НАУЧНОЕ ИЗДАНИЕ МГТУ ИМ. Н. Э. БАУМАНА
НАУКА и ОБРАЗОВАНИЕ
Эл № ФС77 - 48211. Государственная регистрация №0421200025. ISSN 1994-0408
электронный научно-технический журнал

Произвольный доступ к данным архивов телеметрической информации

 # 10, октябрь 2013
DOI: 10.7463/1013.0616065
Сидякин И. М.
УДК 004.632.4

Россия, МГТУ им. Н.Э. Баумана
ivan.sidyakin@gmail.com

1. Введение

Системы регистрации телеметрической информации (ТМИ) обычно разрабатываются для отображения, предварительной обработки и записи поступающего потока данных измерений на какой либо носитель информации в режиме реального времени. Если в состав системы входит компьютер, то носителем информации может быть его жесткий диск или, например, флэш память. Тогда поток ТМИ записывается на диск как обычный файл. Процесс регистрации может быть достаточно длительным, и объем принимаемой информации настолько большим, что для экономии места на диске желательно такие файлы сжимать. Низкочастотная датчиковая телеметрическая информация, то есть, информация, представляющая собой собранные в циклически повторяющиеся кадры оцифрованные отсчеты показаний датчиков, измеряющих медленно меняющиеся параметры, такие например как температура, хорошо сжимается методами сжатия без потерь. Программы gzip, zip или rar, использующие подобные методы сжатия, уменьшают размеры файлов телеметрической информации в пять - десять раз. В системе регистрации ТМИ, процедура сжатия должна быть встроена в процесс регистрации и должна выполняться непосредственно перед записью информации на диск. Записываемые на диск данные сжимаются последовательно, начиная с первого принятого байта, и помещаются в файл архива, который впоследствии при необходимости распаковывается программой

архиватором, для просмотра и обработки содержащейся в нем информации. В некоторых случаях просмотр и обработка записанной на диск телеметрической информации производится последовательно, так же как запись этой информации, с первого и до последнего, записанного в файл, байта. Однако во многих алгоритмах обработки ТМИ, а также для просмотра отдельных фрагментов записи, требуется обеспечить произвольный доступ к записи. Иными словами требуется выполнить чтение данных из любого произвольно заданного места файла. Применяемые архиваторами способы сжатия данных без потерь не вполне отвечают этому требованию. Они рассчитаны на последовательную распаковку данных, когда сжатый файл может быть распакован только с начала. Невозможно начать распаковку данных из любого места файла архива. Для небольших файлов это не имеет значения, так как распаковка выполняется быстро и результат распаковки можно кэшировать в оперативной памяти. Однако, если размер файла ТМИ составляет более сотни мегабайт, динамическая распаковка данных от начала файла до нужного места, становится неэффективной.

В этой работе предлагается метод увеличения скорости чтения данных из произвольно выбранного места архивного файла телеметрической информации большого размера (более 100 Мбайт) за счет добавления к файлу архива дополнительной служебной информации. Прежде чем пояснить его принцип рассмотрим коротко алгоритм deflate различные модификации, которого используются современными программами архивации.

2. Описание используемых алгоритмов

Deflate это комбинация кода Хаффмана [1] и алгоритма сжатия LZ77 [2]. Данные кодируются блоками. Каждый блок кодируется независимо от других. Границы блоков не выровнены на границу байта. Блоки начинаются с трех битового префикса, который содержит признак последнего блока в последовательности и тип сжатия использующегося внутри блока, один из трех: динамический код Хаффмана, статический код Хаффмана или без сжатия. В последнем случае сразу за префиксом в файле записана длина блока в байтах. Используя это значение, можно сразу переместиться к следующему блоку данных.

Для двух других типов блоков кодирование проходит в два этапа. На первом алгоритм LZ77 находит повторяющиеся последовательности символов в пределах окна, размер которого в алгоритме deflate равен 32КБ. Все повторения каждой такой последовательности заменяются на *указатель*, который содержит смещение до места, где последовательность встретилась первый раз и длину этой последовательности. Таким образом, после первого этапа кодирования в блоке остаются указатели LZ77 (они всегда указывают назад) и символы, не попавшие в повторяющиеся последовательности (а также символы, составляющие первые, встреченные в окне экземпляры повторяющихся последовательностей). Символы и значения длин последовательностей кодируются кодом Хаффмана. Собственно для них создается один общий словарь, в котором символам отводится диапазон кодов от 0 до 255. Код 256 используется для индикации конца блока, а коды 257 – 286 используются для указания длин последовательностей. Все эти коды преобразуются в слова кода Хаффмана, длина которых, обратно пропорциональна частоте появления кода в исходных данных блока.

Deflate поддерживает два метода кодирования: статический и динамический. В первом случае используется фиксированный заранее заданный словарь (дерево) кодов Хаффмана. Преимущество этого подхода в том, что он не требует сохранять словарь вместе с данными для их последующего декодирования, а также выполняется быстрее. Недостаток метода в том, что он не адаптируется к данным, статистические характеристики которых могут существенно отличаться от характеристик используемого словаря. Динамическое кодирование позволяет построить словарь по фактическим данным блока. Каждый блок, таким образом, имеет собственный словарь, сохраняемый в начале блока для декодирования. Кодер может выбирать тип сжатия блока и, например, использовать наилучший из трех в каждом конкретном случае, включая и отсутствие сжатия. Если сжатие не дает никакого выигрыша, логично использовать алгоритм, обеспечивающий наивысшую скорость распаковки данных. Как правило, большинство блоков в сжатом файле кодируются динамическим кодом Хаффмана.

Есть некоторые детали реализации этого алгоритма повышающие его производительность, но не имеющие отношения к нашей теме. Подробные сведения об алгоритме deflate можно получить в [3].

Из описания способов кодирования блоков данных видно, что переход к заданному месту файла возможен только после того как декодированы все данные от начала файла до этого места. В случае если данные сжаты алгоритмом Хаффмана, заголовок блока не содержит в явном виде размер блока. Размер определяется с помощью индикатора конца блока (код 256).

Размер блока определяется, как правило, не размером входного буфера данных алгоритма, а размером выходного буфера. Иначе говоря, кодер принимает решение о завершении блока, когда заполняется его выходной буфер и требуется записать этот буфер на диск. Кодер может ставить размер блока в зависимость от фактического значения коэффициента сжатия.

Как уже говорилось выше, блоки кодируются независимо друг от друга, однако это утверждение справедливо только для второй части алгоритма – кодирования Хаффмана. Указатель, заменяющий повторяющуюся последовательность в блоке, может содержать ссылку на цепочку символов расположенную за пределами блока, точнее на последовательность символов, расположенную в одном или нескольких предшествующих блоках. Алгоритм LZ77 использует метод скользящего окна, согласно которому совпадения последовательностей символов в исходных (кодируемых) данных ищутся, начиная с позиции отстоящей ближе к началу файла на заданную и постоянную величину считая от текущей позиции. Размер окна в практических реализациях алгоритма равен 32Кб.

Указанная особенность алгоритма не позволяет начать распаковку файла с произвольного блока, даже если заранее известна позиция этого блока в сжатом файле. Для распаковки следует дополнительно иметь в распоряжении 32 килобайта распакованных данных расположенных непосредственно перед этим блоком. Данных с “образцами ” последовательностей, повторяющихся в блоке, который предстоит распаковать.

3. Описание предлагаемого метода

Из вышесказанного сделаем два вывода:

1) Для того чтобы начать распаковку файла с произвольного символа, требуется знать смещение этого символа, представленного в виде слова кода Хаффмана в

архиве в битах, текущее содержимое словаря кодов Хаффмана, если он формируется динамически и 32 килобайта распакованных данных предшествующих этому байту, для восстановления повторяющихся последовательностей, представленных в виде пары указатель-длина.

2) Для того чтобы начать распаковку файла с произвольного блока достаточно иметь в распоряжении 32 килобайта распакованных данных предшествующих этому блоку. Словарь кодов Хаффмана для каждого блока не требуется, так как в случае, если используется динамическое кодирование, он пересоздается заново для каждого блока, а в двух других случаях или одинаков для всех блоков или не используется.

Учитывая, что обработка 32 килобайтного блока на современных компьютерах выполняется очень быстро а размеры файлов ТМИ как правило на несколько порядков превышают размеры блока, целесообразно остановится на втором варианте, то есть обеспечить распаковку файла с произвольного блока а не с произвольного символа. Конечно, при этом функции программного интерфейса системы доступа к сжатым данным должны принимать в качестве аргумента не номер блока, а позицию байта в исходных (несжатых) данных, с которого требуется начать операцию чтения. Далее функция должна определить позицию блока в сжатых данных, распаковать этот блок с начала до указанной позиции байта, отбросить распакованные данные. Далее с этого места следует продолжить распаковку данных, которые надо вернуть клиенту.

3.1. Подготовка архива

Для того чтобы функция чтения данных из архива была способна определить номер блока в потоке сжатых данных по заданной позиции в исходных данных, одновременно с записью архива требуется создать последовательность контрольных точек. Контрольная точка это структура данных, имеющая следующий простой формат:

```
/* контрольная точка */
struct point {
  off_t out;     /* позиция первого символа блока в исходных данных */
  off_t in;      /* позиция блока в сжатых данных */
  int bits;      /* смещение первого бита блока в сжатых данных */
  unsigned char windows[WINSIZE]; /* окно алгоритма Deflate */
};
```

Позиция начала блока в архиве представлена двумя полями структуры, так как она в общем случае не выровнена на границу байта. Как уже было сказано выше, для начала распаковки блока требуются данные окна алгоритма Deflate, то есть блок исходных (до сжатия) данных файла ТМИ расположенных непосредственно перед блоком. Эти данные, конечно же, имеются в распоряжении процедуры сжатия файла, которая дополнительно отвечает за создание контрольных точек.

В программной реализации алгоритма использовалась модифицированная версия алгоритма deflate широко известной библиотеки zlib ([4], файлы *deflate.c*, *trees.c* версия библиотеки 1.2.3 Июль 2005г.).

Для создания контрольных точек в алгоритм сжатия была вставлена функция OnFlush, которая вызывается в момент начала создания нового блока сжатых данных (в самом конце функции _tr_flush_block в файле *trees.c)*. Кроме этого к внутреннему контексту алгоритма deflate были добавлены поля для отслеживания значений структуры point. К библиотеке также были добавлены функции, возвращающие текущие значения этих полей, включая указатель на окно данных алгоритма deflate. Сокращенный вид функции OnFlush приведен ниже:

```
void OnFlush(z_streamp strm)
{
	…
	getDeflateCounters(strm, &inCount, &outCount, &bitCount);
	PZIP_ARCHIVE zip = (PZIP_ARCHIVE)getDeflateContext(strm);
	…
	if (inCount - zip->LastIndexPosition > zip->accstep)
	{
		…
		getDeflateDictFromCache(&zip->strm, inCount, zip->pwin);
			addpoint(zip->index, …, outByteCount, inCount, …);
		WriteFile(zip->hIndexFile, zip->pwin, WINSIZE, &cbRet, NULL);
		zip->LastIndexPosition = inCount;
		…
	}
}
```

Функция getDeflateCounters возвращает счетчики необходимые для заполнения первых трех полей структуры point. В поле zip->accstep хранится значение шага создания контрольных точек выраженного в байтах. Этот шаг указывается в

настройках процедуры сжатия. Практически для сжатия ТМИ, использовался шаг от 1Мб до 10 Мб. Величина этого шага определяется экспериментально и зависит от производительности компьютера. Из приведенного выше кода видно, что процедура создания новой контрольной точки запускается только в случае, если значение разницы между iCount и последней контрольной точкой превышает значение шага. Следует пояснить, что эти позиции вычисляются в последовательности исходных а не сжатых данных. То есть, контрольные точки располагаются равномерно в “системе координат” исходного файла ТМИ а в сжатом файле располагаются друг от друга на разном расстоянии в зависимости от того насколько эффективно работает алгоритм сжатия. Такой подход приводит к увеличению количества контрольных точек, в случае, если коэффициент сжатия данных увеличивается, однако он ограничивает время необходимое для распаковки фрагмента записи между контрольными точками. Действительно, если бы расстояние между контрольными точками определялось размером сжатых данных, то в случае если данные сжимаются очень хорошо, соответствующий им фрагмент исходных данных оказывается очень длинным и время его распаковки неоправданно большим. Это следует из предположения, что время работы алгоритма распаковки определяется не только временем, затраченным собственно на вычисления, но и временем которое отводится на служебные операции выделения памяти, чтения данных и записи результатов в файл. Впрочем, здесь есть почва для исследований и возможность повышения производительности алгоритма. В защиту используемого подхода, следует также сказать, что на практике для файлов ТМИ имеющих размеры от нескольких сотен мегабайт до десятков гигабайт, значение шага выбирается на несколько порядков выше длины окна алгоритма deflate. А количество контрольных точек в файле не превышает нескольких сотен.

Кроме счетчиков, контрольная точка содержит данные, необходимые для старта процедуры распаковки с указанного места в файле архива. Это содержимое окна deflate или буфер, содержащий распакованные данные (точнее не запакованные данные) расположенные непосредственно перед контрольной точкой. Функция getDeflateContext добавленная к оригинальной реализации алгоритма deflate в библиотеке zlib возвращает указатель на структуру, через которую извлекается указатель этот буфер. Буфер вместе с контрольной точкой сохраняется во временный *индексный* файл. Это файл, после того как упаковка записи ТМИ завершена, содержит все контрольные точки, которые будут использованы для увеличения скорости чтения

сжатых данных. Этот индексный файл рекомендуется поместить в конец архива с записью ТМИ.

3.2. Распаковка архива.

Программный интерфейс, обеспечивающий произвольный доступ к архиву ТМИ, исключая не касающиеся данной темы операции открытия и закрытия файла, а также различные вспомогательные операции, содержит две функции:

```
bool zip_Seek(HANDLE hzip, int offset)
```

Функция перемещает указатель чтения в заданную позицию файла архива.

```
int zip_Read(HANDLE hzip, void * buf, int size)
```

Функция считывает из текущей позиции заданное число распакованных байт. Все параметры функций указаны в координатах исходного несжатого файла. Для того чтобы ориентироваться в этой системе координат, имеется функция возвращающая информацию о файле, включая его исходный (до сжатия) размер. Параметр hzip это ссылка на открытый файл, которую возвращает не показанная здесь функция открытия файла. Следует сказать, что в задачу этой функции, кроме открытия файла архива записи ТМИ, входит загрузка индексного файла. Индексный файл можно поместить в конец архива записи ТМИ и назначить ему какое либо определенное имя. Если процедура открытия файла находит в архиве индексный файл, она загружает список контрольных точек и использует его, так как будет показано ниже. Если файл отсутствует, распаковка файла происходит обычным образом. Если индекс отсутствует, рекомендуется полностью распаковать архив ТМИ во временный файл и считывать данные из этого временного файла.

Основную работу с контрольными точками выполняет функция zip_Seek. В начале, выполняется поиск контрольной точки, за которой располагается указанная в параметре offset позиция. Из найденной контрольной точки извлекается байтовое и битовое смещение начала сжатого блока данных. Далее указатель чтения сжатого файла перемещается в эту полученную позицию и контекст алгоритма deflate инициализируется значениями счетчиков и содержимым буфера окна, считанным из контрольной точки. Это все что необходимо, для того чтобы начать распаковку блока. Блок распаковывается до тех пор, пока позиция в распакованных данных не совпадет с позицией параметра функции offset. Это означает, что указатель чтения установлен в

заданную позицию. Функция zip_Read теперь может продолжить распаковку блока данных заданного размера и вернуть его по указателю buf.

4. Методы повышения производительности алгоритма

В этой статье рассмотрен базовый алгоритм, предложенный для чтения телеметрической информации из потока данных упакованных методом deflate. Этот алгоритм в представленном здесь виде имеет практическую ценность и успешно применяется в реальных системах сбора и обработки телеметрической информации. Очевидно, что есть несколько способов повышения эффективности работы этого алгоритма, а именно следующие:

1. В описанной реализации размер контрольной точки примерно равен длине окна алгоритма deflate, которая составляет 32 килобайта. То есть каждая контрольная точка добавляет к индексному файлу примерно 32К данных. Эту информацию также можно сжать, перед тем как записывать ее в индексный файл.
2. Другое направление для экспериментов это исследование зависимости значения шага между контрольными точками и скорости чтения данных из архива, а также исследование вариантов адаптивной подстройки величины шага к значению коэффициента сжатия на интервале.
3. Некоторый эффект на результаты работы алгоритма может оказать изменение размеров окна алгоритма deflate и также изменение размера блока данных. Эти характеристики, скорее всего, определяются характером сжимаемых данных.

Подробное рассмотрение всех этих вопросов выходит за рамки этой статьи, в задачу которой входит только описание деталей реализации предложенной технологии. Следует отметить, что задача уменьшения объема служебных данных (контрольных точек) становится актуальной в случае, если размер этих данных становится сопоставим с размерами файлов содержащих исходную информацию. Для файлов телеметрической информации объемом в один гигабайт индексный файл с сотней контрольных точек, расположенных друг от друга на расстоянии 10 мегабайт, дает не очень заметную прибавку размером примерно в три мегабайта.

Список литературы

1. Huffman D. A. A Method for the Construction of Minimum Redundancy Codes // Proceedings of the Institute of Radio Engineers. September 1952. Vol. 40, no. 9. P. 1098-1101.

2. Ziv J., Lempel A. A Universal Algorithm for Sequential Data Compression // IEEE Transactions on Information Theory. 1977. Vol. 23, no. 3. P. 337-343. DOI: 10.1109/TIT.1977.1055714

3. DEFLATE Compressed Data Format Specification version 1.3. The Internet Engineering Task Force (IETF). Available at: http://tools.ietf.org/html/rfc1951 , accessed 18.09.2013.

4. A Massively Spiffy yet Delicately Unobtrusive Compression Library // Website Zlib. Available at: http://zlib.net , accessed 18.09.2013.

SCIENTIFIC PERIODICAL OF THE BAUMAN MSTU
SCIENCE and EDUCATION
EL № FS77 - 48211. №0421200025. ISSN 1994-0408
electronic scientific and technical journal

Random access to telemetry data files compressed with deflate algorithm

10, October 2013
DOI: 10.7463/1013.0616065

Sidyakin I.M.

Bauman Moscow State Technical University, 105005, Moscow, Russian Federation
ivan.sidyakin@gmail.com

Telemetric information recording systems are usually designed for display, preprocessing and record of incoming data stream to any kind of storage media in real time In case a computer is part of a system its HDD or flash drive could serve as a storage device. In that case, the data stream is stored as an ordinary file. Telemetry data recording process can take for a long time, while the amount of recorded data could grow up to tens of gigabytes or even more. It looks reasonable to compress these data while saving them to a storage media and keep them there in archived files. Later when post-processing is applied to the recorded data, it is often interesting to work with particular fragments of a record, while fast moving of the “read pointer” within a compressed file and extracting of particular data blocks is required. In other words we need to provide fast read-only random access to the compressed data. OS “built-in” and/or well-known compression software utilities (such as Zip) generally have the same disadvantage: they have to decompress archive file from the very beginning up to a specified point. It is impossible to start unpacking a file from an arbitrary position due to the fact that the deflate algorithm that is in the core of these utilities always needs to know some prehistory in the sequence of uncompressed bytes to proceed with decompression. It is of no importance for small files, because unpacking routines are fast and highly optimized, plus unpacked data can be easily cached. In the meantime, for large telemetry records the problem remains. In this paper the author proposes an effective method that allows to make read-only random access to the packed data faster. This is achieved by inserting some supplementary information into the archive file while it is being recorded. This extra information allows starting decompression from a set of "reference points" within the archived file, but not only from zero index byte. Required modifications in the famous zlib compression library are described.

Publications with keywords: telemetry information, data storage, LZV, deflate
Publications with words: telemetry information, data storage, LZV, deflate

References

1. Huffman D. A. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the Institute of Radio Engineers*, September 1952, vol. 40, no. 9, pp. 1098-1101.

2. Ziv J., Lempel A. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 1977, vol. 23, no. 3, pp. 337-343. DOI: 10.1109/TIT.1977.1055714

3. *DEFLATE Compressed Data Format Specification version 1.3.* The Internet Engineering Task Force (IETF). Available at: http://tools.ietf.org/html/rfc1951 , accessed 18.09.2013.

4. *A Massively Spiffy yet Delicately Unobtrusive Compression Library*. Website Zlib. Available at: http://zlib.net , accessed 18.09.2013.