Нереляционные базы данных

Лабораторная работа 5

MapReduce

MapReduce — это подход к обработке данных, который имеет два серьёзных преимущества по сравнению с традиционными решениями. Первое и самое главное преимущество — это производительность. Теоретически MapReduce может быть распараллелен, что позволяет обрабатывать огромные массивы данных на множестве ядер/процессоров/машин. Как уже упоминалось, это пока не является преимуществом MongoDB. Вторым преимуществом MapReduce является возможность описывать обработку данных нормальным кодом. По сравнению с тем, что можно сделать с помощью SQL, возможности кода внутри MapReduce намного богаче и позволяют расширить рамки возможного даже без использования специализированных решений.

MapReduce — это стремительно приобретающий популярность шаблон, который уже можно использовать почти везде; реализации уже имеются в C#, Ruby, Java, Python. Должен предупредить, что на первый взгляд он может показаться очень непривычным и сложным. Не расстраивайтесь, не торопитесь и поэкспериментируйте с ним самостоятельно. Это стоит того — не важно, используете вы MongoDB или нет.

Теория и практика

MapReduce — процесс двухступенчатый. Сначала делается map *(отображение)*, затем — reduce *(свёртка)*. На этапе отображения входные документы трансформируются *(map)* и порождают *(emit)* пары ключ=>значение (как ключ, так и значение могут быть составными). При свёртке *(reduce)* на входе получается ключ и массив значений, порождённых для этого ключа, а на выходе получается финальный результат. Посмотрим на оба этапа и на их выходные данные.

В нашем примере мы будем генерировать отчёт по дневному количеству хитов для какого-либо ресурса (например, веб-страницы). Это *hello world* для MapReduce. Для наших задач мы воспользуемся коллекцией `hits` с двумя полями: `resource` и `date`. Желаемый результат — это отчёт в разрезе ресурса, года, месяца, дня и количества.

Пусть в `hits` лежат следующие данные:

```
resource     date
index        Jan 20 2010 4:30
index        Jan 20 2010 5:30
about        Jan 20 2010 6:00
index        Jan 20 2010 7:00
about        Jan 21 2010 8:00
about        Jan 21 2010 8:30
index        Jan 21 2010 8:30
about        Jan 21 2010 9:00
index        Jan 21 2010 9:30
index        Jan 22 2010 5:00
```

На выходе мы хотим следующий результат:

```
resource  year   month   day   count
index     2010   1       20    3
about     2010   1       20    1
about     2010   1       21    3
index     2010   1       21    2
index     2010   1       22    1
```

(Прелесть данного подхода заключается в хранении результатов; отчёты генерируются быстро и рост данных контролируется — для одного ресурса в день будет добавляться максимум один документ.)

Давайте теперь сосредоточимся на понимании концепции. В конце главы в качестве примера будут приведены данные и код.

Первым делом рассмотрим функцию отображения. Задача функции отображения — породить значения, которые в дальнейшем будут использоваться при свёртке. Порождать значения можно ноль или более раз. В нашем случае — как чаще всего бывает — это всегда будет делаться один раз. Представьте, что map в цикле перебирает каждый документ в коллекции `hits`. Для каждого документа мы должны породить *ключ*, состоящий из ресурса, года, месяца и дня, и примитивное *значение* — единицу:

```
function() {
    var key = {
        resource: this.resource,
        year: this.date.getFullYear(),
        month: this.date.getMonth(),
        day: this.date.getDate()
    };
    emit(key, {count: 1});
}
```

`this` ссылается на текущий рассматриваемый документ. Надеюсь, результирующие данные прояснят для вас картину происходящего. При использовании наших тестовых данных, в результате получим:

```
{resource: 'index', year: 2010, month: 0, day: 20} => [{count: 1}, {count: 1},
{count:1}]
{resource: 'about', year: 2010, month: 0, day: 20} => [{count: 1}]
{resource: 'about', year: 2010, month: 0, day: 21} => [{count: 1}, {count: 1},
{count:1}]
{resource: 'index', year: 2010, month: 0, day: 21} => [{count: 1}, {count: 1}]
{resource: 'index', year: 2010, month: 0, day: 22} => [{count: 1}]
```

Понимание этого промежуточного этапа даёт ключ к пониманию MapReduce. Порождённые данные собираются в массивы по одинаковому ключу. .NET и Java разработчики могут рассматривать это как тип `IDictionary<object, IList<object>>` (.NET) или`HashMap<Object, ArrayList>` (Java).

Давайте изменим нашу map-функцию несколько надуманным способом:

```
function() {
    var key = {resource: this.resource, year: this.date.getFullYear(), month:
this.date.getMonth(), day: this.date.getDate()};
```

```
    if (this.resource == 'index' && this.date.getHours() == 4) {
        emit(key, {count: 5});
    } else {
        emit(key, {count: 1});
    }
}
```

Первый промежуточный результат теперь изменится на:

```
{resource: 'index', year: 2010, month: 0, day: 20} => [{count: 5}, {count: 1},
{count:1}]
```

Обратите внимание, как каждый emit порождает новое значение, которое группируется по ключу.

Reduce-функция берёт каждое из этих промежуточных значений и выдаёт конечный результат. Вот так будет выглядеть наша функция:

```
function(key, values) {
    var sum = 0;
    values.forEach(function(value) {
        sum += value['count'];
    });
    return {count: sum};
};
```

На выходе получим:

```
{resource: 'index', year: 2010, month: 0, day: 20} => {count: 3}
{resource: 'about', year: 2010, month: 0, day: 20} => {count: 1}
{resource: 'about', year: 2010, month: 0, day: 21} => {count: 3}
{resource: 'index', year: 2010, month: 0, day: 21} => {count: 2}
{resource: 'index', year: 2010, month: 0, day: 22} => {count: 1}
```

Технически в MongoDB результат выглядит так:

```
_id: {resource: 'home', year: 2010, month: 0, day: 20}, value: {count: 3}
```

Это и есть наш конечный результат.

Если вы были внимательны, вы должны были спросить себя: *почему мы просто не написали* `sum = values.length`? Это было бы эффективным подходом, если бы мы суммировали массив единиц. На деле reduce не всегда вызывается с полным и совершенным набором промежуточных данных. Например вместо того, чтобы быть вызванным с:

```
{resource: 'home', year: 2010, month: 0, day: 20} => [{count: 1}, {count: 1},
{count:1}]
```

Reduce может быть вызван с:

```
{resource: 'home', year: 2010, month: 0, day: 20} => [{count: 1}, {count: 1}]
{resource: 'home', year: 2010, month: 0, day: 20} => [{count: 2}, {count: 1}]
```

Конечный результат тот же самый (3), однако он получается немного разными путями. Таким образом, reduce должен всегда быть идемпотентным. То есть,

вызывая reduce несколько раз, мы должны получать такой же результат, что и вызывая его один раз.

Мы не станем рассматривать этого здесь, однако распространена практика последовательных свёрток, когда требуется выполнить сложный анализ.

Чистая практика

С MongoDB мы вызываем у коллекции команду `mapReduce`. `mapReduce` принимает функцию map, функцию reduce и директивы для результата. В консоли мы можем создавать и передавать JavaScript функции. Из большинства библиотек вы будете передавать строковое представление функции (которое может выглядеть немного ужасно). Сперва давайте создадим набор данных:

```
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 4, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 5, 30)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 20, 6, 0)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 7, 0)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 8, 0)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 8, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 21, 8, 30)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 9, 0)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 21, 9, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 22, 5, 0)});
```

Теперь можно создать map и reduce функции (консоль MongoDB позволяет вводить многострочные конструкции):

```
var map = function() {
    var key = {resource: this.resource, year: this.date.getFullYear(), month:
this.date.getMonth(), day: this.date.getDate()};
    emit(key, {count: 1});
};

var reduce = function(key, values) {
    var sum = 0;
    values.forEach(function(value) {
        sum += value['count'];
    });
    return {count: sum};
};
```

Мы выполним команду `mapReduce` над коллекцией `hits` следующим образом:

```
db.hits.mapReduce(map, reduce, {out: {inline:1}})
```

Если вы выполните код, приведённый выше, вы увидите ожидаемый результат. Установив`out` в `inline` мы указываем, что `mapReduce` должен непосредственно вернуть результат в консоль. В данный момент размер результата ограничен 16 мегабайтами. Вместо этого мы могли бы написать `{out: 'hit_stats'}`, и результат был бы сохранён в коллекцию `hit_stats`:

```
db.hits.mapReduce(map, reduce, {out: 'hit_stats'});
db.hit_stats.find();
```

В таком случае все существовавшие данные из коллекции `hit_stats` были бы вначале удалены. Если бы мы написали `{out: {merge: 'hit_stats'}}`, существующие значения по соответствующим ключам были бы заменены на новые, а другие были бы вставлены. И наконец, можно в `out` использовать `reduce` функцию — для более сложных случаев.

Третий параметр принимает дополнительные значения — например, можно сортировать, фильтровать или ограничивать анализируемые данные. Мы также можем передать метод`finalize`, который применится к результату возвращённому этапом `reduce`.

Выводы к лабораторной работе

Это первая глава, в которой мы осветили совершенно новую для вас тему. Если вы испытываете неудобства, всегда можно обратиться к другим средствам агрегирования и более простым сценариям. Впрочем, MapReduce является одной из наиболее важных функций MongoDB. Чтобы научиться писать map и reduce функции, необходимо чётко представлять и понимать, как выглядят ваши данные и как они преобразовываются по пути через `map` и `reduce`.

Требования к сдаче лабораторной работы

1. Отчет к лабораторной работе, содержащий пошаговое исполнение заданий и промежуточные результаты

Задания к лабораторной работе

1. Создайте коллекцию с 5-10 документами с 3-5 полями, содержащими числа, строки и массивы. Для каждого поля (или комбинации полей) выполните произвольных MapReduce и выведите результат на экран. Результат одного из MapReduce сохраните в новую коллекцию.
2. При наличии интернета изучите 1-3 дополнительных значения поля out, продемонстрируйте их работу на произвольных коллекциях из задания 1.
3. Сделайте вывод о плюсах и минусах MapReduce.