УДК 004.7

А.А. Козинский, В.И. Басин

ОБРАБОТКА СОБЫТИЙ В СИСТЕМЕ ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ BRAIN EDUCATION

Статья содержит описание математической модели используемой автором дистанционной системы образования Brain Education для обработки отложенных многофазовых событий, распределения компонентов системы на нескольких рабочих машинах, снижения нагрузки на основной компонет (веб-приложение), распределения нагрузки между разрабатываемыми сервисами для обеспечения отказоустойчивости и обеспечения высокой производительности. Основными компонентами данной модели являются: источники событий, вещатели, очереди, обработчики, машины состояний и триггеры. Каждый компонент выполняет определённую задачу, продвигая сообщение вперёд, к следующим компонентам.

Дистанционная система образования Brain Education, разработанная автором, – многофункциональная высоконагруженная система, предназначенная для организации процесса дистанционного обучения, проведения соревнований, онлайн тестов и алгоритмических решений на различных языках программирования.

В подобной системе возникает потребность обработки большого количества фоновых операций, событий, тригером которых не является конечный пользователь, операции слишком затратные по производительности, однако требуют запуска не очень часто, поэтому их лучше выполнять как можно реже или выполнение которых с ожиданием завершения предоставит неудобство пользователю из-за существенных по длительности моментов ожидания завершения операции. К примеру: создание таблицы результатов соревнования, отправка электронной почты с инструкциями по восстановлению пароля, создание рассылок по почте с новостями, результатами соревнований и контрольных работ, процесс построения рейтинговой таблицы, тестирования алгоритмических задач (этот процесс может длиться несколько минут, поэтому пользователь не захочет ждать отклика системы так долго, будучи заблокированным для каких-либо других действий). Кроме того, существуют события, основанные на целой серии определённых действий пользователя, которые необходимо обработать, быть может, через месяц, на основании данных действий (например, пользователь отправил наилучшее решение задачи в начале соревнования и затем выложил разбор этой задачи в своём блоге – о таком событии необходимо уведомить всех пользователей системы, участвовавших в данном соревновании).

Для решения подобных проблем, а также обеспечения системе масштабируемости путём разнесения её составных компонентов на различные машины была спроектирована следующая математическая модель обработки отложенных событий.

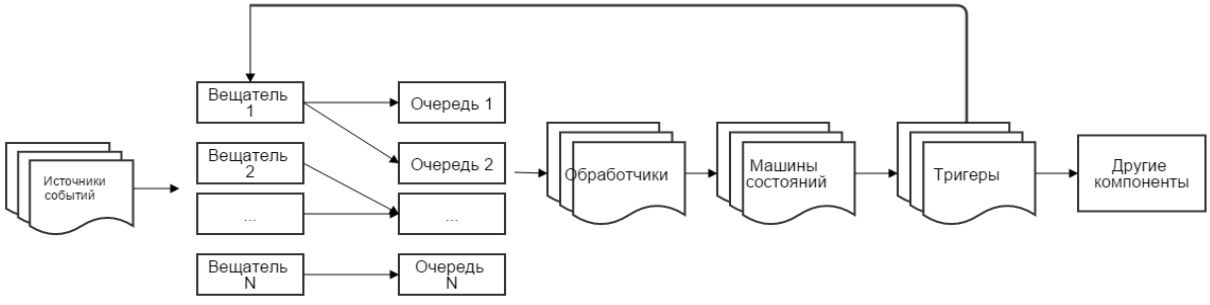

Рисунок 1 – Схема обработки событий

В приведенной схеме (рисунок 1) в качестве источников событий могут выступать любые компоненты системы, которые генерируют события каких-либо необходимых вычислений или обработок. Каждый из таких генераторов сконфигурирован таким образом, чтобы сообщать определённому вещателю о сгенерированном событии.

Вещатели кладут полученные сообщения в специальные очереди сообщений. Вещатель может положить сообщение в несколько очередей одновременно, если событие необходимо обработать несколькими компонентами системы параллельно. Для реализации подобной функциональности существует специальная «карта», позволяющая соответствующим образом сконфигурировать вещатели.

Очереди сообщений хранят записи о сгенерированных событиях и занимаются распределением этих событий между разработчиками, подписанными на определённые очереди. Очереди сообщений также выполняют функцию балансировщика нагрузки путём распределения задач между несколькими обработчиками на основе их загруженности. К примеру, если у нас существует 3 одновременно запущенных сервиса отправки почты, установленных как на одной машине, так и на нескольких, то очереди сообщений будут распределять задания на отправку почты между тремя сервисами в зависимости от их загруженности. Кроме того, подобные очереди обеспечивают возможность масштабирования системы путём разнесения составных компонентов и сервисов на разные машины и привязки их к определённым очередям. В таком случае каждый процесс должен быть независим от другого процесса или использовать те же очереди для обработки событий совместно с другими сервисами.

В качестве обработчиков событий выступают машины (компоненты распределённой системы), на которые установлен разработанный сервис рассылки почты, тестирования алгоритмов, подвода статистики, анализа различного рода активности в системе дистанционного обучения и другие сервисы. Они подписаны на те очереди сообщений, где хранятся события для них, и как-только они поступают, обработчик начинает выполнение необходимой операции.

После того, как обработчик закончил обработку события он переключает состояние соответствующей машины состояния. К примеру, существует событие, когда пользователь решит 100 задач, отправить ему на электронную почту уведомление, что он выиграл подарочную майку. Поэтому как только пользователь успешно решил какую-либо задачу, необходимо обновить счётчик решённых на соответствующей машине состояний.

После того, как состояние события было обновлено, триггеры выполняют анализ текущего состояния, а затем передачу статуса события следующему компоненту системы или снова сообщают вещателю о том, что событие следует положить в очередь для дальнейшей обработки. К примеру, если пользователь решил 56 задач из 100, триггер снова передаст данное событие вещателю.

Полученный подход успешно используется в системе дистанционного обучения Brain Education. Созданы специальные инструменты за мониторингом состояния данной системы и оповещения администраторов о необходимости обслуживания компонентов описанной модели в случае их выхода из строя. Реализация данной модели позволила поднять надёжность системы, отказоустойчивости, распределить нагрузку на обработку сложных событий, требующих больших временных затрат.

В качестве очереди сообщений была использована библиотека RabbitMQ для NET [1]. RabbitMQ – платформа, реализующая систему обмена сообщениями между компонентами программной системы (Message Oriented Middleware) на основе стандарта AMQP (Advanced Message Queuing Protocol). RabbitMQ выпускается под Mozilla Public License. RabbitMQ создан на основе испытанной Open Telecom Platform, обеспе-

чивает высокую надёжность и производительность промышленного уровня и написан на языке Erlang. RabbitMQ состоит из:

- Сервера RabbitMQ
- Поддержки протоколов HTTP, XMPP и STOMP
- Клиентских библиотек AMQP для Java и .NET Framework (поддержка других языков программирования реализована в ПО других производителей)
- Различных плагинов (таких, как плагины для мониторинга и управления через HTTP или веб-интерфейс, или плагин «Shovel» для передачи сообщений между брокерами)

Поддерживается горизонтальное масштабирование для построения кластерной архитектуры. В качестве движка базы данных для хранения сообщений используется Mnesia.

Обработчики и триггеры в системе дистанционнного обучения – это разработанные автором windows-сервисы [2], установленные на дополнительных машинах. Подобные сервисы – это службы ОС Windows – приложения, автоматически (если настроено) запускаемые системой при запуске Windows и выполняющиеся вне зависимости от статуса пользователя.

Конфигурация обработчиков осуществляется с помощью соответствующего XML-элемента в файле конфигурации приложения.

```
        <worker type="Mailer" id="d852966b-c688-4fba-a1d2-cc251a5d435e"
name="Console_Mail_Runner" updateInterval="5000" mqKey="mails_send" />

<messaging host="localhost" user="guest" password="guest">
        <queues>
                <queue key="mails_send" exchange="mails_exchange"
routeKeys="send" queue="mails_send" durable="true" />
        </queues>
        <exchanges>
                <exchange key="mails_exchange" name="mails_exchange"
type="direct" durable="true" />
        </exchanges>
</messaging>

<mails>
        <account host="smtp.gmail.com" sender="bredu.support@gmail.com" pass-
word="****" port="587" enableSSL="true" />
</mails>
```

Листинг 1 – Конфигурация обработчика задач на рассылку почтовых сообщений

В данной конфигурации (Листинг 1) описаны 3 блока: worker, messaging, mails.

Блок worker описывает текущего обработчика. Первый атрибут type означает тип обработчика. Здесь возможны несколько вариантов: Mailer, Tester, StatisticsBuilder, Offerer, Rater. Тип обработчика Tester производит тестирование алгоритмических решений на соответствующей виртуальной машине, где он установлен. StatisticsBuilder производит анализ статистических данных, данных о действиях пользователей системы, генерирует отчёты, основываясь на данных в используемом хранилище данных и передаёт полученную информацию остальным обработчиками, например Offerer на основе успешности определённых пользователей может создавать задачи для Mailer для рассылки электронных писем с поздравлениями на их почтовые адреса. Rater строит таблицы рейтинга по всем компонентам дистанционной системы: движку системы

ведения блогов, олимпиадах, соревнованиях, тестах, успеваемости на соответствующих курсах и так далее.

Атрибут Id в конфигурации обработчика содержит уникальную строку для идентификации данного обработчика, чтобы в дальнейшем использовать значение этого атрибута для анализа статистических данных и логов конкретного обработчика, предоставляя специальный интерфейс для мониторинга работы компонентов системы. Ввиду отсутствия информативности о типе выполняемых обработчиком задач в значении атрибута Id введён дополнительный атрибут Name, предназначенный для более удобного мониторинга состояния системы.

Атрибут UpdateInterval позволяет регулировать время простоя сервиса для более гибкой настройки распределения нагрузки между обработчиками одного и того же типа, где на машинах с большим значением этого поля нагрузка сравнительно меньше машин с малым значением.

Атрибут mqKey используется для подписки на соответствующую очередь событий, предназначенных для процессинга обработчиками данного типа.

Блок messaging описывает каким образом обработчик будет взаимодействовать с системой обмена сообщениями. Описание соответствующих очередей находится в элементе queues, а вещателей – в элементе exchanges. В обоих секциях сообщается информация о схеме маршрутизации сообщений, чтобы перед запуском обработчика произвести соответствующую валидацию на стороне сервера обмены сообщениями.

Блок account описывает необходимые для отправки почты детали: сервер почты, информация для авторизации (имя пользователя и пароль к соответствующему аккаунту), порт и возможность использования защищённого соединения.

При старте сервиса обработчик считывает конфигурационную информацию, запускает проверку её валидности для избежания непредвиденного поведения системы и только потом начинает процесс обработки соответствующих сообщений.

```
static void Main(string[] args)
{
        MainAsync(args).Wait();
        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
        Environment.Exit(0);
}
static async Task MainAsync(string[] args)
{
        try
        {
                if (args.Length > 0 && args[0] == "/setup")
                        await Task.Run(() => SetupMessaging());
                else
                        await RunWorkerAsync();
        }
        catch (Exception e)
        {
                DependencyResolver.Logger.Error("EXCEPTION", e);
        }
}
private static void SetupMessaging()
{
        DependencyResolver.LoadSetupModules();
        DependencyResolver.Logger.Debug("Configuring....");
```

```
        var configurator = DependencyResolver.GetService<Configurator>();
        configurator.Configure();
        DependencyResolver.Logger.Debug("Done");
}
private static async Task RunWorkerAsync()
{
        DependencyResolver.LoadWorkerModules();
        var mailWorker = DependencyResolver.GetService<MailWorker>();
        await mailWorker.StartAsync();
}
```

Листинг 2 – Консольный вариант обработчика

В приведенном фрагменте кода (Листинг 2) показана возможность запуска обработчика в виде обычного консольного приложения. Запуск этого приложения может осуществляться с параметром или без. Если указан параметр “/setup”, то обработчик на основе конфигурационной информации, указанной в файле конфигурации обработчика, произведёт настройку сервера обмена сообщениями (создание каналов вещания, привязка к ним новых очередей сообщений, настройка маршрутизации и др.) для принятия и отправки соответствующих событий через систему другим её компонентам.

Метод RunWorkerAsync запускает непосредственно процесс обработки сообщений в очереди. В первую очередь он загружает модуль со всеми необходимыми для него сервисами и зависимости для последующего автоматического внедрения этих зависимостей в соответствующие объекты, декларирующие определённые зависимости от других компонентов кода. Такой подход в реализации внедрения зависимостей называется Dependency Injection в рамках концепта Inversion of Control [3] – широко используемой практики в мире программирования. Альтернативой подобному подходу является создание класса напрямую, используя его конструктор зависимым от него объектом. Инверсия управления (Inversion of Control, IoC) – важный принцип объектно-ориентированного программирования, используемый для уменьшения связанности в компьютерных программах. Данное понятие подразумевает, что модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и другие должны зависеть от абстракции, а абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций. Одной из реализаций IoC является внедрение зависимостей (dependency injection). Внедрение зависимости используется во многих фреймворках, которые называются IoC-контейнерами. Если сравнить с более низкоуровневыми технологиями, IoC-контейнер – это компоновщик, который собирает не объектные файлы, а объекты ООП (экземпляры класса) во время исполнения программы. Очевидно, для реализации подобной идеи было необходимо создать не только сам компоновщик, но и фабрику, производящую объекты. Аналогом такого компоновщика (естественно, более функциональным) является компилятор, одной из функций которого является создание объектных файлов. Предоставление программисту инструментов внедрения зависимостей даёт значительно бóльшую гибкость в разработке и удобство в тестировании кода.

После загрузки всех необходимых модулей запускается асинхронный процесс, который собственно и выполняет обработку сообщений. Завершить работу обработчика, запущенного в режим консоли можно простым сочетанием клавиш Ctrl + C. Для остановки работы обработчика, запущенного в виде службы Windows, необходимо остановить соответсвующий сервис в Диспетчере задач или с помощью консоли управления всеми сервисами системы.

```
public override async Task ProcessTaskAsync(string queueStr)
{
        var taskId = Guid.Parse(queueStr.Substring(queueStr.IndexOf(':') + 1));
        var mailTask = _mailService.FindTask(taskId);
        mailTask.MarkStarted();
        await _mailService.CommitAsync();
        try
        {
                SendMailImmediatly(mailTask.Destination, mailTask.Subject, mail-
Task.Body);
                mailTask.MarkCompleted(true);
        }
        catch (Exception e)
        {
                mailTask.MarkFailed();
                _log.Error("Error processing task " + mailTask.Id, e);
        }
        await _mailService.CommitAsync();
}
```

Листинг 3. Фрагмент обработчика задач на рассылку электронных писем

В приведенном фрагменте кода (Листинг 3) описано каким образом идёт обработка сообщений в очереди. Изначально из текста сообщения узнаётся ссылка соответствующего задания mailTask на отправку в базе данных. Далее происходит процесс нахождения этого задания в хранилище и ставится дополнительная метка о том, что обработчик начал работу над соответствующим заданием. Отправка сообщения осуществляется специальным методом SendMailImmediatly, который по заданным получателю теме и сообщению создаёт электронное письмо, а затем производит отправку.

Таким образом, описанная система обработки событий позволила успешно распараллелить вычислительный процесс, благодаря модульности системы, возможности разнесения её составных компонентов на несколько рабочих машин, где обработчики и сервисы могут работать независимо от главного процесса. Реализованы несколько алгоритмов распределения нагрузки между сервисами обработки данных, событий, позволяющие эффективно настроить систему во время высоких нагрузок. Созданные инструменты мониторинга позволяют получать уведомления о состоянии системы в любой момент времени и анализа причин отказа любого компонента системы, основываясь на собранных логированной информации и статистике. Таким образом, приведенная модель успешно реализована и применена на практике в дистанционной системе образования Brain Education.

СПИСОК ЛИТЕРАТУРЫ

1. Высоконадёжная очередь сообщений RabbitMQ [Электронный ресурс]. – URL : http://rabbitmq.com. – Дата доступа : 20.10.2014.

2. Windows Servive [Электронный ресурс]. – URL : http://en.wikipedia.org/wiki/Windows_service. – Дата доступа : 20.10.2014.

3. Invertion of Control [Электронный ресурс]. – URL: http://en.wikipedia.org/wiki/Inversion_of_control. – Дата доступа : 20.10.2014.

A. Kazinski, V. Basin **Event Processing in the Brain Education Distance Learning System**

The article contains a description of the mathematical model used by the authors of remote education system Brain Education for deferred processing multiphase event distribution system components on multiple machines working, reduce the load on the main Comp (web application), the distribution of the load between the tailor services for fault tolerance and high performance . The main components of this model are: the sources of the events, broadcasters, line handlers, state machines, and triggers. Each component performs a certain task, promoting the message forward to the following components.

Рукапіс паступіў у рэдакцыю 28.10.2014