УДК 681.3.06

ПРОТОТИПИРОВАНИЕ ПАРАЛЛЕЛЬНОЙ СУБД КАК ОСНОВА УЧЕБНОГО КУРСА ПО ПАРАЛЛЕЛЬНЫМ СИСТЕМАМ БАЗ ДАННЫХ*

М.Л. Цымблер, Л.Б. Соколинский, А.В. Лепихов
Челябинский государственный университет, Челябинск, Россия

В работе описан подход к организации учебного курса "Параллельные системы баз данных" для студентов и аспирантов, специализирующихся в области системного программирования. Основой курса является лабораторный практикум, направленный на разработку прототипа параллельной СУБД. Студенты обеспечиваются библиотекой функций прототипа, справочником по функциям библиотеки и набором контрольных тестов и должны выполнить реализацию прототипа в порядке сверху-вниз. Отладка прототипа выполняется на базе оригинального эмулятора обменов сообщениями по стандарту MPI. Данный подход успешно апробирован на Зимней школе-практикуме молодых ученых и специалистов "Технологии параллельного программирования" 2004 г. (Нижний Новгород). Материалы курса размещены в сети Интернет по адресу http://www.csu.ac.ru/pdbs.

1. Введение

Параллельная система баз данных представляет собой аппаратно-программный комплекс, способный хранить большой объем данных и обеспечивать бесперебойную и эффективную обработку большого количества параллельных транзакций.

Аппаратная компонента параллельных систем баз данных представлена многопроцессорной вычислительной системой с большим количеством процессоров и дисков, в которой процессоры объединены между собой некоторой соединительной сетью, причем время обмена данными по сети сравнимо со временем обмена данными с диском. В основу данной системы могут быть положены архитектура с разделяемой памятью и дисками, архитектура с разделением дисков, архитектура без совместного использования ресурсов, иерархическая или гибридная архитектура [1].

Программной компонентой параллельных систем баз данных является *параллельная система управления базами данных* (СУБД), обеспечивающая параллелизацию и последующую эффективную обработку запросов к базе данных, распределенной по дискам вычислительной системы.

В рамках научного проекта по исследованию моделей и методов проектирования параллельных систем баз данных авторами разработан электронный учебный курс "Параллельные системы баз данных" [http://www.csu.ac.ru/pdbs/] для студентов старших курсов и аспирантов – будущих специалистов в области системного программирования.

Основой курса является лабораторный практикум, предполагающий разработку *прототипа* параллельной СУБД – упрощенной в учебных целях версии системы. Студент обеспечивается библиотекой функций прототипа, справочником по функциям библиотеки и набором контрольных тестов и выполняет реализацию прототипа в порядке сверху-вниз (от головного модуля прототипа к модулям низшего уровня иерархии). Отладка прототипа выполняется на базе оригинального эмулятора обменов сообщениями по стандарту MPI.

Данный подход хорошо зарекомендовал себя на Зимней школе-практикуме молодых ученых и специалистов "Технологии параллельного программирования", проведенной с 25 января по 7 февраля 2004 г. на базе Нижегородского государственного университета при поддержке компании Intel.

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 03-07-90031) и компании Intel.

2. Учебный прототип параллельной СУБД

Для прототипирования нами были выбраны следующие функции, позволяющие имитировать цикл работы реальной СУБД [2]: компиляция запроса, формирование плана запроса, исполнение запроса. Прототип обрабатывает модельную базу данных, отношения которой представлены текстовыми файлами примитивной структуры.

Основной формой параллельной обработки запросов является *фрагментный параллелизм* [3]. Каждое отношение (таблица) базы данных делится на горизонтальные *фрагменты*, распределяемые по процессорным узлам вычислительной системы. Способ фрагментации определяется *функцией фрагментации*, вычисляющей для каждого кортежа отношения номер процессорного узла, на котором должен быть размещен этот кортеж. Запрос выполняется в виде нескольких параллельных процессов (*агентов*), каждый из которых обрабатывает отдельный фрагмент отношения. Полученные фрагменты *сливаются* в результирующее отношение.

2.1. Проектирование прототипа

Модульная структура прототипа параллельной СУБД приведена на Рис. 1.

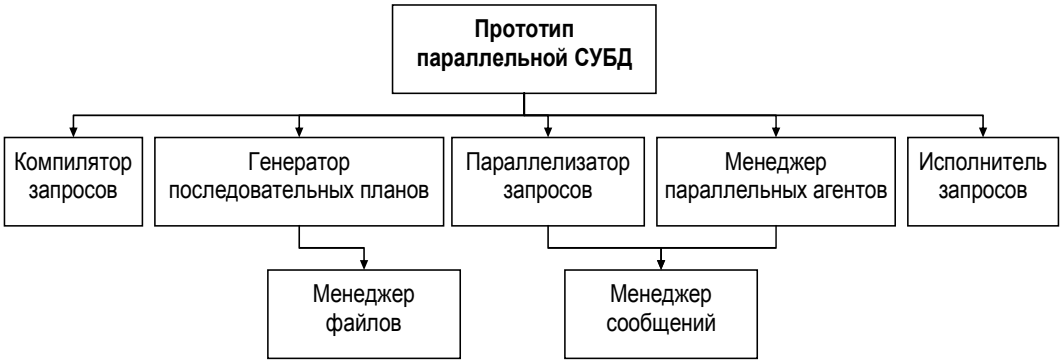

Рис. 1. Модульная структура прототипа параллельной СУБД

Компилятор запросов проверяет синтаксис запроса и формирует его внутреннее представление. В качестве языка запросов в прототипе используется язык, основанный на реляционной алгебре [4]. Внутренним представлением запроса является дерево, в котором листьями являются отношения, а узлами – реляционные операции (выборка, соединение и др.). В качестве сыновей узла выступают операнды реляционной операции, представляющей данный узел.

Для унифицированного представления узлов дерева запроса используется так называемый *скобочный шаблон*, который может получать и посылать данные и выполнять какую-либо одну реляционную операцию [5]. *Генератор последовательных планов* "вставляет" в скобочный шаблон каждого узла дерева запроса реализацию соответствующей реляционной операции.

В простейшем случае запрос параллельно выполняется на всех процессорных узлах без необходимости пересылки кортежей между процессорами. На практике, однако, во время выполнения запроса избежать пересылки кортежей не удается. Например, при выполнении операции соединения [4] отношений возникает необходимость динамически перераспределять кортежи тех отношений, которые фрагментированы не по атрибуту соединения. *Параллелизатор запросов* выполняет преобразование последовательного плана запроса в параллельный план, вставляя в соответствующие места дерева запроса специальный *оператор обмена* **exchange** [6].

Оператор **exchange** имеет два специальных параметра, определяемых пользователем: номер *порта обмена* и указатель на *функцию распределения*. Функция распределения для каждого кортежа вычисляет логический номер процессорного узла, на кото-

ром данный кортеж должен быть обработан. Параметр "порт обмена" позволяет включать в дерево запроса произвольное количество операторов **exchange** (для каждого оператора указывается свой уникальный порт обмена). Структура и методы реализации оператора **exchange** детально рассматриваются в работе [6].

Менеджер параллельных агентов по параллельному плану запроса формирует параллельных агентов для узлов вычислительной системы.

Исполнитель запросов интерпретирует (исполняет) параллельного агента на заданном процессорном узле вычислительной системы.

Менеджер файлов реализует низкоуровневые операции с отношениями базы данных: сканировать отношение, выдать очередной кортеж отношения и др.

Менеджер сообщений предоставляет функции для обмена сообщениями (кортежами) между процессорными узлами вычислительного системы.

2.2. Организация разработки прототипа

В соответствии с описанной выше структурой авторами на языке Си был реализован прототип параллельной СУБД, работающий под управлением ОС Windows. В качестве системы программирования использовалась MS Visual C++. Обмен сообщениями реализован на основе стандарта Message Passing Interface (MPI) [7] с использованием свободно доступной библиотеки MPICH [8]. Размер исходного кода прототипа составляет 7000 строк. Кодирование прототипа осуществлялось в обычном для системного программирования порядке *снизу-вверх* (от модулей низшего уровня иерархии к головному модулю).

Целью практикума является разработка студентами прототипа, аналогичного разработанному авторами. Однако при этом процесс разработки организуется кардинально иным способом.

Студенты обеспечиваются библиотекой функций, справочником по функциям библиотеки и набором контрольных тестов прототипа. *Библиотека функций прототипа* содержит объектный код функций прототипа, разработанного авторами. *Справочник по функциям* содержит снабженные перекрестными ссылками спецификации функций библиотеки в формате HTML. *Контрольный тест* представляет собой совокупность отношений модельной базы данных, запроса к ней на языке запросов прототипа, и результата выполнения запроса.

Разработка студентами прототипа проводится в порядке *сверху-вниз* (от головного модуля к модулям низшего уровня иерархии), характерном для прикладного программирования. Разработка отдельной функции из модуля выглядит следующим образом. Используя справочник по функциям, студент выполняет кодирование данной функции. Затем студент выполняет сборку прототипа с использованием библиотеки, заменив при этом соответствующую библиотечную функцию собственной реализацией. Далее осуществляется запуск полученного прототипа на контрольных тестах. В случае успеха можно переходить к разработке следующей функции данного модуля либо, если список его функций исчерпан – другого модуля. В случае неудачи следует выполнить отладку и повторять данный цикл до устранения ошибок.

Описанный подход позволяет, по нашему мнению, существенно снизить сложность разработки прототипа в рамках практикума, поскольку не требует от студента разработки тестов, тестовых программ и проведения автономного тестирования каждого модуля прототипа (что необходимо в случае разработки системы в порядке снизу-вверх). Кроме того, данный подход позволяет преподавателю по своему усмотрению расставлять акценты практикума, варьируя порядок (в рамках одного уровня иерархии) и/или количество разрабатываемых модулей и функций прототипа.

2.3. Тестирование и отладка прототипа

Запуск прототипа осуществляется с помощью специального загрузчика параллельных программ, входящего в состав библиотеки MPICH. Модификация соответствующих параметров загрузчика обеспечивает возможность запуска прототипа как на многопроцессорной системе, так и на обычном однопроцессорном компьютере. В последнем случае работа экземпляра СУБД как процесса на отдельном процессорном узле эмулируется с помощью процесса ОС Windows.

В случае отсутствия реальной многопроцессорной системы практикум можно ограничить тестированием на обычном компьютере. При наличии таковой можно дополнительно тестировать прототип на вычислительной системе, состоящей из объединенных в локальную сеть компьютеров учебного класса. Переход к тестированию на реальной системе осуществляется после успешного завершения тестирования полностью разработанного прототипа на обычном компьютере.

Сложность параллельных программ как таковых и их недетерминированное поведение превращают отладку прототипа параллельной СУБД в наиболее сложный для студента аспект разработки. В силу этого наличие отладчика параллельных программ в комплексе средств программной поддержки практикума является необходимым.

В нашем случае прямое использование отладчика среды MS Visual C++ невозможно, поскольку запуск прототипа параллельной СУБД осуществляется с помощью загрузчика. Разработка студентом собственного примитивного отладчика, реализующего простейшие функции (выдача значений переменных в заданном процессе и др.), вносит в практикум ненужную техническую сложность. Использование отладчиков сторонних разработчиков требует от студента дополнительного изучения этих продуктов, большинство из которых, кроме того, являются коммерческими (например, TotalView [http://www.etnus.com] и PGDBG [http://www.pgroup.com]). Предложенное нами решение проблемы выбора средств отладки основано на *эмуляции* параллельных процессов и обменов данными между ними с помощью стандартных средств ОС Windows.

Разработанный нами *эмулятор обменов сообщениями* представляет собой динамически компонуемую библиотеку, интерфейс которой есть подмножество функций стандарта MPI, используемое в разработке прототипа. Таким образом, эмулятор позволяет запускать прототип непосредственно из среды MS Visual C++ (без загрузчика) и использовать весь спектр средств встроенного отладчика. Переход от отладки к тестированию и обратно не требует изменения исходных текстов модулей и связан лишь с изменением параметров их компоновки.

Эмулятор обеспечивает представление процессов, запускаемых на процессорных узлах, в виде процессов ОС Windows, каждый из которых представляет собой совокупность следующих *потоков* (нитей): мастер, отправитель, получатель и терминатор. *Поток-мастер* выполняет собственно код процесса. *Поток-отправитель* обрабатывает массив, элементами которого являются очереди сообщений, передаваемых другим процессам программы. *Поток-получатель* обрабатывает очередь сообщений, поступающих от потоков-отправителей других процессов программы. *Поток-терминатор* выполняет аварийное завершение всех процессов программы, если поток-мастер одного из процессов выполнил функцию `MPI_Abort`.

Прием-передача сообщения от процесса *S* процессу *R* выглядит следующим образом (далее мы будем именовать потоки как *Master*, *Sender* и *Receiver*, а индексы имен будут указывать на принадлежность к процессу).

При выполнении функции посылки сообщения поток $Master_S$ добавляет в соответствующую очередь потока $Sender_S$ запись о данном сообщении и открывает ему семафор для начала передачи сообщения.

Поток $Sender_S$ создает в оперативной памяти процесса S область, доступную для потока $Receiver_R$, записывает в нее передаваемое сообщение и генерирует для $Receiver_R$ событие о необходимости начать прием. После этого поток $Sender_S$ переходит к ожиданию подтверждения от потока $Receiver_R$ о завершении приема. При получении подтверждения $Sender_S$ уничтожает ранее созданную область и памяти и закрывает семафор передачи сообщения.

Поток $Receiver_R$ выполняет перманентное ожидание события о необходимости начать прием. При наступлении такого события он обращается к области памяти, которую создал $Sender_S$, и считывает переданное им сообщение. После этого $Receiver_R$ высылает потоку $Sender_S$ подтверждение о завершении приема сообщения.

Разработанный эмулятор имеет самостоятельное значение и может быть использован не только в рамках разработки прототипа параллельной СУБД, но и для отладки параллельных программ, выполняющих обмен сообщениями на основе стандарта MPI.

3. Заключение

В данной работе рассмотрен подход к организации учебного курса "Параллельные системы баз данных" для студентов старших курсов и аспирантов, специализирующихся в области системного программирования.

Целью практикума является разработка прототипа параллельной СУБД. Студенты обеспечиваются библиотекой функций прототипа, справочником по функциям библиотеки и набором контрольных тестов прототипа. С помощью готовых интерфейсов и библиотеки функций прототипа необходимо выполнить реализацию модулей прототипа в порядке сверху-вниз в соответствии с модульной структурой прототипа. При проверке прототипа на контрольных тестах необходимо заменить библиотечные функции собственной реализацией.

Отладка прототипа выполняется на базе оригинального эмулятора обменов сообщениями по стандарту MPI. Процессы и сообщения между ними эмулируются с помощью потоков ОС Windows, что позволяет использовать встроенный отладчик среды программирования MS Visual C++.

Описанный подход успешно апробирован на Зимней школе-практикуме молодых ученых и специалистов "Технологии параллельного программирования" (25 января-7 февраля 2004 г., Нижний Новгород). Материалы курса размещены в сети Интернет по адресу http://www.csu.ac.ru/pdbs.

Литература

1. Соколинский Л.Б. Классификация и анализ параллельных архитектур систем баз данных // Алгоритмы и программные средства параллельных вычислений: [Сб. науч. тр.]. -Екатеринбург: УрО РАН. -2003. -Вып. 7. -С. 185-216.

2. Garcia-Molina H., Ullman J.D., Widom J. Database System Implementation. -Prentice Hall, 2000.

3. DeWitt D.J., Gray J. Parallel Database Systems: The Future of High-Performance Database Systems // Communications of the ACM. -1992. -Vol. 35, No. 6. -P. 85-98.

4. Ullman J.D., Widom J. A First Course in Database Systems. -Prentice Hall, 1997.

5. Graefe G. Query evaluation techniques for large databases // ACM Computing Surveys. -1993. -Vol. 25, No. 2. -P. 73-169.

6. Sokolinsky L.B. Design and Evaluation of Database Multiprocessor Architecture with High Data Availability // Proc. of the 12th Int. DEXA Workshop, Munich, Germany, September 3-7, 2001. -IEEE Computer Society, 2001. -P. 115-120.

7. Pacheco P. Parallel Programming with MPI. -Morgan Kaufmann, 1997.

8. Gropp W., Lusk E. Installation guide for MPICH, a portable implementation of MPI. -Technical Report ANL-96/5, Argonne National Laboratory, 1996.