HOPLANG — РАЗВИТИЕ ЯЗЫКА ОБРАБОТКИ ПОТОКОВ ДАННЫХ МОНИТОРИНГА

А.В. Адинец, П.А. Брызгалов, Вад.В. Воеводин, С.А. Жуматий, Д.А. Никитенко, К.С. Стефанов

В настоящее время актуальной проблемой является анализ эффективности суперкомпьютерных приложений. Основной задачей проекта HOPSA как раз и является исследование эффективности с помощью анализа данных мониторинга. А поскольку суперкомпьютерные приложения — это всегда большие приложения, то объём данных, которые требуется анализировать, достаточно большой.

Основным и наиболее распространённым инструментом обработки больших потоков данных в настоящее время является связка Pig + Hadoop. Но эта связка, как оказалось, не удовлетворяет потребностям проекта. В частности, возникают большие задержки даже при обработке небольших запросов, не поддерживается обработка потоков данных, нет нужного уровня абстракции от различных БД, наконец, не поддерживается работа с индексами в рамках БД. Поэтому для обработки данных кластерного мониторинга требуется свой язык, названный HopLang. Уже сейчас с помощью HopLang можно производить обработку реальных данных мониторинга. В статье освещается развитие языка за последние полгода и перспективы развития.

Введение

Язык HopLang — специализированный язык, основное назначение которого состоит в обработке больших потоков однородных данных [1].

В настоящее время единственным достаточно распространённым инструментом для обработки больших объёмов данных является Hadoop [2]. В паре с Hadoop часто используется Pig, высокоуровневый язык скриптов обработки данных с помощью Hadoop. Использование Pig позволяет абстрагироваться от низкоуровневых деталей реализации и на порядок сократить объём исходного кода по сравнению со связкой Hadoop + Java. Связка Pig + Hadoop уже использовалась в рамках проекта HOPSA для анализа данных потоков задач [3].

Однако, как показал практический опыт, данный подход обладает большим количеством недостатков:

- Pig не поддерживает индексы в БД, поэтому требуется перебирать всю базу даже для выполнения самых простых запросов;
- Поскольку Hadoop не предназначен для интерактивной обработки данных, в начале выполнения каждого запроса возникает сравнительно большая задержка, порядка нескольких секунд;
- Отсутствие нужного уровня абстракции от источника данных не позволяет сменить БД в случае необходимости;
- Отсутствует поддержка обработки потоков данных, например, данных мониторинга в реальном времени;
- Ограничения на вложенность не позволяют выполнять вложенную обработку потоков или вложенные запросы к БД, что ограничивает возможности совместной обработки данных потока задач и данных мониторинга;
- Pig не поддерживает какую-либо форму включаемых файлов или пользовательских функций-запросов (хотя возможны функции на Java), что ограничивает возможности написания переносимых запросов.

Описанные выше недостатки привели к появлению нового языка обработки данных, HopLang, сокращение от HOPsa LANGuage.

Подход, применяемый в HopLang, отличен от подхода MapReduce, применяемого в Hadoop, — он опирается на понятие *потока*, элементы которого могут быть обработаны параллельно, за некоторыми исключениями. Такой подход требует, чтобы структура каждого потока была однородна. Источником потока данных может быть запрос к базе данных, «живой» поток данных, например, от сенсоров системы мониторинга, результат обработки другого потока. Этот подход позволил абстрагироваться от баз данных при написании логики обработки данных, а также дал возможность масштабировать обработку данных, разбивая поток на независимые части.

Первоначальная реализация

В первой реализации языка был применён элементарный анализ синтаксиса на основе регулярных выражений. Строки программы нельзя было переносить, соседние операторы не допускается писать на одной строке. Первая версия языка включала в себя возможность объявления скалярных переменных (оператор **var**),

проверку условий (оператор **if**), а также набор итераторов — ключевых конструкций языка. Первая версия включала в себя итераторы **each**, **sort**, **print**, **top**, **bottom**.

Интерпретатор языка написан на языке Ruby. Применение языка высокого уровня с развитой стандартной библиотекой и широким выборов модулей расширения позволило быстро получить рабочий интерпретатор HopLang. Были реализованы драйверы для баз данных Cassandra (для одномерных и двумерных представлений данных), MongoDB [4] и текстовых данных в формате CSV. Для описания формата данных и привязки их к имени потока в программе используется файл конфигурации `hopsa.conf`. В данном файле задаются общие параметры интерпретатора, значения переменных по умолчанию (если это нужно). В разделе `varmap` описываются имена потоков, допустимых в программе, а также их привязка к базе данных, например:

```
tasks: {
   type: cassandra,
   keyspace: hopsa,
   keyname: taskid,
   cf: tasks_gra,
   max_items: -1,
   push_index: true
 }
```

Здесь описан поток `tasks`, данные которого расположены в БД Cassandra. Так как адрес БД не задан, то предполагается, что база данных работает на этом же сервере. В описании указаны пространство ключей (keyspace, аналог таблиц в Cassandra), имя ключевого поля (keyname) и другие служебные параметры для драйвера. Обязательным в описании является только параметр `type`, который задаёт имя драйвера БД.

Интерфейс для написания собственных драйверов БД открыт, поэтому добавить поддержку нового типа базы данных не составляет большой сложности. Новый драйвер достаточно расположить в специальном каталоге, и он будет автоматически использован компилятором. На данный момент драйвер может быть написан только на языке Ruby.

Основная рабочая конструкция первой версии HopLang — итератор **each**, который выполняет своё тело над каждым элементом входного потока, а по завершении потока — тело секции **final**. Ниже приведён пример подсчёта среднего значения потока, для положительных элементов:

```
var sum, count
out = each x in input where x.value > 0
  sum = sum + x.value
  count = count + 1
final
  yield avg => sum/count
end
```

Первая версия языка не включала в себя агрегатных функций, поэтому минимум, максимум и подобные функции приходилось вычислять вручную.

Кроме итератора **each** были реализованы: **sort,** позволяющий отсортировать по условию входной поток, **print**, выводящий поток на стандартный вывод, а также **top** и **bottom**, позволяющие получить N (параметр итератора) первых или последних элементов потока. **top** и **bottom** не выполняют полной сортировки, а просто создают буфер на N элементов и обновляют его состояние после каждого элемента потока. Такая реализация требует намного меньше памяти, и работает быстрее, чем полноценная сортировка всего потока.

Развитие языка

С момента первой рабочей реализации было проведено значительное расширение конструкций языка. Была добавлена управляющая конструкция **while-end**. Вид итератора **print** был расширен, теперь можно выводить произвольный набор полей, их имена могут отличаться от имён полей в потоке. Кроме того, допускается вывод нескольких потоков в одном итераторе — последовательно, в режиме round-robin или по мере поступления данных.

Например,

```
a = each t in ttt where t.np < 20
      yield t
end
b = each t in ttt2 where t.np >= 20 and t.np < 40
      yield t
end
```

```
print a, b
```

Так как объединение потоков иногда требуется не только для вывода результатов, то в язык добавлена конструкция **union**, «склеивающая» входные потоки. С её помощью, например, можно заменить оператор **print** предыдущего примера строками:

```
c = union a, b
print c
```

В новой версии добавлен новый важный итератор — **group**. Он аналогичен конструкции GROUP BY в SQL, то есть производит группировку данных по заданному выражению. Реализация этого итератора позволила производить подсчёт минимальных, максимальных или средних значений в точках заданной сетки, например, на участках длительностью 10 секунд.

Пример программы с итератором **group**:

```
flow1 = group c by int(c.time*10) in cpu_user where \
            c.time > 10000 and c.time < 20000 and \
            c.node == 'node1' or c.node == 'node2'
final
   yield time => int(c.time*10), \
      val => sum(c.value) / count(c.value), index => 1
   yield time => int(c.time*10), val => max(c.value), index => 2
   yield time => int(c.time*10), val => min(c.value), index => 3
end

print(order=time,val,index) flow1
```

Здесь мы читаем поток с данными загрузки процессора (cpu_user) итератором **group.** Полученный поток данных разбивается (на лету) на группы по значению выражения int(c.time*10), т. е. в одной группе будут данные за 10 секунд. Тело итератора пустое, сразу идёт секция **final**. Тело **final** выполняется каждой группой, в данном случае мы вычисляем минимум, максимум и среднее значение загрузки по всем узлам за 10 секунд.

Тело итератора также выполняется отдельно каждой группой, но уже для каждого значения. Вот пример вычисления среднего значения загрузки процессора, но без агрегатной функции:

```
flow1 = group c by int(c.time*10) in cpu_user where \
            c.time > 10000 and c.time < 20000 and \
            c.node == 'node1' or c.node == 'node2'
   var sum, count
   sum = sum + c.value
   count = count + 1
final
   yield time => int(c.time*10), val => summa / count
end

print(order=time,val) flow1
```

В этих примерах мы также проиллюстрировали новый синтаксис оператора **print**; текущая реализация поддерживает также старый синтаксис.

Другим важным шагом стала реализация делегирования условий выборки базе данных. Данная функциональсть также именуется *проталкиванием* предикатов или фильтров в БД. Если выборка по условию может быть полностью или частично проведена самой базой, то драйвер HopLang протолкнёт всё условие или его часть в базу.

Например, задание условия в выражении

```
x = each y in mydata where y.time > 10 and y.time < 20
```

будет преобразовано в условие запроса к БД, если это Cassandra (и если для time построен индекс либо оно является ключевым полем) или MongoDB. Если преобразование успешно, то скорость выполнения программы существенно возрастает, т. к. объём выборки из БД снижается. В настоящее время для БД Cassandra реализовано проталкивание только конъюнкций простых предикатов. Простой предикат — это числовое или строковое сравнение на равенство, больше или меньше поля записи в БД с выражением, значение которого не

меняется во время выполнения запроса. Для MongoDB реализовано проталкивание любых выражений, для вычисления которых требуются только поля записей БД, стандартные операции и значения, не изменяющиеся за время выполнения запроса.

Хотя проталкивание предикатов позволяет в ряде случаев во много раз повысить скорость обработки, практический опыт показывает, что этой функциональности не всегда хватает. Типичным запросом к данным мониторинга является выборка данных, например, загрузки ЦПУ, по одной задаче. Каждая задача характеризуется временем счёта и набором узлов, на которых она считалась, соответственно, эти два параметра и требуется использовать для фильтрации. Фильтрация по интервалу времени реализуется просто просто как конъюнкция двух сравнений, и легко проталкивается в БД. Однако фильтрацию по узлу просто так в БД не протолкнёшь, ведь узлы не обязательно образуют непрерывный интервал.

Также встаёт вопрос о хранении набора узлов, на которых считалась задача. Можно использовать для этого отдельную таблицу; однако существующие нереляционные БД не поддерживают эффективную операцию соединения. Кроме того, для больших задач количество узлов будет большим, что повлечёт за собой дополнительную нагрузку.

Однако выясняется, что для наборов узлов можно использовать компактное представление. Например, запись «node-[01-08]-[01-32]» не требует больших ресурсов для хранения, и в то же время задаёт набор из 256 узлов, на которых считалась задача. В данном случае задача считалась на всех шасси с 01 по 08, на всех узлах каждого шасси (01-32). В более общем случае набор узлов выражается объединением таких простых интервалов; объединение записывается как несколько интервалов, через запятую. Но и в этом случае такое строковое представление намного более компактно, чем простое хранение набора узлов.

Для упрощения построения запросов, содержащих фильтрацию по узлам, в HopLang встроена новая функция **ins**. Она проверяет вхождение строки в набор узлов, задаваемый в описанном выше формате. Разумеется, функция ins не ограничивается только наборами узлов, и может применяться к другим *наборам диапазонов строк*, но пока других практических потребностей в ней не возникало. В дальнейшем мы будем говорить о наборах диапазонов строк, для сохранения общности.

Задание наборов диапазонов строк производится одной строкой, в которой диапазоны перечисляются через запятую без пробелов. Диапазон задаётся строкой, в которой присутствует одна пара квадратных скобок, в которых задан список и/или диапазон значений. На данный момент поддерживаются только числовые диапазоны значений.

Пример задания набора диапазонов: «node[01-12,14-16,20]». В данном примере будут заданы строки node01, node02 и т. д. до node16, кроме node13, а также строка node20.

Пример использования **ins** в HopLang-программе:

```
x = each y in mydata where y.node ins 'node[01-12,14-16,20]' \
    and y.time < 20
```

Такое задание диапазонов значений строк позволяет оптимизировать проверки как на уровне баз данных, так и на уровне самой программы. Кроме того, запросы с наборами диапазонов строк позволяют иметь единый запрос и передавать различные наборы узлов как параметры, а не генерировать запрос заново для разных наборов узлов, на которых считались разные задачи.

В реализации драйвера MongoDB конструкция **ins** преобразуется в набор отдельных запросов, в каждом из которых выполняется либо сравнение с константой (как node20 в примере), либо производится лексикографическое сравнение с границами диапазона (в примере — node01 и node12). Такая реализация позволяет сократить условия выборки из БД, таким образом, ускорить выборку, не получая при этом лишних данных. В настоящее время также поддерживается преобразования наборов диапазонов строк в регулярное выражение, для тестирования непосредственно в интерпретаторе HopLang, или для проталкивания в БД с эффективной поддержкой регулярных выражений. Имеется также преобразование в набор лексикографических диапазонов строк (используется в драйвере MongoDB), а также просто в множество строк. В будущем возможно преобразование наборов диапазонов строк в маску для оператора LIKE для проталкивания в SQL БД.

В последней версии языка реализованы агрегатные функции min, max, sum, count. Их использование позволило ускорить обработку данных и уменьшить размер программ.

Реализован HopLang-сервер, который позволяет выполнять HopLang-запросы через web-интерфейс. Такое решение позволяет встраивать HopLang в любые сервисы и программы, так как выполнение программы сводится к посылке HTTP-запроса.

Структура url-ов запросов к HopLang-серверу приведена в табл. 1.

Таблица 1.Пути в HTTP-запросах к HopLang-серверу и их семантика

/hopsa	общий префикс
/hopsa/html	префикс для запросов из web-форм
/hopsa/control	управление

/hopsa/auth	аутентификация
/hopsa/info	информационный раздел
/hopsa/info/users	информация по пользователям
/hopsa/info/nodes	информация по узлам (источникам)
/hopsa/info/sensors	информация по сенсорам
/hopsa/info/status	информация по статусу сервера
/hopsa/hoplang	(или /hoplang для legacy) выполнение hoplang-программ

Если URL запроса имеет окончание .html или .json, то ответ выдаётся в соответствующем формате. По умолчанию ответ выдаётся в формате CSV. Для всех путей, кроме /hopsa/hoplang, ответ состоит из строки-статуса, а затем собственно CSV-таблицы ответа.

Например, в ответ на /hopsa/info/nodes:

```
0000-Ihsi-OK, OK: nodes list below
node1, sensors=1:1,1:2,2,3,4,5,10:1,12,13,14,25:7,up
node2, sensors=1:1,1:2,2,3,4,5,10:1,10:2,10:3,10:4,up
node3, sensors=1:1,1:2,up
node4, sensors=163,down
node5, ,sensors=,down
```

А в ответ на /hopsa/info/nodes.json:

```
{status:{code:'0000-Ihsi-OK', message:'OK: nodes list below'},
[{name:'node1', sensors:'1:1,1:2,2,3,4,5,10:1,12,13,14,25:7',state:'up'},
{name:'node2, sensors:'1:1,1:2,2,3,4,5,10:1,10:2,10:3,10:4',state:'up'},
{name:'node3, sensors:'1:1,1:2',state:'up'},
{name:'node4, sensors:'163',state:'down'},
{name:'node5, sensors:'',state:'down'}
]}
```

В данный момент полностью работает только запуск программ, но с реализацией распределённого выполнения будут полностью реализованы и остальные компоненты интерфейса.

Перспективы

На данный момент HopLang уже способен на переработку потоков данных мониторинга в миллионы строк и более за времена порядка нескольких минут. В рамках проекта HOPSA он уже сейчас используется для построения дайджестов для реальных задач, считающихся на кластерах «Чебышев» и «Графит» суперкомпьютерного комплекса МГУ. HopLang также используется для верификации данных мониторинга, получаемых с кластеров и сохраняемых в БД.

Использование HopLang позволило достичь повышения скорости обработки данных по сравнению со связкой Pig + Hadoop. Но чтобы достичь цели в обработке любых объёмов данных, заложенные идеи параллельной обработки требуется реализовать не только в виде конвейера итераторов, как в текущей версии, но и в виде распределённой обработки элементов потоков на кластере.

Любой итератор может быть выполнен на нескольких независимых компьютерах со своим набором данных. В данный момент идёт работа над реализацией этой идеи. Кроме распределения вычислений, будет реализовано и распределение данных — один источник может быть сохранён в нескольких физически разных базах данных (например, данные с узлов node01..node50 на одном компьютере, а с узлов node51..node99 — на другом). Зная распределение данных, можно распределить и вычисления, обрабатывая только локальные данные, находящиеся в локальной базе данных. После этого результаты затем объединяются в единый выходной поток.

Часто для обработки данных потребуется использовать функции, которые написаны не на HopLang. Например, может потребоваться использовать стороннюю библиотеку статистического анализа данных, или же потребуется переписать алгоритм на более низкоуровневом языке для повышения скорости его работы. Для этого к HopLang-программе потребуется подключить код, написанный, например, на C. В настоящее время разрабатывается интерфейс, который позволит использовать Си-подпрограммы в HopLang-программах. Кроме скалярных функций, необходимо позволить пользователю определять собственные функции-редукторы, для них также разрабатывается интерфейс.

Язык HopLang открыт для новых идей и применений, вы можете оставлять свои отзывы, пожелания и даже принять участие в разработке на сайте проекта [5].

Благодарности

Работа выполнена в рамках государственного контракта «Разработка методов и инструментальных систем для анализа эффективности работы параллельных программ и суперкомпьютеров» (ГК 07.514.12.4001).

ЛИТЕРАТУРА:

1. Адинец А.В., Брызгалов П.А., Воеводин Вад.В., Жуматий С.А., Никитенко Д.А. Об одном подходу к мониторингу, анализу и визуализации потока заданий на кластерной системе // Вычислительные методы и программирование. Т. 12 (2011). стр. 90 – 93.
2. «Hadoop homepage». URL: http://hadoop.apache.org/ (дата обращения 5.05.2012).
3. Адинец А.В., Брызгалов П.А., Жуматий С.А., Никитенко Д.А. Система визуализации параметров работы больших вычислительных систем // Труды международной конференции «Параллельные вычислительные технологии-2012 (ПаВТ-2012)», 26 – 30 марта 2012 г., Новосибирск, стр. 714.
4. Kyle Banker «MongoDB in Action». Manning Publications, 2011, 311 стр. ISBN 978-1935182870.
5. HopLang Project Homepage. URL: http://github.com/zhum/hoplang (дата обращения 5.05.2012).