Сбор и визуализация данных о ресурсах, используемых распределенной задачей*

А.Ю. Берсенёв

Институт математики и механики им. Н.Н. Красовского УрО РАН

В данной статье пойдет речь о решении для сбора и визуализации данных, спроектированном в Институте математики и механики им. Н.Н. Красовского УрО РАН (ИММ УрО РАН), которое успешно используется на кластере «Уран».

1. Введение

Как сделать кластер производительнее? Кажется, что ответ на этот вопрос очевиден: закупить новые узлы, добавить памяти, или, скажем, модернизировать систему хранения или сеть. Но приведет ли, например, модернизация системы хранения к реальному уменьшению времени счета большинства задач?

Как правило, у задачи в каждый момент времени есть узкое место – ресурс, который она потребляет на 100%. Это тот ресурс, который имеет смысл оптимизировать, поскольку оптимизация других ресурсов не позволит обойти узкое место и заметно сократить суммарное время счета.

Для поиска узких мест можно собирать некие данные с вычислительных узлов – данные об использовании ресурсов. Так как просмотр этих данных в «сыром» виде малоинформативен, то появляется необходимость в системе анализа и визуализации данных.

В данной статье пойдет речь о решении для сбора и визуализации данных, спроектированном в Институте математики и механики им. Н.Н. Красовского УрО РАН (ИММ УрО РАН), которое успешно используется на кластере «Уран».

2. Принципы построения и архитектура

Система спроектирована с учетом следующих принципов:

1. Использование в качестве компонентов системы только продуктов с открытыми лицензиями;
2. Простота внутреннего устройства, ведущая к простоте развертывания и использования;
3. Минимальные накладные расходы на сбор и хранение данных;
4. Возможность изменения набора собираемых данных;
5. Максимальная безопасность.

Система логически разделена на две составляющие: система сбора данных и система обработки и визуализации этих данных. Такое разделение обеспечивает выполнение второго принципа. Для развертывания системы нужно установить rpm-пакет системы сбора данных и скопировать несколько файлов системы визуализации.

2.1 Система сбора данных

В качестве агентов для сбора данных используется программа collectl. Она выбрана т. к. имеет расширяемую архитектуру и распространяется под лицензией GPL v2. Программа выполняет одну простую задачу – периодически считывает данные со счетчиков в ядре ОС

* Работа выполнена в рамках программы Президиума РАН № 18 "Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности" при поддержке УрО РАН (проект 12-П-1-1034).

(например, загрузку ядер cpu, размер занятой памяти и текущую скорость передачи данных), и записывает их в текстовый файл, находящийся на сетевом диске, доступном со всех узлов.

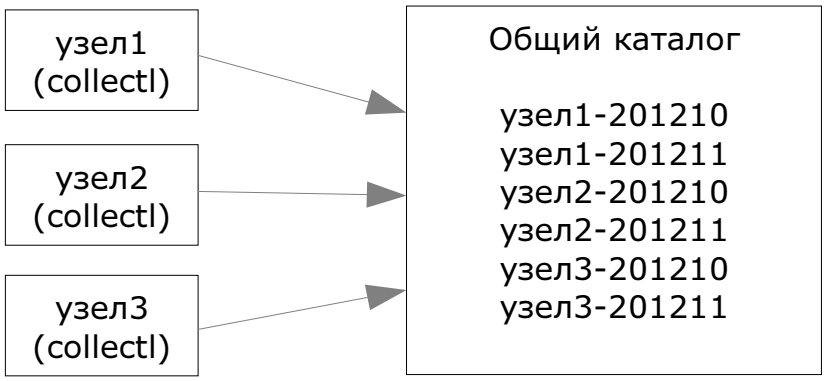

Рис. 1. Схема работы системы сбора данных

Эта программа была доработана под конкретную задачу:

1. Была добавлена возможность записывать данные в файл, имя которого формируется из имени узла, текущего года и месяца. Например, в декабре 2012 года, данные с узла «узел1» попадут в файл узел1-201212. Такая организация файлов позволяет легко делать выборки как по узлам, так и по временным интервалам, а также позволяет организовать удобную систему архивации старых файлов.
2. Для уменьшения размеров выходного файла был изменен его формат. Удаление из выдачи малоинформативных счетчиков уменьшило размер файла в 4 раза.
3. Для уменьшения влияния на запущенные процессы была добавлена буферизация вывода. Модернизированная версия collectl «накапливает» данные за час, а в конце часа записывает их в файл за одну операцию. Таким образом, обеспечивается выполнение третьего принципа.
4. Исправлено несколько ошибок при считывании данных с GPU.

При сборе статистики был выбран интервал в одну минуту, что позволяет сохранить данные о 300 узлах за год в наборе файлов общим размером около 20 гигабайт.

Для упрощения установки доработанной программы был подготовлен стандартный rpm-пакет.

2.2 Система обработки и визуализации данных

Для обработки собранных с узлов данных в ИММ УрО РАН был разработан двухкомпонентный анализатор-визуализатор. Анализатор получает от системы сбора данных информацию о загрузке ресурсов на узлах, а от системы распределения ресурсов кластера – информацию о том, в какое время и на каких узлах была запущена каждая задача. В процессе работы анализатор формирует вспомогательные таблицы использования ресурсов конкретными задачами и подготавливает данные в формате json для системы визуализации.

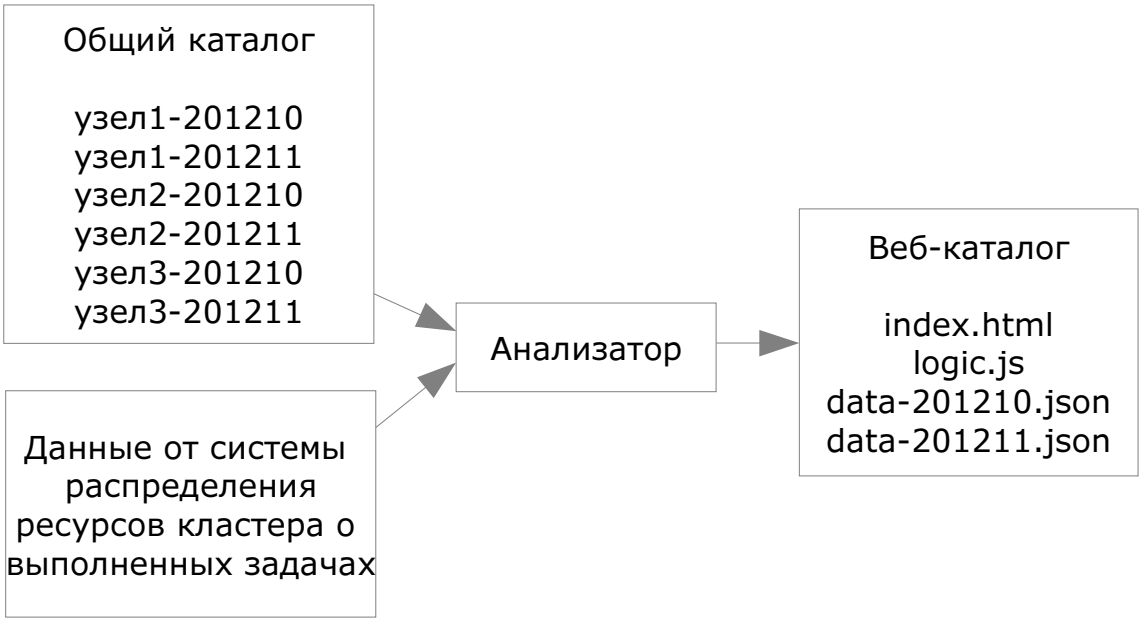

Рис. 2. Схема работы системы визуализации данных

Анализатор написан на языке Python [1]. Использование высокопроизводительного интерпретатора языка Python – PyPy – позволило в три раза ускорить работу анализатора по сравнению с запуском его в среде популярного интерпретатора CPython. Анализ реальных данных кластера «Уран» за три месяца на узле с 8 ГБ памяти и процессором 3 ГГц длится около 10 минут. Узким местом при анализе данных оказался жесткий диск, что удалось выявить с помощью самого анализатора.

Визуализатор написан на JavaScript [2, 3] и является веб-интерфейсом анализатора. Из соображений *безопасности* и предсказуемости нагрузки на сервер технологии динамической генерации страниц на стороне сервера не используются – веб-сервер отдает только статические файлы.

Вся визуализация данных происходит на клиенте с помощью генерации страницы средствами JavaScript. Для загрузки данных, ранее подготовленных анализатором в формате json, используется технология ajax.

Визуализатор позволяет просмотреть информацию о счетных задачах с разной степенью детализации. На основном экране задачи группируются по пользователям, и по каждой задаче формируется краткий ее обзор, например:

Рис. 3. Пример задачи, требовательной к памяти

В данном примере показано, что задача с именем lda_dmft.v.7.1 выполнялась на 32 узлах 20 часов; в пиковый момент на одном из узлов задача занимала 30 ГБ оперативной памяти; в среднем использовалось 19% возможностей CPU; было передано по сети всеми узлами в сумме 2.45 ГБ информации; сделано 5400 запросов ввода-вывода.

Можно сделать вывод, что задача требовательна к памяти – она использует практически всю доступную память. Запуск этой задачи на узлах с большим объемом памяти может оказаться для нее очень полезным.

При нажатии мышью на область описания задачи, можно узнать более подробные сведения по узлам:

NODE	RAM	CPU USER								CPU SYSTEM								IB	ETH	NFS READ	NFS WRITE
umt10	13.6	0	100	0	100	99	0	0	0	0	0	0	0	0	0	0	0	0.02	0	0.284	0.029
umt100	14.9	0	0	0	52	52	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.182	0.004
umt102	13.7	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.163	0
umt103	14.1	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.168	0
umt104	13.6	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.163	0
umt105	13.7	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.169	0
umt106	13.6	0	100	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.178	0.001
umt107	13.6	100	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.162	0
umt11	13.6	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0.185	0.001
umt112	13.6	0	100	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.165	0.001
umt117	15.2	100	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.16	0
umt118	14	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.161	0
umt119	14	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.165	0

Рис. 4. Подробные сведения по узлам

Видно, что задача использует два-три ядра на 100% и не использует остальные. Возможно, имеет смысл сильнее «распараллелить» задачу.

Еще несколько примеров.

Обычные счетные задачи без обмена данными между узлами:

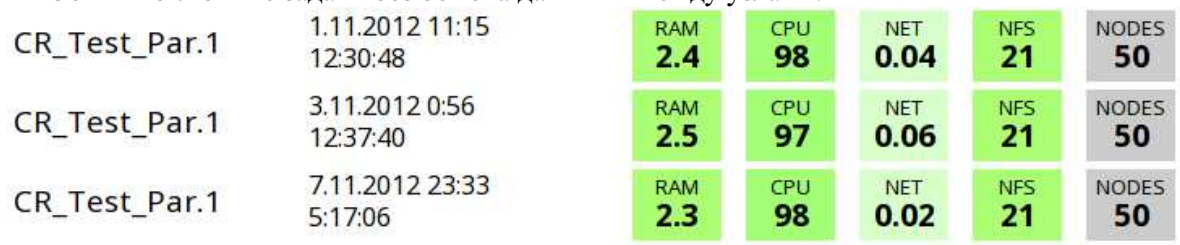

Рис. 5. Пример счетных задач

Задачи, использующие только одно ядро из доступных восьми:

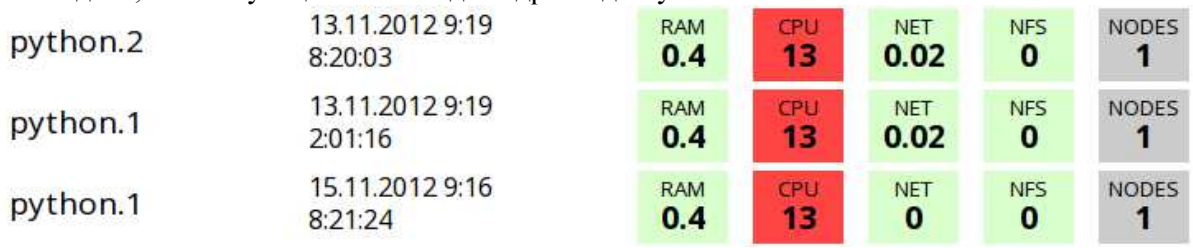

Рис. 6. Пример задач, использующих только одно ядро

Задача с интенсивным дисковым вводом-выводом:

Рис. 7. Пример задачи с интенсивным дисковым вводом-выводом

Задачи с интенсивным сетевым вводом-выводом:

		RAM	CPU	NET	NFS	NODES
rungms_imm.1	6.11.2012 17:22 3:49:41	1	40	179	2	8
rungms_imm.2	6.11.2012 18:23 1:00:01	0.8	41	79	3	8
rungms_imm.1	7.11.2012 4:54 2:16:41	1	41	132	4.2	8

Рис. 8. Пример задачи с интенсивным сетевым вводом-выводом

Восемь узлов второй задачи обменялись 79ГБ данных за час.

По каждой задаче доступны графики, показывающие, как менялось потребление ресурсов в динамике:

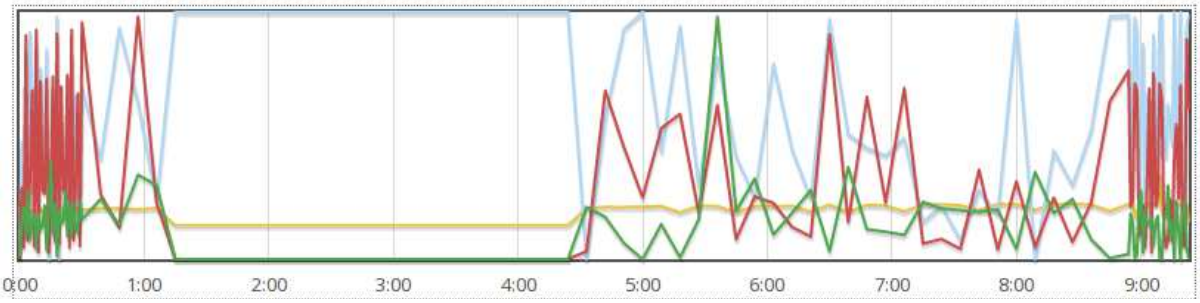

Рис. 9. Пример визуализации потребления ресурсов задачей

Бледно голубым цветом показано потребление cpu, зеленым — дисковый ввод-вывод, желто-оранжевым – потребление памяти и красным – сетевая активность.

Здесь ясно виден «счетный» участок – начиная с первого часа. Длительность этапа можно уменьшить, повысив быстродействие или количество cpu.

В среднем за месяц на кластере «Уран» запускается около 3 000 задач. Для того, чтобы посмотреть в списке только проблемные задачи, в интерфейсе предусмотрен механизм фильтрации по потребляемым ресурсам:

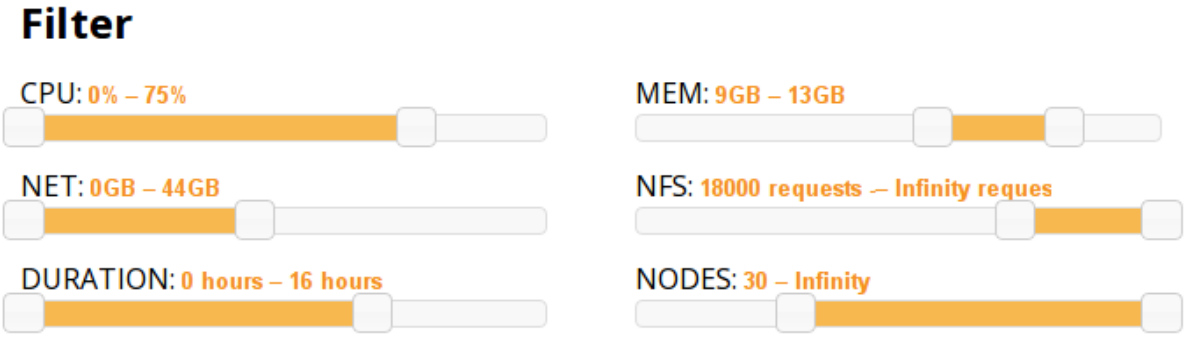

Рис. 10. Интерфейс механизма фильтрации

3. Выводы

В статье были приведены конкретные примеры того, как сбор и визуализация данных о распределенных задачах помогает определить направления дальнейшего развития кластера. Описанная система проста в установке и в использовании. Написанные в ИММ УрО РАН части доступны под лицензией GPL v2. Примеры кода доступны по адресу: https://alexbers.com/stat/src/, а визуализатор в действии – по адресу: https://alexbers.com/stat/.

Литература

1. Lutz M. Learning Python, 4th Edition. O'Reilly, 2009.
2. Bibeault B., Katz Y. jQuery in Action, Second Edition. Manning, 2010.
3. Sawyer D. M. JavaScript & jQuery: The Missing Manual. O'Reilly, 2011.
4. А.В. Адинец, П.А. Брызгалов, Вад. В. Воеводин, С.А. Жуматий, Д.А. Никитенко, К.С. Стефанов. JOB DIGEST – ПОДХОД К ИССЛЕДОВАНИЮ ДИНАМИЧЕСКИХ СВОЙСТВ ЗАДАЧ НА СУПЕРКОМПЬЮТЕРНЫХ СИСТЕМАХ//Научный сервис в сети Интернет: поиск новых решений: Труды Международной суперкомпьютерной конференции (17-22 сентября 2012 г., г. Новороссийск). - М.: Изд-во МГУ, 2012. стр.9-15.