиске оптимального шага в ходе работы системы.

1. Анализ предметной области

1.1. Балансировка нагрузки: общие сведения

Балансировка нагрузки – метод распределения заданий между несколькими устройствами (например, серверами) с определенной целью. Среди целей балансировки можно выделить: оптимизацию использования ресурсов, сокращение времени обслуживания запросов, масштабирование и отказоустойчивость [24].

1.1.1. Задача балансировки нагрузки узлов вычислительной сети

Проблема балансировки нагрузки вычислительной сети возникает по нескольким причинам:

- поступающие задания имеют различную степень сложности, поэтому требуется определить количество ресурсов, достаточное для их выполнения;
- структура вычислительной сети также неоднородна, узлы обладают разной производительностью, которая в большинстве случаев неизвестна;
- структура межузлового взаимодействия неоднородна, так как вычислительные узлы могут быть географически распределенными, в результате чего образуется различная задержка.

Для решения некоторых из описанных выше проблем успешно применяются адаптивные алгоритмы с возможностью оценивания неизвестных параметров сети, таких как, например, производительность узлов (см. [4, 8, 11, 13]).

1.1.2. Стратегии балансировки

Балансировка предполагает равномерную нагрузку вычислительных узлов в сети (см. например [8]). При появлении нового задания блансировщик должен

принять решение о том, на каком узле следует обработать поступивший запрос. В зависимости от архитектуры сети, применяются различные стратегии балансировки.

Сети условно можно разделить на два класса: централизованные и децентрализованные (распределенные). В первом случае характерной особенностью является наличие управляющего узла, который необходим для функционирования сети в целом. Для распределенных систем характерно распределение заданий между множеством узлов и отсутствие единого управляющего центра, поэтому выход из строя одного из узлов не приводит к полной остановке всей системы [23].

При централизованной стратегии один из узлов собирает информацию о состоянии всей вычислительной системы и/или принимает решение о назначении заданий для каждого сервера. При полностью распределенной стратегии на каждом узле выполняется алгоритм балансировки нагрузки, использующий взаимодействие между узлами [17].

1.1.3. Виды балансировок

Различают статическую и динамическую балансировки. В первом случае выбор узла для выполнения пришедшего задания производится заранее на основе какого-либо правила. При распределении заданий может использоваться опыт предыдущих итераций [14].

Динамическая стратегия предусматривает перераспределение вычислительной нагрузки на другие узлы во время работы.

Балансировщик определяет:

- загрузку вычислительных узлов;
- величину сетевых задержек и другие параметры.

На основании собранных данных принимается решение о переносе заданий с одного узла на другой [7].

Следует отметить, что выбор стратегии напрямую зависит от типа выполняемых заданий, в том смысле, что в некоторых случаях применение статической стратегии вместо динамической и наоборот нецелесообразно. Так, например, при выборе стратегии для балансировки поступающих запросов на веб-серверы стоит обратить внимание на обоснованность перераспределения заданий в процессе динамической балансировки. В этом случае следует применять статическую балансировку.

В некоторых случаях считается, что статическая балансировка не дает желаемых результатов в случае изменения состояния системы. Однако, если к таким изменениям отнести например выход из стоя одного или нескольких узлов, или появление нагрузки, не связанной с прямым выполнением заданий, то следует учесть следующее:

- в современных системах с балансировщиком применяется сервис определения состояния узла, благодаря этому, что в статической, что в динамической стратегии балансировщик не отправит задание на неработающий узел, а текущие задания в любом случае будут потеряны;
- появление дополнительной нагрузки может учитываться некоторыми статическими алгоритмами, основывающимися на сборе данных о текущей загруженности узла, поэтому балансировщик может с легкостью адаптироваться к изменившемуся состоянию системы.

В любом случае, выбор стратегии стоит осуществлять на основе рациональных представлений о работе системе и предметной области.

1.1.4. Оценка загрузки

Оценка загрузки вычислительного узла может производиться несколькими способами. Один из способов (аналитический), который состоит в приблизительной оценке загрузки каждого объекта на основе знаний о поступивших заданиях [7].

Преимущество аналитического метода состоит в том, что он достаточно точно может оценить трудоёмкость задачи. Недостаток же этого метода состоит в том, что он может быть довольно неточным в случае, если модель для оценки скорости выполнения задания неточна.

Другой способ сбора данных о загрузке состоит в измерении загрузки узлов. Большинство современных машин снабжено счетчиками времени (с точностью до микросекунд), которые могут быть использованы для измерения времени выполнения каждой задачи. Преимущество метода состоит в том, что он является точным в большинстве случаев. К недостаткам можно отнести следующее: стратегии балансировки, основанные на этом методе учитывают прошлое распределение нагрузок. Если загрузка задач меняется непредсказуемым образом, то метод будет неточным.

В диссертационной работе применяется метод, основанный на оценивании производительностей узлов. Для этого применяется рандомизированный алгоритм стохастической аппроксимации (англ. SPSA – Simultaneous perturbation stochastic approximation) [1-3, 5, 9, 18-20], детально описанный в пункте 3.1.

1.2. Актуальность проблемы балансировки Web-серверов

Балансировка может применяться для различных типов устройств, однако, наибольшее применение она получила в сфере передачи данных за счет постоянного увеличения трафика в сети.

По данным исследования [29] среди сетевых протоколов наиболее выделяется TCP, который значительно превосходит все остальные (Рис. 1.1).

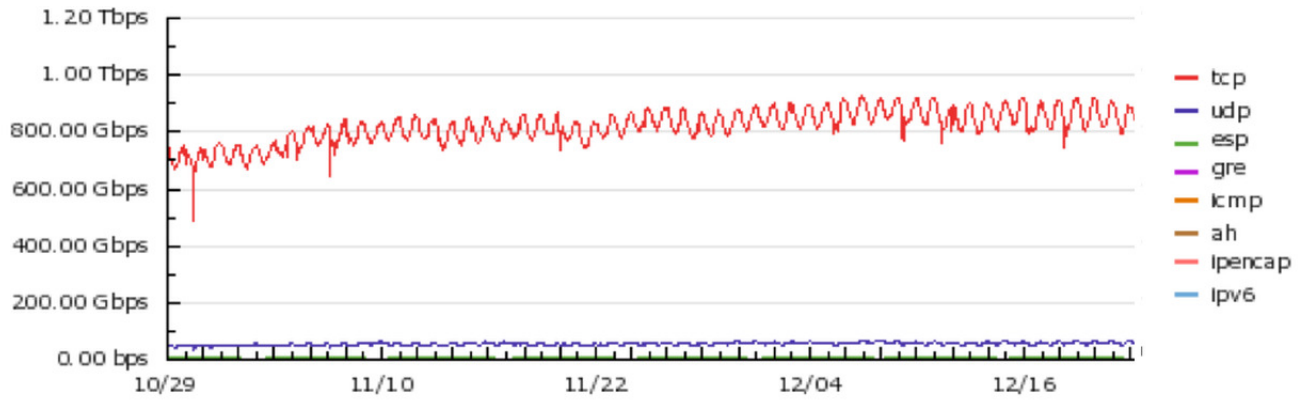

Рис. 1.1 90-дневное отслеживание используемых протоколов

Транспортный протокол передачи данных TCP используется большим количеством приложений, но несмотря на это, среди них также выделяется лидирующий сегмент, которым являются сервисы на порту 80 (Рис. 1.2).

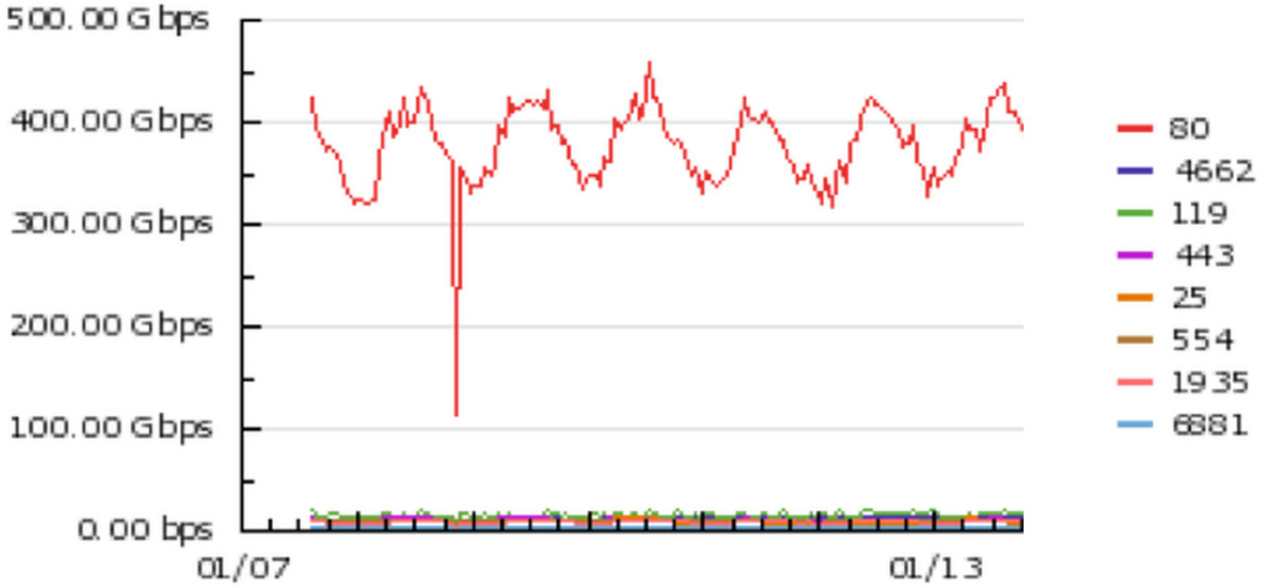

Рис. 1.2 60-дневное отслеживание используемых портов

С учетом этих данных и тенденции к увеличению роста количества пользователей сети можно сделать вывод об актуальности проблемы балансировки нагрузки Web-серверов.

1.3. Анализ компонентов существующих решений

На основе данных, полученных в результате анализа веб-сайтов компаний-разработчиков из области сетевых технологий и наиболее известных компаний, предоставляющих услуги, напрямую связанных с балансировкой нагрузки вычислительной сети, и других источников [25-28] выделены следующие существующие программные решения (ПО и IaaS), реализующие в той или иной форме решение задачи балансировки:

- HAProxy;
- Nginx Plus;
- Amazon Web Services.

Далее приводится краткая характеристика этих программных решений с учетом как положительных, так и отрицательных сторон.

1.3.1. HAProxy

HAProxy – это бесплатное, быстрое и надежное решение, которое предлагает высокую отказоустойчивость, балансировку нагрузки, и прокси для TCP и HTTP-приложений. Такая функциональность подходит для высоконагруженных веб-сайтов с достаточно большим количеством посещений. За годы совего существования это решение с открытым исходным кодом стало стандартом, де-факто, поставляется с большинством основных дистрибутивов и часто развертывается по-умолчанию в облачных платформах [26].

Режим работы этого приложения делает его интеграцию в существующие архитектуры очень легким, как видно из Рис. 1.3.

1.3.2. Nginx Plus

В NGINX Plus Release 6 (R6) был представлен новый алгорим балансировки под названием Least Time. Этот алгоритм отслеживает число одновременных

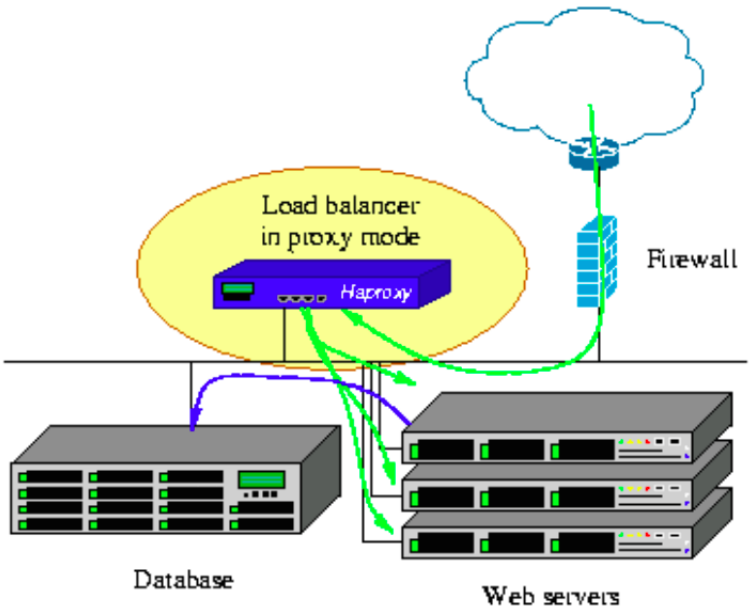

Рис. 1.3 Архитектура системы с HAProxy

подключений и среднее время отклика от каждого узла из пула серверов, доступных для распределения нагрузки. Также он использует эту информацию для того, чтобы выбрать наиболее подходящий узел для каждого запроса.

По информации производителя, алгоритм дает преимущество по сравнению с остальными вариантами балансировки в той ситуации, когда задержка между узлами является значительной. Один из распространенных случаев – это нахождение узлов в различных географически распределенных центров обработки информации. Локальные узлы, как правило, имеют низкую задержку по сравнению с удаленными узлами. Least Time алгоритм выбирает узел с наименьшей задержкой, однако, если выбранный узел недоступен, система перенаправит запрос более медленному узлу [25].

Схема работы алгоритма представлена на Рис. 1.4.

1.3.3. Amazon Web Services

Elastic Load Balancing является одним из компонентов, предоставляемых Amazon Web Services. Балансировщик автоматически распределяет трафик приложений по нескольким вируальным машинам, которые находятся в облаке.

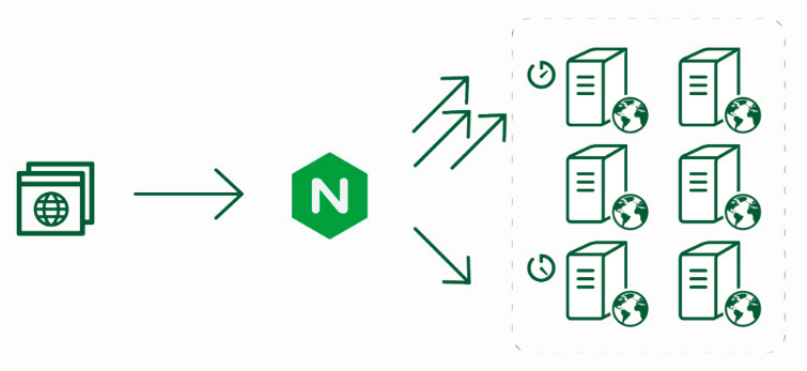

Рис. 1.4 Архитектура системы NGINX Plus Release 6

Это позволяет достичь более высокого уровня отказоустойчивости в приложениях [27].

Производителем заявлены следующие положительные стороны применения Elastic Load Balancing:

- отказоустойчивость (благодаря наличию системы отслеживания статуса виртуальной машины, трафик будет направляться только на работающие сервера, независимо от того, где они находятся – локально или территориально-распределенно);
- масштабируемость (если существующие виртуальные машины перестанут справляться с поступающим трафиком, то балансировщик автоматически развернет еще один сервер, с идентичными характеристиками);
- безопасность (виртуальные машины находятся в виртуальной частной сети (англ. VPC – Amazon Virtual Private Cloud), к которой нет доступа извне, то есть все внешние запросы направляются только к балансировщику).

2. Постановка задачи

2.1. Балансировка нагрузки узлов вычислительной сети

Рассматривается формальная постановка задачи балансировки вычислительных узлов из [4]. Пусть вычислительная система (сеть) состоит из d узлов. Система работает последовательно, обрабатывая на каждой итерации $t, t = 1, 2, \ldots$, поступивший в систему ранее набор заданий известного размера q_{t-1}, который можно произвольно разделить на d наборов подзадач x^j, $j = 1, \ldots, d$:

$$\|\mathbf{x}\|_1 = \sum_{j=1}^{d} x^j = q_{t-1} \tag{2.1}$$

(здесь и далее $\mathbf{x} = col(x^1, \ldots, x^d)$, верхний индекс j используется не как степень, а для обозначения номера узла).

Пусть $\theta^j \in \mathbb{R}$ - производительность узла j. Время вычислений узлом $j = 1, \ldots, d$ обозначается как $time^j(x^j) = x^j/\theta^j$.

Требуется минимизировать общее время обработки пакета заданий q_{t-1}:

$$T(\mathbf{x}) = \max_{j=1,\ldots,d} time^j(u^j) \to \min_{\mathbf{x}}. \tag{2.2}$$

Идеальный алгоритм распределения заданий поддерживает для всех узлов равномерную “занятость” выполнением поставленных задач, минимизируя коммуникации между узлами, необходимые для составления плана работ и пересылки данных [13]. Задача о планировании загрузки усложняется, когда задачи генерируются динамически (с течением времени) и непредсказуемо.

В условиях, когда производительности известны, лучшая стратегия распределения задач (стратегия управления) – пропорциональное разделение заданий:

$$\frac{x^1}{\theta^1} = \frac{x^2}{\theta^2} = \ldots = \frac{x^d}{\theta^d}. \tag{2.3}$$

Такая стратегия управления загрузкой называется балансировкой загруз-

ки. Формальное определение следующее:

$$\mathbf{x} = U(\theta, q_{t-1}) : x^j = \frac{\theta^j}{\|\theta\|_1} q_{t-1}, j = 1, \ldots, d, \theta = col(\theta^1, \ldots, \theta^d). \tag{2.4}$$

На практике производительности узлов $\theta \in R^d$ могут быть неизвестными. Более того, они могут меняться со временем из-за параллельного выполнения сторонних работ: $\theta_t = \theta + w_t$, или вообще меняться со временем: $\theta_t = \theta_{t-1} + w_t$, где $w_t \in R^d$ - независимые случайные элементы.

Обычный способ – использовать в стратегии управления оценки производительностей $\hat{\theta}_t$ на каждой итерации t, которые определялись бы из условия:

$$\|\hat{\theta}_t - \theta_t\|^2 \to \min \tag{2.5}$$

в каком-нибудь разумном смысле, и с их помощью вычислять разбиение x_t как $x_t = U(\hat{\theta}_t, q_{t-1})$.

2.2. Оптимизация функционала среднего риска

Задача оптимизации (2.5) является частным случаем более общей схемы нахождения оптимального значения функционала среднего риска.

Рассматривается семейство дифференцируемых функций $\{f_w(\theta)\}_{w \in \mathbb{W}}$, $f_w(\theta) : \mathbb{R}^m \to \mathbb{R}$, пусть $\mathbf{x}_1, \mathbf{x}_2, \ldots$ - последовательность точек наблюдения (измерения), выбираемая экспериментатором (план наблюдений), в которых в каждый момент времени $t = 1, 2, \ldots$ доступны наблюдению значения $y_1, y_2, \ldots$ функций $f_w(\cdot)$ с аддитивными внешними помехами v_t

$$y_t = f_{w_t}(\mathbf{x}_t) + v_t, \tag{2.6}$$

где $\{w_t\}$ - неконтролируемая последовательность, $w_t \in \mathbb{W}$.

Пусть $(\Omega, \mathcal{F}, P)$ - основное вероятностное пространство, а $\mathcal{F}_{t-1}$ - σ-алгебра всех вероятностных событий, которые реализовались до момента времени $t = 1, 2, \ldots$.

Нестационарная постановка задачи: найти “дрейфующую” точку минимума $\theta_t^\star$ функции:

$$F_t(\theta) = E_{\mathcal{F}_{t-1}} f_{w_t}(\theta) \to \min_\theta \tag{2.7}$$

при линейных ограничениях

$$H_t\theta = \mathbf{q}_{t-1} \tag{2.8}$$

с задаваемыми матрицей H_t размерности $k \times d$ и векторами $\mathbf{q}_{t-1} \in \mathbb{R}^k$, $0 \le k < d$.

Здесь и далее E — символ математического ожидания, $E_{\mathcal{F}_{t-1}}$ — символ условного математического ожидания по отношению к σ-алгебре $\mathcal{F}_{t-1}$.

Если матрица H полного ранга, т. е. $rank H_t = k$, то из линейной алгебры известно, что существуют линейное отображение $h_t : \mathbb{R}^d \to \mathbb{R}^{d-k}$ и обратные к ней функции $g_t : \mathbb{R}^{d-k} \to \mathbb{R}^d$ такие, что

$$\mathbf{x} = g_t(h_t(\mathbf{x})),\ \forall \mathbf{x} \in \mathbb{M}_t = \{H_t\mathbf{x} = \mathbf{q}_{t-1}\}.$$

Пусть $\mathbf{\Delta}_n,\ n = 1, 2, \ldots$ – последовательность бернуллиевских случайных векторов из $\mathbb{R}^{d-k}$, принимающих значения ± 1 с вероятностью $\frac{1}{2}$. Задается начальный вектор $\widehat{\theta_0} \in \mathbb{R}^m$, положительные константы α и β, принимающие значения больше нуля. Рассматривается следующий алгоритм:

$$\begin{cases} \mathbf{x}_n^{\pm} = g_{2n-1\pm\frac{1}{2}}(h_{2n-1\pm\frac{1}{2}}(\widehat{\theta}_{2n-2}) \pm \beta\mathbf{\Delta}_n), \\ \widehat{\theta}_{2n-1} = g_{2n-1}(h_{2n-1}(\widehat{\theta}_{2n-2})), \\ \widehat{\theta}_{2n} = g_{2n}(h_{2n}(\widehat{\theta}_{2n-1}) - \alpha\mathbf{\Delta}_n\frac{y_n^+ - y_n^-}{2\beta}). \end{cases} \tag{2.9}$$

Алгоритм (2.9) в [4] применяется для отслеживания изменений $\mathbf{x}_t^\star$ с применением оптимальных шагов α и β, выбранных на основе серии экспериментов. Пусть $h_t(\mathbf{x}) = h(\mathbf{x}) = col(x^1, \ldots, x^{m-1})$ и

$$g_t(\mathbf{z}) = col(z^1, \ldots, z^{d-1}, \frac{\sum_{i=1}^d q_{t-1}^i - \sum_{j=1}^{d-1} z^j\theta_t^i}{\theta_t^d}).$$

Задается начальный вектор $\widehat{\mathbf{x}}_0$ вычисляемый по формуле: $q_0^i/\theta_0^i,\ i = 1, \ldots, d$.

На каждой итерации производятся следующие действия:

• вычисляются два значения

$$y_n^{\pm} = \Phi_{2n-1\pm\frac{1}{2}}(g_{2n-1\pm\frac{1}{2}}(h(\widehat{\mathbf{x}}_{t-1}) \pm \beta\mathbf{\Delta}_t)); \tag{2.10}$$

• вычисляется квазиградиент

$$\widehat{\nabla}_n = \mathbf{\Delta}_n \frac{y_n^+ - y_n^-}{2\beta}; \tag{2.11}$$

• получение новой оценки

$$\widehat{\mathbf{x}}_{2n-1} = g_{2n-1}(h(\widehat{\mathbf{x}}_{2n-2});$$

$$\widehat{\mathbf{x}}_{2n} = g_{2n}(h(\widehat{\mathbf{x}}_{2n-2}) - \alpha\widehat{\nabla}_n). \tag{2.12}$$

В [4] рассматривается один из возможных эмпирических функционалов качества:

$$\Phi_t(\widehat{\theta_t}) = \frac{1}{2(d-1)q_{t-1}^2} \sum_{j,k=1}^{d} (time_t^j - time_t^k)^2 \to \min_{\widehat{\theta_t}}, \tag{2.13}$$

в котором $time_t^j = time^j(U^j(\hat{\theta}_t, q_{t-1})), j = 1, \ldots, d$.

Функция $\Phi_t(\widehat{\theta_t})$ имеет точку минимума $\widehat{\theta_t} = \theta_t$, которая соответствует оптимальной стратегии управления, минимизирующей общее время обработки заданий.

2.3. Адаптация шага алгоритма

В случае (2.12) использовался некоторый фиксированный размер шага α. Задачей диссертационной работы является исследование зависимости качества балансировки от шага алгоритма α и возможности применения алгоритма SPSA для отслеживания оптимального значения параметра в процессе балансировки

системы. Подобная оптимизация позволит сделать алгоритм балансировки наиболее приспосабливающимся к динамически меняющимся условиям среды, в следствие чего должна повыситься скорость обработки заданий.

Для достижения поставленной задачи необходимо выполнить следующие шаги:

- реализовать алгоритм SPSA для отслеживания производительностей с использованием постоянного шага α;
- определить факторы, взаимосвязанные с влиянием параметра α на работу алгоритма балансировки;
- предложить адаптацию алгоритма на базе SPSA для отслеживания производительностей совместно с параметром α и реализовать ее;
- провести сравнительный анализ полученного алгоритма с существующим, который использует постоянный шаг;
- спроектировать архитектуру прототипа некоторой реальной системы, использующей исследуемый в работе алгоритм;
- реализовать модуль балансировки.

3. Адаптивный алгоритм с использованием SPSA

3.1. Исследование зависимости работы алгоритма от параметров

Как отмечалось в главе 2.3, использование алгоритма (2.12) предполагает задание (выбор) некоторого размера шага α. Для того, чтобы была возможность использовать алгоритм SPSA для оптимизации некого параметра необходимо, чтобы целевая функция была выпуклой. Таким образом, для исследования возможности адаптации SPSA с постоянным шагом были проведены имитационные эксперименты с целью выяснения зависимости скорости достижения баланса системы от размера шага α.

Пусть в систему по равномерному распределению постоянно поступает некоторое количество заданий из интервала [15;150], а также в произвольный момент времени один из узлов получает пиковую нагрузку. Фиксируется длина временного такта и количество узлов – 15 узлов и 400 тактов. В результате выбора малого шага алгоритма α, относительно большого и оптимального (получен приблизительно, в результате экспериментов) получены результаты, представленные на Рис. 3.1-3.3.

Пусть показателем баланса системы будет среднеквадратическое отклонение следующей величины:

$$x_i = \frac{q_i}{\hat{\theta}_i} \tag{3.1}$$

где q_i – длина очереди заданий узла i, $\hat{\theta}_i$ – производительность узла i (в данном случае пропускная способность каждого узла, характеризующая количество обработанных запросов за последний такт).

Здесь и далее, единицей измерения x_i является количество запросов в такт времени. Пусть такт времени измеряется в секундах, тогда будет использоваться следующее обозначение единицы измерения x_i - з/сек.

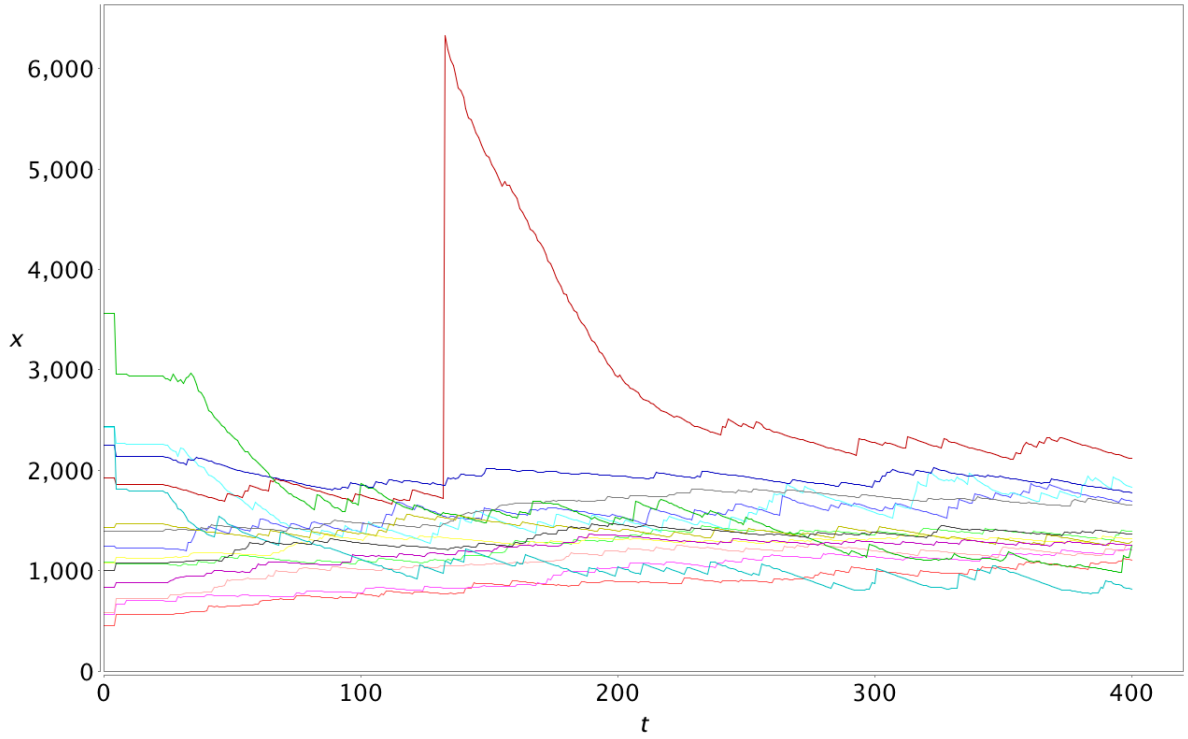

Рис. 3.1 Поведение системы при малом α

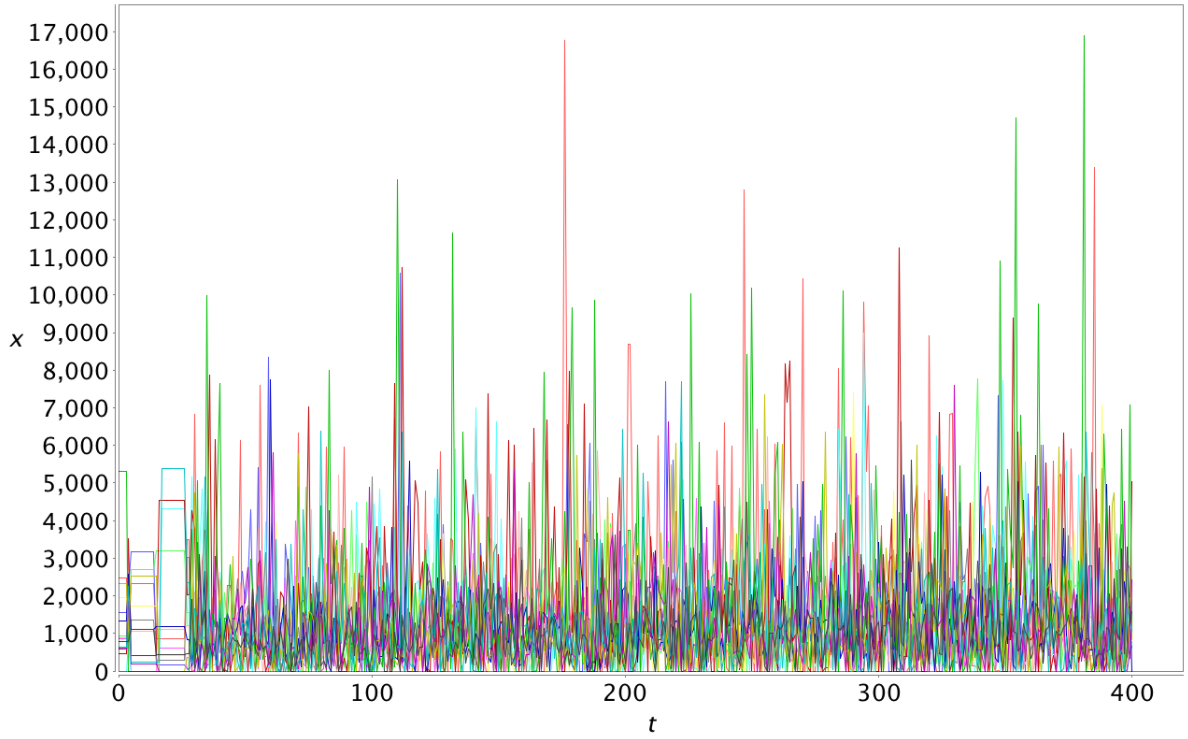

Рис. 3.2 Поведение системы при большом α

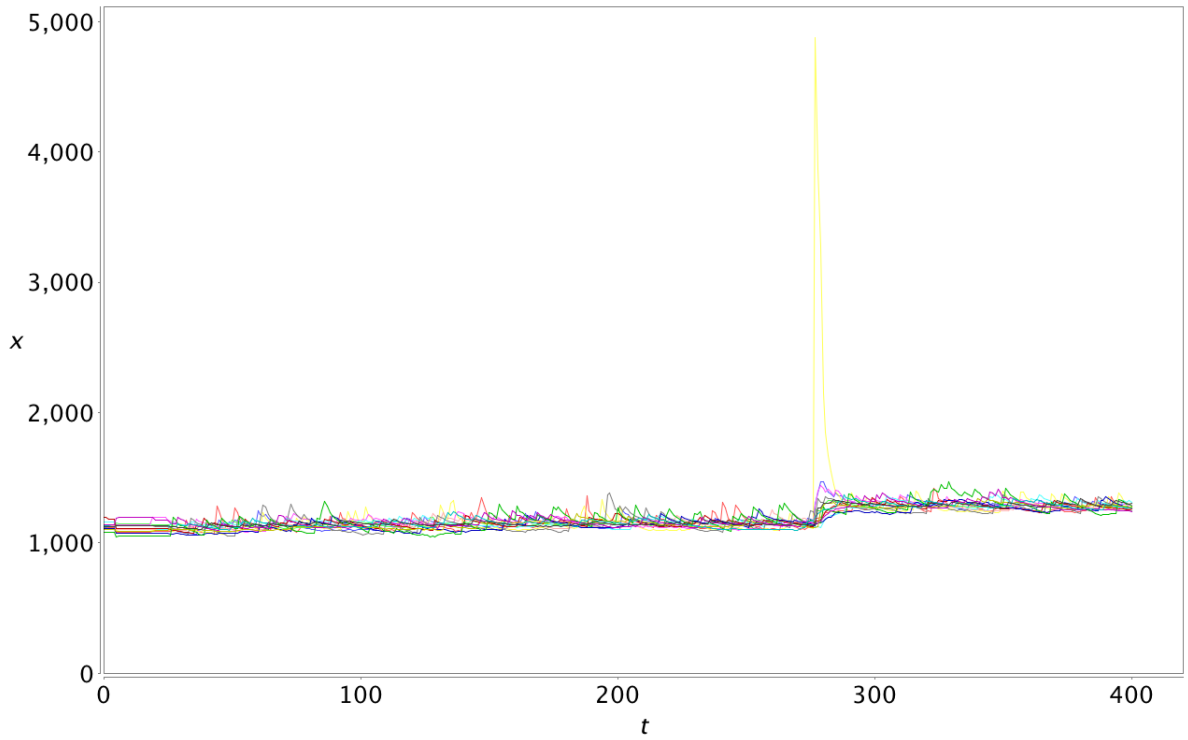

Рис. 3.3 Поведение системы при оптимальном α

Как видно из полученных данных (Рис. 3.1-3.3), использование неоптимального шага приводит к отсутствию баланса в системе.

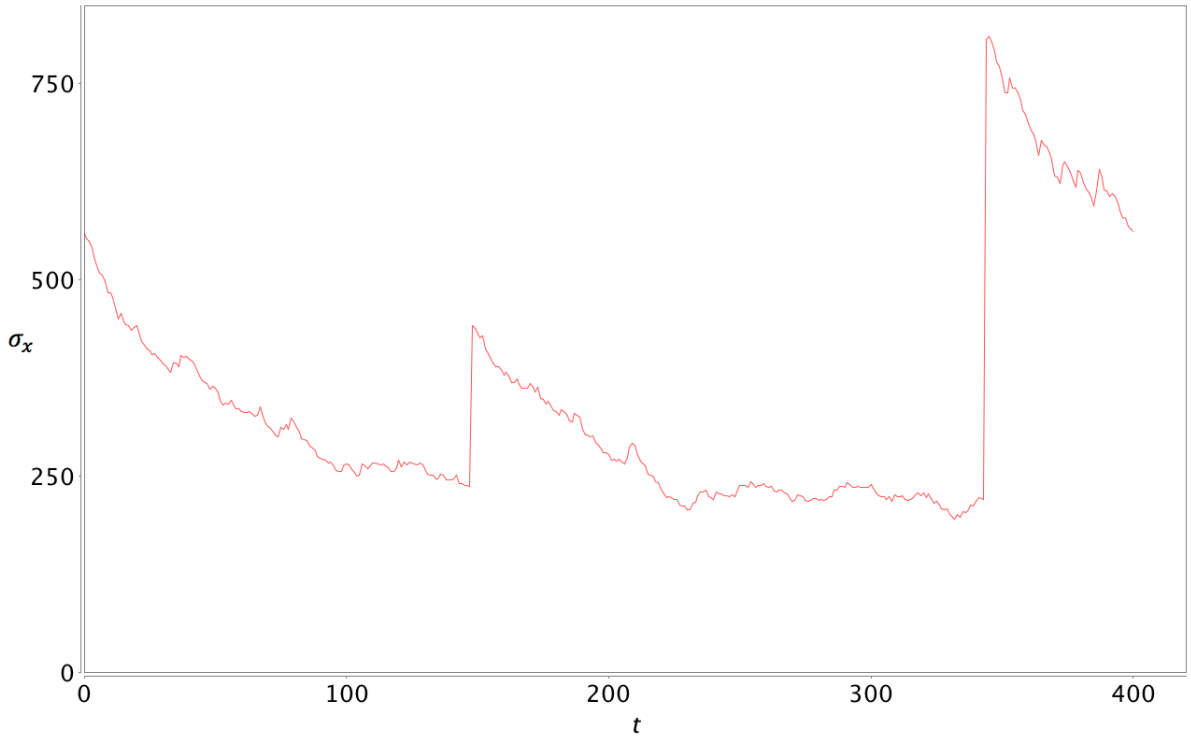

Рис. 3.4 Среднеквадратическое отклонение x_i при малом α

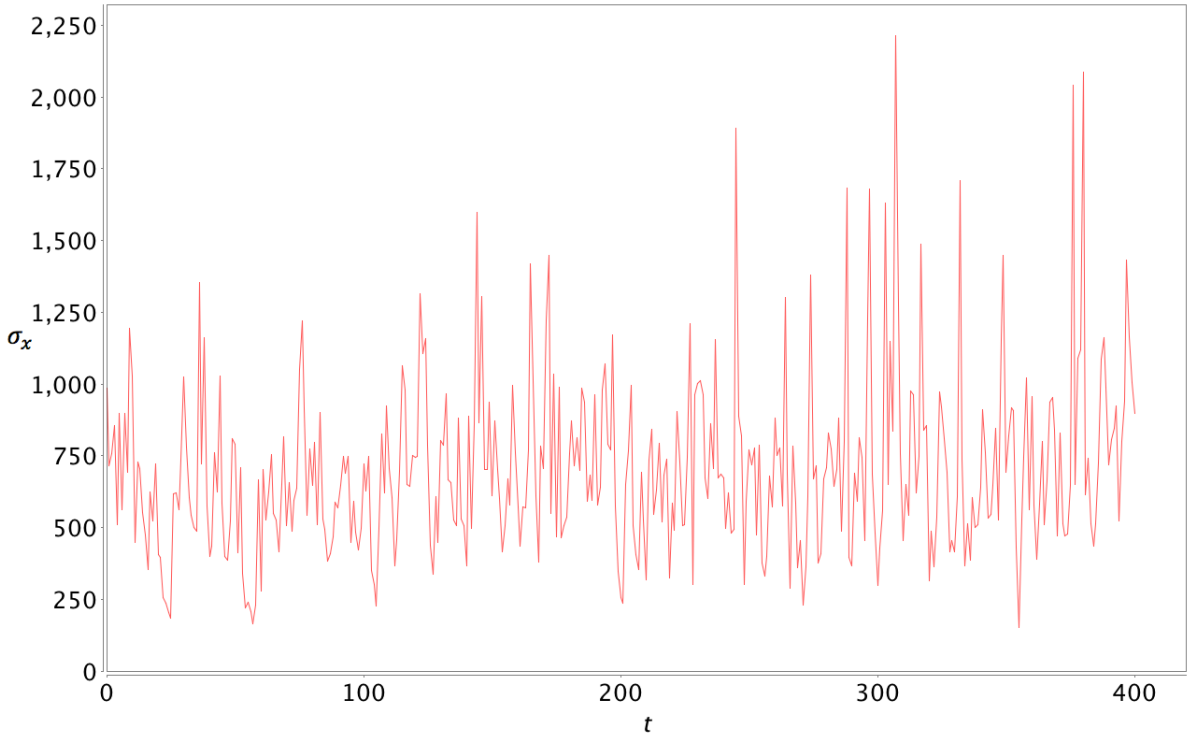

Рис. 3.5 Среднеквадратическое отклонение x_i при большом α

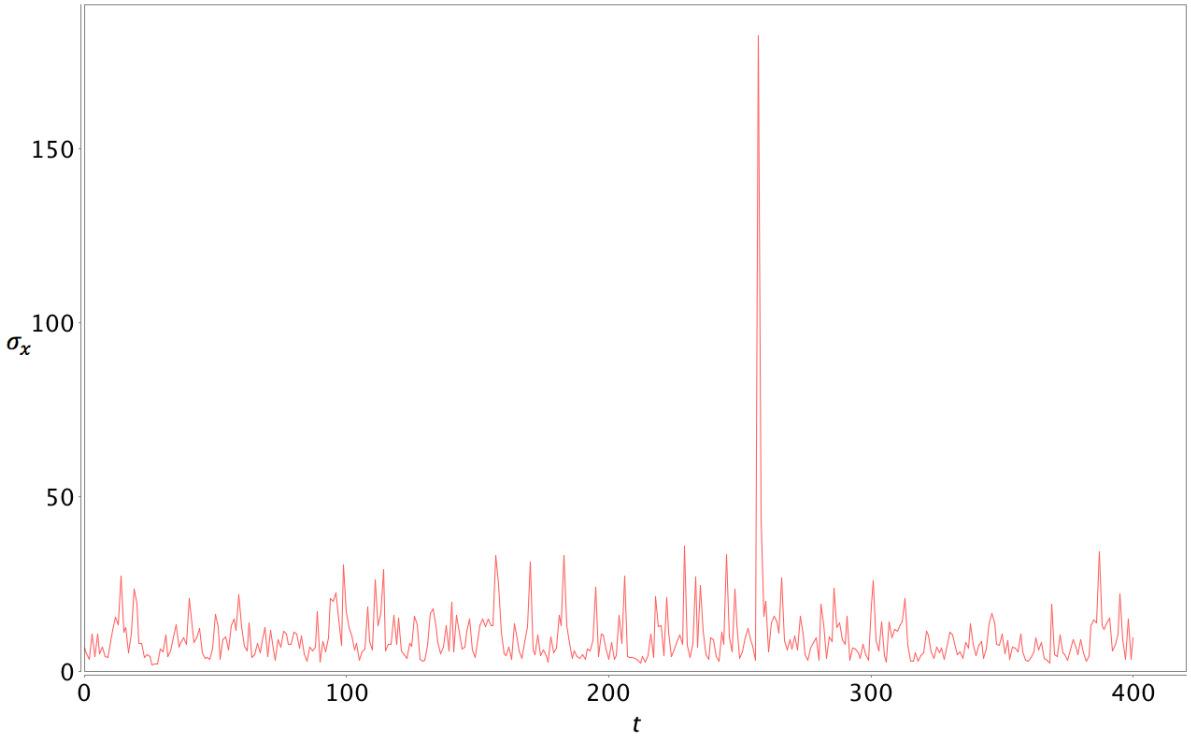

Рис. 3.6 Среднеквадратическое отклонение x_i при оптимальном α

По Рис. 3.4-3.6 можно судить о том, что чем меньше среднеквадратическое отклонение, тем больше система приближена к состоянию баланса.

Значения среднеквадратического отклонения x_i в зависимости от шага α определена следующим образом: шаг алгоритма изменяется в заранее исследованном диапазоне значений, меньше или больше которых сходимость алгоритма заметно ухудшается, при текущем значении шага вычисляется значение среднеквадратического отклонения.

Таблица 3.1. Значения среднеквадратического отклонения x_i при различном шаге алгоритма

№ п/п	Шаг α	Среднеквадратическое отклонение x_i, з/сек				
		2	3	4	5	6
1	0.0005	928.204825	396.483004	644.316195	614.218799	516.532748
2	0.001	415.118151	270.222106	304.013423	301.891068	263.916150
3	0.005	65.906657	61.705239	80.430630	60.583292	86.688483
4	0.01	57.769075	53.272472	45.645000	30.562524	39.844338
5	0.05	14.379905	15.991839	15.358416	10.985569	15.398492
6	0.07	9.832618	14.252967	14.625289	9.807895	10.891037
7	0.1	12.841626	12.817092	15.305487	10.434856	9.149602
8	0.13	14.764335	11.729830	21.141965	21.141965	11.107989
9	0.15	24.997534	12.720037	25.723547	28.038499	34.705907
10	0.2	231.893560	174.771698	354.068806	201.461212	275.177030
11	0.3	1394.01852	883.311296	994.170567	1026.18197	881.073026

Пусть система в начале работы получает некоторое количество заданий, а также отсутствуют постоянные и пиковые нагрузки, тогда Рис. 3.7 показывает зависимость количества тактов, необходимых для выполнения начального количества заданий, от шага алгоритма α.

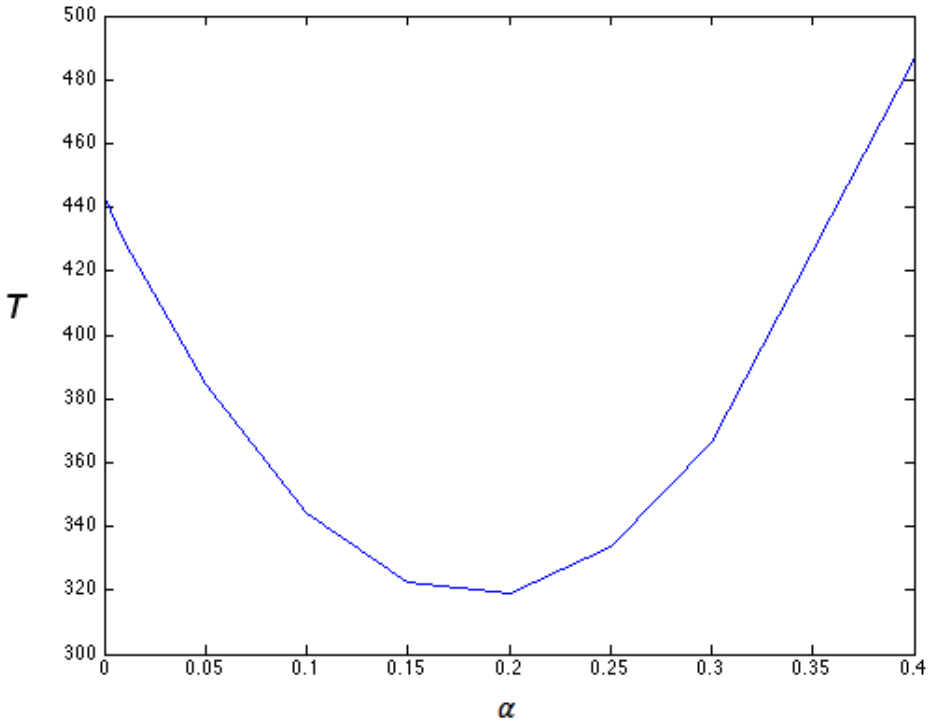

Рис. 3.7 Зависимость времени выполнения заданий T от шага α ($d = 15$)

Из последнего графика (Рис. 3.7) можно сделать вывод о том, что α определяет поведение системы, работающей в оптимальном режиме только тогда, когда шаг алгоритма оптимален. Вследствие этого требуется усовершенствовать алгоритм с целью достижения наилучшего времени выполнения заданий.

Так как в алгоритме SPSA присутствует еще один параметр - β, требовалось исследовать его влияние на работу системы.

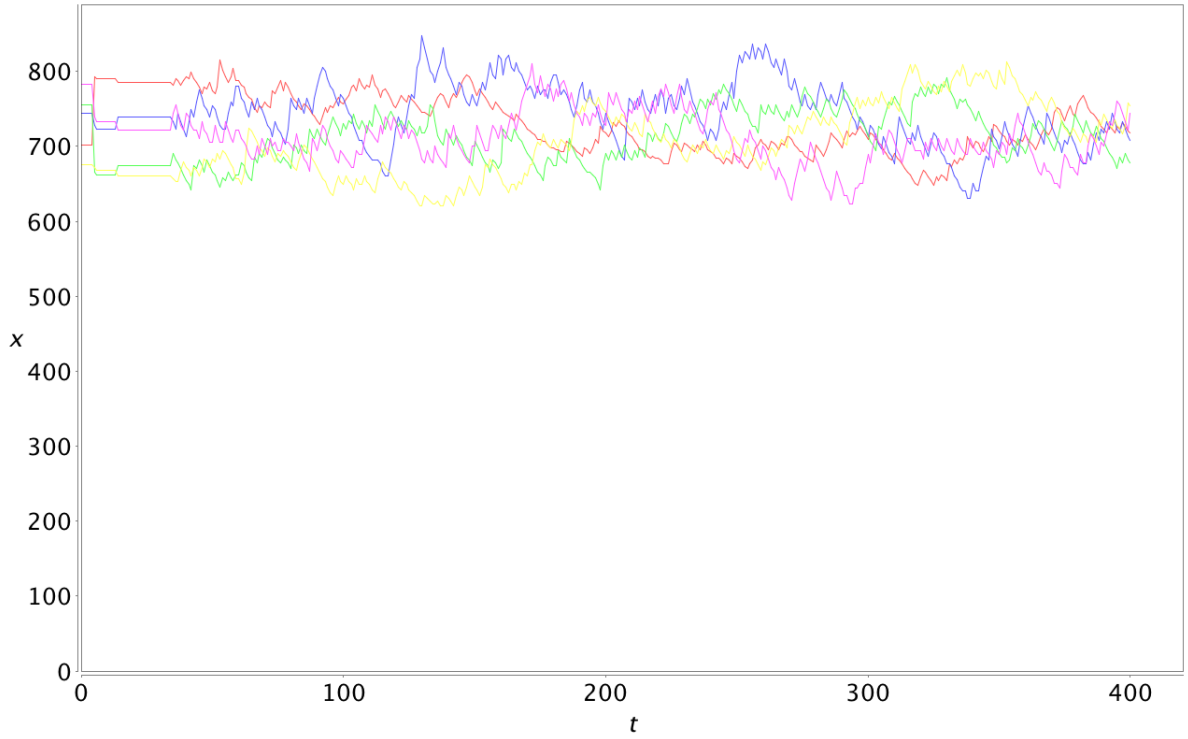

Рис. 3.8 Поведение системы при $\beta = 0.1$

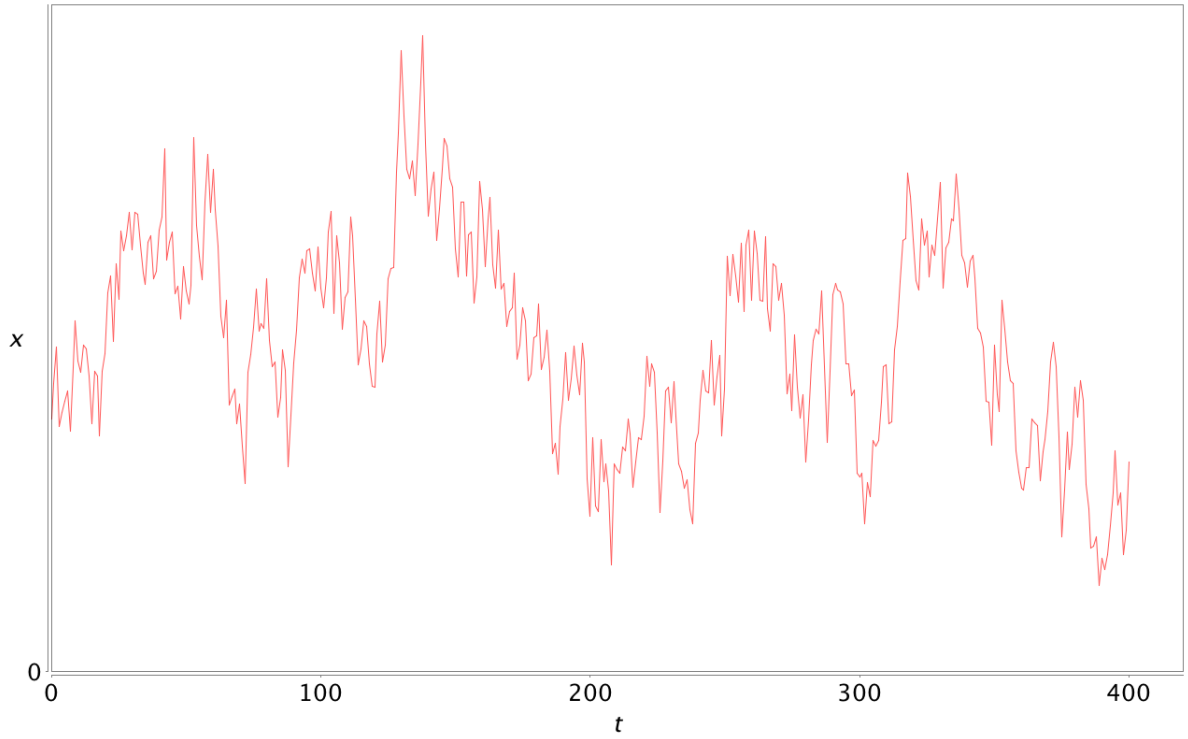

Рис. 3.9 Среднеквадратическое отклонение x_i при $\beta = 0.1$

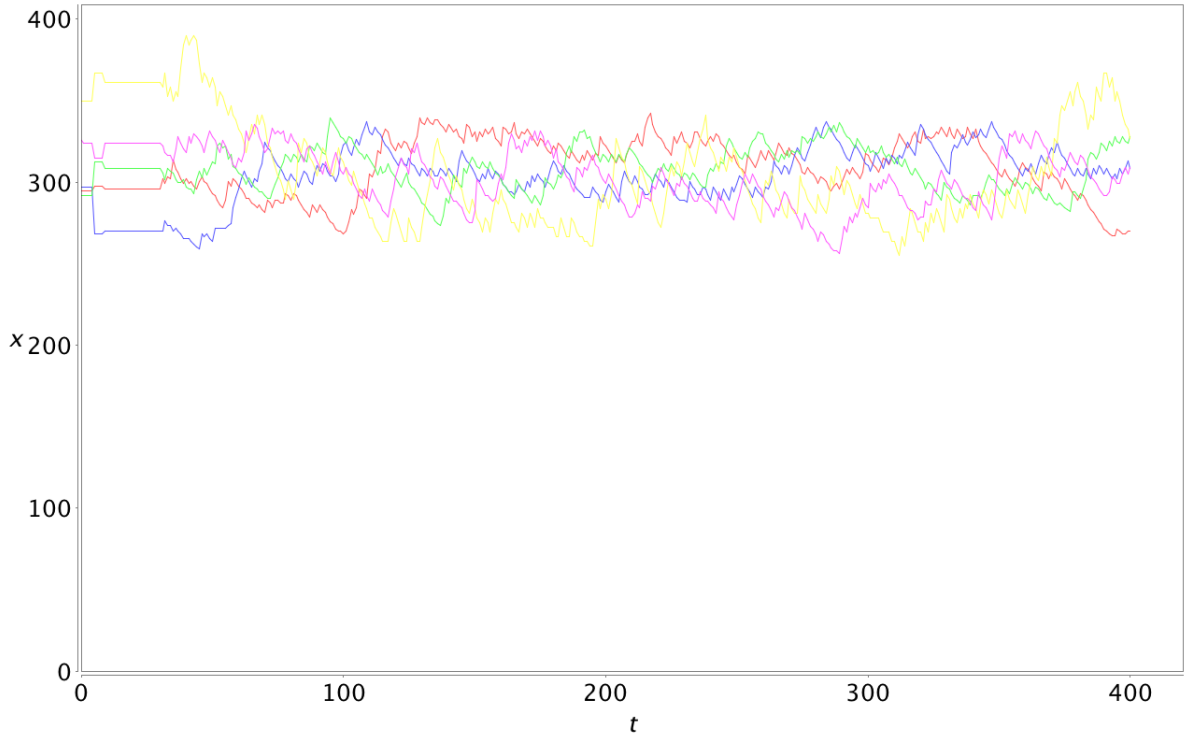

Рис. 3.10 Поведение системы при $\beta = 0.5$

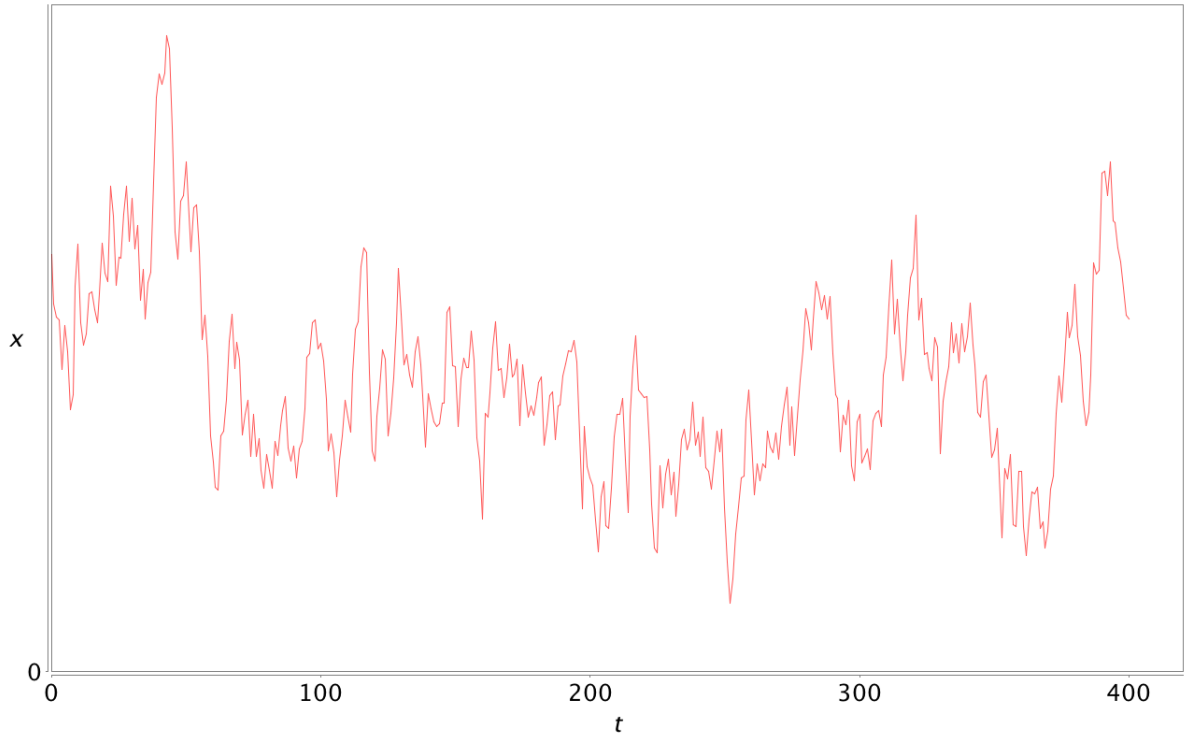

Рис. 3.11 Среднеквадратическое отклонение x_i при $\beta = 0.5$

Полученные результаты (Рис. 3.8-3.11) свидетельствуют о том, что β незначительно влияет на работу системы, поэтому имеет смысл зафиксировать этот параметр. В диссертационной работе β принимается равным 0.1.

3.2. Оптимизация шага алгоритма

Выбирается натуральное достаточно большое число N (например, $N = 500$). Общее время работы сервера разбивается на определенные интервалы (такты) $t_0, t_1, \ldots, t_k, \ldots$, по правилу:

t_0 - время начала работы сервера;

t_1 - время окончания выполнения первых N заданий;

...

t_k - время окончания выполнения k-ой серии из N заданий;

...

Также стоит отметить, что такты можно задавать различными способами, например, фиксированным интервалом времени сервера.

В качестве функционала качества, исходя из проведенных имитационных экспериментов, целесообразно выбрать следующую функцию:

$$\Phi_t(\alpha_t) = \sum_{i=1}^{N} \sqrt{\frac{1}{d} \sum_{j=1}^{d} \left(x_t^j(\alpha_t) - \bar{x}(\alpha_t)\right)^2} \to \min_{\alpha_t} \tag{3.2}$$

где j - верхний индекс, обозначающий номер узла.

$\Phi_t(\alpha_t)$ характеризует степень сбалансированности системы, таким образом, если система пришла к балансу, то $\Phi_t(\alpha_t) = 0$. Стоит отметить то, что достижение баланса является относительным, и нулевого значения функции достигнуть невозможно для случая, когда в систему постоянно поступают задания. В реальности $\Phi_t(\alpha_t)$ будет только стремиться к нулевому значению.

В общем виде алгоритм оптимизации α выглядит следующим образом:

Задается начальное значение α_0 из интервала $[0.05; 0.5]$, $\gamma > 0, t = 0, 1, \ldots$

Определяются новые значения

$$\alpha_t^{\pm} = \alpha_{t-1} \pm \gamma\delta_t \tag{3.3}$$

Находятся значения функционала качества

$$s_t^{\pm} = \Phi_{2t-1}(\alpha_t^{\pm}) \tag{3.4}$$

Вычисляется квазиградиент

$$\hat{\nabla}_t = \triangle_t \frac{s_t^+ - s_t^-}{2\gamma} \tag{3.5}$$

Получение новой оценки

$$\hat{\alpha}_{2t-1} = \hat{\alpha}_{2t-2} \tag{3.6}$$

$$\hat{\alpha}_{2t} = \hat{\alpha}_{2t} - \gamma\hat{\nabla}_t \tag{3.7}$$

4. Реализация балансировщика

4.1. Программная платформа «Akka»

Для реализации алгоритма был выбран Akka framework, который позволяет создать распределенную систему на основе акторов. Актор представляет собой сущность, которая предоставляет следующие возможности:

- простые высокоуровневые абстракции для параллельной работы;
- высокопроизводительную асинхронную неблокирующую модель для реализации событий;
- легковесное управление процессами.

Выбор вышеописанного ПО обусловлен несколькими причинами:

- необходимость моделирования распределенной системы;
- взаимодействие между узлами, приближенное к реальному;
- возможность отключения оптимизирующей логики, которая могла бы повлиять на результаты моделирования алгоритма.

4.2. Архитектура прототипа системы

Поступающие запросы из внешней сети направляются к Frontend серверу, служащему связующим звеном между внешним миром и системой Web-серверов. Каждый Web-сервер представляет собой виртуальную машину. Frontend сервер перенаправляет запрос к одному из балансирующих акторов, который в свою очередь регулирует поток заданий, отдаваемых Backend серверу. Использование данной структуры позволяет распределять нагрузку между виртуальными машинами посредством взаимодействия акторов друг с другом.

Система контроля служит для определения состояния серверов, для сбора статистики, а также для автоматизированного развертывания виртуальных машин.

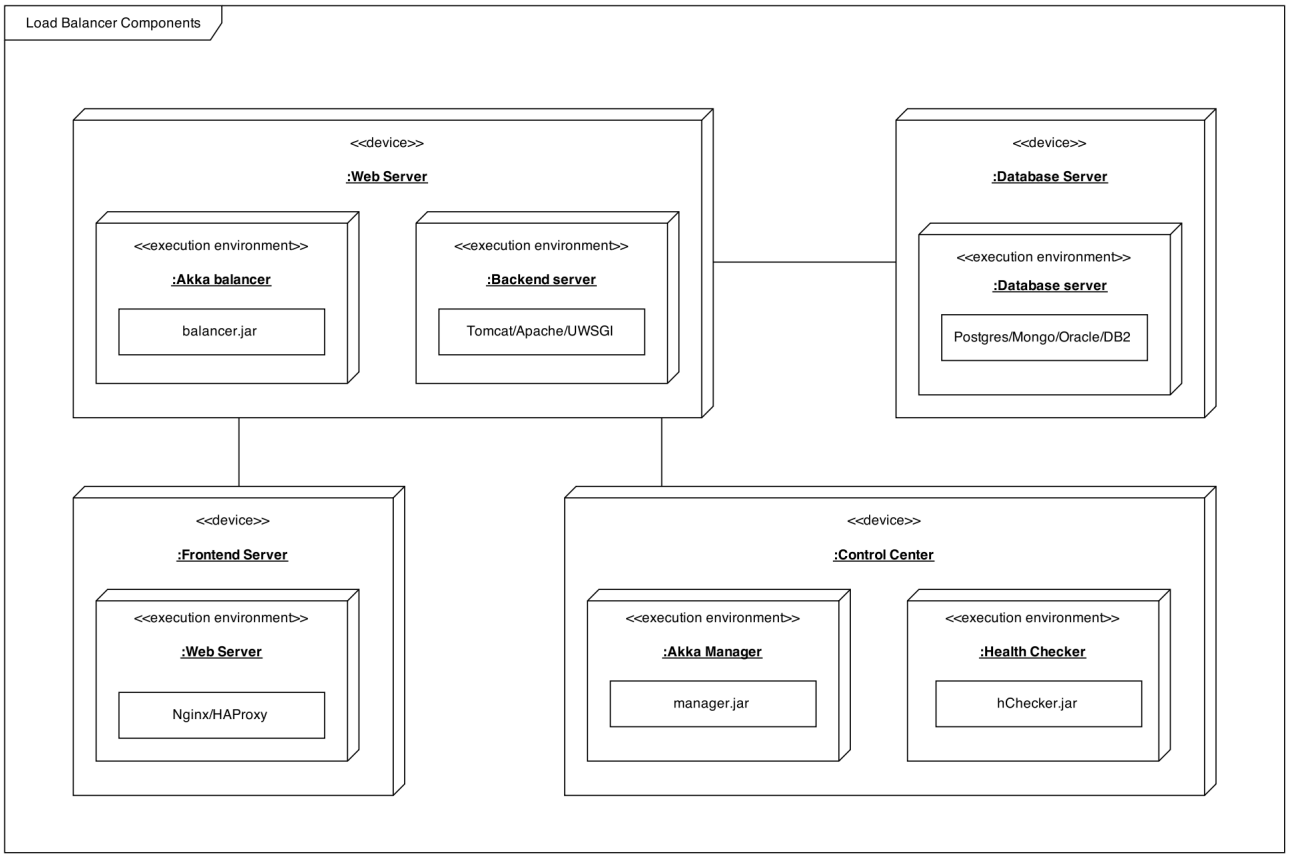

Рис. 4.1 Диаграмма развертывания

4.3. Результаты

4.3.1. Сравнение адаптивной и неадаптивной версий алгоритма SPSA

В текущем разделе приведены результаты имитационных экспериментов для сравнения работы адаптивного алгоритма и первоначального при одинаковых условиях.

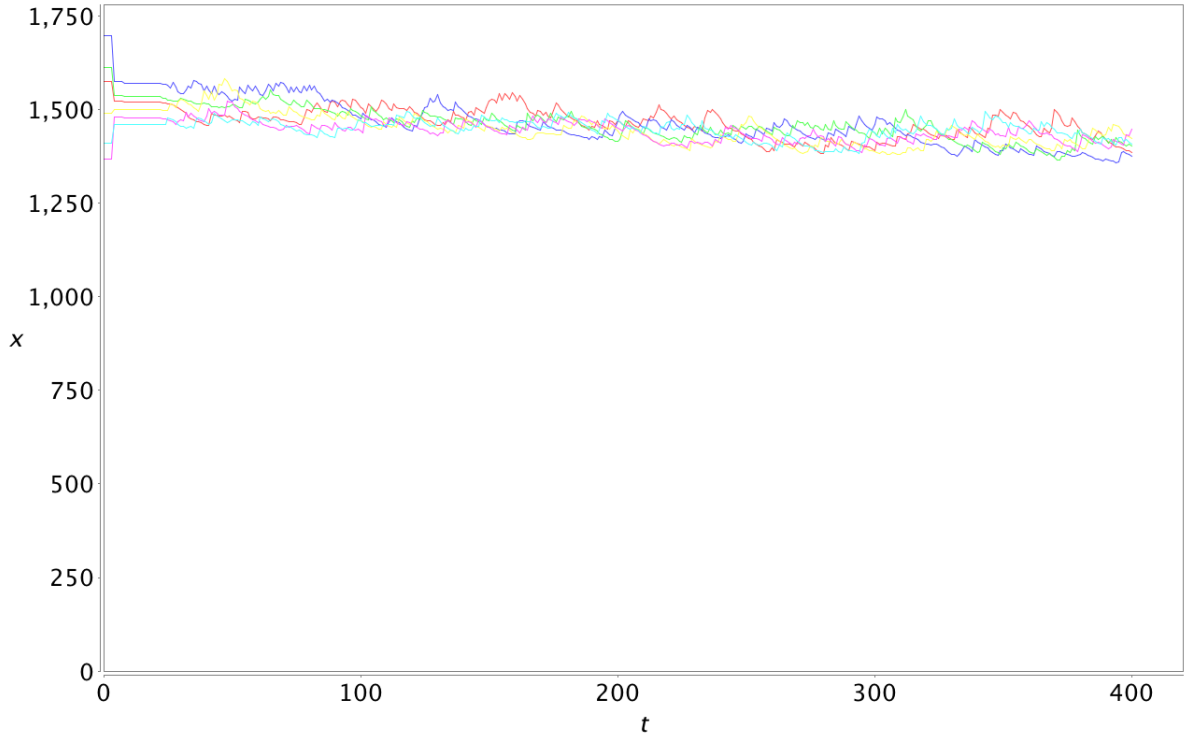

Рис. 4.2 Адаптивный алгоритм с шагом $\alpha_0 = 0.006$

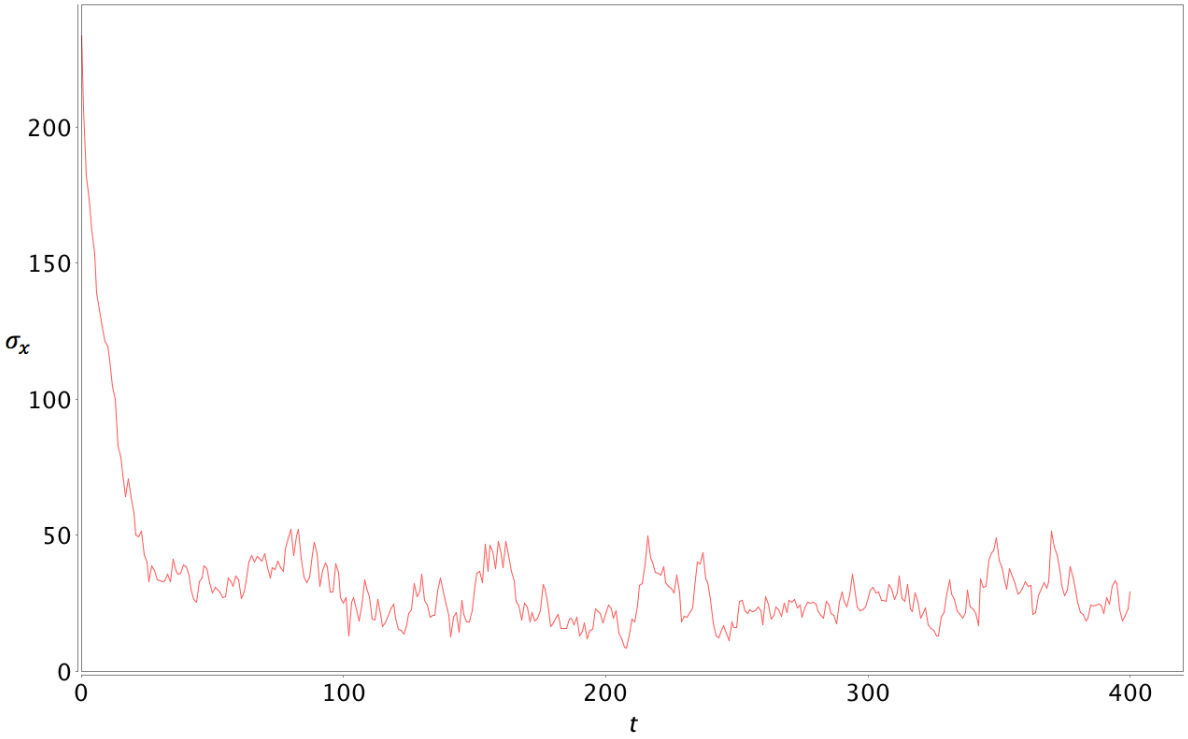

Рис. 4.3 Среднеквадратическое отклонение x_i при $\alpha_0 = 0.006$ в адаптивном алгоритме

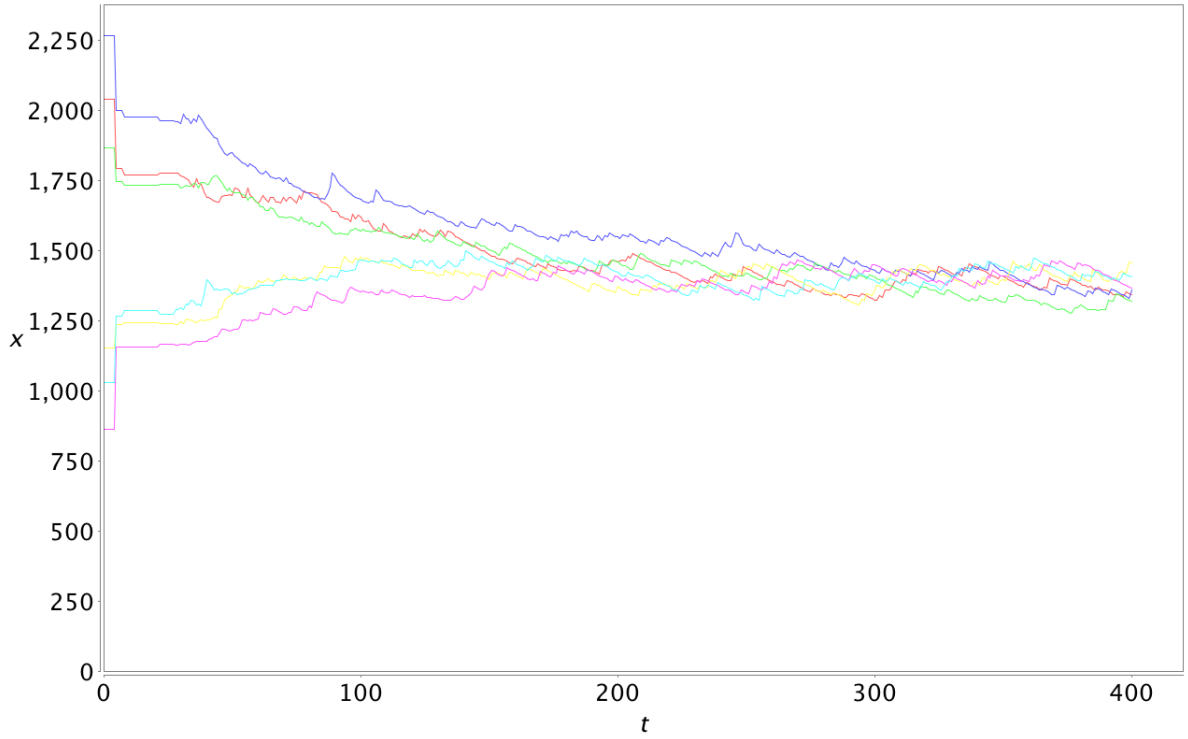

Рис. 4.4 Неадаптивный алгоритм с шагом $\alpha_0 = 0.006$

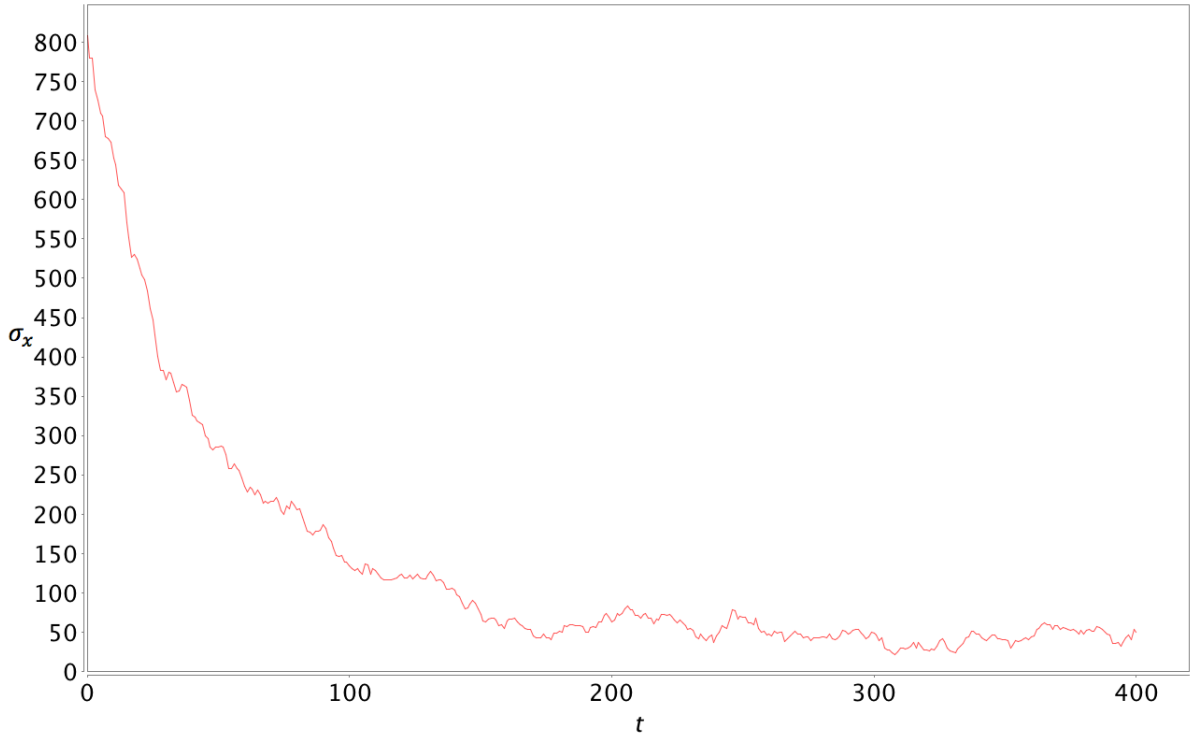

Рис. 4.5 Среднеквадратическое отклонение x_i при $\alpha_0 = 0.006$ в неадаптивном алгоритме

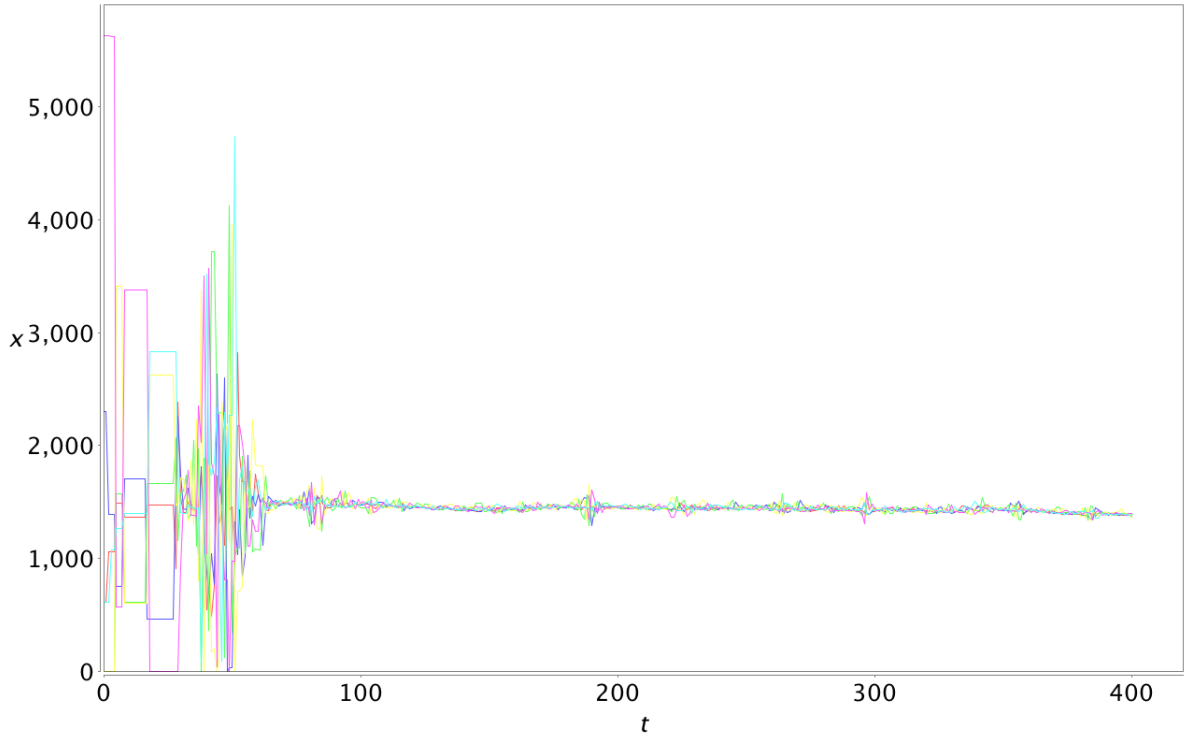

Рис. 4.6 Адаптивный алгоритм с шагом $\alpha_0 = 0.4$

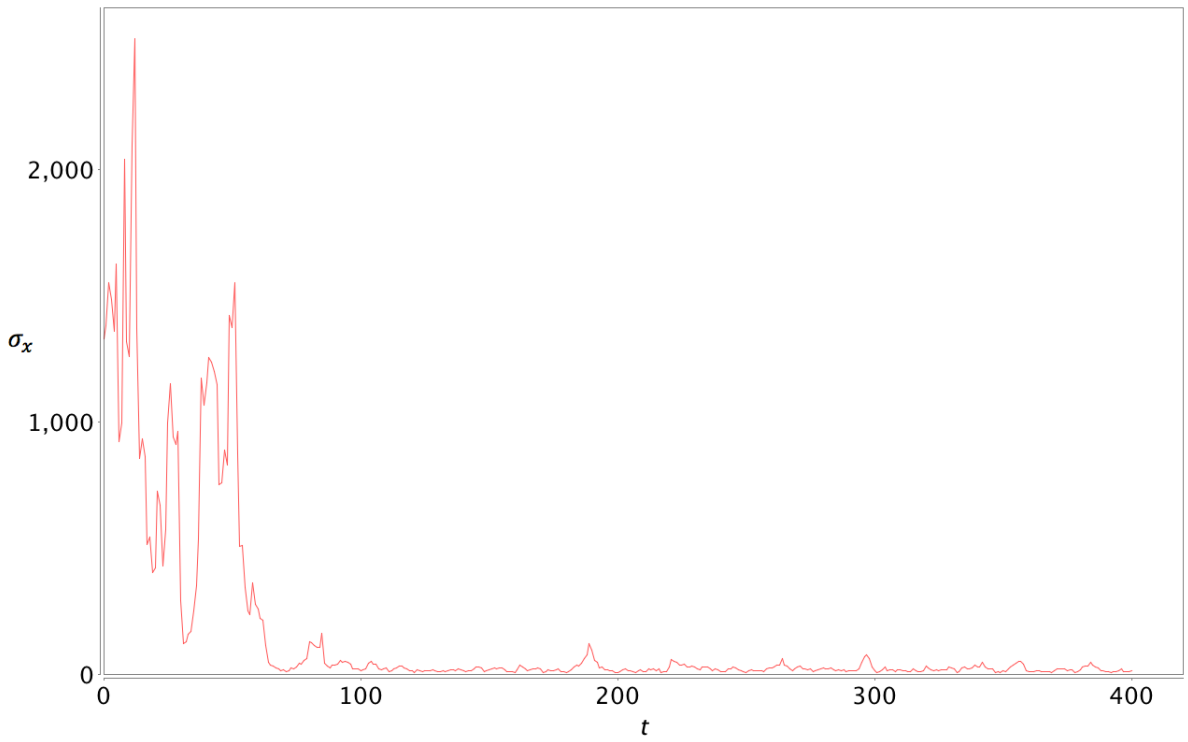

Рис. 4.7 Среднеквадратическое отклонение x_i при $\alpha_0 = 0.4$ в адаптивном алгоритме

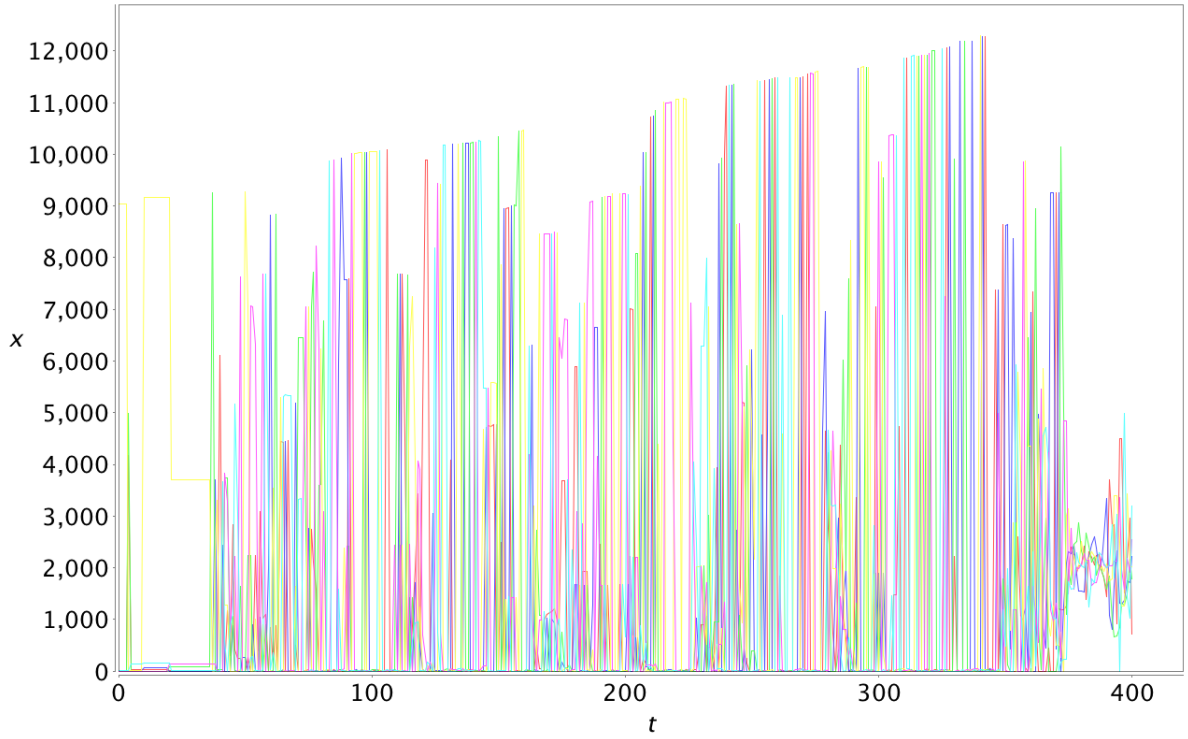

Рис. 4.8 Неадаптивный алгоритм с шагом $\alpha_0 = 0.4$

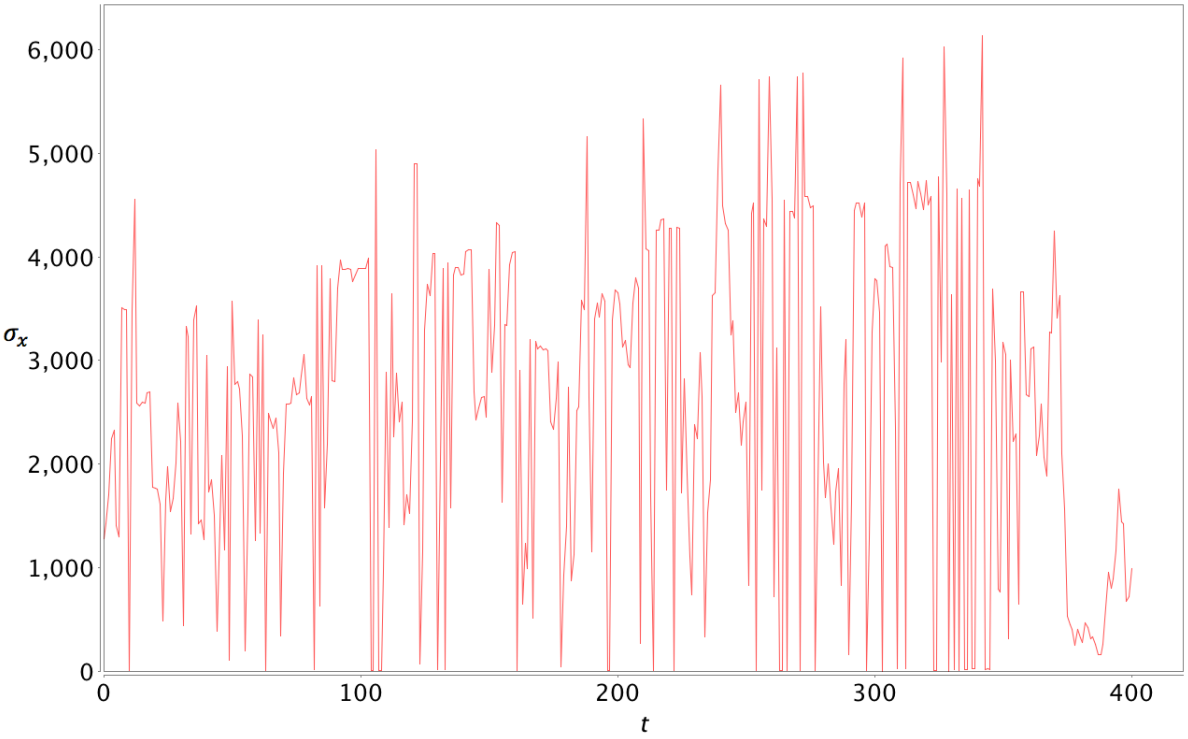

Рис. 4.9 Среднеквадратическое отклонение x_i при $\alpha_0 = 0.4$ в неадаптивном алгоритме

4.3.2. Сравнение адаптивного алгоритма SPSA с другими алгоритмами

Для сравнения адаптивного SPSA с другими алгоритмами был выбран Round Robin, из-за его распространенности [30].

Round Robin, или алгоритм кругового обслуживания, представляет собой перебор по круговому циклу: первый запрос передаётся одному серверу, затем следующий запрос передаётся другому и так до достижения последнего сервера, а затем всё начинается сначала.

Самой распространёной имплементацией этого алгоритма является, метод балансировки Round Robin DNS. Как известно, любой DNS-сервер хранит пару «имя хоста — IP-адрес» для каждой машины в определённом домене. С каждым именем из списка можно ассоциировать несколько IP-адресов. DNS-сервер проходит по всем записям таблицы и отдаёт на каждый новый запрос следующий IP-адрес. В результате все серверы в кластере получают одинаковое количество запросов.

Пусть в системе имеется 5 узлов разной производительности, по распределению Пуассона генерируется некоторое количество заданий, поступающих в систему. Рассматривается работа двух алгоритмов при одинаковых условиях.

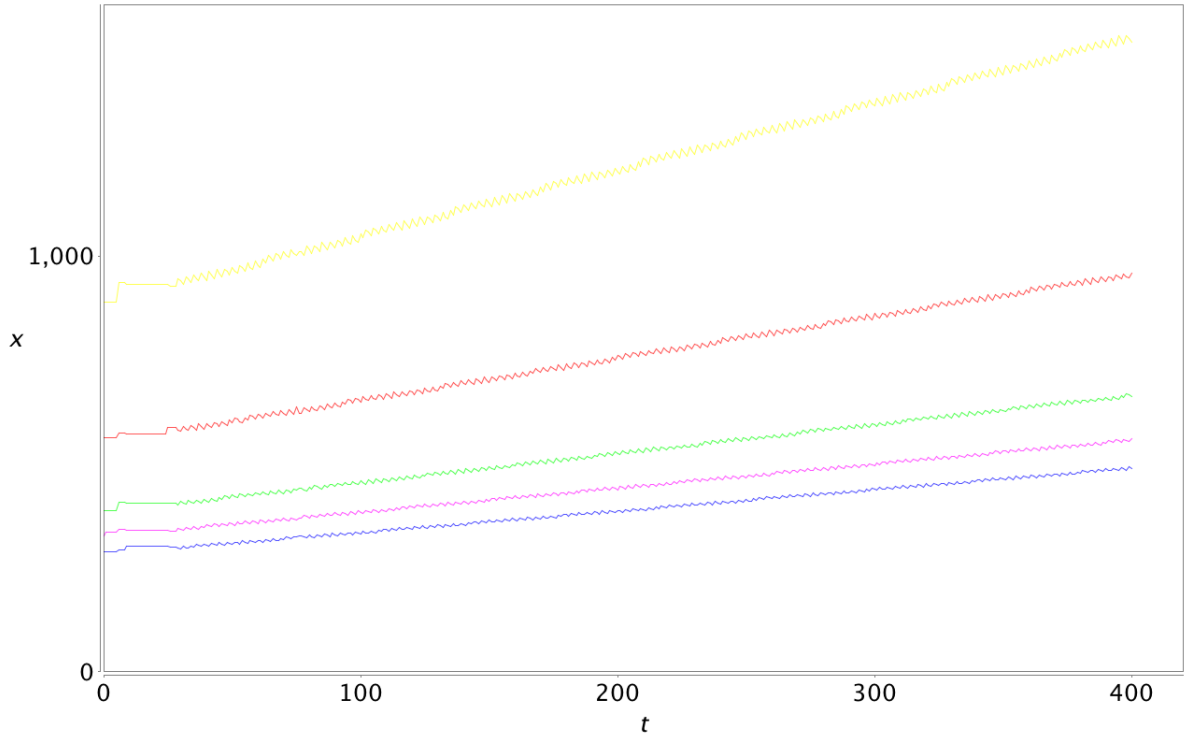

Рис. 4.10 Round Robin

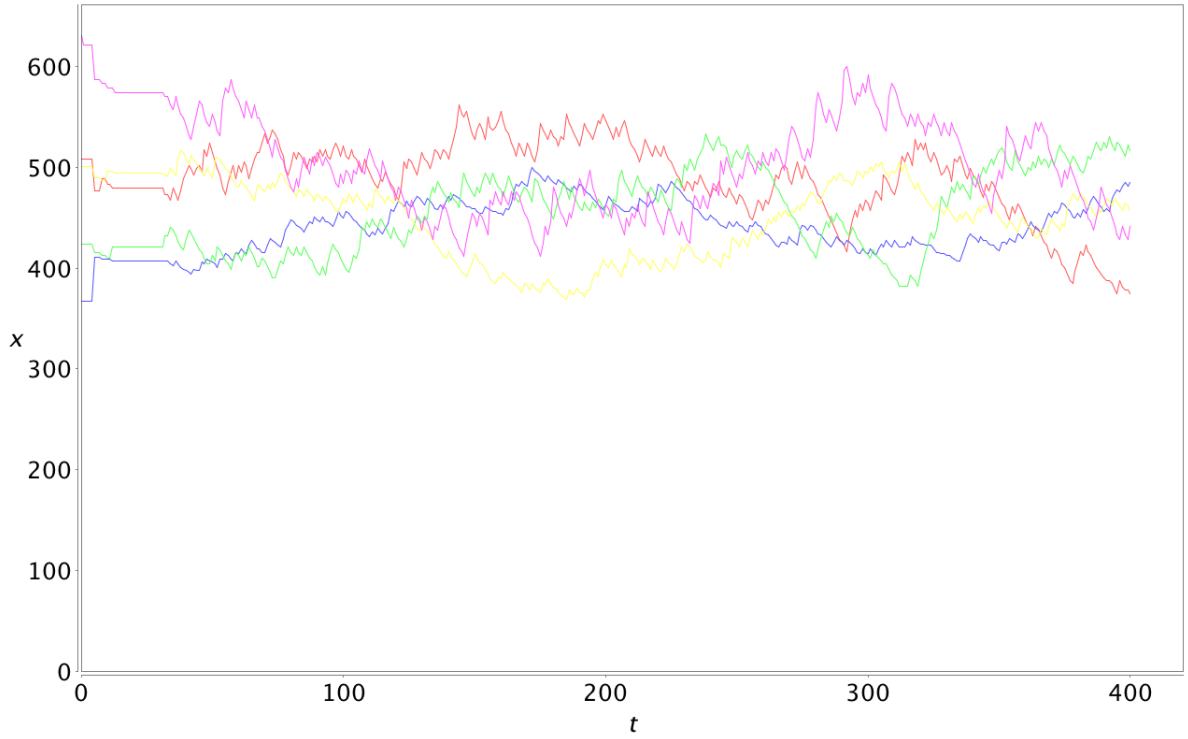

Рис. 4.11 Адаптивный SPSA

Как видно из полученных графиков (Рис. 4.10-4.11), Round Robin не справляется с количеством поступающих заданий. Однако, работа SPSA при тех же условиях не приводит к накапливанию заданий в системе.

Заключение

В рамках диссертационной работы проведено исследование возможности применения алгоритма SPSA для отслеживания оптимального параметра α в процессе балансировки системы. Определены факторы, взаимосвязанные с влиянием параметра α на работу алгоритма балансировки. Реализован алгоритм SPSA для отслеживания производительностей с использованием постоянного шага α. Предложена и реализована адаптация алгоритма на базе SPSA для отслеживания производительностей совместно с параметром α. Проведен сравнительный анализ полученного алгоритма с существующим, который использует постоянный шаг. Также спроектирована архитектура прототипа системы, использующей балансировку нагрузки и реализован компонент, отвечающий непосредственно за балансировку.

Список литературы

1. Амелина Н. О., Фрадков А. Л. Приближенный консенсус в стохастической динамической сети с неполной информацией и задержками в измерениях // Автоматика и телемеханика. 2012. № 11. С. 6–29.
2. Граничин О. Об одной стохастической рекуррентной процедуре при зависимых помехах в наблюдении, использующей на входе пробные возмущения // Вестник Ленинградского университета. 1989. № 1. С. 19–21.
3. Граничин О. Процедура стохастической аппроксимации с возмущением на входе // Автоматика и телемеханика. 1992. № 2. С. 97–104.
4. Граничин О. Поисковые алгоритмы стохастической аппроксимации с рандомизацией на входе // Автоматика и телемеханика. 2015. № 5. С. 43–59.
5. Граничин О., Поляк Б. Рандомизированные алгоритмы оценивания и оптимизации при почти произвольных помехах. Москва: Наука, 2003.
6. Поляк Б., Цыбаков А. Б. Оптимальные порядки точности поисковых алгоритмов стохастической аппроксимации. 1990. Т. 26. С. 126–133.
7. Хританков А. С. Модели и алгоритмы распределения нагрузки // Информационные технологии и вычислительные системы. 2009. № 2. С. 65–80.
8. Amelina N., Fradkov A., Jiang Y., Vergados D. Approximate consensus in stochastic networks with application to load balancing // IEEE Transactions on Information Theory. 2015. Vol. 61, no. 4. P. 1739–1752.
9. Blum J. Multidimensional stochastic appoximation // The Annals of Mathematical Statistics. 1954. Vol. 9. P. 737–744.
10. Borkar V. Stochastic Approximation. A Dynamical Systems Viewpoint. Cambridge University Press, 2008.
11. Granichin O., Volkovich V., Toledano-Kitai D. Randomized Algorithms in Automatic Control and Data Mining. Springer, 2015.
12. Granichin O., Gurevich L., Vakhitov A. Discrete-time minimum tracking based on stochastic approximation algorithm with randomized differences. Shanghai,

China: 48th Conf. Decision and Control, 2009. P. 5763–5767.

13. Granichin O., Amelina N. Simultaneous Perturbation Stochastic Approximation for Tracking under Unknown but Bounded Disturbances // IEEE Transactions on Automatic Control. 2015. Vol. 60. P. 1653–1658.
14. Goldberg D. Genetic algorithms. Pearson Education India, 2006.
15. Kushner H., Yin G. Stochastic Approximation Algorithms and Applications. New York: Springer–Verlag, 2002.
16. Kiefer J., Wolfowitz J. Statistical estimation on the maximum of a regression function // The Annals of Mathematical Statistics. 1952. Vol. 23. P. 462–466.
17. Kameda H. Optimal Load Balancing In Distributed Computer Systems. Springer, 2011.
18. Robbins H., Monro S. A stochastic approximation method // The Annals of Mathematical Statistics. 1951. Vol. 22. P. 400–407.
19. Spall J. Multivariate Stochastic Aproximation Using a Simultaneous Perturbation Gradient Aproximation // IEEE Transactions on Automatic Control. 1992. Vol. 37, no. 3. P. 332–341.
20. Spall J. A one–measurement form of simultaneous perturbation stochastic approximation // Automatica. 1997. Vol. 33. P. 109–112.
21. Tempo R., Calafiore G., Dabbene F. Algorithms for Analysis and Control of Uncertain Systems: with Applications. New York: Springer–Verlag, 2013.
22. Tsitsiklis J., Bertsekas D., Athans M. Distributed asynchronous deterministic and stochastic gradient optimization algorithms // IEEE Transactions on Automatic Control. 1986. Vol. 31, no. 9. P. 803–812.
23. Tanenbaum A., Steen M. V. Distributed systems. Pearson Prentice Hall, 2007.
24. Балансировка нагрузки. URL: https://en.wikipedia.org/wiki/Load_balancing_(computing) (дата обращения: 30.01.2015).
25. Least Time Load-Balancing Algorithm. URL: http://nginx.com/blog/nginx-plus-r6-released/ (дата обращения: 30.01.2015).
26. HAProxy. URL: http://www.haproxy.org/ (дата обращения: 30.01.2015).

27. Elastic Load Balancing. URL: http://aws.amazon.com/elasticloadbalancing/ (дата обращения: 30.01.2015).
28. Google HTTP/HTTPS. URL: https://cloud.google.com/compute (дата обращения: 30.01.2015).
29. Internet Traffic Trends. URL: https://www.nanog.org/meetings/nanog43/presentations/Labovitz_internetstats_N43.pdf (дата обращения: 30.01.2015).
30. Балансировка нагрузки: основные алгоритмы и методы. URL: http://blog.selectel.ru/balansirovka-nagruzki-osnovnye-algoritmy-i-metody/ (дата обращения: 30.01.2015).