

МОНИТОРИНГ, АНАЛИЗ И ВИЗУАЛИЗАЦИЯ ПОТОКА ЗАДАНИЙ НА КЛАСТЕРНОЙ СИСТЕМЕ

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

НИВЦ МГУ им. М.В. Ломоносова, Москва

Введение

В связи с распространением больших кластерных систем в настоящее время актуальной является задача эффективного использования таких систем. Для этого необходимо выявлять проблемы, возникающие в процессе счета задачи, извещать пользователей о существовании проблем и предлагать возможные пути их решения.

Одним из путей достижения этих целей является непрерывный мониторинг выполняемых на кластере задач и анализ полученных данных. Прежде всего, это данные программных и аппаратных счетчиков узлов, такие как загрузка процессора, количество переданных по сети пакетов и данных, объем занимаемой памяти или количество обращений к файловой системе. Также сюда входят данные, касающиеся самой аппаратуры, например, информация, снимаемая с температурных датчиков. Наконец, сюда входит и информация, поставляемая стоящей на кластере системой очередей – чтобы можно было привязать данные мониторинга к конкретным задачам и пользователям. Все эти данные необходимо уметь собирать, обрабатывать, анализировать и строить на их основе отчеты и графики, которые помогли бы пользователям понять, в чем проблема, а в идеале – разобраться с ней.

Общая архитектура системы

Для решения поставленных задач была предложена следующая архитектура (рис.1). Здесь данные о задачах поступают от системы управления задачами в «слой агрегации» (aggregation layer), где все они приводятся к единому формату. Это нужно для того, чтобы обрабатывать данные от любой системы управления задачами. Далее данные передаются одному или нескольким модулям обработки и/или сохранения данных (analysis modules, db modules), которые могут немедленно произвести какие-то действия (например, обновить данные об использованной квоте, выдать пользователю отчет о работе его задачи и т. п.), или сохранить полученные данные для дальнейшей обработки.

Сохраненные данные можно получать из базы данных (БД) и обрабатывать единообразно – это может сделать любой клиент системы. Запрос клиента формируется на языке обработки данных и не зависит от используемых баз данных, их структуры и т.п. Запрос обрабатывается модулями анализа (analysis modules). Результат обработки передается обратно клиенту.

Один из наиболее востребованных клиентов – web-интерфейс. Он выделен на рисунке отдельно, т. к. клиент системы выступает и в качестве сервера для запросов пользователя (call center). Работа web-сервера (и одновременно клиента системы) состоит не только в посылке запросов к модулям анализа, но и визуализации полученных данных, предоставления пользователю удобного интерфейса к анализу данных.

Управление обработкой данных и хранение информации о конфигурации всей системы (баз данных, источников данных, загруженности модулей обработки) – задача модулей управления.

Данная архитектура предполагает, что все ее компоненты могут быть дублированы. Кроме отказоустойчивости это позволяет масштабировать обработку данных при наличии большого числа запросов или большого объема данных.

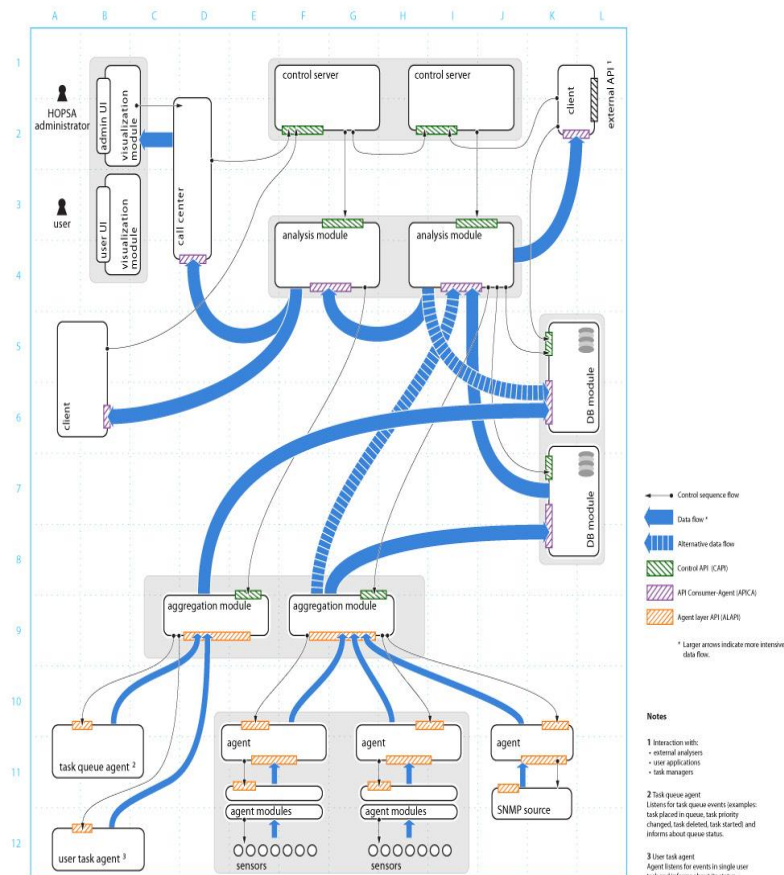


Рис. 1. Общая архитектура создаваемой системы

Сбор, анализ и обработка данных

Выбор конкретных используемых технологий обусловлен большим объемом данных, которые требуется обрабатывать. Даже элементарные данные о прохождении задач, такие как имя пользователя и очереди, командная строка, количество узлов и времена постановки в очередь, начала счета и завершения, собранные за несколько лет на кластере «Чебышев», занимают 0.5 Гб. Если добавить сюда данные с узлов, то объем данных достигнет нескольких гигабайт. Но и это – сравнительно немного по сравнению с полным набором датчиков с узлов, объем информации по которым может достигать десятков Гб за сутки. Запросы могут одновременно поступать от нескольких пользователей и требуется их обрабатывать как можно быстрее.

Ресурсами одного компьютера в таких условиях не обойтись. Поэтому были выбраны технологии обработки больших объемов данных, которые масштабируются на вычислительные кластеры. Для хранения данных используется распределенная БД Cassandra [1], поддерживающая хранение данных в виде таблиц «ключ-словарь значений», эффективную выборку по ключам и ограниченные возможности индексирования. Для обработки данных на кластерах используется технология Hadoop [2], реализация подхода MapReduce, при помощи которой выполняются запросы, написанные на высокоуровневом языке запросов Pig [3].

В текущем прототипе реализован сбор данных только для системы очередей. Компонент изображенный на рис. 1 как task queue agent, представляет собой набор из трех

скриптов, которые получают данные непосредственно из системы очередей за последние сутки и импортируют данные в БД Cassandra. В настоящее время поддерживается импорт данных из систем очередей Cleo и SLURM.

После импорта в БД Cassandra данные доступны для обработки при помощи технологии Hadoop. Однако сама технология Hadoop является довольно низкоуровневой, и написание для нее запросов требует серьезных затрат. Чтобы сократить время написания и объем кода запросов, используется высокоуровневый язык запросов Pig – SQL-подобный скриптовый язык для запросов, выполняемых распределенно с использованием технологии Hadoop. Pig поддерживает работу с большим количеством источников данных, среди которых CSV, HBase и Cassandra. В отличие от SQL, Pig является скорее процедурным языком, а не декларативным, и Pig-операторы, как правило, выполняются именно так, как они записаны в программе. В случае обработки данных о потоках задач гибкость процедурного подхода оказывается достоинством. Так, это позволяет, например, получать данные о частоте использования того или иного пакета (под пакетом подразумевается имя исполняемого файла), не требуя построения для этого специального индекса.

На рис. 2 изображен пример Pig-запроса, определяющий 100 пакетов с наибольшим количеством использующих их пользователей. Директивы %default используются для задания параметров запроса. Оператор LOAD загружает данные, в данном случае – из БД Cassandra; переданный URL определяет БД и таблицу, из которой требуется брать данные. Следующий за ним оператор FOREACH .. GENERATE преобразует данные из набора пар «ключ – набор имен столбцов и их значений» в набор кортежей, более привычный для обработки. После этого при помощи регулярного выражения и встроенной функции REGEX_EXTRACT из командной строки извлекается имя исполняемого файла. Далее идет двойная группировка и агрегация – сперва по пакетам, затем по пользователям – чтобы для каждого пакета получить количество его пользователей. Наконец, полученные значения сортируются при помощи оператора ORDER и из них выбирается топ-100 при помощи оператора LIMIT.

```
%default cf 'tasks_cheb'
%default ntop 100
%default resdir '/does/not/exist'

REGISTER 'hopsa-udfs.jar';

tsr = LOAD 'cassandra://hopsa/$cf' USING CassandraStorage() AS (k:bytearray, va:{t:(
  n:bytearray, v:bytearray)});
ts1 = FOREACH tsr {
  un = FILTER va BY n=='user';
  cl = FILTER va BY n=='cmdline';
  GENERATE k AS key, FLATTEN(un.v) AS user,
    FLATTEN(cl.v) AS cmdline;
};
ts = FOREACH ts1 GENERATE key, user, REGEX_EXTRACT(cmdline,
  '/?([~._a-zA-Z0-9]*)*([~._a-zA-Z0-9]{1,})', 2) AS pack;
grp = GROUP ts BY (pack, user);
ucs = FOREACH grp GENERATE group.pack AS pack, group.user AS user, COUNT(ts) AS nruns;
grp2 = GROUP ucs BY pack;
packs = FOREACH grp2 GENERATE group AS pack, COUNT(ucs) AS nusers,
  SUM(ucs.nruns) AS nruns;
lpacks = ORDER packs BY nusers DESC;
topacks = LIMIT lpacks $ntop;
STORE topacks INTO '$resdir' USING PigStorage('\t');
```

Рис. 2. Pig-запрос, находящий 100 пакетов с наибольшим числом использующих их пользователей

Большие Pig-запросы могут исполняться достаточно эффективно; например, запрос на рис. 2 на данных с кластера «Чебышев» за время его активной работы (3 года) выполняется всего лишь за 2 минуты.

Визуализация

Данные, полученные в результате обработки, должны быть представлены пользователю в удобном для восприятия виде. Эту задачу решает call center, который как предоставляет web-интерфейс пользователю, так и является клиентом системы обработки данных. Пользователь может выбрать один из наиболее частых запросов, например, распределение числа задач за период времени, и его параметры – интервалы времени, список интересующих пользователей и т.д.

Если пользователю требуется запрос, которого нет в списке, он может сформировать собственный запрос на языке обработки данных, возможно, взяв за основу один из предложенных ранее.

Полученный от системы ответ всегда представляет собой данные в виде таблицы – набор именованных столбцов и набор строк собственно данных. Используя библиотеку Google Chart, полученные данные отображаются в виде графиков и диаграмм. Для каждого имени данных заранее известен его тип, а значит по полученным данным можно быстро подобрать набор подходящих диаграмм. Пользователь может менять варианты отображения – тип диаграммы, масштаб и т. п., не посылая заново запрос, что позволяет быстро проводить разносторонний анализ полученных данных.

На рис. 3 приведен пример интерактивного отображения средней длины очередей суперкомпьютера «Ломоносов». В верхней части можно выбрать закладку с типом диаграммы (пузырьковая, столбцовая или линейная), внизу – дату отображения, а справа – набор очередей.

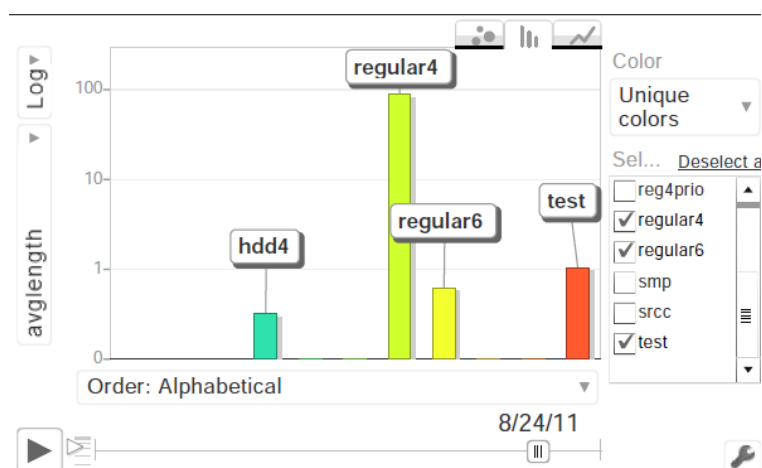


Рис. 3. Пример визуализации данных с кластера «Ломоносов», построенных при помощи разработанной системы

Набор вариантов отображения не ограничивается приведенным на рисунке – инструментарий Google Charts Toolkit позволяет строить несколько десятков типов интерактивных диаграмм. Встроенная логика call center позволяет использовать только те диаграммы, которые подходят по типу отображаемых данных.

Построенные диаграммы можно сохранить в файл для дальнейшего изучения или архива. Визуализация данных работает во всех современных браузерах, поддерживающих JavaScript или HTML5 – Internet Explorer, Mozilla Firefox, Opera, Safari.

Реализованная версия – только начальный шаг в реализации интерфейса визуализации. В ближайших планах – реализовать визуальный конструктор запросов, добавить авторизацию и аутентификацию пользователей.

Заключение

В настоящее время разработанная система развернута в НИВЦ МГУ и используется для мониторинга кластеров «Графит», «Ломоносов» и «Чебышев». Система находится в состоянии активного развития и тестирования.

Вместе с тем, становятся очевидными недостатки технологий, заложенных в основу системы. Прежде всего это касается системы хранения и обработки данных. Поддержка индексов в Cassandra, как и в большинстве других NoSQL БД в настоящее время сильно ограничена. Как следствие, приходится перебирать весь объем данных при обработке даже сравнительно небольших запросов. Далее, технология Hadoop вносит большие накладные расходы как на этапе начальной инициализации (порядка нескольких секунд на запрос), так и при последующей обработке данных. Естественно, эти расходы приемлемы при обработке действительно больших запросов, но для менее ресурсоемких запросов их хочется избежать. Язык Pig, хотя и предоставляет интерфейс к большому количеству различных БД, не поддерживает прозрачного перехода между различными БД и требует в этом случае изменения скриптов запросов. Наконец, Pig не поддерживает запросов произвольной вложенности и обработки потоков данных в реальном времени.

Ограничения появляются также на стороне клиента – со сравнительно небольшим объемом данных, получаемых, например, при вычислении средней длины очереди по дням за несколько лет, потребление памяти браузера вырастает до 1 Гб, а интерфейс пользователя начинает тормозить.

Поэтому основным направлением работы в ближайшее время будет переход на более гибкие технологии, прежде всего в области обработки данных. В частности, уже разрабатывается язык запросов Hoplang, который также будет поддерживать распределенное исполнение, но будет лишен перечисленных выше недостатков. Разрабатываемый язык будет поддерживать работу с различными БД, в т.ч. Cassandra, что позволит более плавно перейти на него в системе.

Также возможным направлением дальнейшей работы будет более тесное взаимодействие с европейскими партнерами в рамках проекта «HOPSA». Планируется, во-первых, поддержка единого европейского формата данных OTF как источника данных в нашей системе, и во-вторых, обеспечение доступа инструментов европейской стороны к нашим данным.

Работа выполнена при поддержке госконтракта Министерства образования и науки РФ № 07.514.12.4001 и гранта РФФИ № 09-07-00168-а.

Литература

1. Eben Hewitt. Cassandra: the Definitive Guide. O'Reilly Media, November 2010, 330 p. ISBN: 978-1-4493-9041-9.
2. Tom White. Hadoop: the Definitive Guide. O'Reilly Media, May 2009, 528 p. ISBN: 978-0-596-52197-4.
3. Pig project – [<http://pig.apache.org/>]. Accessed 28.09.2011.