ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ПО ДАННЫМ МОНИТОРИНГА

Д.А. Никитенко, К.С. Стефанов

Введение

Задача создания эффективного программного обеспечения является крайне актуальной в области высокопроизводительных вычислений. Однако существующие методы оценки эффективности и отладки зачастую сложны в использовании и вызывают при работе массу накладных расходов, искажающих наблюдаемую картину, и затрудняющих интерпретацию полученных результатов [1].

В данной статье предлагается подход к анализу эффективности параллельных программ на основе данных мониторинга вычислительной системы, на которой исполняется параллельная программа. Мы будет опираться на данные мониторинга, доступные при работе под ОС семейства Linux.

Описание предлагаемого подхода

Мы считаем, что программа работает эффективно, если в течение всего времени работы полностью использует выделенный ей ресурс процессора. Основным критерием эффективности работы программы мы будем считать приближение реальной достигнутой на исследуемой программе производительности во флопсах к пиковой производительности той части вычислительной системы, на которой работала программа. По нашему мнению, применение аналогичных методов возможно и к программам, основная рабочая «нагрузка» которых приходится не на операции с плавающей точкой. Например, можно «целевым» показателем считать производительность в инструкциях в секунду (IPS), однако в данной работе акцент делается именно на вещественной арифметике.

Если программа не полностью использует предоставленный ей ресурс, мы считаем, что в программе имеются недостаточно эффективно реализованные фрагменты. Низкая эффективность фрагментов отражается в особенностях данных мониторинга, которые мы в дальнейшем будем называть аномалиями. В зависимости от числа наблюдений данных мониторинга, мы делим аномалии на статические, динамические и корреляции.

Статические аномалии определяются по одному значению, сравнивая его с заранее установленным порогом. Для исключения «ложных срабатываний» можно считать аномалией только тот случай, когда условие выполняется в течение некоторого времени. Приведем примеры статических аномалий:

- пользовательская загрузка процессора меньше 95%,
- занятое место в файле подкачки больше нуля,
- объем свободной памяти равен нулю,
- количество операций вытеснения на диск больше нуля.

Динамические аномалии определяются путем сравнения следующего значения какого-то датчика с предыдущим в течение некоторого временного интервала. Это может быть «полка» – датчик в течение некоторого времени сохраняет одно значение, превышающее определенный установленный порог. Смысл – выход какого-то компонента на «насыщение». Например:

- загрузка InfiniBand сохраняется примерно на одном уровне и больше определенного значения,
- число дисковых операций сохраняется примерно на одном уровне и ненулевое.

Динамической аномалией является и постоянное увеличение (уменьшение) значения датчика, например, уменьшение объема свободной памяти во время работы программы, что может сигнализировать об утечке памяти. Другими динамическими аномалиями могут быть: резкий скачок значения или периодические изменения значения какого-то датчика.

Последний класс выделяемых аномалий — это зависимости между значениями разных датчиков, которые мы условно их называем «корреляции». Примеры корреляций:

- загрузка процессора упала и увеличилась дисковая активность,
- увеличилась активность ввода-вывода и появилась «полка» на графике загрузки интерконнекта.

Не все аномалии представляют собой признаки неэффективности программы. Например, сохраняющееся количество дисковых операций может быть необходимо для нормальной работы (хранение больших объемов временных данных), но в любом случае это повод обратить внимание на соответствующие конструкции.

Данные мониторинга поступают от агентов, работающих на узлах вычислительной системы. Агент имеет набор датчиков, каждый из которых выдает значение, характеризующие какой-либо параметр работы системы. Эти значения выдаются с определенной периодичностью. Анализ потока данных от узлов, на которых работает программа вместе с данными от компонент, общих для всей вычислительной системы (файл-серверы, коммутаторы и т. п.) позволяет выявлять свойства параллельной программы.

Под причиной низкой эффективности программы мы понимаем особенности ее структры или алгоритма, приводящие к неполному использованию предоставленных программе ресурсов вычислительной

системы. Эти причины во время работы программы находят отражение в виде признаков низкой эффективности, то есть особенностей данных мониторинга, которые сигнализируют о возможных причинах низкой эффективности.

Для анализа причин низкой эффективности программ разработано дерево принятия решений, которое приведено на рис. 1

Рис. 1. Дерево принятия решений о причинах неэффективности исследуемой программы

Анализ начинается с рассмотрения занятости процессора выполнением пользовательских задач (показатель cpu user time ядра Linux). Если занятость процессора все время работы задачи близка к максимуму (100%), необходимо оценить степень приближения достигнутой производительности к пиковой. Для этого могут быть использованы аппаратные счетчики событий, имеющиеся во всех широко применяемых в построении больших вычислительных систем процессорах, в том числе процессорах архитектуры x86_64 производства компаний Intel и AMD. Эти счетчики могут показывать количество выполненных операций с плавающей точкой, количество обращений в память, количество промахов при обращении в кэш-память разных уровней и т. п. В данном случае нам нужны сначала счетчики операций с плавающей точкой. В случае, если производительность во флопсах, измеренная по этим счетчикам, не достигает заданного порога, то надо искать причины уменьшения этой производительности. Для этого надо оценивать работу с памятью, в частности кэш-промахи, количество операций чтения-записи в память и т. п. Кроме того, не все операции с плавающей точкой конвейеризуются, что тоже может служить причиной снижения производительности. Например, для процессоров Intel не конвейеризуется операция извлечения квадратного корня.

Отметим, что установление здесь правильного порога — задача нетривиальная. Этот порог будет нашей целевой оценкой желаемой реальной производительности исследуемой программы. Если мы приблизим этот порог к пиковой производительности, мы получим ситуацию, что все программы будут объявлены неэффективными (ни одна реальная программа не может выйти на пиковую производительность). Если же этот порог будет занижен, то мы можем оказаться в ситуации, когда мы прекращаем оптимизацию программы, в то время как потенциал для оптимизации еще остается.

В случае, если пользовательская загрузка процессора не достигает максимума, необходимо определить, что именно не дает загрузить процессор полностью. Для этого надо рассматривать другие показатели мониторинга. Низкая загрузка может определяться наличием подкачки, обменов с диском, сетевых обменов и т. п.

Большой уровень других типов загрузки процессора (системная загрузка, ввод-вывод, прерывания и т. п.) указывает, что процессор загружен какой-то другой деятельностью. Это могут быть слишком частые переключения контекстов (например, при частом обращении к сервисам ядра или между нитями программы) либо же каким-то коммуникациями, при которых используется активное ожидание.

Коммуникации мы подразделяем на файловые и сетевые. Файловые — это операции с файлами, которые могут в свою очередь быть расположенными на сетевых файловых системах и вызывать еще и сетевые обмены. Сетевые коммуникации — коммуникации MPI, TCP и т. п., инициированные непосредственно пользовательской программой (а не косвенно через сетевые файловые системы).

Файловые операции с локальными дисками оцениваются по среднем размеру операций (мониторинг может выдавать число операций и суммарный размер данных). Много мелких операции может стать причиной неэффективной работы с диском. Также играет роль тип доступа: последовательный или случайный. Это оценивается косвенно, по среднему размеру данных, передаваемых за одну операцию общения с диском, и количеству операций, если они вышли на «полку». Для случайного доступа характерен средний размер данных в одной операции, примерно равный размеру блока файловой системы, при последовательном вводе-выводе средний размер будет больше. Для одиночных современных дисков количество случайных операций ввода-вывода составляет величину до нескольких сотен в секунду. Более подробно о методике расчета можно прочитать в [2].

Файловые операции на сетевых файловых системах анализируются аналогично, однако определить порог «насыщения» общего файл-сервера гораздо труднее, так как его архитектура может быть неизвестна или сложна для анализа. Кроме того, на производительность общих файловых систем влияют другие задачи,

работающие одновременно с исследуемой, поэтому анализ возможной неэффективности при работе с общими файловыми системами должен проводиться с привлечением данных мониторинга файл-серверов или на основе множественных запусков.

Анализ сетевых коммуникаций должен учитывать средний размер передаваемых сообщений. Множество мелких сообщений влечет повышенные накладные расходы на их обработку, что может сильно снижать эффективность параллельной программы. Кроме того, выход показателей коммуникаций на «полку» вместе со снижением загрузки процессора является признаком потери времени на коммуникациях.

Ненулевой размер файла подкачки (или активность подкачки) свидетельствует о проблемах при работе с памятью. Это может говорить об утечках памяти или о выделении слишком большого объема памяти. Утечки памяти, как правило, приводят к постепенному увеличению занятой области в файле подкачки (уменьшению объема свободной оперативной памяти) в процессе работы программы. Единовременное выделение большого объема дает более быстрые изменения в объемах подкачки и занятой памяти (реальное увеличение объема занятой памяти и места в файле подкачки происходит не в момент выделения памяти, а в момент первого обращения, поэтому изменения параметров мониторинга могут оказаться растянуты по времени).

Эксперименты по применению предлагаемого подхода.

Для проверки предлагаемого подхода в НИВЦ МГУ реализуется система мониторинга, способная работать на суперкомпьютерах, имеющих десятки тысяч ядер. Основным инструментом исследования потенциала сверхмасштабируемости программ является информация мониторинга во время работы программы.

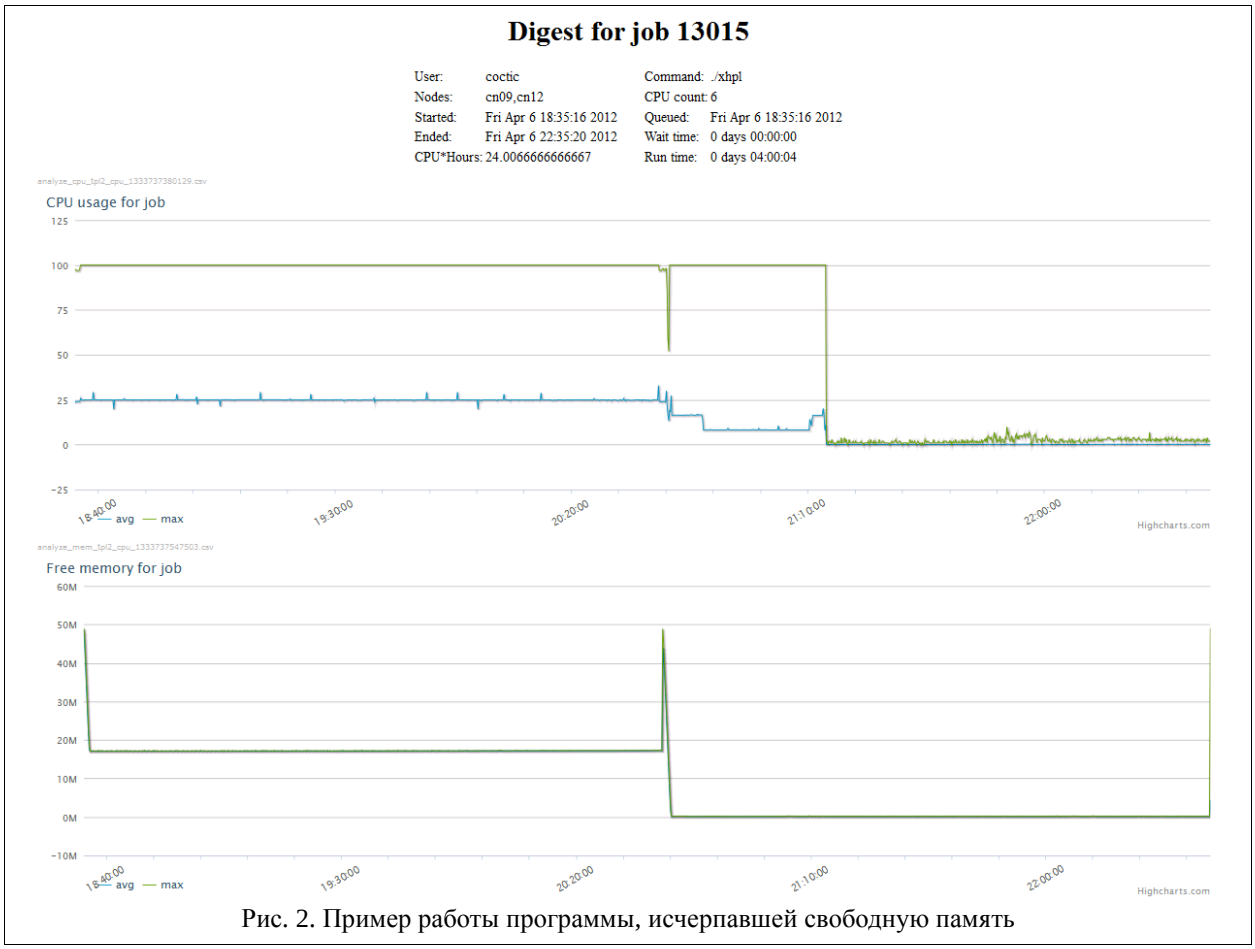

Рис. 2. Пример работы программы, исчерпавшей свободную память

На рис. 2 показан пример работы программы, которая в процессе работы уменьшила загрузку процессора (статическая аномалия — загрузка процессора меньше 100%). График свободной памяти показывает, что в этот же момент количество свободной памяти уменьшилось до нуля (статическая аномалия — объем свободной памяти упал до нуля). Одновременный анализ обоих графиков дает аномалию типа «корреляция» — уменьшение загрузки процессора при одновременном исчерпании сводобной памяти. Таким образом, в данном случае программа начинает работать неэффективно из-за слишком больших требований по памяти. При этом изменение объема памяти происходит скачком (динамическая аномалия), т. е. программа в процессе работы запрашивает сразу слишком много памяти.

Другой пример приведен на рис. 3.

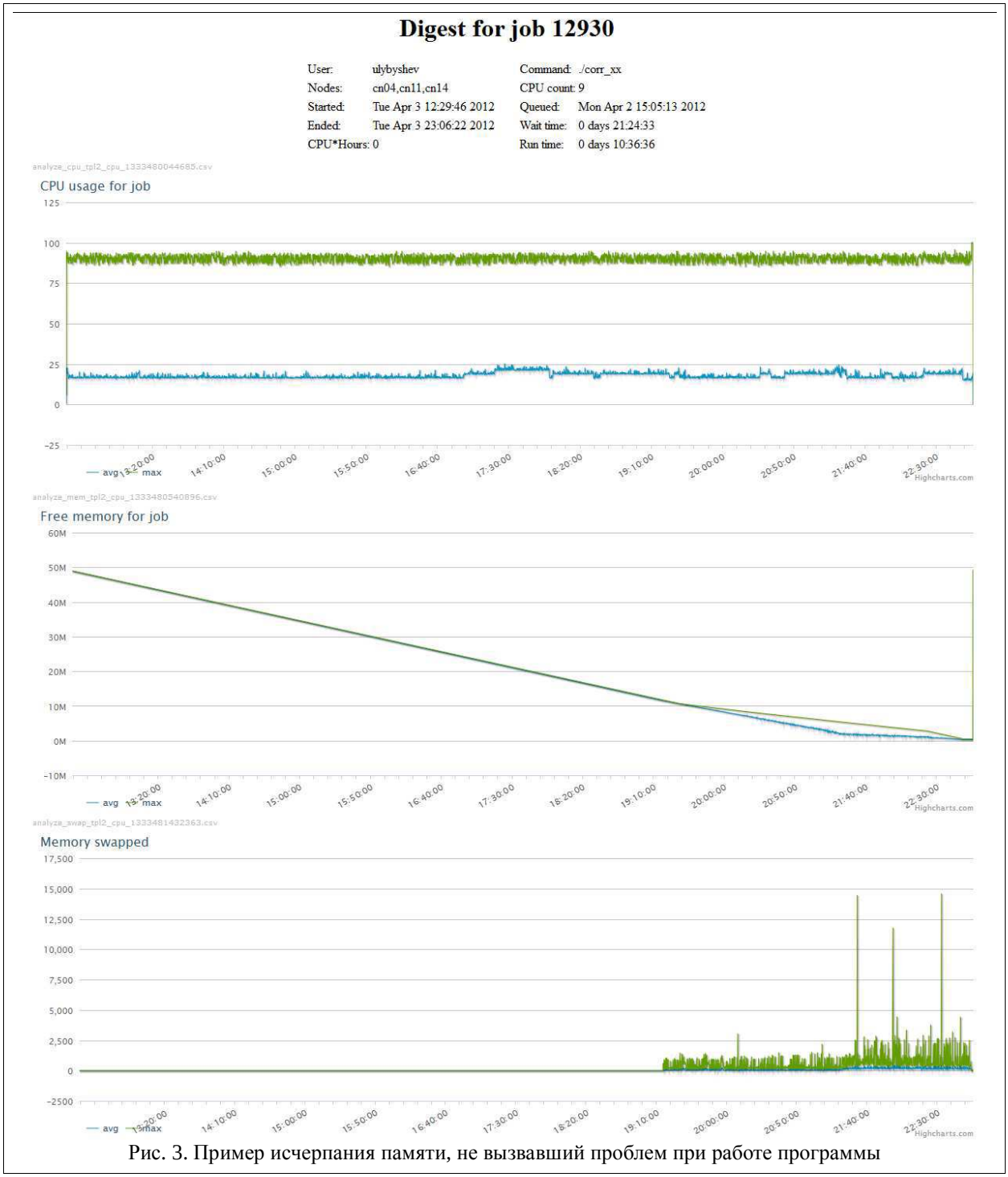

Рис. 3. Пример исчерпания памяти, не вызвавший проблем при работе программы

Как видно, в этом случае также была исчерпана свободная память (статическая аномалия), и даже начались операции подкачки (также статическая аномалия). Однако это не уменьшило загрузки процессора, из чего можно сделать вывод, что в область подкачки вытеснялись те области памяти, которые не содержат данных, активно используемых в настоящий момент. Возможно, в программе имеет место утечка памяти (на это указывает динамическая аномалия — постоянно уменьшающийся объем свободной памяти), то есть выделенные области памяти не помечаются как свободные, но обращений к ним больше не происходит. В данном случае это не привело к проблемам с производительностью, но при увеличении времени работы этой программы такие проблемы возможны (в область подкачки начнут вытесняться активно используемые данные или же место в области подкачки тоже исчерпается).

Кроме того, анализ этих двух примеров показывает, что при максимальной загрузке процессора (верхний график), близкой к полной, средняя загрузка процессора (средний график) при этом составляет 25% и менее. То есть часть процессорных ядер простаивала. В данном случае это объясняется настройками системы

очередей, которая по умолчанию выделяет под пользовательские процессы 3 ядра из установленных на каждом вычислительном узле 12 (два шестиядерных процессора).

Анализ серии запусков производится аналогично анализу одного запуска. При этом анализируются аномалии, возникающие при увеличении количества процессоров, на котором работает исследуемая программа. Появление таких аномалий указывает на проблемы с масштабируемостью при использовании соответствующего ресурса. Например появление подкачки при увеличении числа процессоров свидетельствует о том, что требуемый объем оперативной памяти становится больше доступного, и необходимо уменьшить ее потребление.

Заключение

Первые эксперименты по применению предложенного подхода показали его перспективность и возможность получать нетривиальные выводы о поведении программ. Серьезной задачей на будущее является разработка эффективных алгоритмов анализа данных мониторинга с учетом времени, необходимого для обработки поступающего объема данных. Дальнейшим направлением работы является увеличение экспериментальной базы и совершенствование данного подхода, как уточнение связи данных мониторинга с проблемами эффективности в реальных программах, так и введение в рассмотрение новых датчиков.

Работа выполнена при финансовой поддержке Министерства образования и науки Российской Федерации, ГК № 07.514.11.4017.

ЛИТЕРАТУРА:

1. Walden, J., Messer, A., and Kuhl, A. Idea: Measuring the Effect of Code Complexity on Static Analysis Results. In Proceedings of ESSoS. 2009, 195-199.
2. How to calculate IOPS for your storage system? .— http://www.techshell.org/2010/04/11/calculate_iops/