Глава 7.
Пользователь в кластерной среде

Создание кластерной системы – это не цель, а только шаг к решению реальных задач. После установки узлов, настройки ОС, средств удаленного доступа и параллельного программирования, кластер готов к работе. В принципе, пользователи уже могут заходить на него со своих рабочих мест, компилировать и запускать программы. Но вот будут ли они его использовать? Окажется ли данный инструмент удобным и эффективным, станет ли он востребованным вычислительным сообществом? Сложные вопросы, но от них в конечном итоге и зависит успех проекта в целом. Программа-минимум предполагает создание комфортной и полноценной среды для работы пользователей, установки набора вспомогательных программ и систем. Развернутый вариант включает профессиональный консультационный центр, создание службы поддержки работы пользователей, систему их обучения и многие другие шаги. Если использование кластерной системы стало неотъемлемой частью работы пользователей, если кластер прочно занял место в цепочке получения новых результатов, то только в этом случае проект можно назвать успешным. Попробуем наметить некоторые шаги в этом направлении.

Вопросов у пользователей возникнет много, и к этому нужно приготовиться сразу. Будут вопросы “периода роста”, как правило, они простые и быстро заканчиваются. Сложнее отвечать на содержательные вопросы, связанные с тонкими характеристиками работы параллельных программ. Нам приходилось работать в рамках многих проектов на большом числе параллельных компьютеров, и практически всегда перед нами стояла задача – помощь в создании действительно эффективных параллельных программ. Интересно то, что почти всегда все множество

потенциальных проблем и вопросов пользователей сводилось к двум формулировкам: “Что-то не так с моей программой” либо “Это плохой компьютер, предыдущий был лучше”. Как мы увидим, жизнь намного богаче, чем столь категоричные заявления, просто истинная причина низкой эффективности не так очевидна и кроется где-то в другом месте.

Иногда проблема снималась легко – достаточно было, например, разобраться в документации по работе с системой или компилятором. В некоторых случаях приходилось довольно глубоко вникать в суть решаемой задачи. Чем дольше мы работаем в этой области, тем отчетливей проявляется многогранность исходной проблемы. Как и для любых параллельных компьютеров, речь должна идти о **создании целостной инфраструктуры, сопровождающей большинство прикладных работ на кластерных системах**.

Давайте посмотрим, на какие составные части может разбиваться исходная проблема пользователя, и почему нужен комплексный подход к анализу ситуации в каждом конкретном случае. Практика показала, что если “с программой что-то не так”, то вопросы могут возникать на самых разных уровнях:

АНАЛИЗ КОНФИГУРАЦИИ КОМПЬЮТЕРА
↓
АНАЛИЗ ЭФФЕКТИВНОСТИ СИСТЕМНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
↓
АНАЛИЗ СТРУКТУРЫ ПРИКЛАДНОЙ ПРОГРАММЫ
↓
АНАЛИЗ АЛГОРИТМИЧЕСКОГО ПОДХОДА

Начинать исследование приходилось с анализа конфигурации конкретного компьютера: тип и число процессоров, уровни и объем памяти, параметры и топология коммуникационной среды, особенности

организации ввода/вывода и т.п. Даже эти, казалось бы, простые и очевидные понятия, на практике могут оказаться причиной вопросов и недовольства пользователей.

Сильно увлеченный прикладной областью пользователь до поры до времени может и не знать, что число процессоров для запуска пакета, если не задано явно, по умолчанию берется из конфигурационного файла. Запускает задачу, получает результат, все нормально. Перешел на другой кластер, запускает такой же вариант, время получения ответа увеличилось в два раза, хотя “в программе он ничего не менял, а процессоры такие же”. Все верно, и программа не испортилась, и новый кластер не хуже, разница лишь в числе процессоров, используемых пакетом по умолчанию на каждой из этих кластерных систем.

Похожие вопросы вызывает изменение структуры памяти, сильно влияющей на эффективность работы программ: объем и частота работы оперативной памяти, количество уровней кэш-памяти и объем каждого уровня, возможность разделения какого-либо уровня несколькими ядрами или процессорами.

Еще одна ситуация. Человек приходит и говорит, что его программа стала “неожиданно” работать медленнее, почему? Стали разбираться, и оказалось, что раньше он работал на кластере, у которого было по два жестких диска на узел, а у нового кластера на каждом узле установлено лишь по одному. На всех узлах кластеров по два процессора, все процессы приложения исключительно интенсивно работают с локальными дисками. На узлах первого кластера потоки ввода/вывода от двух процессоров бесконфликтно разводились по разным дискам, а на втором кластере единственный жесткий диск на каждом узле становился узким местом и сдерживал работу процессоров. Пользователь знал об этой особенности своей программы, но на конфигурацию кластеров не обратил внимания.

Другой пользователь, но проблема та же: медленная работа программы. Стали разбираться, и оказалось, что человек изменил всего

лишь один параметр программы, но не учел, что при этом значительно увеличились области памяти, выделяемые под массивы данных. В результате программа перестала помещаться в оперативную память, появился отсутствующий прежде свопинг, из-за чего время работы программы выросло на порядок.

Почему латентность в моей программе получается намного выше той, что мне обещали, с моей программой что-то не так? Оказалось, что пользователь ориентировался на характеристики коммуникационной сети нового поколения, которая уже была анонсирована, но реально еще не установлена. С программой было “все так”, и в рамках существующей конфигурации она работала идеально. Одновременно заметим, что негативного осадка этот вопрос не оставил, напротив, порадовал весьма высокий уровень подготовки пользователя, который отслеживает столь тонкие параметры и может задавать такие вопросы.

Обсуждая подобные случаи, мы не ставим своей целью обвинить пользователя в некомпетентности, в нежелании или неспособности разобраться с возникающими проблемами. Задача обеспечения эффективного выполнения программ сложна и многогранна, в расчет нужно принимать весьма тонкие свойства программно-аппаратной среды кластера, для исследования которых у пользователя просто может и не быть нужного инструментария. Посмотрим на данные мониторинга динамики выполнения параллельной программы, показанные на рис. 7.1.

Классическая ситуация. Сначала задача на всех узлах считается с максимальной эффективностью: левые области всех картинок верхнего ряда закрашены полностью, что говорит об использовании процессора на 100%. Затем в некоторый момент эффективность резко падает. Начинаем исследовать причины: одновременно с падением эффективности видим усиление свопинга (третий ряд сверху) на фоне отсутствия каких-либо иных процессов, которые могли бы мешать работе программы (второй ряд сверху, значение параметра `loadaverage` равно 2, т.е. числу процессоров на узле) и отсутствия обменов в сети (нижний ряд). После выполненного

анализа становится понятным источник проблем: недостаток оперативной памяти.

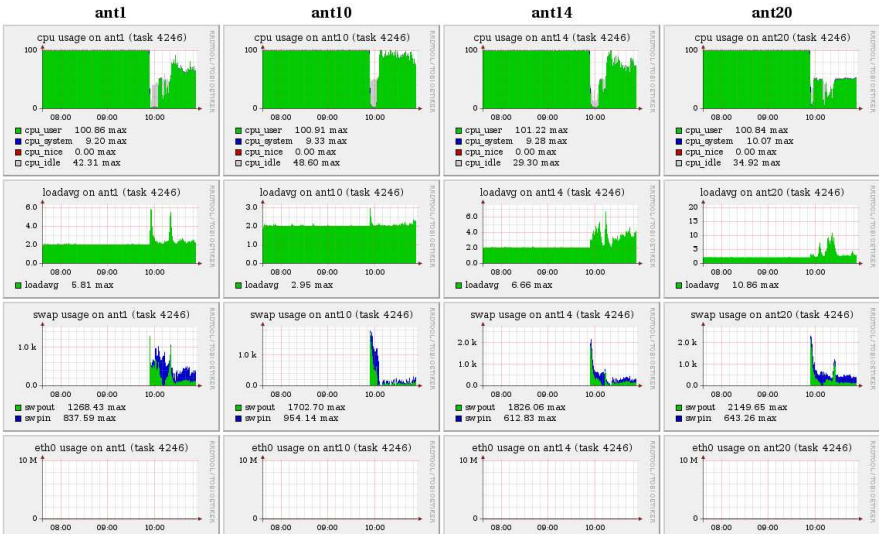

Рис. 7.1. Снижение эффективности выполнения программы из-за недостатка оперативной памяти и активизации свопинга

Увы, несмотря на очевидную полезность, с такими данными, как правило, работает только системный администратор, пользователю они редко бывают доступны. Рассмотрим еще два примера использования данных системного мониторинга характеристик программно-аппаратной среды кластера.

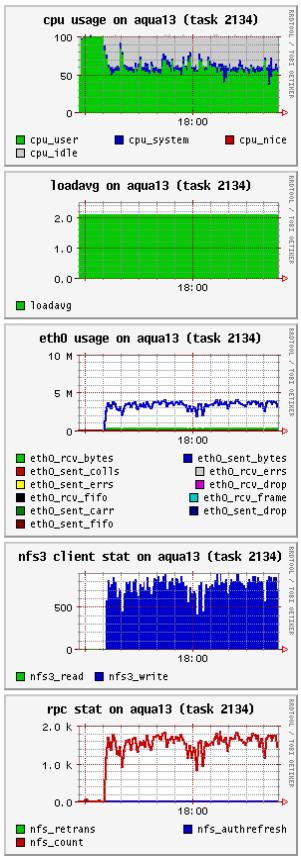

Рис. 7.2. Снижение эффективности работы программы из-за регулярного появления посторонних процессов

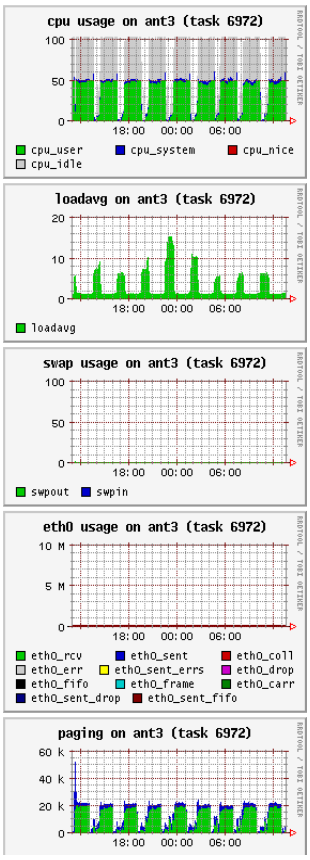

Рис. 7.3. Падение эффективности выполнения программы из-за медленной работы сетевого диска

Весьма поучительный пример для администраторов кластерных систем показан на рис. 7.2. Из него ясно следует, что эффективность работы программ определяется не только квалификацией пользователей, но и аккуратной настройкой системного окружения администраторами. Более того, пользователям разобраться самостоятельно в ситуациях подобного рода почти невозможно. По серому цвету верхней половины верхней диаграммы видно, что двухпроцессорный узел используется лишь наполовину, один процессор все время простаивает – на узле выполняется однопроцессорная задача. Работа другого процессора (нижняя часть верхней диаграммы) регулярно прерывается, причем с той же периодичностью резко возрастает как значение параметра `loadaverage` (вторая диаграмма сверху), так и активность обмена страницами оперативной памяти (нижняя диаграмма) при отсутствии сколько-нибудь заметного использования области свопинга (вторая диаграмма снизу).

В системе явно порождаются новые более приоритетные процессы, захватывающие процессор на значительное время. Детальный разбор показал, что причиной такой ситуации стала ошибка в одной строке таблицы программы `cron`, в результате которой трудоемкая системная операция, обычно запускаемая раз в неделю, выполнялась намного чаще.

Падение эффективности работы программы, которое показано на рис. 7.3, вызвано другими причинами. С параметром `loadaverage` никаких проблем нет, его значение всегда около 2. Зато резко возрастает активность работы с сетевым диском (вторая диаграмма снизу, `nfs_client`), и пользовательский процесс все время проводит в ожидании завершения операций записи на диск. Кстати, плохо организованная работа с сетевыми дисками очень часто является причиной низкой эффективности программ. Причем слова “плохо организованная работа” могут относиться и к архитекторам кластерного проекта, и к администраторам, и к пользователям. Архитекторы могли бы учесть специфику будущих задач и предусмотреть для передачи файлов, скажем, не Gigabit Ethernet, а

подумать об использовании намного более эффективных решений, например, на основе Panasas. Администраторы могли бы проверить эффективность установки NFS на своей системе, воспользовавшись уже существующим набором тестов, и изменить значения ключевых параметров, выставляемых автоматически по умолчанию в некоторые средние значения. Пользователь вполне мог бы задуматься о том, что на кластере есть как сетевые, так и локальные диски.

Сетевые диски разделяются между всеми пользователями, доступны через сетевую файловую систему и, как правило, физически расположены на отдельных устройствах. Все это приводит к дополнительным задержкам, и работа с локальными дисками вычислительных узлов проходит намного быстрее. В нашей практике был случай, когда в программе изменили лишь путь к каталогу для размещения временных файлов, за счет чего время работы программы уменьшилось в 15 раз: исключили взаимодействие с сетевым диском, а всю активность перенесли на локальные диски узлов.

Следующий большой срез – это анализ эффективности системного и прикладного программного обеспечения. Отчасти мы его уже затронули, обсуждая аккуратную настройку сетевой файловой системы, но спектр возможных вопросов намного шире. У многих в процессе работы на компьютере были нарекания на работу штатных компиляторов, что можно отнести к проблемам этого же уровня. Просмотрите документацию к компилятору, попробуйте подобрать оптимальный набор ключей и опций для его работы на данной платформе, а если есть такая возможность, то поэкспериментируйте с альтернативными вариантами компиляторов. Очень полезно посоветоваться с системными администраторами, которые помогут решить и ряд смежных вопросов. Один из пользователей самостоятельно установил в свой домашний каталог прикладной пакет, который по умолчанию вызывал стандартный, но совсем не оптимизированный вариант MPI. Естественно, что характеристики работы программы были ужасны, однако ситуация была полностью исправлена за

три минуты внесением одной строки в конфигурационный файл пакета. Проблема с программой? Да нет, опять-таки не в самой программе дело... Компьютер плох? И этого нельзя сказать.

Быстро найти причину бед пользователей удается далеко не всегда, иногда приходится проводить аккуратный детальный анализ характеристик программного окружения. В одном проекте при переходе на новый компьютер программа явно замедлилась, хотя никаких изменений в ее текст не вносилось. Причина была найдена достаточно быстро: чрезмерно большие задержки при выполнении операции барьерной синхронизации процессов. Почему? Оказалось, что на систему поставили экспериментальный вариант реализации MPI, в которой каждый процесс, доходя до точки синхронизации, выполнял следующие действия. Если он оказывался последним, и все ждали только его, то процесс рассылал всем уведомление о своем приходе и продолжал выполнение дальше. В противном случае он “засыпал” на достаточно продолжительный промежуток времени, после которого проверял, двигаться ли дальше или опять “заснуть”. Время на “сон” процессам было отведено достаточно большим, и ничего хорошего от такой реализации нельзя ожидать.

Важно осознать и то, что решить самостоятельно проблемы этого уровня пользователь, как правило, не может: знаний-то может и хватает, но прав что-либо менять в установках системы маловато. Здесь он должен работать в тесном контакте с системными администраторами, которые, в свою очередь, должны внимательно прислушиваться к вопросам и пожеланиям своих пользователей.

Основная задача в рамках анализа структуры прикладной программы состоит в поиске ответа на вопрос: “Можно ли не меняя алгоритма улучшить эффективность работы программы?”. Алгоритм может обладать всеми нужными свойствами, в частности достаточной степенью параллелизма или локальностью использования данных, однако форма записи программы может скрыть эти особенности, и компилятор не сможет сгенерировать эффективный код. Нужно изменить структуру программы,

сделать ее особенности явными и видимыми для компилятора, для чего в распоряжении есть целый арсенал приемов и методов эквивалентного преобразования программ. Занимаясь исследованиями в данном направлении в течение двух десятков лет, мы разработали и используем на практике комбинацию методов статического и динамического анализа, реализованных в экспериментальной системе исследования структуры программ V-Ray. Эффективных вариантов решения проблем подобного сорта может быть много, в конечном итоге все зависит от личного опыта, предпочтений, наработанных технологий, сложившихся правил по разработке больших программных комплексов авторскими коллективами.

Проведите простой эксперимент. Возьмите программу перемножения двух плотных квадратных матриц `A=B*C` размера, скажем, `N=5000`, примерно такого вида:

```
DO I = 1, N
  DO J = 1, N
    DO K = 1, N
      A(I,J) = A(I,J) + B(I,K) * C(K,J)
```

Информационная структура данного фрагмента позволяет выбрать любой порядок следования данных трех циклов, на правильность результата работы программы это никак не повлияет. Однако выбранный порядок сильно повлияет на время ее работы. Запустите шесть различных вариантов программы, определяемых возможными комбинациями циклов на компьютере с любым из процессоров Intel, AMD, Power, PA-RISC, и замерьте время работы. Получите шесть разных значений, причем минимальное время будет отличаться от максимального в несколько (!) раз. Слова “программа примерно такого вида” означают, что совершенно не важно, на каком языке будет записана программа: C, Pascal, Fortran или на каком-то ином, подобный эффект будет наблюдаться для каждого из них.

И, наконец, если анализ приложения показал, что структура программы не соответствует особенностям архитектуры компьютера, то проблемы эффективности исходной программы кроются в свойствах используемых алгоритмов. Единственное, что остается – это перейти к

алгоритмическому анализу. Возможно, что в результате исследований этого этапа пользователю придется поменять алгоритм и полностью переписать некоторые части программы. В ряде случаев другого пути просто нет, к этому нужно быть готовым. В идеальном варианте **разработка программы для кластера должна сразу проходить с учетом его архитектуры**, тогда недоразумений со свойствами алгоритмов не возникнет.

Как мы видим, проблема анализа и повышения эффективности работы параллельных программ является комплексной. Именно поэтому мы говорим не о каком-то одном средстве, системе или методе для ее решения, а о целой инфраструктуре. В полной мере это касается и всего спектра программного обеспечения. Понимая сложность проблемы и острую необходимость в разного рода вспомогательных инструментах, к настоящему моменту разработано немало полезных систем, входящих в состав штатного программного обеспечения компьютеров. Компиляторы, отладчики, анализаторы, конверторы, профилировщики – всем этим богатством нужно уметь пользоваться, и нужно учить пользоваться, если есть желание получать реальный эффект от уникальных возможностей современных кластерных систем. Насколько это реально, насколько развит программный сервис на доступном кластере – это вопрос к сервисной службе и администраторам каждой конкретной кластерной системы.