Д. Н. Степанов

Использование технологии XS в разработке информационной системы для Университета города Переславля

Научный руководитель: д.ф.-м.н С. В. Знаменский

Аннотация. Данная работа посвящена использованию технологии XS в разработке информационной системы (ИС) для УГП. Технология позволяет использовать возможности языка C в программах, написанных на языке Perl, для того, чтобы увеличить их быстродействие и производительность. Продукт под названием TokyoCabinet был выбран в качестве системы управления базами данных (СУБД). На языке C реализован алгоритм чтения данных из базы данных (БД).

Ключевые слова и фразы: УГП, СУБД, TokyoCabinet, XS, контекст, ретроспективная система, Perl API.

1. Введение

Летом 2009 года в УГП имени А. К. Айламазяна была начата работа по созданию и реализации принципиально новой модели информационной системы ([1]), которая могла бы обеспечить возможности, труднодостижимые при использовании традиционных архитектур, а именно:

- возможность доступа к любым данным, имеющимся в информационной системе, при условии возможных изменений функционирования сервисов и внутренних структур данных (при этом изменения могут быть весьма значительными). Обычно в таких случаях создается еще одна или несколько информационных систем, которые, как и исходная ИС, требуют поддержки. Приходится думать и о согласованности тех данных, которые дублируются в разных ИС (например, информация о персонах, которые задействованы во всех ИС);
- ретроспективность **абсолютно всех** данных, с которыми работает ИС (а не их части, что присутствует в некоторых реализациях с использованием традиционных подходов на основе SQL). Ретроспективность позволяет получить доступ к информации,

актуальной в прошлом. Исполняемый код тоже можно считать данными;

- отсутствие проблемы взаимных блокировок, которые порой сильно снижают эффективность традиционных СУБД на основе SQL.

2. Выбранный инструментарий

Пусть все данные представлены в виде множества пар «идентификатор => значение» (и то, и другое — строки произвольной длины). Назовем *контекстом* совокупность данных с фиксированным началом строки идентификатора. Назовем информационную структуру *контекстно-автономной*, если организация доступа к данным любого контекста может быть изменена (как и сами данные) независимо от остальной части системы без приостановки сервисов и пользовательских приложений, в которых контекст не участвует.

Для хранения данных в виде таких пар была выбрана СУБД *TokyoCabinet* [2]. Это open-source продукт, работающий на операционной системе Linux, распространяется под лицензией GPL. Она обладает гораздо более высоким быстродействием, чем СУБД на основе SQL (благодаря тому, что она работает на более низком уровне), но к ней неприменимы обычные SQL-запросы.

TokyoCabinet позволяет работать с несколькими типами организации данных, нами выбран тип **Btree**. В нем множество пар «идентификатор => значение» организовано в виде упорядоченного списка (по умолчанию порядок сортировки лексикографический, но можно определить свой). Также есть индекс, где все ключи организованы в виде бинарного сбалансированного дерева, что позволяет получить запись по данному ключу за время, зависящее логарифмически от количество записей в БД. Если нет точного совпадения на ключи, то берется самая первая запись с ключом, «большим» того, который мы ищем (понятие «больше» зависит от способа сортировки ключей). Также есть курсоры, позволяющие встать на некоторую запись и двигаться от нее вверх или вниз по записям, таким образом осуществляя итерацию, подобно курсорам в традиционных архитектурах.

Таким образом, вся наша БД представлена в виде одного файла (в SQL-архитектурах обычно каждая таблица базы данных — отдельный набор файлов), и естественно решается проблема взаимных блокировок.

3. Строение ключей в БД

Ключ записи является конкатенацией строк, среди которых обычно есть:

- префикс, идентифицирующий данное;
- указание автора записи либо специфической информации о её происхождени;
- строка отсчета времени в секундах с заданной для контекста точностью;
- строка отсчета реального времени в секундах и/или идентификатор пользователя или процесса, сохранившего запись;
- байт особенности интерпретации. Это необходимо для различения данных разных типов (строки в разных кодировках, исполняемый код, сериализованная структура, ссылка на другую запись).

Если для заданного ключа в БД нет особого описания прав доступа, то права доступа определяются минимальным контекстом, для которого задано описание прав. В терминах дерева данных это означает, что если права доступа не указаны для заданного листа, то используются права для минимальной содержащей этот ключ ветки. Идентифицирующий данные префикс обычно является составным и формируется так, чтобы максимально использовать быстрый механизм балансированного двоичного дерева для проведения наследования данных при авторизации доступа к страничкам и выборе нужной версии исполняемого кода, формируюего страницу, которую видит пользователь.

Конечно, хранение полной истории для всех данных в БД довольно скоро может обернуться нехваткой дисковой памяти. Но мы можем выделять некоторые промежутки времени («белые пятна истории»), и внутри них убирать всю историю изменений для всех ключей, которые попали в этот промежуток, оставляя для каждого ключа только его самое свежее значение. Основанием для такой операции служит то, что вероятность обращения кого-либо из пользователей к истории изменений в этот промежуток времени очень мала. Такую стратегию можно применить, например, к старым log-файлам.

4. Алгоритм *db_read*

Далее встает вопрос о реализации алгоритма, который позволял бы взять значение по ключу из БД, учитывая права доступа. Необходимо учитывать, что могут потребоваться данные, актуальные только в указанный период времени (если нам необходим доступ к истории). Псевдокод алгоритма описан здесь: [3] (с применением кода на языке **Perl**, который является основным языком разработки ИС). Поскольку чтение данных должно происходить как можно быстрее и эффективнее, то к реализации алгоритма предъявляются особые требования по времени выполнения кода и использованию памяти.

5. Технология XS

Язык программирования **Perl** — интерпретируемый, интерпретатор написан на языке **C**. Есть возможность писать на **C** код (набор функций), который компилируется и оформляется в виде динамически подгружаемой библиотеки, а затем эти функции можно вызывать, используя обычный синтаксис языка **Perl** ([4]). Как правило, эти функции обладают значительно большей производительностью, чем написанные на **Perl**, иногда в десятки и даже в сотни раз. Возникла идея изучить и применить технологию, позволяющую использовать возможности языка **C** в **Perl**-коде.

Интерфейс прикладного программирования (API) у интерпретатора языка **Perl** достаточно обширен и включает в себя более 3000 различных функций и макросов, их предназначение:

- работа со строками (в том числе с кодировками Unicode и UTF-8);
- работа с числами;
- работа со структурами данных языка Perl (скалярами, массивами, хешами);
- изменение процесса интерпретации исходного кода и т. д.

XS (eXternal Subroutine) — это набор макросов, позволяющих писать функции на языке **C**, используя синтаксис, почти идентичный синтаксису языка **C**. Затем перед компиляцией специальный скрипт заменяет эти макросы на развернутый **C**-код. После компиляции эти функции можно использовать в **Perl**.

XS часто используется в реализации следующих модулей языка **Perl**:

- работа с графикой;
- работа с базами данных (в том числе с XML);
- использование различных численных методов;
- mod_perl ([5]).

Можно вообще взять почти любую библиотеку, написанную на **C** или **C++** и написать необходимое кол-во функций-врапперов, чтобы использовать возможности этой библиотеки в **Perl**.

6. Результаты

В дополнение к уже существующему варианту реализации алгоритма *db_read* на языке **Perl** был написан ее аналог на языке **C** и оформлен в виде динамически подгружаемой библиотеки. В процессе сравнения их быстродействия было выявлено, что при обращении к БД по разным ключам выигрыш варианта на **C** бывает разный: от нескольких процентов до более чем в двадцать раз. Алгоритм *db_read* содержит в себе и элементы рекурсии (проверка прав доступа, работа с «белыми пятнами истории»), и итерации (просмотр БД с помощью курсора). Таким образом, время, которое требуется на поиск необходимого ключа, зависит от кол-ва итераций и глубины рекурсий. А они зависят от общего размера БД и количества «белых пятен истории». Следует заметить, что эти результаты проводились на тестовой БД небольшого размера, где общее количество записей невелико. И поэтому следует ожидать, что с ростом БД выигрыш от использования реализации алгоритма на **C** будет все больше и больше.

Возникла идея сравнить прозводительность **C** и **Perl** в «предельных» случаях, на примере простого итеративного (подсчет суммы всех целых чисел от 1 до m) и простого рекурсивного алгоритма (факториал от числа n). Реализации обоих алгоритмов на обоих языках очень компактны, не более 5 строчек кода. Вариант на **C** был оформлен в виде динамически подгружаемой библиотеки и затем вызывался из **Perl**-скрипта. Оказалось, что в первом случае (подсчет суммы) вариант на **C** быстрее примерно в 110 раз, причем выигрыш почти не зависил от количества итераций (при $m = 1000000$ и при $m = 10000000$). Во втором случае (рекурсивное вычисление факториала) выигрыш был пропорционален глубине рекурсии. В частности, при $n = 150$ он составлял 65-75 раз.

Отсюда следует, что в проектах, использующих **Perl**, при реализации итеративных или рекурсивных алгоритмов, от которых требуется высокая производительность, кажется целесообразным использовать технологию **XS** (разумеется, при соответствующей подготовке программистов). Напротив, реализации линейных алгоритмов почти не отличаются по производительности в **Perl** и в **C**.

7. Выводы и перспективы

В процессе проектирования ИС для УГП были освоены широкие возможности технологии **XS**, был написан вариант алгоритма *db_ read* на языке **C**, который оказался эффективнее варианта на языке **Perl**. В дальнейшем предполагается применить эту технологию при реализации других алгоритмов, которые используются в процессе функционирования новой ИС и для которых критично быстродействие и производительность. Недостатком технологии **XS** является сравнительная сложность в изучении, и поэтому крайне желательно для реализаций всех алгоритмов на языке **C** иметь их «запасные» аналоги на языке **Perl**.

Список литературы

[1] Абрамов С. М, Живчикова Н. С., Знаменский С. В., Котомин А. В., Степанов Д. Н., Титова Е. В., Юмагужина В. Н. *Архитектура системы для разработки технологий организации сложной совместной деятельности*, 2009.

[2] Официальный сайт разработки СУБД TokyoCabinet и родственных ей программных продуктов: `http://1978th.net`.

[3] Алгоритм *db_ read*: `http://wiki.botik.ru/IS4UGP/ReadFromDatabase`.

[4] Документация к языку Perl: man-страницы *perlxstut, perlguts, perlxs, perlcall, perlapi, perlunicode, perlapio, perlclib, perlreapi*.

[5] Технология mod_perl: `http://perl.apache.org`.

D. N. Stepanov. *Use of XS technology in development of information system of Pereslavl University* // Proceedings of Junior research and development conference of Ailamazyan Pereslavl university. — Pereslavl, 2010. — p. 55–61. (*in Russian*).

Abstract. This paper is devoted to use of XS technology in development of information system of Pereslavl University. This technology allows to use the capacity of the language C in programs, written on language Perl to enlarge their speed and power. Product under name TokyoCabinet was chosen as DBMS. The algorithm of reading data from database was realized.

Key Words and Phrases: UGP, DBMS, TokyoCabinet, XS, context, retrospective system, Perl API.