Реинжениринговая технология распределенных вычислений в локальной сети

Сапожников А.П. (sap@jinr.ru), Сапожникова Т.Ф. (tsap@jinr.ru)

Лаборатория Информационных Технологий

Объединенного Института Ядерных Исследований

Россия, 141980, Дубна, Московской обл., ул.Жолио-Кюри, 6

В середине 90-х годов в мире персональных компьютеров началось бурное развитие объектно-ориентированных технологий программирования на основе языков C++ и Паскаль. Сейчас уже невозможно представить себе солидную прикладную систему, работающую в стиле MS DOS, т.е. без современной оконной графики и "мышки". В то же время основная масса программ численных расчетов, лежащих в основе большинства прикладных систем, пришла в мир ПК еще из древней эпохи больших (по размеру!) компьютеров. Практически это означает, что программы написаны на Фортране.

Статическая природа Фортрана не позволяет ему стать инструментом объектно-ориентированного программирования. К тому же этот язык уже почти не развивается, а новые поколения программистов предпочитают C++ и Delphi. С другой стороны, трудоемкость программирования алгоритмов численных расчетов практически не зависит от используемого языка. Переписать серьезную программу с Фортрана на C как правило, способен только человек, хорошо разбирающийся в алгоритме, т.е. автор, а он скорее всего уже далек от активного программирования. Использование же конверторов вроде известного F2C [1] абсолютно не улучшает ситуацию, поскольку:

- программа как была "черным ящиком", так им и остается;
- пока не известны конверторы Фортрана в Паскаль и C++.

Поэтому можно предположить, что большие вычислительные программы, реализованные в свое время на Фортране, вынужденно останутся в своем фортранном виде по крайней мере в ближайшее обозримое время.

Другой важной особенностью современного программирования является тенденция к распределению вычислений между несколькими, в общем случае различными компьютерами вычислительной сети. Существующие стандартные технологии распараллеливания программ, такие как MPI [2,3] и OpenMP [4,5], ориентированы на разбиение вычислительной задачи на более мелкие процессы. Как правило, это разбиение осуществляется вручную при разработке или модернизации программного обеспечения. В то же время технологии объединения готовых вычислительных блоков в более крупные распределенные системы практически отсутствуют.

Ключевым моментом нашей работы является идея построения вычислительного сервера из готовых, автономно разработанных фортранных программ методом их автоматического реинжениринга, исключающего ручное преобразование исходных текстов. Этим достигается возможность интеграции старых, проверенных временем программ, созданных предыдущими поколениями разработчиков, в состав более крупных современных систем обработки информации, включающих развитые средства визуализации, базы данных и прочие механизмы человеко-машинного общения. Кроме того, такой подход позволяет привлекать к разработке вычислительных серверов опытных специалистов по численным методам, не желающих выходить за рамки привычных им инструментальных средств.

Основное отличие нашего подхода к организации распределенных вычислений от традиционного заключается в том, что речь идет не о технологии разбиения, а о технологии объединения. Эта технология по существу является реинжениринговой, поскольку основным инструментальным средством создания вычислительных серверов становится конвертор для автоматического преобразования автономно разработанных программ с целью интеграции их в более крупные распределенные системы.

Итак, цели нашей работы формулируются следующим образом:

- Ускорение счета больших вычислительных задач путем распределения работы между независимыми вычислительными серверами.

- Разработка технологии и инструментальных средств для автоматического построения вычислительного сервера из готовых программ, созданных ранее для сугубо автономного применения.
- Интеграция старых, проверенных временем программ, созданных предыдущими поколениями разработчиков, в состав более крупных современных систем обработки информации.

Предлагается достаточно нетрадиционная архитектура распределенной системы для решения больших вычислительных задач (рис.1). Система состоит из единственного клиента и одного или нескольких вычислительных серверов, работающих либо прямо на клиентской рабочей станции, либо на удаленных компьютерах, входящих в состав локальной сети. Каждый такой сервер - это пара процессов, один из которых всегда готов к диалогу с клиентом, а другой исполняет конкретную вычислительную программу. Количество серверов и их местонахождение определяются потребностями клиента. Целью сервера является решение конкретной вычислительной задачи, поставленной клиентом. В процессе решения сервер может потреблять информацию как из автономно подготовленных файлов данных, так и в диалоге с клиентом. Необходимые в ходе решения задачи операции ввода-вывода инициируются в сервере с помощью традиционных фортранных операторов READ и WRITE, но интерпретируются клиентом. Это достигается путем автоматической замены в тексте программы сервера операторов ввода-вывода на вызов интерфейсных подпрограмм. Такая замена делается программой-конвертором F2F, являющейся ядром предлагаемой технологии. Именно наличие конвертора позволяет вычислительному серверу оставаться привычной фортранной программой, но при этом приобрести новые качества.

Стоит сразу же отметить, что использование терминов “клиент” и “сервер” здесь также весьма нетрадиционно: обычно сервер – один, а клиентов – много. Поменять термины местами значило бы поступиться смыслом. Новых же терминов для обозначения этих понятий у нас пока нет.

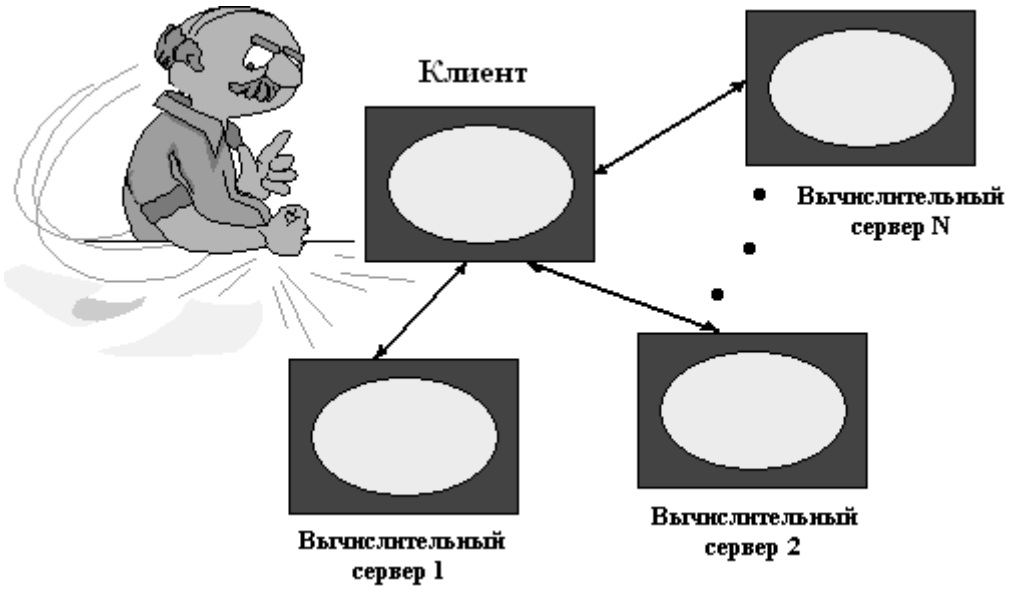

Рис.1. Организация распределенной системы для решения больших вычислительных задач.

Программа - клиент

Рабочая станция пользователя содержит программу-клиент, распределяющую работы между независимыми вычислительными серверами. Кроме запуска вычислительных серверов и интерпретации их запросов на ввод-вывод клиентская программа (и только она) осуществляет интерактивное общение с пользователем. Такая организация позволяет полностью отделить этап решения вычислительной задачи от этапа интерпретации полученных результатов, что в свою очередь позволяет использовать на каждом из этапов как наиболее подходящие инструментальные средства, так и независимые друг от друга коллективы разработчиков.

Разработан и программно реализован типовой клиент, осуществляющий стандартную интерпретацию запросов на ввод-вывод, поступающих от вычислительного сервера. Он оформлен как базовый класс, все свойства которого могут быть унаследованы конкретным клиентским приложением, порождаемым для нужд конкретной задачи. Переопределению и перепрограммированию подлежат только те части клиентского программного обеспечения, поведение которых существенно отличается от стандартного поведения базового класса. Этот базовый класс идейно

близок известному классу TThread, входящему в состав библиотек классов, распространяемых фирмами Inprise(Borland) и Microsoft [6].

Вычислительный сервер (Computational Server)

Вычислительный сервер - это программа, выполняющая большой объем вычислений при малой интенсивности ввода/вывода. Вся необходимая входная информация либо уже находится в файлах, доступных серверу, либо поставляется программой-клиентом по запросу от сервера. Вычислительный сервер строится из обычной фортранной программы с помощью инструментального средства, названного нами F2F – конвертор из Фортрана в Фортран.

Количество серверов и их местонахождение в сети определяются потребностями клиента. В частности, возможно использование сервера, выполняющегося на той же самой рабочей станции. С точки зрения клиента, работающего в среде Windows, сервер – это Automation Object, поэтому их взаимоотношения определяются правилами COM – технологии [6,7].

Разработан программный интерфейс между вычислительным сервером и программой-клиентом. Программная реализация этого интерфейса сконцентрирована в модуле ComFort.F90. Использование этого интерфейса и составляет обязательную основу предлагаемой технологии построения распределенных программных систем. Сервер организован в виде двух процессов, один из которых постоянно готов к диалогу с клиентом, а другой выполняет заказанную вычислительную работу. Диалог с клиентом сводится к приему и исполнению следующих запросов :

- Запуск заданной вычислительной процедуры (execute).
- Приостановка (suspend).
- Продолжение работы (resume).
- Принудительное завершение работы (terminate).
- Опрос времени счета (get_cpu_time).

Как видно, этот набор запросов весьма компактен, но вполне покрывает потребности любого клиента, не требуя никаких специальных действий в прикладной части сервера, на которые она и не была изначально рассчитана.

Коммуникации клиент-сервер осуществляются в рамках COM-технологии. Единицей обмена при этом является строка текста. Это позволяет локализовать все обмены в единственном месте и упрощает маршаллинг данных. При кажущейся ограниченности этого интерфейса он вполне достаточен, т.к. со стороны сервера в обмене участвует всегда только текстовая информация, а на стороне клиента необходимые форматные преобразования необременительны.

В ходе проектирования коммуникационного уровня выяснилось, что традиционная модель взаимодействия клиента и сервера, когда подавший очередной запрос клиент дожидается окончания отработки этого запроса сервером, весьма неудобна для организации именно больших вычислительных задач: одна из сторон всегда вынуждена ждать. Мы использовали практически неопубликованную в отечественной литературе методику взаимодействия с использованием механизма обратных вызовов клиента сервером. При этом серверу предоставляется возможность своевременно получить у клиента необходимую для работы информацию, а по окончании работы сервер просто заканчивает свою деятельность, освобождая машинные ресурсы. Клиент же, в промежутках между отработкой сравнительно редких обратных вызовов, может заниматься своей основной деятельностью: интерактивным общением с пользователем, например, визуализацией получаемых от сервера результатов. Такой подход позволяет клиенту взаимодействовать одновременно с несколькими независимо запущенными серверами, практически не усложняя программирование ни со стороны серверов, ни со стороны клиента.

Таким образом, возникает новая, простая и симметричная модель взаимодействия клиента и сервера, состоящая всего из двух базовых запросов:

1. RequestFromClient запускает вычислительный сервер, поручая ему конкретную работу;
2. RequestFromServer запрашивает у клиента дополнительную информацию.

Оба эти базовых запроса оперируют только текстовой информацией, что существенно упрощает интерфейс. Возникающая при этом необходимость в форматных преобразованиях для вычислительных задач не является серьезным ограничивающим фактором.

Конвертор F2F

Разработан программный конвертор фортранных программ F2F, осуществляющий преобразования исходных текстов вычислительных программ, необходимые для их работы в составе вычислительного сервера. Преобразованные программы работают в отдельном адресном пространстве, в общем случае, на удаленном компьютере. Это дополнительно повышает надежность всей системы, поскольку возможные ошибки на клиентской стороне не могут повлиять на ход выполнения программ вычислительного сервера. F2F организует необходимое для взаимодействия с клиентом программное окружение сервера. Операторы ввода-вывода в исходном тексте программы заменяются на обращения к подпрограммам, реализующим протокол взаимодействия с клиентом. В эти подпрограммы встраивается информация о выполняемом операторе ввода-вывода: текст оператора, номер его строки в исходной программе, имя текущего файла данных, участвующего в обмене, и номер записи. Это позволяет при ошибках ввода-вывода получить максимально подробную информацию о месте возникновения ошибки, поэтому преобразованная программа становится даже в чем-то лучше исходной. Кроме того, F2F заменяет операторы OPEN открытия файлов данных на вызов подпрограммы, которая может осуществлять поиск файлов не только в конкретно указанном справочнике (директории), но и в независимо заданном списке справочников. Во все операции ввода-вывода F2F встраивает перехват возможных ошибок, что позволяет исключить неконтролируемые "зависания" программ вычислительного сервера.

Целью конвертора F2F является превращение программ, предназначавшихся для взаимодействия непосредственно с человеком, в программы, взаимодействующие с процессом-клиентом и работающие в составе распределенной системы. Для обеспечения независимости конфигурации исходной программы от возможных требований конвертора F2F встраивает в преобразуемую программу определение управляющей переменной периода компиляции. Эта идея позволяет конфигурировать сколь угодно сложно программное окружение сервера, не заботясь о том, как это окружение сказывается на исходном тексте вычислительной программы, из которой этот сервер был построен. По существу F2F является транслятором с Фортрана на

Фортран. Именно F2F и является основным инструментальным средством реинжениринга в предлагаемой нами технологии.

Первоначально предполагалось, что конвертор F2F будет работать в полуавтоматическом режиме, беря на себя только основную часть рутинной работы по преобразованию операторов ввода-вывода, синтаксис которых в языке Фортран весьма сложен. В отдельных случаях допускалась помощь со стороны человека. Однако со всеми синтаксическими проблемами удалось успешно справиться. Сейчас F2F практически не требует ручной доработки получаемой им программы. Его работоспособность в автоматическом режиме проверена на вычислительных программах, состоящих из сотен тысяч строк текста.

Пример работы конвертора

```
Subroutine S          !  Вот фрагмент исходной программы:
.  .  .
Open(unit=Lun,file='myfile.dat',status='old')
Write(1),a,b,c        !  вывод двоичных данных в файл
Write(*,*),a,b,c      !  вывод на консоль
Read(*,'(i5)') n      !  ввод с консоли
.  .  .
Stop
End
```

```
Subroutine S          !  Вот во что это превращается:
Use ComFort           ! здесь весь интерфейс с клиентом
.  .  .
mess = 'subroutine S,  line 14:   Open(unit=Lun,file="myfile.dat",…'
Call Open_File( Lun, 'myfile.dat', 'old', 'formatted' )
if(fo_error.ne.0) goto 12345
mess = 'subroutine S,  line 15:   Write(1),a,b,c'
Write( 1, err=12345 )  a,b,c          !  реакция на возможные ошибки
```

```
      mess = ‘subroutine S,  line 16:   Write(*,*),a,b,c’
      Write ( iobuf, err=12345 )  a,b,c
      Call InterfaceIO ( jwrite )              ! переадресация вывода
      mess = ‘subroutine S,  line 17:   Read(*,”(i5)”),n’
      Call InterfaceIO ( jread )               ! переадресация ввода
      Read ( iobuf, ’(i5)’,  err=12345 )  n
      .  .  .
      Call Instead_Of_Stop                     ! возврат в головную программу сервера
12345 Call IOError ( mess )                    ! на выдачу сообщений об ошибках
      End
```

Перспективы дальнейшего развития

Идея автоматического преобразования исходных текстов представляется весьма плодотворной. В частности, многие преобразования текста, необходимые для интеграции с пакетом MPI, могут быть успешно выполнены конвертором F2F. Так например, во всех MPI-программах операторы ввода-вывода должны выполняться не всеми процессами, как того требует MPI-парадигма SPMD /2,5/, а практически всегда – единственным выделенным процессом. Необходимые команды пакета MPI с успехом могут быть добавлены F2F-конвертором в текст программы при переносе ее под управление пакетом MPI. Подобная автоматизация рутинных работ при распараллеливании программ, созданных для работы на одном процессоре, должна явиться важным направлением дальнейшего развития F2F-технологии.

Выводы:

Предложена и программно реализована реинженеринговая технология автоматизированного построения распределенных вычислительных систем на основе готовых, автономно разработанных фортранных программ. Существенными чертами этой технологии являются:

1. использование конвертора F2F для автоматической генерации программы вычислительного сервера;

2. построение клиентского приложения на базе программного обеспечения типового клиента.

Эта технология вовсе не требует построения серверной части системы именно из готовых программ. Если вычислительный сервер разрабатывается заново, то у разработчика появляется выбор:

- сделать программу "в старом стиле", используя привычные операторы ввода-вывода, а потом конвертировать ее;
- использовать собственные инструментальные средства с соблюдением необременительных требований интерфейса.

Весьма новой является возможность работы клиентского приложения одновременно с несколькими вычислительными серверами. Это позволяет динамично и в широких пределах масштабировать строящиеся на основе этой технологии вычислительные системы.

Важно отметить, что программная реализация всех элементов этой технологии осуществлена исключительно собственными силами, без привлечения какого-либо коммерческого программного обеспечения.

В качестве рабочей станции используется IBM PC, оснащенная MS Windows и инструментальными средствами разработки графических клиентских приложений, таких как C++ Builder или Delphi. Там же происходит начальная подготовка текстов программ для вычислительных серверов, включая и необходимое конвертирование. Эти программы, будучи машинно-независимыми, после однократной компиляции готовы к постоянной эксплуатации на серверных компьютерах, тип которых может быть практически произвольным.

Работа выполнена в Лаборатории информационных технологий ОИЯИ в рамках проекта, поддержанного грантом РФФИ 03-07-90347.

Литература:

1. S.I.Feldman, P.J.Weinberger, A Portable Fortran 77 Compiler. UNIX Time Sharing System Programmer's Manual, Tenth Edition, Volume 2, AT&T, Bell Laboratories, 1990.
2. MPI: The complete Reference. MIT Press, Cambridge, Massachusetts. 1997.
3. http://parallel.ru
4. http://www.openmp.org
5. В.В.Воеводин, Вл.В.Воеводин. Параллельные вычисления. БХВ-Петербург, 2002.
6. *Н. Елманова,* Delphi, C++Builder и COM: вопросы и ответы. *Компьютер Пресс - CD, 1999, N 7.*
7. D.Rogerson. Inside COM. MicroSoft Press, 1997.