Планирование запросов над множеством неоднородных распределенных информационных ресурсов в архитектуре средств поддержки предметных посредников ♣

© Вовченко А.Е., Крупа А.В.
ИПИ РАН
itsnein@gmail.com, akrupa@gmail.com

Аннотация

Планировщик запросов играет одну из важнейших ролей в архитектуре средств поддержки предметных посредников. Планировщик отвечает за декомпозицию и выполнение пользовательского запроса на множестве распределенных неоднородных информационных ресурсов. Задача декомпозиции в архитектуре посредников всегда имеет множество решений, поэтому одной из важнейших задач планировщика является выбор наиболее эффективного решения.

В статье приводится описание планировщика, который способен эффективно выполнять запросы, сформулированные над информационными ресурсами, содержащими значительные (по сравнению с пропускной способностью каналов связи) объёмы данных и требующие продолжительного выполнения. Также в статье приводится описание требований к адаптерам ресурсов, для возможности построения эффективного плана выполнения запроса.

1 Введение

В различных областях науки наблюдается экспоненциальный рост объема получаемых экспериментальных (наблюдательных) данных. Например, в астрономии текущий и ожидаемый темп роста данных от наземных и космических инструментов удваивается в течение периода от шести месяцев до одного года. Сложность использования таких данных увеличивается еще и вследствие их естественной разнородности. Число организаций, получающих данные наблюдений в отдельных областях науки в мире, велико. Разнообразие (информационная несогласованность) получаемой информации вызывается, в частности, не только большим числом организаций, производящих наблюдения, и их независимостью, но и разнообразием объектов наблюдения, непрерывным и быстрым совершенствованием техники наблюдений, вызывающим адекватные изменения структуры и содержания накапливаемой информации. Это приводит к необходимости использования неоднородной, распределенной информации, накопленной в течение значительного периода наблюдений технологически различными инструментами.

Основной идеей в инфраструктуре доступа к множественным неоднородным информационным ресурсам является введение промежуточного слоя между ресурсами, и потребителями информации. Основными компонентами промежуточного слоя являются предметные посредники [1,5], существующие независимо от информационных ресурсов. Использование предметных посредников в среде неоднородных информационных ресурсов представляет интеллектуальный подход к интеграции информации, который может быть эффективно использован при создании новых средств поддержки электронных библиотек.

Архитектура средств поддержки посредников разрабатывалась для обеспечения интеграции широкого класса информационных ресурсов, различающихся возможностями выполнения запросов. Она призвана обеспечить интеграцию разнообразных видов информационных ресурсов, включая:

- базы данных (в том числе объектные) в СУБД, обладающих возможностью создания временных коллекций;
- базы данных (в том числе и объектные) в СУБД с доступом только для чтения;
- веб-сервисы;
- простые таблицы (HTML, excel), текстовые коллекции, коллекции объектов, без встроенных средств выполнения запросов.

Система исполнения запросов в такой среде должна корректно функционировать вне зависимости от объемов содержащейся в коллекциях информации. Предполагается, что каждый ресурс может взаимодействовать с

Труды 11[ой] Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2009, Петрозаводск, Россия, 2008.

каждым. В случае если не требуется передачи данных от ресурсов к ресурсам, взаимодействие происходит только между посредником и ресурсами. Здесь рассматриваются вопросы адаптивного планирования запросов[11] в среде посредников для обеспечения ее приемлемой эффективности.

Одним из основных требований к планировщику запросов является предсказуемость времени вычисления запросов. Планировщик должен построить эффективный план и выдать для него оценку времени исполнения. Поскольку окружение, в котором исполняется запрос, непостоянно, планировщик должен следить за выполнением запроса, изменяя прогноз в зависимости от изменений в окружении. В случае существенных изменений в окружении планировщик должен уметь изменить план, не прерывая выполнения запроса, если это возможно.

В посреднике различаются два уровня представления информации: локальный уровень, представляющий метаинформацию о разнородных коллекциях, и федеративный уровень, поддерживающий глобальную метаинформацию на языке канонической модели, формируемую на основе подхода GLAV [3]. Канонической моделью данных [2] в посреднике является информационная модель языка СИНТЕЗ [6]. В качестве языка запросов в этой модели в настоящее время используется подмножество языка формул СИНТЕЗа - язык Syfs (Synthesis Formula Subset), а точнее его алгебраическая форма – Asyfs[6].

После получения запроса, переписанного в терминах локальных схем, выраженных однородно в канонической модели, планировщик осуществляет его декомпозицию во множество локальных запросов и составляет план их выполнения. Глобальный запрос формулируется в терминах федеративной схемы посредника. Локальные запросы формулируются в терминах локальной схемы для конкретной коллекции. Посредник инициирует, а планировщик должен контролировать процесс выполнения запроса согласно составленному плану.

Целью настоящей работы является обсуждение подхода к реализации эффективного алгоритма планирования в среде предметных посредников. В следующем разделе рассматривается реализованный алгоритм планирования. Затем в третьем и в четвертом разделе обсуждаются ключевые моменты алгоритма планирования. Затем в пятом разделе представлен ряд требований к ресурсам, а точнее к адаптерам ресурсов, которые должны выполняться для эффективной работы алгоритма, после чего описана архитектура разработанного адаптера ресурсов а также способности адаптеров (capabilities) в соответствии с выработанными требованиями.

2 Алгоритм планирования запросов

Описываемый планировщик является частью архитектуры средств поддержки посредников и отвечает в ней за эффективное выполнение запросов. Следующие свойства характеризуют класс рассматриваемой архитектуры:

- выполнением запроса управляет клиентское приложение, представляющее собой интерфейс взаимодействия с посредником;
- средства координации выполнения запросов между клиентами отсутствуют;
- координация выполнения запросов между адаптерами ресурсов реализуется исключительно клиентским приложением;
- планировщик взаимодействует непосредственно с ресурсами;
- ресурсы не предоставляют информации о других клиентах их использующих;
- информация о внутреннем устройстве ресурса, способе и скорости выполнения в нем операций, отсутствует.

В такой системе невозможно построить план выполнения запроса [8-10] со строгим указанием выделяемого времени того или иного адаптера как вычислительного ресурса каждому из запросов для выполнения определенной операции. Планировщик в этой системе формулирует для каждого информационного ресурса набор независимых заданий [7], которые могут выполняться одновременно, а могут быть поставлены в очередь, и передает их адаптеру. Если выполнение следующего задания требует завершения предыдущего, то планировщик должен ждать завершения первого, и только потом делать запрос на выполнение второго. Задания от разных клиентов также независимы. Адаптерам ресурсов дана почти полная свобода в выборе порядка исполнения всех заданий. В общем случае, они могут выполняться параллельно, если это позволяет ресурс. Задания могут быть выполнены не полностью, а по частям. Это позволяет адаптерам достаточно гибко планировать и распределять нагрузку и другие ограниченные вычислительные ресурсы между всеми заданиями.

Задача планировщика в системе - формулировать и раздавать задания адаптерам и координировать их выполнение в пределах одного запроса. Любой запрос, обращающийся более чем к одному информационному ресурсу, может быть разбит на задания/подзапросы более чем одним способом. Выбор того или иного варианта разбиения может существенно повлиять на время выполнения запроса. Поиск эффективного варианта разбиения – важная задача планировщика в архитектуре посредников. Рассматриваемая реализация планировщика (в системе допускается одновременная работа любого количества различных реализаций планировщиков) использует для поиска эффективного разбиения эвристические предположения и имитацию выполнения запроса [4]

на небольших подмножествах данных там, где предположения не работают. В процессе имитации измеряются такие параметры, как:

- Скорость передачи данных;
- Скорость выполнения определенных заданий;
- Время ожидания в очереди.

Для оценки таких характеристик, как, например, селективность операций выборки по условию, используются данные, полученные в результате имитации. Таким же образом оцениваются результаты выполнения операций соединения. В результате, выбирается определенное разбиение запроса на задания. В процессе выполнения запроса планировщик контролирует, соответствуют ли измерения на реальных данных оценке, полученной на этапе имитации. Результаты измерений во время выполнения могут существенно отличаться по многим причинам, например, из-за существенного изменения нагрузки на тот или иной ресурс. Если планировщик обнаружит, что зафиксированные изменения существенно влияют на ожидаемое время выполнения запроса, то он, не прерывая исполнения запроса, попытается найти более эффективный способ выполнения оставшейся его части. Таким образом, алгоритм обработки запроса выглядит следующим образом:

1. Обработка запроса:
   - Разбиение на элементарные операции;
   - Оптимизирующие преобразования на основе правил;
2. Разбиение множества невыполненных операций на задания:
   - Оценка различных вариантов разбиения на основе эвристических предположений;
   - Имитация в случае, когда предположений недостаточно;
3. Выполнение плана:
   - Выполнение операций согласно плану;
   - Контроль соответствия результатов имитации/предположений реальной ситуации;
   - Возврат к шагу 2, если различия оказывают существенное влияние на прогноз времени выполнения запроса.

Эффективность такого способа планирования запросов, в сочетании с наличием прогноза времени выполнения запроса до начала выполнения достаточна для практического применения на запросах, требующих продолжительного исполнения и использующих большие объемы данных.

При реализации планировщика, функционирующего данным способом, очень важно правильно выбрать пространство планов, в котором будет осуществляться поиск, а также четко сформулировать правила адаптации плана и условия, при которых изменение плана считается необходимым.

3 Пространство возможных планов

Языки запросов Syfs и Asys поддерживают операции Join и Union. По грубой оценке, одно это обеспечивает рост количества возможных планов для данного запроса, как O(N!), где N, это количество задействованных в запросе коллекций. На практике, оно может быть больше и зависит также от количества обращениям к функциям в запросе. В планировщике применяются два распространенных способа сужения пространства поиска.

Во-первых, все операции Union переставляются в корень запроса, т.е. запрос представляется в виде объединения нескольких подзапросов, каждый из которых не содержит операций Union. Каждый подобный подзапрос планируется отдельно.

Во-вторых, при планировании подзапросов, не содержащих операций Union, рассматриваются только те планы, которые удовлетворяют следующим условиям:

- план не должен выполнять операцию перекрестного соединения, если этого не требуется пользователю;
- правый подзапрос-операнд каждой из операций Join не должен содержать операций Join.

Перечисленным ограничениям удовлетворяют только те планы, которые допускают вычисление посредством конвейерной обработки. Рост количества планов, удовлетворяющих заданным ограничениям можно оценить как O(NI), что приемлемо при использовании на практике.

4 Итерационное выполнение

В процессе вычисления запросов, для выполнения очередной операции Join часто возникает необходимость передать данные из одного ресурса в другой. В случае наличия альтернативных ресурсов, на которых может быть выполнена операция, выбор наилучшего варианта осуществляется посредством имитации выполнения подпланов на произвольных частях реальных данных. Адаптеры ресурсов накладывают ограничения на объем данных, который они могут единовременно загружать из других адаптеров или посредника. Объем данных, которые могут быть задействованы в запросе, теоретически не ограничен (в реализации есть ограничение в 2 в 63-ей степени байт). Для обработки больших массивов данных на запросе открывается ридер, из которого данные можно брать блоками любой заданной длины. Каждый из блоков обрабатывается отдельно и независимо и проходит все последующие этапы выполнения.

Планировщик контролирует время выполнения оставшейся части плана (и каждого этапа) для каждого из блоков данных. Если в определенный момент время выполнения становится существенно больше полученного во время имитации, то для всех

последующих этапов, начиная с данного, заново запускается поиск плана, и производится переоценка. Если более эффективный план найден, то оставшиеся блоки данных будут обрабатываться в соответствие с новым планом.

Таким образом, при выполнении запросов, использующих большие объемы данных, планировщик адаптирует план под изменяющиеся условия. При выполнении запросов, для выполнения которых нет необходимости в разбиении данных на блоки, изменений плана во время выполнения не происходит. Тем не менее, на таких запросах применимы и работают оценки на основе имитации.

5 Требования к адаптерам ресурсов

5.1 Традиционная архтектура адаптера

Спецификация интерфейса адаптеров неоднородных информационных ресурсов, применяемых в настоящее время в архитектуре средств поддержки предметных посредников, включает два метода:

```
public interface Adapter
{
        long executeAsyfsQuery(Plan plan) throws
AdapterException;
        void getAsyfsResult(long id, VOClassWriter
writer) throws AdapterException;
}
```

Таким образом, осуществляется возможность выполнения запроса на информационном ресурсе, и получение результата этого запроса. Отметим основные функциональные недостатки применяемых адаптеров:

1) Не поддерживаются сессии. Поскольку адаптеры функционируют в многопользовательских интеграционных средах, наличие сессий, изолированных друг от друга, с изолированными ресурсами является необходимым.

2) Не поддерживается возможность повторного использования результата запроса. В случае, когда одни и те же данные требуются в нескольких частях плана, запрос, получающий эти данные, придется выполнять несколько раз. Подобный подход неприемлем для эффективного планирования.

3) Не поддерживается возможность управления загрузкой данных в адаптер супервизором, а также возможность материализации запросов в адаптерах. В текущей реализации адаптерам передается план, в котором содержится URI и ID результата, по которому можно забрать результат. Для эффективного планирования требуется, чтобы момент загрузки данных и их объем определялись планировщиком динамически, а не заранее построенным планом.

4) Не поддерживаются учет способностей (Capabilities) адаптеров. Ресурсы обладают различными возможностями, например способностью хранить временные данные, выполнять соединения, и.пр. Для эффективного планирования необходимо учитывать все такие особенности.

5) Не поддерживаются оценочные запросы, необходимые для имитации. Для эффективного планирования необходимы оценки объема результата.

Все описанные недостатки были устранены в новой версии адаптера.

5.2 Перспективная архитектура адаптера

Спецификация интерфейса новых адаптеров является более сложной:

```
public interface RemoteAdapter
{
   boolean isSameResource(RemoteAdapter ra);
   void echoRequest() throws ConnectionException;
   void remoteEchoRequest(RemoteAdapter adapter);
   long getAdapterCapability(AdapterCapability
capability);
   boolean adapterSelectExtendedCapability(boolean
opOnSchemaData,String op, boolean
left_op_const,String left_op_type,boolean
right_op_const,String right_op_type);
   Module getAdapterScheme();

   int openSession();/*returns SessionID*/
   void closeSession(int sessionID);
   void keepAlive(int sessionID);
   long getSessionCapability(int sessionID,
SessionCapability capability);

   int registerQuery(int sessionID,Plan plan);  /*returns
QueryID*/
   void closeQuery(int sessionID,int queryID);

   int openReaderOnQuery(int sessionID, int
queryID,String rowIdAttributeName);
   /*returns ReaderID*/

   int materializeReader(int sessionID,int readerID);
   int materializeReader(int sessionID,int readerID,long
maxBytes,long maxRows);
   /*returns DataSetID*/

   int loadRemoteData(int sessionID,RemoteAdapter
adapter,int remoteSessionID,int remoteReaderID);
   /*returns DataSetID*/
   int loadRemoteData(int sessionID,RemoteAdapter
adapter,int remoteSessionID,int remoteReaderID,long
maxBytes,long maxRows);
   /*returns DataSetID*/

   void removeData(int sessionID,int dataSetID);


   long freeSpaceAvailable(int sessionID);   /*bytes*/
```

```
   String getData(int sessionID, int readerID, long
maxBytes, long maxRows);
   /*    *returns VOCLASS    */

   long dataAvailable(int sessionID, int readerID);
   long rowsSent(int sessionID,int readerID);
   long bytesSent(int sessionID,int readerID);
   void closeReader(int sessionID, int readerID);

   /*Query Evaluating Functions*/
   long estimateRows(int sessionID,int queryID);
   long estimateDataSize(int sessionID,int queryID);
   long averageRowSize(int sessionID,int queryID);
   int getSampleDataReader(int sessionID, int
queryID,long maxBytes, long maxRows, String
rowIdAttributeName);
   /*returns ReaderID*/

}
```

Функции *isSameResource, echoRequest, remoteEchoRequest, getAdapterScheme* необходимы для корректной работы с удаленными ресурсами.

Функции *getAdapterCapability, adapterSelectExtendedCapability, getSessionCapability, freeSpaceAvailable* осуществляют поддержку возможностей ресурсов, обладающих различными способностями.

Функции *openSession, closeSession, keepAlive* отвечают за поддержку сессий.

Функции *registerQuery, closeQuery* необходимы для передачи запросов ресурсам.

Функции *openReaderOnQuery, getData, dataAvailable, rowsSent, bytesSent, closeReader* осуществляют поддержку многократного использования результата запроса. На запрос открывается *reader*, посредством которого данные могут быть считаны как целиком, так и по частям. Кроме того предоставляются возможности получения статистики (сколько данных считано, остались ли данные).

Функции materializeReader, loadRemoteData, removeData отвечают за работу с временными коллекциями. В качестве источника для временных данных могут выступать как данные, полученные от других удаленных адаптеров, так и собственные данные, являющиеся результатом некоторого запроса, над которым открыт *reader*. Временные коллекции могут участвовать в запросах, как если бы это были внутренние данные ресурса.

Функции *estimateRows, estimateDataSize, averageRowSize, getSampleDataReader* предоставляют некоторые оценочные данные, на основе которых строится эффективный план.

5.3 Способности адаптеров (capabilities)

Различают два вида способностей (capabilities): способности сессии, и способности адаптера. Способности сессии описывают ресурсы выделенные под конкретную сессию. Способности адаптера описывают принципиальные возможности адаптера, как например возможность выполнения некоторых операций. Планы, которые не удовлетворяют ограничениям по ресурсам сессии и ограничениям способностей адаптера отбрасываются планировщиком во время поиска последовательным перебором возможных планов в заданном пространстве.

Рассмотрим подробнее все возможности.

Описание способностей сессии.

```
public class SessionCapability {
   public final int value;
      /*IntValues*/
   public static final SessionCapability
iMaxRowsInDataSet = new SessionCapability(0);
   public static final SessionCapability
iMaxDataSetSize = new SessionCapability(1);
   public static final SessionCapability
iMaxDataSpacePerSession = new SessionCapability(2);
   public static final SessionCapability
iMaxDataSetsPerSession = new SessionCapability(3);
   public static final SessionCapability
iMaxRegistredQuerysPerSession = new
SessionCapability(4);
   public static final SessionCapability
iMaxOpenReadersPerSession = new
SessionCapability(5);
   public SessionCapability(int v)
   {      value=v;   }
}
```

iMaxRowsInDataSet – результатом вызова getSessionCapability должно быть максимальное количество кортежей в одной временной коллекции или 0, если временные коллекции не поддерживаются данным адаптером.

iMaxDataSetSize – результатом вызова getSessionCapability должен быть максимальный размер одной временной коллекции в байтах или 0, если временные коллекции не поддерживаются данным адаптером.

iMaxDataSpacePerSession - результатом вызова getSessionCapability должен быть максимальный суммарный размер временных коллекций в пределах данной сессии, или 0, если временные коллекции не поддерживаются данным адаптером.

iMaxDataSetsPerSession - результатом вызова getSessionCapability должно быть максимальное количество одновременно существующих временных коллекций в пределах данной сессии или 0, если временные коллекции не поддерживаются данным адаптером.

iMaxRegistredQuerysPerSession - результатом вызова getSessionCapability должно быть максимальное количество одновременно существующих зарегистрированных запросов в пределах сессии. Минимальное значение 1.

iMaxOpenReadersPerSession - результатом вызова getSessionCapability должно быть максимальное количество одновременно открытых ридеров в пределах сессии. Минимальное значение – 1.

На всех остальных значениях getSessionCapability должна возвращать 0.

Описание способностей адаптера.

```
public class AdapterCapability {
   public final int value;
   public static final AdapterCapability
UserDataSetsSupport = new AdapterCapability(0);
   public static final AdapterCapability
UserDataSetsComplexQuery = new
AdapterCapability(1);
   public static final AdapterCapability
RemoteUserDataLoad = new AdapterCapability(2);
   public static final AdapterCapability
CanExecuteSchemaJoin = new AdapterCapability(4);
   public static final AdapterCapability
CanExecuteTempJoin = new AdapterCapability(5);
   public static final AdapterCapability
CanExecuteMixedJoin = new AdapterCapability(6);
   public static final AdapterCapability
CanExecuteSchemaUnion = new AdapterCapability(7);
   public static final AdapterCapability
CanExecuteTempUnion = new AdapterCapability(8);
   public static final AdapterCapability
CanExecuteMixedUnion = new AdapterCapability(9);
   public static final AdapterCapability
CanExecuteSchemaSelect = new
AdapterCapability(10);
   public static final AdapterCapability
CanExecuteTempSelect = new AdapterCapability(11);
   public static final AdapterCapability
SupportsExtendedSelectCapability = new
AdapterCapability(18);
   public static final AdapterCapability
CanExecuteSchemaSetTypeOperations = new
AdapterCapability(12);
   public static final AdapterCapability
CanExecuteSetTypeOperationsOnTempData = new
AdapterCapability(13);
   public static final AdapterCapability
CanAcceptTempSetTypeData = new
AdapterCapability(14);
   public static final AdapterCapability
AllowSchemaMethodsOnTempData = new
AdapterCapability(15);
   public static final AdapterCapability
CanExecuteTempAppendID = new
AdapterCapability(16);
   public static final AdapterCapability
CanExecuteSchemaAppendID = new
AdapterCapability(17);
   public AdapterCapability(int v)
   {      value=v;      }
}
```

UserDataSetsSupport - Если возвращается значение отличное от нуля, это значит, что адаптер поддерживает временные коллекции. Это в свою очередь означает, что в этом адаптере обязательно должны быть реализованы функции materializeReader, removeData, freeSpaceAvailable.

UserDataSetsComplexQuery - Если возвращается значение равное нулю, это означает, что использование временных коллекций в запросах сильно ограничено. Единственным допустимым запросом с использованием временных коллекций при этом будет запрос, который запрашивает без каких либо изменений все данные, содержащиеся в одной конкретной коллекции.

RemoteUserDataLoad - Если возвращается отличное от нуля значение, это означает, что адаптер реализует и поддерживает методы loadRomoteData и remoteEchoRequest. Фактически, это означает, что адаптер поддерживает загрузку данных, поступающих от других адаптеров.

CanExecuteSchemaJoin - Если возвращается отличное от нуля значение, это означает что поддерживается операция Join над двумя и более коллекциями, объявленными в схеме источника (не временными).

CanExecuteTempJoin - Если возвращается отличное от нуля значение, это означает, что на адаптере допустимо выполнение операции Join над двумя или более временными коллекциями.

CanExecuteMixedJoin - Если возвращается отличное от нуля значение, это означает, что адаптер способен выполнить операцию Join над парой операндов, один из которых является результатом выполнения каких-либо допустимых на этом адаптере операций над коллекциями, объявленными в схеме, а второй над временными коллекциями. Над результатом выполнения подобной операции должны быть выполнимы любые операции, которые разрешено выполнять над временной коллекцией.

CanExecuteSchemaUnion - Если возвращается отличное от нуля значение, это означает что поддерживается операция Union над двумя и более коллекциями, объявленными в схеме источника (не временными).

CanExecuteTempUnion - Если возвращается отличное от нуля значение, это означает, что на адаптере допустимо выполнение операции Union над двумя или более временными коллекциями.

CanExecuteMixedUnion - Если возвращается отличное от нуля значение, это означает, что адаптер способен выполнить операцию Union над парой операндов, один из которых является результатом выполнения каких-либо допустимых на этом адаптере операций над коллекциями, объявленными в схеме, а второй над временными коллекциями. Над результатом выполнения подобной операции должны быть выполнимы любые операции, которые разрешено выполнять над временной коллекцией.

CanExecuteSchemaSelect - Если возвращается отличное от нуля значение при вызове с этим параметром, это означает что поддерживается операция Select над данными в составе схемы (и любыми данными, которые можно получить в результате допустимых операций над данными в схеме) в полном объеме.

CanExecuteTempSelect - Если возвращается отличное от нуля значение при вызове с этим параметром, это означает что поддерживается операция Select над временными данными в составе

схемы (и любыми данными, которые можно получить в результате допустимых операций над временными данными) в полном объеме.

SupportsExtendedSelectCapability - Если функция с этим параметром возвращает ненулевое значение, то в адаптере должна быть корректно реализована функция adapterSelectExtendedCapability.

CanExecuteSchemaSetTypeOperations - Если возвращается отличое от нуля значение при вызове с этим параметром, это означает что поддерживаются операции над множествами, над данными объявленными в схеме источника. Если в схеме источника содержатся множества, то операции над множествами обязаны поддерживаться.

CanAcceptTempSetTypeData - Если возвращается отличное от нуля значение при вызове с этим параметром, это означает, что во временных коллекциях могут содержаться атрибуты - множества.

CanExecuteSetTypeOperationsOnTempData - Если возвращается отличное от нуля значение, допустимо использование операций над множествами над временными данными

AllowSchemaMethodsOnTempData - Если возвращается отличное от нуля значение при вызове с этим параметром, это означает что все методы, объявленные в схеме источника, могут быть использованы, в том числе и над временными данными.

CanExecuteSchemaAppendID - Если возвращается ненулевое значение при вызове с этим параметром, это означает, что адаптер может выполнять операции Append с Id предикатом над данными, объявленными в схеме источника. В виде Id предикатов будут представлены все операции преобразования типов атрибутов, а также арифметические операции (могут быть логические, над булевыми аргументами).

CanExecuteTempAppendID - Если возвращается ненулевое значение при вызове с этим параметром, это означает, что адаптер может выполнять операции Append с Id предикатом над временными данными. В виде Id предикатов будут представлены все операции преобразования типов атрибутов, а также арифметические операции (могут быть логические операции над булевыми аргументами).

6 Заключение

Предложенное применение оценок на основе имитации выполнения планов, а также итерационное выполнение запросов, позволяет уже в достаточно простой задаче интеграции информационных ресурсов, такой, как задача поиска далеких галактик, описанная в публикации [1], для решения которой необходимо интегрировать несколько объемных каталогов астрономических объектов, снять ограничения на объем задействованных в выполнении запроса данных и, в зависимости от загрузки задействованных в её решении ресурсов, повысить эффективность выполнения запросов.

Литература

[1] Брюхов Д.О., Вовченко А.Е., Захаров В.Н., Желенкова О.П., Мартынов Д.О., Скворцов Н.А., Ступников С.А. Архитектура промежуточного слоя предметных посредников для решения задач над множеством интегрируемых неоднородных распределенных информационных ресурсов в гибридной грид-инфраструктуре виртуальных обсерваторий. Информатика и ее применения, Т. 2, вып. 1, 2008, стр. 2--34.

[2] Захаров В.Н., Калиниченко Л.А., Соколов И.А., Ступников С.А. Конструирование канонических информационных моделей для интегрированных информационных систем. Информатика и ее применения, Т. 1, вып. 2, 2007, стр. 15--38.

[3] Briukhov D.O., Kalinichenko L.A., Martynov D.O. Source Registration and Query Rewriting Applying LAV/GLAV Techniques in a Typed Subject Mediator. Proc. of the Ninth Russian Conference on Digital Libraries RCDL'2007.

[4] Christian Wiesner. Query Evaluation Techniques for Data Integration Systems. 2004. http://www.opus-bayern.de/uni-passau/volltexte/2004/40/pdf/QETechniquesForDISystems.pdf

[5] Kalinichenko L.A., Briukhov D.O., Martynov D.O., Skvortsov N.A., Stupnikov S.A. Mediation Framework for Enterprise Information System Infrastructures. Proc. of the 9th International Conference on Enterprise Information Systems ICEIS 2007. -- Funchal, 2007. -- Volume Databases and Information Systems Integration. -- P. 246--251.

[6] Kalinichenko L.A., Stupnikov S.A., Martynov D.O. SYNTHESIS: a Language for Canonical Information Modeling and Mediator Definition for Problem Solving in Heterogeneous Information Resource Environments. Moscow: IPI RAN, 2007.

[7] Mostafa Elhemali, Cйsar A. Galindo-Legaria, Torsten Grabs, Milind M. Josh. Execution Strategies for SQL Subqueries. Proceedings of the 2007 ACM SIGMOD international conference on Management of data. P. 993 – 1004.

[8] Surajit Chaudhuri. An Overview of Query Optimization in Relational Systems. Symposium on Principles of Database Systems. 1998. P.. 34 - 43.

[9] Volker Markl, Vijayshankar Raman, David Simmen, Guy Lohman, Hamid Pirahesh, Miso Cilimdzic. Robust Query Processing through Progressive Optimization. Proceedings of the ACM SIGMOD international conference on Management of data. P. 659 – 670. 2004.

[10] Yannis E. Ioannidis. Query Optimization. ACM Computing Surveys, Volume 28, Issue 1. March 1996. P. 121 – 123.

[11] Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Levy, Daniel S. Weld An Adaptive Query Execution System for Data Integration. Proceedings of the 1999 ACM SIGMOD international conference on Management of data. P. 299 – 310.

Query planning over heterogeneous distributed information resources in the architecture of the subject mediators

© Vovchenko A.E., Krupa A.V.

Query planner plays one of the major roles in the architecture of subject mediators. Planner is responsible for decomposition and execution of the user query over a set of heterogeneous information resources. The decomposition problem in the mediator architecture implies a set of acceptable decisions, therefore one of the major tasks of the planner is the choice of the most effective one.

In the article the description of the planner which is able to effectively execute the queries formulated over information resources, containing large-scale sets of data (in comparison with the bandwidth of communication channels). Such queries usually require long time for execution. Also in the article the description of requirements to resource wrappers is presented. A wrapper architecture was developed to provide the planner with a possibility of construction of effective execution plans.

♣ Работа выполнена при частичной финансовой поддержке РФФИ (грант 08-07-00157-а) и Программы фундаментальных исследований Президиума РАН № 3 «Фундаментальные проблемы системного программирования» (проект «Исследование методов и средств промежуточного слоя предметных посредников, обеспечивающего решение задач над множеством неоднородных распределенных информационных ресурсов»).