УДК 004.021

HOPLANG: РАЗВИТИЕ ЯЗЫКА ОБРАБОТКИ ПОТОКОВ ДАННЫХ МОНИТОРИНГА

А. В. Адинец[1], П. А. Брызгалов[1], Вад. В. Воеводин[1], С. А. Жуматий[1], Д. А. Никитенко[1], К. С. Стефанов[1]

В настоящее время актуальной проблемой является анализ эффективности суперкомпьютерных приложений. Основная задача проекта HOPSA — исследование эффективности с помощью анализа данных мониторинга. Поскольку суперкомпьютерные приложения — это всегда ресурсоемкие приложения, то объем данных, которые требуется анализировать, достаточно большой. Основным и наиболее распространенным инструментом обработки больших потоков данных в настоящее время является связка Pig + Hadoop, однако эта связка, как оказалось, не удовлетворяет потребностям проекта. В частности, возникают значительные задержки даже при обработке небольших запросов, не поддерживается обработка потоков данных, нет нужного уровня абстрагирования от различных баз данных и, наконец, не поддерживается работа с индексами в рамках баз данных. В этой связи для обработки данных кластерного мониторинга требуется специальный язык, получивший название HopLang. В статье освещается развитие языка за последние полгода и перспективы его развития. Статья рекомендована к публикации программным комитетом Международной научной конференции “Научный сервис в сети Интернет: поиск новых решений” (http://agora.guru.ru/abrau).

Ключевые слова: параллельные вычисления, мониторинг, суперкомпьютеры.

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

В настоящее время единственным достаточно распространенным инструментом для обработки больших объемов данных является технология Hadoop [2]. В паре с Hadoop часто используется язык Pig, представляющий собой высокоуровневый язык скриптов обработки данных с помощью инструментальных средств, встроенных в Hadoop. Использование языка Pig позволяет абстрагироваться от низкоуровневых деталей реализации и на порядок сократить объем 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, такой подход обладает многими недостатками, из которых перечислим следующие:

— язык Pig не поддерживает индексы в базах данных, поэтому требуется перебирать всю базу даже для выполнения самых простых запросов;

— поскольку технология Hadoop не предназначена 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 базу данных 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 базам данных, что затрудняет совместную обработку данных потока задач и данных мониторинга;

– язык Pig не поддерживает какую-либо форму включаемых файлов или пользовательских функций-запросов (хотя возможны функции на языке Java), что ограничивает возможности написания переносимых запросов.

[1] Научно-исследовательский вычислительный центр, Московский государственный университет им. М. В. Ломоносова, Ленинские горы, д. 1., стр. 4, 119992, Москва; А. В. Адинец, мл. науч. сотр., e-mail: adinetz@gmail.com; П. А. Брызгалов, науч. сотр., e-mail: pyotr@parallel.ru; Вад. В. Воеводин, мл. науч. сотр., e-mail: vadim_voevodin@mail.ru; С. А. Жуматий, ст. науч. сотр., e-mail: serg@parallel.ru; Д. А. Никитенко, мл. науч. сотр., e-mail: dan@parallel.ru; К. С. Стефанов, науч. сотр., e-mail: cstef@parallel.ru

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

Подход, применяемый в HopLang, отличен от модели MapReduce, применяемой 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, или 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 применен элементарный анализ синтаксиса на основе регулярных выражений. Строки программы нельзя было переносить, а соседние операторы писать на одной строке. Первая версия языка включала в себя возможность объявления скалярных переменных (оператор var), проверки условий (оператор if), а также набор итераторов — ключевых конструкций языка: итераторы each, sort, print, top и bottom. Интерпретатор языка написан на языке Ruby. Применение языка высокого уровня с развитой стандартной библиотекой и широким выбором модулей расширения позволило быстро получить рабочий интерпретатор HopLang. Были реализованы драйверы для баз данных Cassandra (для одномерных и двумерных представлений данных) и MongoDB [4], а также для текстовых данных в формате CSV (Comma-Separated Values). Для описания формата данных и привязки их к имени потока в программе используется файл конфигурации hopsa.conf. В этом 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. Приведем пример описания потока:

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

В этом примере описан поток tasks, данные которого расположены в базе данных Cassandra. Так как адрес базы данных не задан, то предполагается, что база данных работает на том 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 задает имя драйвера базы данных. Интерфейс для написания собственных драйверов базы данных 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 текущий момент драйвер может быть написан только на языке Ruby.

Основная рабочая конструкция первой версии HopLang — это итератор each, который выполняет свое тело над каждым элементом входного потока, а по завершении потока — тело секции final. Ниже приведен пример подсчета 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, и print, выводящий поток на стандартный вывод, а также top и bottom, позволяющие получить N (параметр итератора) первых или последних элементов потока. Итераторы 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 имен 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 ее помощью, например, можно заменить оператор print предыдущего примера следующими строками:

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

В новой версии добавлен новый важный итератор group, который аналогичен конструкции GROUP BY в языке SQL (Structured Query Language), т.е. производит группировку данных по заданному выражению. Реализация этого итератора позволила производить подсчет 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 идет 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. Приведем 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; отметим, что текущая реализация поддерживает и старый синтаксис.

Другим важным шагом стала реализация делегирования условий выборки базе данных. Эта функциональность также именуется проталкиванием предикатов или фильтров в базу данных. Если выборка по условию может быть полностью или частично проведена самой базой, то драйвер HopLang поместит

(протолкнет) все условие или его часть в базу.

Приведем пример задания условия в выражении:

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

В этом примере условие будет преобразовано в условие запроса к базе данных, 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, поскольку объем выборки из базы данных снижается. В настоящее время для базы данных Cassandra реализовано проталкивание только конъюнкций простых предикатов. Простой предикат — это числовое или строковое сравнение на равенство, больше или меньше поля записи в базе данных с выражением, значение которого не меняется во время выполнения запроса. Для базы данных MongoDB реализовано проталкивание любых выражений, для вычисления которых требуются только поля записей базы данных, 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 счета и набором узлов, на которых она считалась; соответственно, эти два параметра и требуется использовать для фильтрации. Фильтрация по интервалу времени реализуется как конъюнкция двух сравнений и легко проталкивается в базу данных. Однако фильтрацию по узлу просто так в базу данных не протолкнешь, ведь узлы не обязательно образуют непрерывный интервал.

Также встает вопрос о хранении набора узлов, на которых считалась задача. Можно использовать для этого отдельную таблицу, однако существующие нереляционные базы данных не поддерживают эффективную операцию соединения. Кроме того, для больших задач количество узлов будет большим, что повлечет 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 задает набор из 256 узлов, на которых считалась задача. В этом 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. Однако 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, проверяющая 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, однако 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 задается 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 этом примере заданы строки node01, node02 и т. д. до node16, кроме node13, а также строка node20.

Приведем пример использования функции 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 базы данных MongoDB конструкция ins преобразуется в набор отдельных запросов, в каждом из которых либо выполняется 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 базы данных и, таким образом, ускорить выборку, не получая при этом лишних данных. Кроме того, в настоящее время поддерживается преобразование наборов диапазонов строк в регулярное выражение для тестирования непосредственно в интерпретаторе HopLang или для проталкивания в базу данных 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 базы данных.

В последней версии языка реализованы агрегатные функции min, max, sum и 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-запросов к HopLang-серверу приведена в следующей таблице.

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 выдается в соответствующем формате. По умолчанию ответ выдается 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 получим

```
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
```

Ответ на /hopsa/info/nodes.json имеет вид

```
{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 настоящее время полностью работает только запуск программ, но с реализацией распределенного 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. Кроме того, Язык HopLang используется для верификации данных мониторинга, получаемых с кластеров и сохраняемых в базе данных.

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

Любой итератор может быть выполнен на нескольких независимых компьютерах со своим набором данных. В данный момент идет работа над реализацией этой идеи. Кроме распределения вычислений, будет реализовано и распределение данных — один источник может быть сохранен 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 объединяются в единый выходной поток.

Часто для обработки данных требуется использовать функции, которые написаны не на HopLang. Например, может потребоваться использовать стороннюю библиотеку статистического анализа данных

или же переписать алгоритм на более низкоуровневом языке для повышения скорости его работы. Для этого к HopLang-программе потребуется подключить код, написанный, например, на языке Си. 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].

Работа выполнена при финансовой поддержке Минобрнауки России по государственному контракту от 24.12.2010 г. № 07.514.12.4001 в рамках ФЦП “Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2013 годы”.

СПИСОК ЛИТЕРАТУРЫ

1. *Адинец А.В., Брызгалов П.А., Воеводин Вад.В., Жуматий С.А., Никитенко Д.А.* Об одном подходе к мониторингу, анализу и визуализации потока заданий на кластерной системе // Вычислительные методы и программирование. 2011. **12**. 90–93.
2. Hadoop homepage (http://hadoop.apache.org/).
3. *Адинец А.В., Брызгалов П.А., Жуматий С.А., Никитенко Д.А.* Система визуализации параметров работы больших вычислительных систем // Тр. Междунар. конф. “Параллельные вычислительные технологии-2012 (ПаВТ-2012)”. Новосибирск, 26–30 марта 2012 г. Челябинск: Издательский Центр ЮУрГУ, 2012. 714.
4. *Banker K.* MongoDB in action. Shelter Island: Manning Publications, 2011.
5. HopLang Project Homepage (http://github.com/zhum/hoplang).

Поступила в редакцию
29.09.2012