Решение задачи устранения дефокусировки и смаза на сервис-ориентированной распределенной вычислительной системе

А.В. Гаврилов, И.И. Доровских, И.Д. Красинский

Обработка большеформатных изображений относится к числу задач, требующих значительных вычислительных ресурсов. Для решения таких задач является актуальным создание распределенных вычислительных систем (РВС) Определен набор требований к РВС, на основе которых предложена архитектура системы. В рамках реализации сервис-ориентированной РВС создан модуль решения задачи устранения дефокусировки и смаза на изображениях.

1. Введение

Распределенная вычислительная система (РВС) [1] представляет собой систему, обеспечивающую инфраструктуру для запуска и выполнения задач на множестве входящих в нее узлов – персональных компьютерах, объединенных сетью. В настоящее время такие системы составляют серьезную конкуренцию кластерам в силу активного распространения и роста компьютерных сетей. Кроме того, пропагандируется более полное использование имеющихся вычислительных ресурсов (мощности персональных компьютеров) не только в научных учреждениях, но и на производстве. Для РВС адаптируют алгоритмы решения различных задач: расчета напряжений в сложных механизмах, расчета микро- и нано-оптических элементов, обработки большеформатных изображений, различных задач математической физики. В большинстве своем это задачи, требующие сверхбольших вычислительных и временных затрат.

Существует несколько подходов к организации РВС. Наиболее распространенным из них является реализация РВС, специализированной для решения конкретной задачи. В этом случае способ общения между узлами вычислительной системы и организация процесса решения ограничены выбранными предметной областью, алгоритмом и средствами реализации. Второй подход – это использование уже реализованных универсальных РВС (например, Condor и X-Com), которые могут охватить решение некоторого класса задач. И, наконец, третьим и одним из самых молодых подходов является использование образцов проектирования и фрэймворков. Набор инструментов Globus Toolkit – один из представителей этой области – является широко известным и достаточно популярным.

В данной работе описано иное видение архитектуры РВС, которое было положено в основу разработанной авторами системы. Результаты реализации показывают не только возможность существования такого подхода, но и некоторые его преимущества. Также приведено описание реализации для разработанной РВС алгоритма устранения дефокусировки и смаза, иллюстрирующее простоту и эффективность рассматриваемого подхода.

2. Существующие подходы к реализации РВС

2.1 Задачи РВС

РВС является инфраструктурой для распределенных программ и должна реализовывать общие для многих алгоритмов задачи: передачу данных по сети, репликацию данных, миграцию задач, подсчет различных метрик, построение оценок вычислительной сложности поступивших задач и вычислительного ресурса узлов РВС. Кроме этого РВС должна планировать, распределять и балансировать нагрузку по узлам для оптимизации времени решения задач и минимизации простоев узлов [4-5].

В процессе работы РВС могут происходить исключительные ситуации (ИС), влияющие на выполнение вычислительных задач и функционирование РВС в целом. Возможны следующие риски: отказ вычислителя, временная потеря связи с вычислителем, потеря начальных данных для всей задачи или ее ветви, сбой работы ветви вычислительной задачи, отказ узла, на котором находятся жизненно важные для работы системы сервисы, отсутствие исполняемого кода распределенного алгоритма на вычислителе. Каждый из этих рисков влияет либо на время решения задачи, либо на принципиальную возможность ее решения.

Обеспечение жизненного цикла самой РВС является одной из главных задач РВС. В условиях нестабильной и ненадежной работы сети и вычислительных узлов РВС должна реплицировать собственные системные данные и быть способной развернуть жизненно важные сервисы на любом узле.

2.2 Подходы к реализации

Существуют технологии, позволяющие разработчику самостоятельно реализовать РВС. В частности, с использованием функциональных языков Erlang или MPI. При этом разработчик должен спроектировать архитектуру системы и реализовать всю требуемую функциональность РВС. Как правило, такого рода системы проектируются для конкретной задачи, вследствие чего являются специализированными и часто неприменимыми в других условиях эксплуатации, например с новым протоколом коммуникации.

Другой подход – использование существующей РВС, например, Condor или X-COM. Обычно такая РВС предоставляет набор базовых интерфейсов программирования приложений (Application Programming Interface, API). Следовательно, возможности разработчика ограничены предоставляемым API.

Наибольшие возможности для реализации РВС предоставляет набор инструментов Globus Toolkit. Однако Globus скорее является набором инструментальных средств, чем замкнутым комплектом модулей. Так же стоит отметить большую сложность при организации и администрировании РВС, реализованной на основе Globus.

Таким образом, при использовании существующих подходов возникают сложности адаптации РВС под конкретные условия эксплуатации.

Модульный подход проектирования архитектуры программных комплексов на основе взаимодействующих сервисов зарекомендовал себя как эффективный подход для создания масштабируемых стабильных систем. Концепция таких систем позволяет независимым разработчикам реализовывать необходимые модули, легко встраиваемые в систему и интегрируемые с уже существующими сервисами, то есть упрощает разработчикам создание новых сервисов, модификацию и обновление старых для специализации РВС для работы в конкретных условиях.

3. Экспериментальная платформа XPDC

3.1 Архитектура

При проектировании архитектуры экспериментальной платформы XPDC (eXtensible Platform for Distributed Calculations) разработчики руководствовались следующими требованиями:

1. простота разработки распределенных приложений в будущей системе (обеспечение прозрачной передачи данных между узлами, а так же быстрой адаптации алгоритмов для использования на РВС),
2. гибкость системы (возможность добавления новых и обновления существующих сервисов, предоставление пользователю системы широкого набора инструментов для решения задач).

Выделим основные задачи РВС:

1. предоставление системы хранения и передачи данных через протоколы коммуникации,
2. формирование среды для запуска и выполнения вычислительных задач,
3. поддержка репликации данных,
4. управление жизненным циклом запущенных вычислительных задач.

Как видно, вышеизложенные задачи достаточно независимы и могут решаться независимыми модулями – сервисами, согласованно взаимодействующими на каждом узле и во всей РВС в целом.

Предложенный в [6] подход так же предусматривает наличие в РВС единой информационного среды (ЕИС) – системы хранения и коммуникации данных, позволяющей отделить решение вычислительной задачи от решения коммуникационной. Все операции, связанные с хранением и передачей данных, а также коммуникацией между узлами (например, на основе событий) берет на себя ЕИС.

Таким образом, на вычислительном узле системы модули вычислительных задач будут выполняться в некоторой среде, предоставляемой узлом РВС. Такая среда называется контейнером и обеспечивает жизненный цикл вычислительных модулей, предоставляет им API РВС для получения начальных данных, коммуникаций между модулями, сохранения результата, а также позволяет мигрировать задачи в другой контейнер при отказе узла.

3.2 Реализация

По рассмотренному выше принципу построения архитектуры был разработан экспериментальный фреймворк XPDC.Slavery, позволяющий решать вычислительные задачи в одноранговой локальной сети. Это ограничение определено выбором протоколов коммуникации (сервис Connector) и не является ограничением архитектуры РВС. В качестве языка программирования был выбран язык Java, делающий возможным кроссплатформенную работу РВС.

Структурно следует выделить 3 архитектурных слоя: 1) набор сервисов Slavery, реализующих контейнер и сервисы распределенных вычислительных алгоритмов; 2) набор сервисов XPDC, реализующих ЕИС; 3) сервис-ориентированная платформа на основе спецификации OSGi.

3.3 Основные сервисы

Фреймворк XPDC.Slavery предоставляет множество сервисов, реализующих функциональность вычислительного узла РВС. Многие из них носят чисто служебные функции, не предоставляя разработчику вычислительных сервисов возможности использовать их напрямую.

Непосредственное отношение к реализации жизненного цикла РВС имеют следующие сервисы:

- сервис хранения и данных сохраняет и распределяет данные с помощью сервиса коммуникации (по запросу);
- сервис коммуникаций реализует коммуникацию между вычислительными узлами;
- сервис репликации реплицирует данные, обеспечивая их сохранность в случае возможных отказов узлов РВС;
- сервисы сбора метрик и их расчета, отвечающие за сбор, расчет и хранение метрик генерируемых системой и распределенными программами;
- сервис событий позволяет создавать и распространять события по РВС;
- сервис миграции, отслеживающий работу платформы и запущенных на ней задач, при отказе узла обеспечивает миграцию вычислительных задач на другие узлы и запускает их с текущей контрольной точки;

- сервис запуска вычислительных задач представляет базовые классы для создания вычислительной задачи при реализации распределенного алгоритма и механизмы запуска задач как изнутри системы, так и из внешних клиентов, поддерживает эффективный механизм сохранения контрольных точек для миграции задач между узлами.

Так же в платформе реализованы сервисы: распространения исполняемого кода (автоматического обновления сервисов), поддержки БД SQL Lite; и ряд других.

Особого внимания заслуживает сервисехранения и передачи данных. Он представляет собой механизм поддержки распределенного хранилища данных системы. Такое хранилище может быть использовано для унификации интерфейса обмена данными. Обмен данными с использованием хранилища происходит прозрачно и не требует знания местоположения адресата, которому предназначены данные. Отправитель сохраняет данные с некоторым строковым идентификатором, а получатель, зная этот идентификатор, может синхронно либо асинхронно загрузить данные на свой узел. В случае синхронной загрузки запросившая данные задача блокируется до тех пор, пока они не будут получены, в случае же асинхронной загрузки задача не блокируется. По окончании асинхронной загрузки генерируется локальное событие, которое может быть использовано для обработки данных по мере их поступления. После использования данные могут быть удалены. На каждом узле РВС размещается два сервиса – сервис хранилища данных и сервис реестра данных. Сервис хранилища данных активен на каждом узле РВС, в то время как сервис реестра данных активен одновременно только на одном узле РВС. Сервисы предоставляют интерфейсы взаимодействия с одной из двух таблиц служебной БД xpdc_storage. В таблице storage для каждого идентификатора данных хранятся данные, в таблице storage_registry для каждого идентификатора данных хранится набор URL хранилищ тех узлов, которые содержат копии данных, сохраненных с этим идентификатором.

Хранение системных данных внутри БД дает ряд преимуществ, в тои числе повышает скорость работы, улучшает надежность хранения, обеспечивает безопасность, предоставляет гибкие возможности запроса системной информации. В частности, БД незаменима для хранения информации о метриках системы (в связи с большим объемом этой информации).

Приведенный набор сервисов позволяет работать с широким классом распределенных алгоритмов. Проиллюстрируем реализацию распределенного алгоритма для использования на платформе XPDC на примере алгоритма обработки изображения.

4. Реализация распределенной версии алгоритма нелинейной адаптивной обработки изображения (устранение дефокусировки и смаза)

Рассмотрим реализованную по описанным выше требованиям РВС с точки зрения разработчика распределенных программ. В качестве алгоритма для демонстрации воспользуемся адаптивным алгоритмом устранения дефокусировки и смаза на изображении.

Алгоритм восстановления качества изображения [7] состоит из следующих этапов: выбор на искаженном изображении фрагментов, наиболее подходящих для формирования тестовых образцов; формирование из выбранных фрагментов тестовых образцов (компьютерное ретуширование изображений на фрагментах); идентификация параметров искажающей системы и/или восстанавливающего фильтра по отобранным и тестовым (отретушированным) фрагментам.

В [8] было установлено, что для эффективной реализации данного алгоритма необходимо проводить обработку изображения в определенном цветовом пространстве. Таким образом, алгоритм состоит из шести шагов:

1) преобразование цветового пространства;
2) поиск тестового фрагмента;
3) ретуширование тестового фрагмента;
4) построение восстанавливающего фильтра;

5) обработка изображения полученным фильтром;

6) обратное преобразование цветового пространства.

В описанном алгоритме могут быть реализованы параллельно (при этом каждой его ветке передается своя часть исходного изображения [9]) следующие шаги: 1, 2, 5 и 6.

Теперь рассмотрим переход к распределенной реализации данного алгоритма. Каждый его этап представляет собой соответствующий метод в некоторой библиотеке.

В соответствии с алгоритмом, пользователю системы необходимо создать шесть классов наследников класса Solver (предоставляется модулем вычислений), которые будут отвечать за каждый из этапов.

```
public Object perform() {
//Получение данных для алготма
      String fileType = (String) storage.load("/imageType",getSessionId());
      byte[] fragmentFile = (byte[]) storage.load("/fragment"+id, getSes-
sionId());
//Вызов метода, реализующего алгоритм
      Image fragment = new Image(fragmentFile);
      fragment.rgb2hsl();
//Сохранение результатов
      ByteArrayOutputStream bars = new ByteArrayOutputStream();
            ImageIO.write(fragment.getImage(), fileType, bars);
      storage.save("/fragment_hsl"+id, getSessionId(),bars.toByteArray());
      }
```

Рис. 1. Пример наследника класса Solver

Как видно из Рис. 1, в каждом из этих классов переопределен единственный метод, в котором осуществляются следующие действия:

1) загрузка данных для обработки,

2) вызов метода-обработчика из библиотеки,

3) сохранение обработанных данных.

Для осуществления процесса управления распределенным алгоритмом пользователю необходимо создать класс-наследник класса Director (предоставляется модулем, отвечающим за определение жизненного цикла распределенного алгоритма). Задачей этого класса является запуск на предоставленных узлах наследников класса Solver.

```
public Object perform() throws Exception {
  switch (point) {
  case INITIAL_STATE:
    slaveList = storage.load(this.getName()+"/slaveList", getSessionId());
    Image inImage = new Image(inFile);
    inImage.decomposition(slaveList.size(), decompositionType);
// Запускаем задачу на предоставленных узлах
    for (int i = 0; i < slaveList.size(); i++) {
      Task task = new RGB2HSL(getTaskName(), i, getSessionId());
      ByteArrayOutputStream bars = new ByteArrayOutputStream();
      ImageIO.write(inImage.getFragment(i), fileType, bars);
      storage.save("/fragment" + i, getSessionId(), bars.toByteArray());
      sendTask(slaveList.get(i).getUrl(), task);
    }
// Сохраняем контрольную точку
    saveControlPoint(RGB2HSL_SOlVE);
…
```

Рис. 2. Пример наследника класса Director

Для успешной миграции задачи на каждом этапе алгоритма сохранялись контрольные точки, в случае сбоя вычислительного узла сервис миграции задач переместит алгоритм на другой вычислительный узел с восстановлением всех данных из последней сохраненной контрольной точки.

Набор приведенных классов является для рассматриваемой РВС реализацией выбранного нами распределенного алгоритма.

Теперь учтем, что некоторые из этапов могут выполняться параллельно. То есть перед рассылкой задач на узлы системы необходимо произвести декомпозицию исходных данных (для рассмотренной задачи она будет заключаться в разделении изображения на фрагменты), а после завершения алгоритма получить общий результат (собрать обработанные фрагменты в общее изображение). Декомпозицию и сборку результата лучше реализовать в отдельных классах (Distributor и Combiner) по следующим соображениям:

1) эти процессы могут быть запущены асинхронно на других узлах РВС,
2) такой подход упрощает написание программы и улучшает ее восприятие.

Результат работы алгоритмов, как и все промежуточные вычисления, сохраненные при помощи сервиса хранения и передачи данных, доступны для любого участника данной вычислительной сессии. При необходимости идентификатор записи может быть опубликован для других алгоритмов, запускаемых на РВС, или данные могут быть выложены в глобальную сессию, общую для всех запускаемых на РВС алгоритмов.

Нетрудно заметить, что реализация распределенной версии вычислительного алгоритма действительно становится достаточно простой и требует малых временных затрат (в отличие от большинства классических подходов к реализации распределенных и параллельных алгоритмов)

Для реализации более сложного взаимодействия узлов в процессе решения вычислительной задачи можно воспользоваться работой с распределенными событиями, предоставляемой сервисом событий. Использование данного сервиса позволяет более гибко управлять процессом вычислений на вычислительных узлах при помощи событийного подхода.

6. Заключение

Рассмотренная задача устранения дефокусировки и смаза на большеформатных изображениях может быть эффективно решена с помощью распределенных вычислений и является характерным представителем класса задач, требующих значительных вычислительных ресурсов.

Для параллельных и распределенных вычислений традиционно используется ряд технологий, каждая из которых обладает своими преимуществами и недостатками. Одним из существенных недостатков часто является сложность разработки и понимания распределенных программ.

На основе анализа недостатков ряда существующих подходов была предложена концепция РВС, основанная на наборе взаимодействующих сервисов. Реализованный экспериментальный фреймворк XPDC.Slavery предлагает набор базовых сервисов для организации распределенных вычислений, каждый из которых может быть заменен специализированной версией, созданной сторонними разработчиками для конкретных условий эксплуатации РВС.

Наличие единой информационной системы позволяет разработчику прикладных программ не заботиться о физическом местонахождении и миграции данных. Система предоставляет быстрый и удобный доступ к данным, отвечая за решение задач поиска, доставки и их репликации.

Таким образом, использование фреймворка XPDC.Slavery снимает с разработчика распределенных алгоритмов необходимость реализации инфраструктурных задач РВС. При этом пре-

доставляются мощные средства наращивания функциональности РВС, модификации и обновления базовых сервисов.

В виде вычислительного модуля для разработанной РВС был реализован распределенный алгоритм устранения дефокусировки и смаза на большеформатных изображениях. Благодаря использованию средств контейнера реализация алгоритма крайне проста в написании и понимании.

Литература

1. Хорошевский В.Г. Архитектура вычислительных систем. – М., МГТУ им. Н.Э. Баумана. - 2005.
2. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. – М., Мир. - 1991.
3. Голуб Дж., Ван Лоун Ч. Матричные вычисления. – М., Мир. - 1999.
4. Седельников М.С. Точный алгоритм распределения набора задач по машинам вычислительной системы // Мат. Всерос. науч. конф. молодых ученых «Наука. Техника. Инновации» / - Новосибирск: НГТУ, –2004. – С. 65–78.
5. Седельников М.С. Алгоритм распределения набора задач с переменными параметрами по машинам вычислительной системы. – М, 2006 (Автометрия, №1).
6. Гаврилов А.В., Доровских И.И., Красинский И.Д. Архитектура сервис-ориентированной распределенной вычислительной системы и ее реализация на основе фреймворка OSGi // Научно-технический вестник СПбГУ ИТМО. - 2008. - №47. – С. 23-31.
7. Фурсов В.А. Идентификация моделей систем формирования изображений по малому числу наблюдений. – С., СГАУ. - 1998.
8. Зимин Д.И. Технология определения восстанавливающего фильтра и обработки цветных изображений // Компьютерная оптика. – 2005 г. - Вып. 27.- С. 170 - 176.
9. Доровских И.И. Реализация типовых алгоритмов обработки изображений на мультиядерных процессорах // Материалы седьмой международной конференции-семинара «Высокопроизводительные параллельные вычисления на кластерных системах» / Н.Н.: ННГУ, - 2007. – С. 306 - 313.