

## Архитектуры грид-облаков

Jinesh Varia

Технолог Веб службы Amazon

Cloud Architectures

Jinesh Varia Technology

Evangelist Amazon Web Services ([jvaria@amazon.com](mailto:jvaria@amazon.com))

<http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf>

### **Введение**

В этой статье содержатся примеры построения приложений в стиле, использующем службы, к которым можно получить доступ в облаке Интернет.

*Архитектуры облаков* – это методы разработки приложений, в которых используются службы, доступные в Интернете по требованию. Построенные этим методом приложения таковы, что к поддерживающей вычислительной инфраструктуре обращаются только тогда, когда это безусловно необходимо: (например, для обработки пользовательского запроса), чтобы выбрать необходимый ресурс по требованию (определить сервера и хранилища данных), выполнить конкретную работу и затем освободить ставшие ненужными ресурсы, а часто и выйти из приложения при завершении работы. При выполнении приложение масштабируется легко и естественно, опираясь на свойства тех ресурсов, в которых возникла необходимость.

Эта статья состоит из двух разделов. В первом разделе мы описываем приложение, которое использует инфраструктуру, предоставляемую веб-службами Amazon (крупнейшая в мире по обороту компания, продающая товары и услуги через Интернет.) Это приложение находится в данное время в эксплуатации. Оно позволяет разработчику производить сравнение образцов на миллионах веб-документов. Приложение выполняет обращения к сотням виртуальных серверов по требованию, выполняет на них параллельные вычисления, используя распределенную инфраструктуру обработки с открытым кодом, называемую *Nadoop*, затем закрывает все виртуальные сервера, возвращая все их ресурсы обратно в облако – все это не требует больших программистских усилий и предоставляется вызывающей стороне за вполне приемлемую плату.

Во втором разделе мы обсуждаем некоторые из "лучших" практик, использующих веб-службы Amazon – Amazon S3, Amazon SQS, Amazon SimpleDB и Amazon EC2; в этих инфраструктурах строятся масштабируемые приложения промышленного уровня..

Ключевые слова:

Веб-службы Amazon, Amazon S3, Amazon EC2, Amazon SimpleDB, Amazon SQS, Hadoop, MapReduce, cloud-компьютинг

### **Что нам дают архитектуры облака?**

Архитектуры, называемые архитектурами облака, нацелены на решение трудных ключевых проблем, возникающих при крупномасштабной обработке данных сложной структуры. В традиционной обработке часто приходится сталкиваться с проблемой нехватки вычислительной мощности, достаточной для удовлетворения нужд приложения. Во-вторых, трудно получить доступ к машинам тогда, когда это конкретно необходимо. Трудно распределить большую работу на отдельные машины и скоординировать на них выполнение процессов для распределенной работы, а также не просто обеспечить переход на новую машину при выходе из строя работающей. Трудными задачами являются реализация автомасштабирования при динамической загрузке работ и освобождение машин при завершении выполнения всей задачи. Архитектуры облака с этими трудностями справляются.

Приложения, построенные на принципах архитектуры облака, выполняются "на облаке", т.е. на инфраструктуре, физическое расположение которой задается провайдером. Эти приложения в полной мере используют преимущества простого API (*Application Programming Interface*), доступные в Интернете служб, которые легко масштабируются по требованию. Используемые службы жестко соблюдают промышленные стандарты, требующие реализации сложной логики надежности и масштабирования служб более низкого уровня; реализации, полностью локализованной "внутри облака". Использование ресурсов в архитектурах облака в полной мере следует принципу "использования только того, что необходимо", аккуратно обрабатывая случайные или периодически возникающие запросы. Отсюда следует высокий "коэффициент полезного действия" и оптимальная реакция на ошибки.

### **Польза от архитектур облака для бизнеса**

Разработка приложений в архитектурах облака имеет несколько очевидных преимуществ. Некоторые из них перечислены ниже:

1. *Почти никаких предварительных инфраструктурных вложений*: когда вам приходится создавать широкомасштабную систему, то вы должны быть готовы потратить целое состояние на землю, машинное оборудование (шкафы, компьютеры, маршрутизаторы, аппараты аварийного питания), управление машинным оборудованием (электропитание, охлаждение) и на обслуживающий персонал. Легитимизация этих расходов потребует, как правило, несколько раундов менеджментских переговоров прежде, чем проект начнет реализовываться. Ну, а для компьютеринга в стиле утилит никаких фиксированных или стартовых расходов не требуется.

2. *Как раз такая инфраструктура, которая сейчас требуется:* раньше, когда ваши системы добивались общественного признания, ни они, ни поддерживающая их инфраструктура не умели масштабироваться, и вы становились жертвой своего собственного успеха. И наоборот, когда про вас никто не знал, несмотря на огромные инвестиции, вы становились жертвой своей неудачи. При развертывании приложений "в облаке", где поддерживается динамическое управление, архитекторам программного обеспечения не приходится беспокоиться о поддержке систем большого масштаба. Ваши решения менее рискованны, так как вы масштабируетесь по мере вашего роста. В архитектурах облака также хорошо решается задача быстрого освобождения инфраструктуры (за минуты).
3. *Более эффективное использование ресурса:* обычной заботой системных администраторов является обеспечение аппаратуры (при исчерпании имеющейся мощности) и экономичная утилизация инфраструктуры (когда мощности простаивают). Архитектуры облака позволяют администраторам распоряжаться ресурсами более эффективно и эффектно, так как приложения имеют возможность запрашивать и освобождать ресурсы именно тогда, когда это нужно (по требованию).
4. *Платим только за то, чем пользуемся:* переход на работу с утилитами и соответствующая ему политика цен позволяют клиенту платить только за те инфраструктурные возможности, которые были им реально использованы. Клиент не абонирует всю инфраструктуру целиком. Здесь проявляется тонкое различие между приложениями для настольных компьютеров и веб-приложениями. Приложение для настольных компьютеров, или традиционное клиент-серверное приложение, выполняется на собственной инфраструктуре пользователя (ПК или сервера), тогда как при работе с приложением в архитектурах облака пользователь работает на инфраструктуре третьей стороны и платит только за частичку того, что его обслуживает.
5. *Потенциальные возможности сокращения времени обработки:* параллелизация – это один из мощнейших способов ускорить выполнение задания. Пусть некоторая очень трудоемкая задача требует для своей обработки 500 часов рабочего времени одного компьютера. Если эта задача может быть распараллелена, то в архитектурах облака можно было бы запустить 500 экземпляров этой задачи и выполнить её за 1 час. Эластичная инфраструктура дает возможность использовать распараллеливание и получить большие экономические выгоды.

### **Примеры архитектур облака**

Имеется много примеров приложений, которые могли бы взять многое у архитектур облака. Сюда входят как большие офисные системы, так и веб-приложения. Вот некоторые из них:

- Обработка связанных в цепочку процессов
  - Обработка связанных в цепочку документов – конвертирование сотен тысяч документов из формата Microsoft Word в формат PDF, преобразование средствами OCR<sup>1</sup> миллионов страниц/образов в стандартно обрабатываемый текст;
  - Обработка связанных в цепочку изображений – создание ярлыков или вариантов изображения низкого разрешения, изменение размеров миллионов изображений ;
  - Цепочки видео-преобразований – преобразование AVI в фильмы MPEG;
  - Индексирование – создание индекса найденных на вебе данных;
  - Глубокая обработка данных – выполнение поиска на множестве миллионов записей.
  
- Системы пакетной обработки
  - Приложения административной и управленческой служб (в финансовом, страховом или торговом секторе);
  - Анализ протоколов – анализ и изготовление ежедневных/еженедельных отчётов;
  - Автоматическая компиляция – параллельная автоматическая обработка репозитория кодов-источников;
  - Тестирование автоматизированных устройств – развертывание и тестирование автоматизированных устройств (проверка функциональности, загрузки, качества) каждую ночь на различных конфигурациях развёртывания.
  
- Веб-сайты
  - Веб-сайты, которые "спят" ночью и автоматически масштабируются днем;
  - Актуальные веб-сайты – веб-сайты для конференций и спортивных событий;
  - Развитие веб-сайтов ;
  - "Периодические веб-сайты" – веб-сайты, которые работают только в строго определенном временном режиме ("Чёрные пятницы" или Рождество);

В нашей статье мы подробно обсудим пример приложения под кодовым названием "GrepTheWeb".

### **Пример архитектуры облака: GrepTheWeb ("найди на вебе по шаблону")**

Веб-служба AlexaWebSearch (Веб-поиск-Alexa) позволяет разработчику создавать настраиваемые поисковые машины на больших массивах данных, прорабатываемых службой Alexa каждую ночь. Эта служба, в частности, позволяет пользователям запрашивать у Alexa поисковый индекс и получать по нему на выходе миллионы результатов. Разработчики могут задавать запросы, возвращающие до 10 миллионов результатов.

---

<sup>1</sup> **Оптическое распознавание символов** (*Optical Character Recognition, OCR*) — механическая или электронная конвертация изображений символов и букв в текст, редактируемый на компьютере. Перевод осуществляется программным путём, после получения изображения со сканера или фото (Википедия).

Результирующее множество, которое представляет очень небольшое подмножество всех документов на вебе, может затем обрабатываться далее, используя язык регулярных выражений. Это позволяет разработчикам фильтровать результаты своих поисков, используя критерий, который не индексируется службами Alexa (Alexa индексирует документы на основе пятидесяти различных документных атрибутов), давая, таким образом, разработчику возможность выполнять более интеллектуальные поиски. Разработчики могут проверять регулярные выражения на фактических документах, если даже их количество исчисляется миллионами, выбирая подмножества документов, которые соответствуют данному регулярному выражению.

Мы рассматриваем приложение, которое в настоящее время является продуктом в Amazon.com, оно называется GrepTheWeb, потому что оно может выполнять команду "grep(Global Regular Expression Print)" – общеизвестная утилита командной строки системы Unix для поиска по шаблону на фактических веб-документах. GrepTheWeb предоставляет разработчику возможность выполнять достаточно сложный специализированный поиск, такой, как выбор документов, имеющих особенный HTML-тэг, или META-тэг или выбор документов со специфицированной пунктуацией ("Hey!", he said, «Why Wait?» - "Эй!", сказал он, "Чего ждать?"), или поиск математических уравнений (" $f(x) = ?x + W$ "), поиск текста на языке программирования, адреса электронной почты или каких-либо других соответствий шаблону типа "(dis)integration of life"

Познакомимся ближе с работой службы с такой замечательной функциональностью. В следующем разделе архитектура программы GrepTheWeb будет рассмотрена на различных уровнях.

Результаты работы службы Million Search Results Service (миллионы результатов поиска) содержатся в одном файле в виде отсортированного списка ссылок; результаты заархивированы с помощью Unix-утилиты gzip. Этот файл передается на первый вход службы GrepTheWeb. На второй вход передается некоторое регулярное выражение. Результатом обработки будет отфильтрованное подмножество ссылок на документы, хранящееся в одном файле в отсортированном и заархивированном виде. Так как весь процесс выполняется асинхронно, разработчики могут получить статус завершения своих работ, обратившись к функции GetStatus().

Обработка регулярного выражения на миллионах документов дело далеко не тривиальное. На время выполнения процесса, которое может оказаться очень большим, могут влиять различные факторы:

- Регулярные выражения могут быть сложными;
- Множество данных может быть очень большим, возможно сотни терабайт;
- Неизвестные шаблоны запросов, т.е., любое количество посетителей, могут обратиться к приложению в любой заданный момент времени.

Поэтому при разработке GrepTheWeb одной из поставленных целей было масштабирование по всем направлениям (более мощные языки сопоставления шаблонов, большее число конкурирующих пользователей обычных наборов данных, большие наборы данных, повышение качества результатов), удерживая при этом рост стоимости обработки.

Нам было важно построить приложение, которое не только масштабируется по требованию, но также и не требует больших предварительных инвестиций, и не требует платы за поддержку простаивающих машин. Чтобы получить ответ за приемлемое время, было важно уметь распределить работу на много более мелких задач, а операции распределенного Grep'a выполнять так, чтобы эти задачи просчитывались параллельно на множестве узлов.

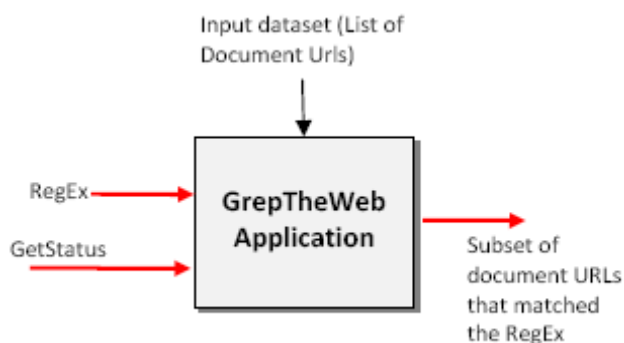


Рис. 1. Архитектура GrepTheWeb – уровень 1.

Расписывая архитектуру подробнее, изобразим её на рисунке 2.

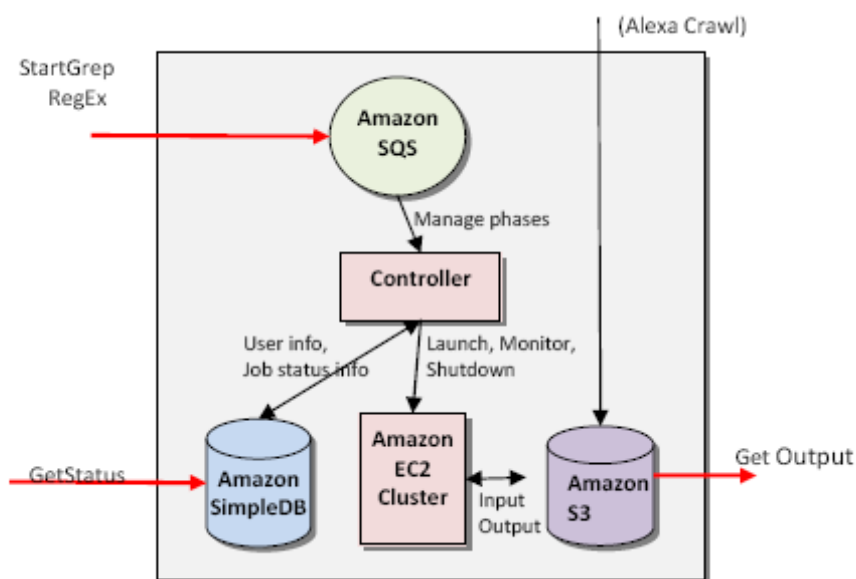


Рис. 2. Архитектура GrepTheWeb – уровень 2

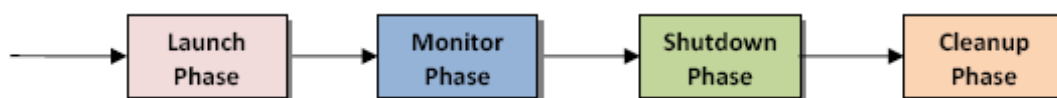
В этой архитектуре используются следующие компоненты AWS (Amazon Web Services, веб-службы Amazon):

- **Amazon S3** для выбора входных наборов данных и для запоминания выходных наборов данных;
- **Amazon SQS** для долговременной буферизации запросов, исполняя роль "мягкой связки" между контроллерами;

- **Amazon SimpleDB** (простая база данных Amazon) для хранения промежуточных результатов, протоколов и для данных пользователя, относящихся к задачам;
- **Amazon EC2** для исполнения больших кластеров Hadoop по требованию;
- **Hadoop** для распределенной обработки, автоматического распараллеливания и планирования работ.

### Рабочий поток

Система GrepTheWeb построена по модульному принципу. Обработка данных происходит за четыре этапа, как показано на рисунке 3. Этап запуска отвечает за проверку корректности данных и инициализацию обслуживания запроса к GrepTheWeb, за инициализацию экземпляров Amazon EC2, за запуск кластера Hadoop на этих экземплярах и начало выполнения всех рабочих процессов. На этапе управления происходит мониторинг кластера EC2, карт, модификаций и проверки корректности завершения. На этапе останова происходит расчет биллинга и завершение всех процессов Hadoop и экземпляров Amazon EC2. На этапе очистки удаляются все промежуточные данные из базы Amazon SimpleDB.



**Рис. 3** Последовательность этапов в архитектуре GrepTheWeb

Подробное описание работы схемы, изображённой на рисунке 4:

1. При старте приложения создаются очереди (если они еще не созданы) и выполняется запуск всех контроллеров нитей. Каждая нить контроллера начинает опрашивать соответственно свои очереди на предмет наличия сообщений.
2. При получении от пользователя запроса StartGrep сообщение о запуске перемещается в очередь запуска.
3. *Этап запуска:* Нить контроллера запуска выбирает сообщение запуска и выполняет указанную задачу, обновляет статус и регистрируется по времени в домене Amazon SimpleDB, заводит новое сообщение в очереди монитора и удаляет сообщение из очереди запуска при завершении обработки.
  - a. Задача запуска начинает обработку экземпляров Amazon EC2, используя прединсталлированное АМІ в JRE<sup>1</sup>, развёртывает требуемые библиотеки Hadoop и стартует Hadoop Job (выполняет задачи Map/Reduce).
  - b. Hadoop выполняет задачи карт на подчиненных узлах Amazon EC2 в параллельном режиме. Каждая задача карты выбирает файлы, разбитые на нити в фоновом режиме, из системы Amazon S3, обрабатывает регулярное выражение (атрибут очереди сообщений) для файла из Amazon S3 и записывает совпавшие результаты вместе с описанием совпавших результатов (не более 5) локально и затем задача объединяет/свёртывает результаты и фиксирует их как выходные данные.
  - c. Окончательные результаты запоминаются на Amazon S3 в корзине выходных результатов.

<sup>1</sup> **JRE(Java Runtime Environment)** — минимальная реализация виртуальной машины, необходимая для исполнения Java-приложений, (Википедия).

4. *Этап монитора:* Нить контроллера монитора выбирает это сообщение, проверяет по базе Amazon SimpleDB статус или наличие признака ошибки и выполняет задачу монитора, обновляет статус в домене Amazon SimpleDB, заносит новое сообщение в очередь завершения работы и очередь биллинга и удаляет обработанное сообщение из очереди монитора по завершении обработки.
  - a. Задача монитора проверяет статус Hadoop (признак успешно/ошибка в протоколе работы) через фиксированные промежутки времени, обновляет элементы базы SimpleDB со статусом ошибки и выходной файл Amazon S3.
5. *Этап завершения:* Нить контроллера завершения выбирает это сообщение из очереди завершения и выполняет задачу завершения, обновляет статус и фиксируется по времени в домене Amazon SimpleDB, удаляет обработанное сообщение из очереди после обработки.
  - a. Задача завершения убивает процессы Hadoop, завершает экземпляры EC2 после получения информации о топологии EC2 от Amazon SimpleDB и выходит из инфраструктуры.
  - b. Задача биллинга получает информацию о топологии EC2, файл текущего использования базы Simple DB, файл Amazon S3 и данные о входных запросах; затем она модифицирует биллинг и передает его службе биллинга.
6. *Этап чистки:* записывает в архив данные базы SimpleDB с пользовательской информацией.
7. Пользователи могут выполнить операцию GetStatus при выходе из службы для получения статуса всей системы (всех контроллеров и Hadoop) и загрузить отфильтрованные результаты из Amazon S3 после завершения.

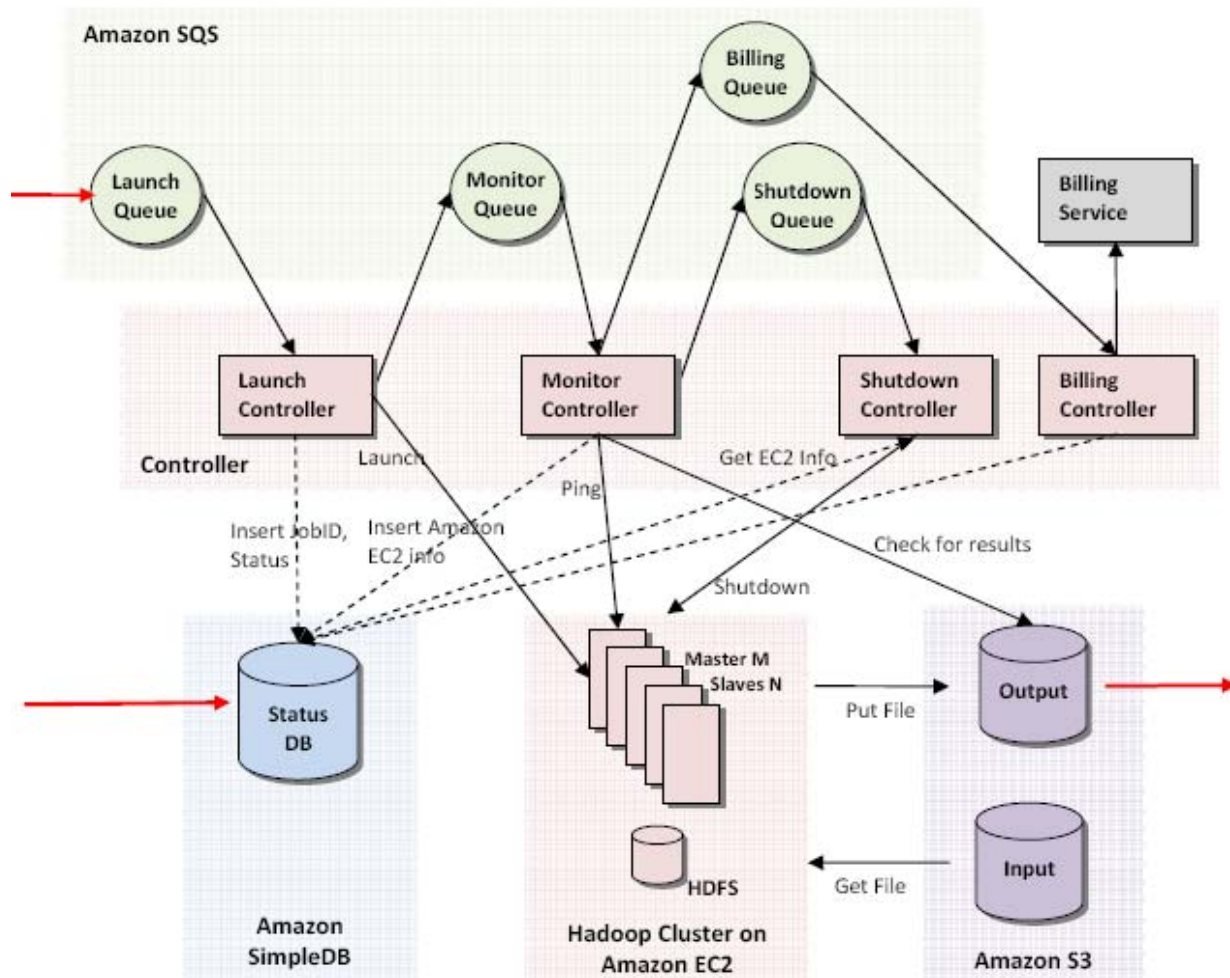


Рис. 4 Архитектура GrepTheWeb – уровень 3

### Использование веб-служб Amazon

В следующих четырех подразделах мы опишем практику использования системы GrepTheWeb и то, как она работает со службами AWS.

#### **Как использовался Amazon S3<sup>1</sup>**

В программе GrepTheWeb элемент Amazon S3 исполняет роль хранилища как входных, так и выходных данных. Входные данные для GrepTheWeb выбираются из самого веба (в заархивированном виде из Alexa's Web Crawl), они хранятся в Amazon S3 в виде объектов и часто подвергаются обновлению. Так как "свитковые" множества данных на вебе могут быть очень большими (обычно это терабайты) и постоянно увеличивающимися в размере, необходимо наличие распределенного, "безразмерного" долговременного хранилища данных. Amazon S3 доказал свое отличное соответствие этой задаче.

<sup>1</sup> Amazon S3 (Amazon Simple Storage Service ()) — сервис-хранилище.

## **Как использовался Amazon SQS**<sup>1</sup>

Amazon SQS – это механизм пересылки сообщений между компонентами. Он является тем связующим элементом, который объединяет различные функциональные компоненты в одно целое. Это не только способствовало созданию множества различных компонент без жестких связей, но и построению всей системы, более устойчивой к ошибкам и отказам.

### ***Буфер***

Если одна из компонент получает и обрабатывает запросы быстрее, чем другие компоненты (ситуация несбалансированных отношений поставщик-потребитель), буферизация поможет всей системе легче справиться с "пробками" трафика (или загрузки). Amazon SQS выполняет роль транзитного буфера между двумя компонентами (контроллерами) в системе GrepTheWeb. Если сообщение отправляется непосредственно какой-то компоненте, то получатель обязательно должен принять ее со скоростью, диктуемой отправителем. Например, если система биллинга работает медленно, или если время запуска кластера Hadoop превышает ожидаемое, работа всей системы замедлится, так как ей придется просто ждать. При наличии очередей сообщений отправитель и получатель развязаны, и служба обработки очереди сглаживает любые иррегулярности, связанные с "пробками" трафика сообщений.

### ***Изоляция***

Взаимодействие любых двух контроллеров в системе GrepTheWeb осуществляется через очередь сообщений и никакой контроллер не обращается непосредственно к любому другому контроллеру. Вся коммуникация и взаимодействие требуют записи сообщений в соответствующую очередь и выбора сообщений из очереди. Это делает всю общую систему развязанной по компонентам, а интерфейс становится простым и ясным. Подсистема Amazon SQS предоставила возможность унифицированного способа передачи информации между различными компонентами приложения. Функция каждого контроллера сводится к выбору сообщения, обработке сообщения (выполнения функции) и записи сообщения в другую очередь, так как контроллеры полностью изолированы друг от друга.

### ***Асинхронность***

Так как узнать сколько времени потребуются для выполнения каждого этапа достаточно трудно, (так, например, на этапе запуска решается динамически, сколько экземпляров подлежит запуску для данного запроса и поэтому время выполнения этапа неизвестно) Amazon SQS предоставляет возможность построения асинхронных систем. Теперь, когда на этапе запуска для обработки требуется больше предполагаемого времени или происходит отказ на этапе монитора, на остальные компоненты системы это никак не влияет и вся система оказывается более стабильной и в большинстве случаев доступной.

---

<sup>1</sup> Amazon SQS (Amazon Simple Queue Service) — сервис принимает очереди сообщений для хранения

## Как использовалась база Amazon SimpleDB<sup>1</sup>

Одной из причин использовать базу данных в архитектуре облака является отслеживание статусов. Так как компоненты системы работают в асинхронном режиме, нужно уметь получать статус системы в любой момент времени. Более того, так как все компоненты автономны и дискретны, необходимо иметь место хранения данных, в котором можно производить поиск, и где будут храниться статусы системы.

Поскольку база Amazon SimpleDB не предполагает схемы данных, структуру записи необязательно определять заранее. Каждый контроллер может ввести свою собственную структуру и подключать данные к элементу "работа". Например: если необходимо "прокрутить адрес почты regex<sup>2</sup> для не менее 10 миллионов документов", контроллер запуска добавит/обновит атрибут "статус запуска" вместе с атрибутом "время начала запуска", в то время, как контроллер монитора добавит/обновит атрибуты "статус монитора" и "статус hadoop" значениями состояния (выполняется, завершено, ошибка, ничего). Функция GetStatus() отошлет запрос к базе Amazon SimpleDB и возвратит состояние каждого контроллера, а также и интегрированный статус всей системы.

Службы компонент могут обратиться с запросом к базе Amazon SimpleDB в любое время, так как контроллеры хранят свои статусы независимо друг от друга – ещё один удобный способ создавать асинхронные службы с хорошим режимом готовности. Хотя в GrepTheWeb при реализации работы с Amazon SimpleDB был принят упрощенный подход, имеется возможность выхода и на более продвинутый метод с использованием полномасштабного мониторинга, почти что в реальном времени. Например, запоминание статуса Hadoop JobTracker, показывающего, сколько было обработано карт на заданный момент.

База Amazon SimpleDB используется также для хранения активных идентификаторов запросов Request ID для хронологических целей и аудита/биллинга.

Подводя итог, база Amazon SimpleDB используется как статусная база, в которой хранятся различные статусы компонент и как протокольная база для отслеживания данных о производительности системы.

## Как использовалась компонента Amazon EC2

В системе GrepTheWeb все программы контроллеров работают на экземплярах Amazon EC2. Контроллер запуска запускает процессы главного и подчиненных экземпляров, используя предконфигурированный образ машины Amazon Machine Image (AMI). Так как динамическая поставка и отправка данных происходит с помощью простых вызовов веб-служб, GrepTheWeb знает, сколько главных и подчиненных экземпляров подлежит запуску.

---

<sup>1</sup> Amazon SimpleDB - сервис, предоставляющий возможности индексирования данных и выполнения запросов

<sup>2</sup> Regex-Регулярные выражения (*regular expressions*, сокр. **RegExp**, **RegEx**, жарг. *регэкспы* или *рэксы*) — система синтаксического разбора текстовых фрагментов по формализованному шаблону (Википедия)

Контроллер запуска Amazon Web Services The launch делает предположение, базирующееся на опыте и логике "бронирования", о количестве необходимых подчиненных процессов для выполнения каждой конкретной работы. Логика бронирования базируется на сложности запроса (количество предикатов и т.д.) и размере входного множества данных (количество просматриваемых документов). Здесь также возможно конфигурирование, так что мы можем сократить время обработки, просто задав количество запускаемых экземпляров.

После запуска экземпляров и старта кластера Hadoop для этих экземпляров, кластер укажет нужные экземпляры, произведёт согласование и распределение файлов (ключи SSH, сертификаты) перейдет к выполнению работы gper.

## Программный продукт Hadoop

Программный продукт Hadoop<sup>1</sup> является распределенной системой обработки данных открытого кода, позволяющий обрабатывать очень большие наборы данных путем их разбиения на отдельные фрагменты приемлемого размера, распределяемого на совокупности доступных машин. Система управляет всем процессом запуска работ, выполняя их независимо от того, где физически располагаются обрабатываемые данные; в конце общего процесса система объединяет отдельные результаты в общий конечный результат.

Обычно работа происходит в три этапа. На этапе отображения входные данные преобразуются в промежуточное представление в виде пар ключевых значений; на этапе обработки (за которую отвечает сама система Hadoop) данные обрабатываются и сортируются в соответствии с их ключами; и на этапе редуцирования происходит рекомбинация промежуточного представления в конечный результат. Разработчики приложения реализуют два интерфейса, Отображатель шрифта и Преобразователь данных, а за всей распределенной обработкой следит сама Hadoop (автоматическое распараллеливание, планирование работ, мониторинг работ и агрегация результата).

В самой системе Hadoop имеется главный процесс, работающий на одном из узлов, который следит за пулом подчиненных процессов (называемых также исполнителями); последние работают на отдельных узлах. Hadoop разбивает входные данные на фрагменты. Эти фрагменты приписываются подчиненным процессам, каждый из которых выполняет задачу отображения (логику отображения задает пользователь) на каждой паре ключей, найденных в текущем фрагменте, записывает результат локально и информирует главный процесс о статусе завершения. Hadoop объединяет все результаты и сортирует их в соответствии со значениями ключей. Затем он передает ключи Преобразователю данных, который выбирает результаты с помощью программы итератора (организатора

---

<sup>1</sup> **Apache Hadoop** является свободным Java Фреймворком, поддерживающим выполнение распределённых приложений, работающих на больших кластерах, построенных на обычном оборудовании. Hadoop прозрачно предоставляет приложениям надёжность и быстродействие операций с данными. В Hadoop реализована вычислительная парадигма, известная как MapReduce Согласно этой парадигме приложение разделяется на большое количество небольших заданий, каждое из которых может быть выполнено на любом из узлов кластера. (Википедия)

циклов) запускает соответствующую задачу (логику специфицирует пользователь) и отправляет "конечные" выходные данные обратно в распределенную файловую систему.

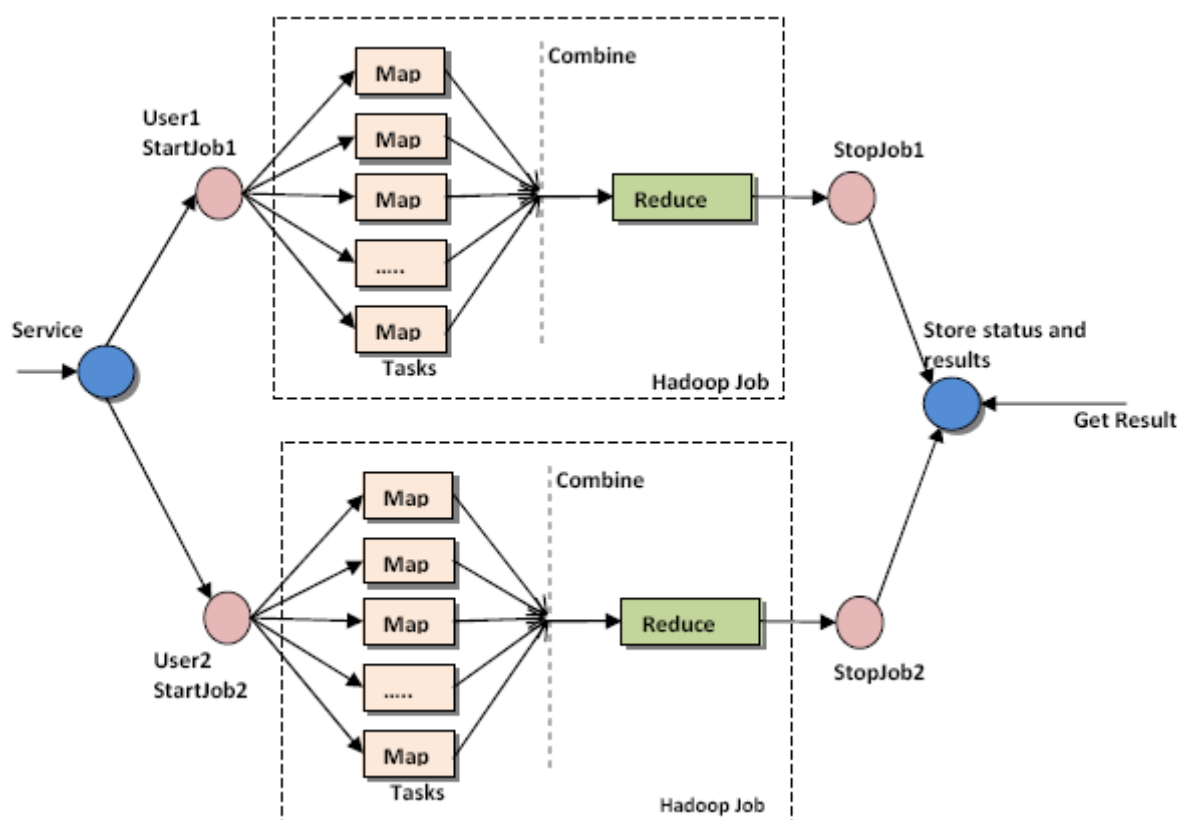


Рис. 5. Операция экономного отображения (в GrepTheWeb)

### Реализация Hadoop в системе GrepTheWeb

Hadoop хорошо согласуется с приложением GrepTheWeb, так как каждая grep-задача может выполняться параллельно, независимо от остальных grep-задач, используя встроенный в Hadoop механизм параллелизма. Для приложения GrepTheWeb фактические документы (веб) выбираются заблаговременно и передаются на хранение базе Amazon S3. Каждый пользователь запускает grep-работу, обращаясь к функции StartGrep из конечной точки службы. При получении сигнала запуска главный и вспомогательные узлы (Hadoop кластер) начинают обрабатывать экземпляры из Amazon EC2. Hadoop разбивает входные данные (документы со ссылками на объекты Amazon S3) на некоторое количество поддающихся обработке фрагментов, каждый по 100 строчек, и каждый фрагмент приписывает некоторому вспомогательному узлу для выполнения на нём операции отображения. Задача отображения читает эти строки (взять этот файл из Amazon S3 входит в её обязанности), вычисляет на них регулярное выражение и сохраняет результаты в локальном хранилище. Если совпадения нет – нет и результата. Затем задача отображения передает результаты этапу преобразования, на котором формируется конечный результат простым объединением частичных результатов. Выходные данные в качестве "конечного" результата записываются обратно в базу Amazon S3.

## Реализация программы отображения

1. Key = номер строки, и value = строка во входном наборе данных
2. Создать URL (используя реквизиты доступа Amazon AWS) по содержимому ключа значения строки
3. Перенести на буфер объект Amazon S3 (файл)
4. Проверить регулярное выражение на содержимом буфера
5. Если проверка оказалась успешной, занести результат в новую пару ключевых значений (ключ = строка, значение = до 5 совпадений)

Реализация программы преобразования – выполнение встроенной функции идентичности и запись результата обратно на S3. («Black Friday» or Christmas)

## Советы по разработке приложения в архитектуре облака

1. Обеспечьте масштабируемость вашего приложения масштабируемостью каждой **компоненты** в отдельности. Если каждая компонента реализует интерфейс службы, отвечающий **за свою собственную масштабируемость** в подходящих измерениях, то у общей системы будет масштабируемая база.
2. Для того, чтобы управление было более удобно и чтобы были высокими характеристики доступа к компонентам, убедитесь, что компоненты являются **слабо связанными**. Главным здесь является такая конструкция компонент, чтобы между ними не было жесткой взаимозависимости. То есть, если одна из компонент по той или иной причине выходит из строя, спит (не отвечает) или продолжает работу (медленно отвечает), остальные компоненты системы могут продолжать работать в обычном режиме.
3. Реализуйте **параллелизации** для лучшего использования всей инфраструктуры и для повышения производительности. Одними из методов повышения эксплуатационных характеристик инфраструктуры являются дистрибуция задач на совокупности машин, многоканальная обработка запросов и эффективное объединение результатов параллельного исполнения.
4. После разработки базовой функциональности задайте себе вопрос "Что, если это не будет работать?" Используйте методы и подходы, обеспечивающие корректность работы в непредвиденных обстоятельствах. При отказе одной и компонент (а ошибки случаются все время) система должна выдавать предупреждения, уметь обходить ошибки, и возвращаться назад к "последнему известному состоянию".
5. Не забывайте про **фактор стоимости**. Для получения эффективного по стоимости приложения важным является использование ресурсов по требованию. Недопустимая расточительность – платить за простаивающую инфраструктуру.

Каждый из этих аспектов обсуждается далее в контексте GrepTheWeb.

## Использование масштабируемых ингредиентов

Приложение GrepTheWeb использует компоненты с высокими характеристиками масштабируемости из инфраструктуры веб-служб Amazon, которые не только масштабируются по требованию, но и требуют за это оплаты.

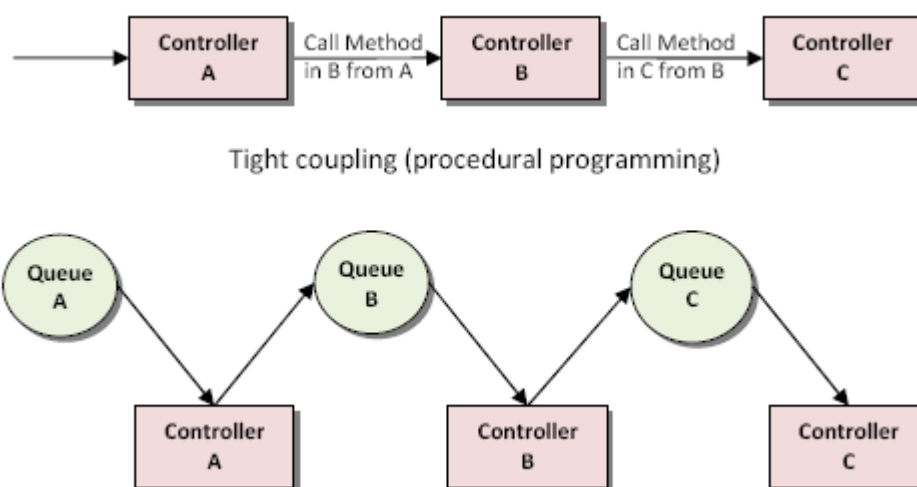
Все компоненты GrepTheWeb предлагают интерфейс, который определяет функции, вызываемые по запросам HTTP с ответами в XML. Для удобства программирования в небольших клиентских библиотеках конкретный код службы хранится в свернутой и абстрагированной форме.

Каждая компонента независима от остальных компонент и может масштабироваться по всем измерениям. Например, если на базу Amazon SimpleDB обрушивается многотысячный поток запросов, то требования могут быть обработаны, так как предусмотрена обработка большого количества параллельных запросов.

Таким же образом, в такой схеме распределенной обработки как Nadoor, предусмотрено масштабирование. Nadoor автоматически распределяет работы, заново запускает работы с отказами, и для обработки терабайтовых массивов данных обращается к услугам множественных узлов.

## Нужны свободно связанные системы

Команда разработчиков GrepTheWeb построила свободно связанную систему, используя очереди сообщений. Если для "связывания" любых двух компонент используется очередь или буфер, то может поддерживаться конкурентность, высокая степень доступности и можно не бояться пиковых нагрузок. В результате вся система продолжает функционировать, если даже часть компонент становится недоступной. Если одна из компонент выйдет из строя или станет временно недоступной, система запомнит запрос на буфере и приступит к обработке этого запроса, когда критическая компонента вновь будет готова к работе.



**Рис 6: Свободное связывание – независимые этапы веб-служб Amazon**

Если в GrepTheWeb, например, на сервер приходит внезапно много запросов (вызванная интернетом форс-мажорная ситуация) или обработка регулярных выражений занимает больше медианного времени (низкий коэффициент ответа компоненты), программа Amazon SQS организует на буфере очередь запросов, так что эти задержки не оказывают влияния на остальные компоненты.

Так как в многопользовательской системе важно иметь доступ к статусам сообщений и запросов, в системе GrepTheWeb это возможно. Для этого в отдельной области памяти, по которой можно производить поиск, поддерживается обновленный статус каждого запроса; такой областью памяти является Amazon SimpleDB. Комбинация Amazon SQS (для работы с очередями) и Amazon SimpleDB (для работы с состояниями) позволяет добиться большой гибкости в решении вопроса свободного связывания.

### **Переходите на "параллельное мышление"**

В наше время "эра-тера" и многопроцессорных машин программирование требует рассуждений в понятиях мультипоточности.

В GrepTheWeb везде, где это возможно, потоки защищены; философским лозунгом при этом является "ни с кем не делись". К многопоточности прибегают всегда, когда это может увеличить производительность. Например, объекты выбираются из Amazon S3 во множественных конкурентных потоках, так как такой доступ более быстрый, чем выбор объектов последовательно по одному. Если многопоточность недостаточна, подумайте об использовании множественных узлов.

До последнего времени параллельный компьютеринг на большом кластере машин был не только дорог, но и трудно доступен. Во-первых, трудно получить финансирование большого кластера машин, а после его приобретения сталкиваешься с проблемами управления и поддержки. Во-вторых, в дальнейшем возникают технические задачи. Трудно исполнять глобально распределенные задачи на этих машинах, хранить и получать доступ к большим наборам данных. Параллелизация не простая задача, и планирование работ сопряжено с появлением ошибок. Более того, при отказе узлов их обнаружение трудно, а восстановление очень дорого. Прослеживанию узлов и статусов часто не уделяется должное внимание, так как по мере увеличения числа машин в кластере это становится сложной задачей.

Но теперь компьютеринг изменился. С появлением Amazon EC2 обеспечение многих вычислительных экземпляров стало простым. Эта задача может быть решена за несколько минут простыми обращениями к API, обратная задача решается также просто. С приходом такой распределенной инфраструктуры как Hadoop при развертывании параллельного приложения необязательно обращаться к помощи высококвалифицированных консультантов. Разработчики, не обладающие опытом работы с параллельным компьютерингом, могут реализовать нужные интерфейсы с помощью нескольких команд, и запараллелить работу, не утруждая себя вопросами планирования, мониторинга или объединения.

## **Работа по требованию**

В GrepTheWeb каждая входящая в рабочий блок компонента доступна по Интернету через веб-службы. Это означает, что приложение может запрашивать дополнительные ресурсы (серверы, память, базы данных очереди) или освобождать их, когда потребность в них прошла. Вся инфраструктура целиком иницируется в облаке запросом работы gger и в дальнейшем возвращается в облако при завершении работы.

Дополнительная прелесть GrepTheWeb – это почти нулевые инфраструктурные накладные расходы до и после выполнения приложения. Вся инфраструктура располагается в облаке и начинает функционировать при запросе на выполнение работы (gгер), затем при выполнении она возвращается в облако. Кроме того, во время функционирования она может быть по требованию масштабирована; т.е. эластичность масштабирования зависит от количества сообщений и объема входного набора данных, сложности регулярного выражения и т.д.

Для GrepTheWeb предусмотрена логика резервирования, определяющая количество запускаемых подчиненных Nadoor экземпляров на основе сложности регулярного выражения и входных данных. Например, если у регулярного выражения немного предикатов, или если во входных данных около 500 документов, будет обрабатываться только 2 случая. Впрочем, если во входных данных 10 миллионов, будет обрабатываться до 100 случаев.

## **Разрабатываются схемы, гибкие для перезагрузки и перезапуска**

Простое правило: относись к облачной архитектуре с пессимизмом; жди отказов. Другими словами, ориентируйся при разработке, реализации и развертывании на автоматическое восстановление в исключительных ситуациях.

Более конкретно, не рассчитывай на безотказную работу оборудования. Не рассчитывай на качество нового оборудования. Не рассчитывай на то, что с вашим приложением не случится катастрофа. Не рассчитывай на то, что вы всегда справитесь с темпом поступления запросов. Становясь пессимистом, вы перестаете заниматься вопросами стратегии восстановления на этапе разработки, что благотворно сказывается на структуре системы в целом. Например, следующие методы могут быть полезными в критических случаях:

1. Следите за сохранением корректных копий и за восстановлением данных;
2. Создавайте потоки процессов, восстанавливаемые при перезагрузке;
3. Предусмотрите возможность ресинхронизации состояния системы при перезагрузке сообщений из очередей;
4. Храните предконфигурированные и предоптимизированные параметры системы для поддержки пунктов 2 и 3 при загрузке.

Хорошая архитектура облака должна быть защищена от стороннего вмешательства при запуске и перезагрузках. В системе GrepTheWeb с этой задачей надежно справляется совместная работа подсистем Amazon SQS и Amazon SimpleDB. Так, например, если

элемент, на котором выполнялся управляющий поток, выходит из строя, он может быть восстановлен, и продолжать работу с предыдущего состояния так же, как и при нормальном выполнении. Это обеспечивается созданием предконфигурируемого образа Amazon Machine, которая при запуске выбирает все сообщения из очереди Amazon SQS и их состояния из элемента домена Amazon SimpleDB при перезагрузке.

При выходе из строя вспомогательного узла в связи с ошибками оборудования Nadoor автоматически перепланирует эту задачу на другой узел. Эта устойчивость к неполадкам позволяет Nadoor работать на больших стандартных серверных кластерах.

### **Результаты и стоимость**

Мы провели ряд экспериментов. Регулярное выражение для почтовых адресов исполнялось на 10 миллион документов. Для обработки на 48 конкурентных экземплярах потребовалось 48 минут, для обработки на 92 экземплярах - менее 6 минут. В это время входит время запуска экземпляра и время старта кластера Nadoor. Общая стоимость для 48 экземпляров равна примерно 5 долларам, а в случае 92 экземпляров - менее 10 долларов.

### **Заключение**

Вместо создания своих приложений на фиксированных и жестких инфраструктурах, пользуйтесь новым методом построения приложений на инфраструктурах по требованию.

Примером создания таких приложений является система GrepTheWeb.

Без предварительных инвестиций мы смогли выполнить работу в параллельном режиме на многих узлах. При этом мы прибегали к пошаговому масштабированию по требованию (пользователи, размер набора исходных данных). Инфраструктура приложения никогда не использовалась впусую из-за простоев.

В следующем разделе мы узнаем, как использовалась каждая из служб инфраструктуры Amazon (Amazon EC2, Amazon S3, Amazon SimpleDB и Amazon SQS), и мы поделимся с вами тем, чему мы научились, и что мы считаем лучшей практикой.

### **Лучшие практики нашего опыта**

В данном разделе мы расскажем о некоторых лучших практиках, к которым мы пришли при разработке GrepTheWeb.

#### **Лучшие практики в Amazon S3**

##### **Загрузить большие файлы**

Выполнить Upload Large Files (загрузить большие файлы), определить скорость передачи данных в режиме точка-точка утилитой Retrieve Small Offsets End-to-end – в Amazon S3 наиболее удобно тогда, когда хранятся большие файлы (единица измерения килобайты). В Amazon S3 индивидуальные файлы хранятся в объединенном и сжатом

(gzip) виде в единице хранения gz как объекты. Выбор конкретных файлов выполняется с помощью задания стандартного запроса HTTP GET при задании URL (алгоритма контроля и ключа), местоположения (область байтов) и размера (длина в байтах). В результате этого общая стоимость хранения уменьшается в связи с сокращением общего размера набора данных (ввиду компрессии) и, следовательно, меньшего числа запросов PUT.

### **Сортировать ключи и затем выполнить загрузку вашего набора данных**

Реконсайлеры Amazon S3 показывают, что производительность увеличивается, когда перед загрузкой производится предварительная сортировка ключей. Напишите небольшой скрипт для загрузки в Amazon S3 отсортированных ключей (указателей URL).

### **Используйте многопоточный выбор данных**

Вместо того, чтобы выбирать объекты из Amazon S3 по одному, в каждой задаче отображения при выборе объектов запускались конкурентные потоки. Впрочем, не рекомендуется использовать все возможные потоки, так как у каждого узла свои ограничения на полосу пропускания. Идеально пользователям следовало бы попытаться последовательно менять их число до точки, в которой добавление новых потоков не сказывается положительно на увеличение скорости.

### **Пользуйтесь экспоненциальным откатом**

Работая с приложением, всегда полезно при каждом отказе повторять запрос к веб-службе. Что не очевидно, так это какую выбрать стратегию определения интервалов времени между попытками. Мы рекомендуем схему ограниченного экспоненциального отката, где точное время ожидания между повторными попытками определяется комбинацией последовательных удвоений числа секунд, определяющих максимальное время ожидания и случайного выбора значения в этом интервале.

Мы рекомендуем вам встроить в программу обработки ошибок вашего клиента экспоненциальный откат, ожидание и логику повторных вызовов. Экспоненциальный откат сокращает число запросов в Amazon S3 и уменьшает, таким образом, общие расходы, потому что не приходится перезагружать отдельные части системы.

## **Лучшие практики для Amazon SQS**

### **Хранение ссылочной информации в Message**

Amazon SQS очень хорошо подходит для сообщений с небольшим сроком жизни в рабочих потоках и обрабатывающих процессах. Чтобы оставаться в границах размера записи, мы советуем хранить ссылочную информацию как часть самого сообщения, а файл конкретных входных данных хранить на Amazon S3.

В GrepTheWeb сообщение из очереди запуска содержит URL входного файла (.dat.gz), который является небольшим подмножеством результирующего множества, (миллион поисковых элементов, содержащих до 10 миллионов ссылок). Аналогично

этому, сообщение из завершающей очереди содержит URL выходного файла (.dat.gz), являющегося отфильтрованными результирующим множеством, содержащим ссылки, соответствующие регулярному выражению.

### **Пользуйтесь сообщениями, ориентированными на процессы и документы**

Имеется два подхода работы с сообщениями, которые нам показались эффективными: сообщения, ориентированные на процессы, и сообщения, ориентированные на документы. Первые часто определяются процессами и действиями, что сводится к удалению старого сообщения из очереди "из" и затем добавлению нового сообщения с новыми атрибутами к новой очереди "к".

Работа с ориентированными на документы сообщениями происходит тогда, когда на поток пользователь/работа одно сообщение проходит по всей системе с различными атрибутами. Реализация этого часто происходит на базе XML/JSON(*JavaScript Object Notation*), так как их модель расширяема. В этом решении сообщение может модифицироваться, получателю сообщения нужно только понимать те части сообщения, которые для него важны. Таким образом, одно-единственное сообщение может проходить по всей системе и различные компоненты Amazon Web Services работают только с теми частями сообщения, которые для них важны.

Для GrepTheWeb мы использовали подход, ориентированный на процессы.

### **Пользуйтесь преимуществами наглядности**

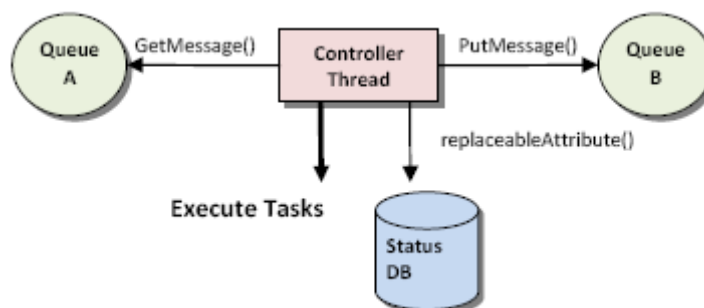
В Amazon SQS предусмотрена специальная функциональность, которой нет в большинстве других систем сообщений; при чтении сообщения из очереди она видна и остальным, кто работает с этой очередью, оно не удаляется автоматически из очереди. Потребитель должен явно удалить это сообщение из очереди. Если этого не произошло за определенное время после прочтения сообщения, считается, что у потребителя произошла ошибка и сообщение появится в очереди снова. Это реализуется с помощью фиксирования так называемого времени задержки видимости при создании очереди. В GrepTheWeb этот параметр времени задержки очень важен, так как некоторые процессы (такие, как контроллер выключения) могут выйти из строя и не отвечать (например, при остановке выполнения экземпляров). При установке параметр видимости на определенное время задержки, другой контроллер потока выберет старое сообщение и возобновит задачу (или окончательно ее выключит).

### **Лучшие практики для Amazon SimpleDB**

#### **Атрибуты нитей**

В Amazon SimpleDB в доменах имеются элементы (items), а у элементов есть атрибуты. Запрос к базе Amazon SimpleDB возвращает некоторое множество элементов. Но часто при выполнении отдельных задач требуются значения атрибутов. В этом случае для получения атрибутов каждого элемента списка за командой запроса следует поместить несколько обращений GetAttributes. Ясно, что время выполнения замедлится.

Чтобы как-то выправить ситуацию, мы очень рекомендуем оформить такие запросы, как мультипоточковые и выполнять их в параллельном режиме. В этом случае общая производительность существенно увеличивается (даже иногда раз в 50). В приложениях GrepTheWeb этот подход поможет создавать более динамичные отчеты месячной активности.



**Рис.7. Архитектура контроллера и Технологический процесс**

### **Используйте Amazon SimpleDB совместно с другими службами**

Создавайте такие инфраструктуры, библиотеки и утилиты, которые используют одновременно функциональности двух или более служб. Для GrepTheWeb мы создали небольшую инфраструктуру, которая совместно использует службы Amazon SQS и Amazon SimpleDB для глобального предоставления подходящего обобщенного состояния. Например, все контроллеры наследуются от класса BaseController. Основная задача этого класса – сортировка очереди сообщений "из", проверка корректности статусов для конкретного домена Amazon SimpleDB, выполнение функций, обновление статусов новой меткой времени и статусом и помещение нового сообщения в очередь "к". Преимущество такой установки состоит в том, что в случае отказа аппаратных средств или когда умирает экземпляр контроллера, новый узел может быть поднят почти немедленно, он восстановит операционное состояние, вновь выбрав сообщения из очереди Amazon SQS, а их статус из базы Amazon SimpleDB при перезагрузке, что делает всю систему более гибкой.

Общей практикой, хотя и не используемой в нашем случае, является хранение фактических файлов в виде объектов на Amazon S3 и хранение всех метаданных, относящихся к объекту, на Amazon SimpleDB. Кроме этого, использование в Amazon S3 ключей к объекту как имя элемента на Amazon SimpleDB также широко распространено

### **Лучшие практики в Amazon EC2**

#### **Запустите множественные экземпляры все и сразу**

Не ждите, когда ваши экземпляры EC2 будут загружаться один за другим; в простой команде запуска экземпляров укажите количество экземпляров каждого типа.

#### **Автоматизируйте всё, насколько это возможно**

. Мы специально говорим про это, потому что про автоматизацию в Amazon EC2 часто забывают. Автоматизация дает возможность разработчику запускать динамически

программируемый центр данных, который расширяется и сжимается по мере надобности. Например, автоматизация вашего тестового цикла установки в форме Amazon Machine Image (AMI) и затем запуск этого образа каждый вечер автоматически на Amazon EC2 (используя работу CRON<sup>1</sup>) сэкономит вам много времени. Автоматизируя процесс создания AMI, вы сэкономите время на конфигурации и оптимизации.

### **Запускайте новые вычислительные экземпляры сразу же**

Нужно просто запустить новые вычислительные экземпляры и стартовать на них процессы Hadoop, установить ссылки на головной узел и динамически наращивать кластер (или его свертывать) в реальном времени для повышения скорости прохождения общего процесса.

### **Сохраните ваши аттестационные данные AWS при подключении к AMI**

Если на вашем AMI исполняются процессы, которым нужно общаться с другими веб-службами AWS (для работы с очередью Amazon SQS или для выбора объектов из Amazon S3), не делайте распространенной ошибки и не храните ваши аттестационные данные AWS на AMI. Вместо этого, обязательно передайте их по сети как аргументы в зашифрованном виде. Общая процедура следующая:

1. Сгенерируйте новую ключевую пару RSA<sup>2</sup>, используйте инструменты OpenSSL, (криптографический пакет);
2. Скопируйте приватный ключ на образ (так, чтобы он был расположен на конечном AMI)
3. Отправьте (Post) открытый ключ вместе с остальной информацией образа для возможности доступа к нему остальных пользователей.
4. При запуске образа секретный ключ AWS необходимо зашифровать (или приватный ключ, если вы хотите пользоваться SOAP) с помощью открытого ключа, который был получен на шаге 3.
5. В дальнейшем ваш образ может это расшифровать на этапе загрузки и использовать для расшифровки данных, которые требуются для контакта с Amazon S3. Если пользователям не нужен такой доступ, то вам не нужно удалять личный ключ, просто убедитесь, что он не может бы прочитан другими пользователями, кроме корневого.

### **Благодарности**

Особую благодарность выражаем Kenji Matsuoka и Tinou Bao – главному коллективу разработчиков архитектуры GrepTheWeb.

### **Рекомендации**

Так как службы Amazon Web Services это службы очень простых строительных блоков, наиболее эффективно использовать их совместно с другими службами.

---

<sup>1</sup> CRON-Демон-планировщик задач в UNIX-подобных операционных системах.

<sup>2</sup> RSA- аббревиатура от трёх фамилий, криптографический алгоритм.

- **Используйте Amazon S3 и Amazon SimpleDB совместно, всегда, когда вы захотите искать объекты Amazon S3 по их метаданным**  
Мы рекомендуем хранить большие файлы на Amazon S3, а связанные с ними метаданные и ссылочную информацию на Amazon SimpleDB, так, чтобы разработчики имели доступ к этим метаданным. Данные, предназначенные только для чтения, можно сохранять также на Amazon S3 как метаданные объекта (т.н. автор, создать дату и т.д);
- **Используйте Amazon SimpleDB и Amazon SQS совместно, всегда, когда вы хотите, чтобы ваше приложение было на этапе Store transient messages**  
Храните транзитные сообщения в Amazon SQS и статусы работы/сообщения в Amazon SimpleDB, так, чтобы вы могли статусы обновлять часто и выбрать статус любого запроса в любое время простым запросом. Это особенно полезно в асинхронных системах;
- **Используйте Amazon S3 и Amazon SQS совместно, всегда, когда вы хотите создавать процессные цепочки или решения, касающиеся производителя и потребителя.**  
Храните исходные файлы в Amazon S3 и отправляйте соответствующие сообщения в очередь Amazon SQS вместе со ссылками и метаданными (S3 URI и проч.).