Сегодня мир систем управления базами данных претерпевает значительные изменения. Всемогущество реляционных систем теперь вполне законно подвергается сомнению ввиду того, что определился ряд задач, с которыми системы других классов справляются гораздо лучше. Одним из примеров этому служит задача хранения временных рядов, решение которой требуется, в частности, и в области управления технологическими процессами - для хранения данных истории процессов. Для этих целей и был создан описываемый в статье продукт InfinityHistoryServer 3.0

Дмитрий **Москвитин**
компания Элеси

InfinityHistoryServer 3.0
ИСТОРИЯ В ДЕТАЛЯХ

ЗАДАЧА ХРАНЕНИЯ ДАННЫХ ИСТОРИИ ПРОЦЕССОВ

В сфере промышленной автоматизации, как и во многих других областях, где выполняется мониторинг и управление процессами, актуальна задача регистрации информации о протекании этих процессов. Такая информация, которую можно именовать историей процесса, представляет собой совокупность временных рядов, описывающих поведение наблюдаемой системы. Каждый временной ряд является набором кортежей вида <t, запись>, где t – метка времени, а запись описывает событие, возникшее в момент времени t, при этом кортежи упорядочены по метке времени. Структура и содержание самой записи зависит от того, что именно характеризует временной ряд. Например, в промышленной автоматизации это может быть пара <значение, качество>, содержащая, соответственно, значение некоторого параметра, или это может быть набор полей, описывающих событие, требующее внимания оператора. Хранение подобной информации крайне важно, поскольку она дает широкие возможности проведения ретроспективного анализа, а также может быть использована при прогнозировании.

Для задачи хранения данных истории процессов характерна специфика, делающая вопрос о выборе средств для ее решения особенно важным:

- значительный объем хранимой информации;
- данные поступают в реальном времени, причем возможны всплески интенсивности их поступления.

Например, пусть регистрируются значения по 5 тыс. параметров (что вполне реально для современной АСУ ТП). Допустим, по каждому из параметров значения обновляются в среднем хотя бы раз в 5 секунд. При таких условиях за сутки система накопит 86,4 млн записей, а за год - 31,536 млрд. Подобные объемы хранимой информации, вкупе с характером ее поступления, требуют как можно большего увеличения производительности и записи, и чтения, а также сокращения размера записей для экономии дискового пространства. Однако существует и «смягчающее» обстоятельство: в основном данные поступают в порядке возрастания метки времени. Т. е. вероятность получения инверсных записей (по сути, записей, поступающих «задним числом») мала.

Итак, какое решение удовлетворяло бы поставленным требованиям? Разумеется, можно прибегнуть к одному из радикальных способов - использовать прямую запись в суточные, недельные и т.п. файлы: все поступающие данные последовательно сохранять в файл. Такое решение очень просто реализуемо, причем позволяет достичь максимально возможной производительности записи. Однако операция чтения будет выполняться крайне медленно, так как для выборки записей по одному параметру потребуется линейный просмотр всех записей в запрошенном временном интервале. Кроме этого, оно не дает возможности эффективно выполнять вставки «задним числом», вероятность возникновения которых мала, но все же существует, а такие записи могут нести очень важную информацию. Казалось бы, применение в этом случае общеизвестных индексных структур способствовало бы устранению указанных недостатков, но на самом деле это не так. Во-первых, не удалось бы достичь значительного ускорения операций чтения, если не использовать механизм кластеризации данных (как это возможно в Oracle). А во-вторых, самое главное, это устранило бы и положительные ка-

чества: значительно снизилась бы производительность записи, причем сложность реализации возросла бы многократно.

Таким образом, очевидно, что решение задачи «в лоб» вряд ли даст удовлетворительные результаты. Она требует использования более сложных алгоритмов и структур данных, влекущих за собой реализацию множества непростых задач: поддержания целостности хранилища, управления параллельным доступом и пр. Именно для таких целей и существуют системы управления базами данных (СУБД). Поэтому, можно сказать, поиск решения на начальном этапе можно свести к вопросу: какую СУБД использовать?

КОРОТКО ОБ ИСТОРИИ РАЗВИТИЯ СУБД И СОВРЕМЕННЫХ ТЕНДЕНЦИЯХ. РЕШЕНИЕ - СПЕЦИАЛИЗИРОВАННАЯ СУБД

История развития систем управления базами данных насчитывает уже не один десяток лет. Благодаря неугасающему и по сей день интересу к этой теме, за прошедшие годы было проведено множество исследований, давших значительные результаты. Так, появление механизмов управления параллельным доступом, защиты целостности, эффективных алгоритмов и многое другое позволило превратить некогда примитивные наборы данных в огромные хранилища, обеспечивающие одновременную работу множества пользователей. Эти достижения были воплощены во всех развитых СУБД и стали их неотъемлемой частью.

Развитие СУБД происходило в соответствии с тем, как трансформировалась идеология моделирования предметной области. Изначально казалось достаточным представления любых данных в виде иерархически связанных сущностей. Очень скоро выяснилось, что хранение неиерархических связей столь же необходимо, и появилась сетевая модель данных. Примерно в то же время была разработана реляционная модель, в основе которой лежит математический аппарат, давший способ построения произвольных запросов к базе данных. В результате, в соответствии с этой моделью, были созданы реляционные СУБД, господствующие и по сей день.

Примерно с начала 80-х и вплоть до конца 90-х предпринимались активные попытки создания систем, способных заменить реляционные. Исследования проводились в направлении создания объектно-ориентированных систем, которые воспринимали бы описание данных в терминах объектно-ориентированной парадигмы, ставшей в итоге привычной для подавляющего большинства разработчиков благодаря своей естественности. В тот период считалось, что такие системы заменят реляционные, однако ввиду определенных причин этого не произошло, и уже созданные экспериментальные и даже коммерческие продукты такого класса стали считаться мертворожденными (не следует относить к ним объектно-реляционные системы, являющиеся, по сути, надстройками над реляционными).

Вместе с тем, господство реляционных СУБД так и не стало абсолютным, несмотря на все их положительные качества и заслуги. И тому есть веские основания, опровергающие, в том числе, и бытующее убеждение, что иерархические, сетевые, объектно-ориентированные и прочие системы (не являющиеся реляционными) не имеют шансов на существование.

Не будем рассматривать здесь одно из таких оснований, заключающееся в усложнении труда разработчика базы данных по причине постоянно возрастающих возможностей глубже и шире охватывать предметную область. С этой проблемой довольно успешно справляются объектно-реляционные «прослойки». В контексте данной статьи наиболее важным является иной аспект: более низкая производительность. И в некоторых случаях значительно более низкая.

Сравнительная форма была употреблена не случайно: опыт использования реляционных СУБД показал, что существует ряд специфичных задач, где они ведут себя хуже в сравнении с системами других классов. Например, на сегодняшний день в самых разных областях успешно применяются сетевые, объектно-ориентированные, пространственные и многие специализированные системы. Как минимум, причина кроется в том, что в реляционных СУБД механизмы хранения данных унифицированы и не позволяют использовать более эффективные для конкретного случая алгоритмы и структуры данных. Если же перейти к общим размышлениям на эту тему, то напрашивается вполне закономерное сомнение в приписываемом реляционным системам всемогуществе, поскольку вряд ли какая-либо система будет удовлетворять всем возможным требованиям пользователей.

Не стоит оставлять без внимания и тот факт, что реляционные СУБД, кроме всего прочего, утратили свою монополию на язык SQL. Этот язык стал стандартным вообще для всяких СУБД.

Итак, необходимо оценить возможные решения задачи хранения временных рядов. Сразу же возникает вопрос: а не подходят ли для этих целей реляционные системы? Действительно, они относительно дешевы и широко распространены, и если бы они были способны удовлетворить поставленным требованиям, то это было бы идеальным решением. Однако на сегодняшний день существует множество решений такой задачи, использующих именно реляционные СУБД. И для всех них характерна производительность, не удовлетворяющая предъявляемым требованиям. Еще один весомый недостаток - высокая избыточность данных. Например, опыт использования СУБД Interbase показал, что на одну запись приходится порядка 80 байт, причем полезную информацию несут всего 18 байт, т. е. ~22,5 % от всего размера записи, остальные байты содержат служебную информацию, используемую самой СУБД.

Сетевые СУБД также мало пригодны, поскольку в контексте рассматриваемой задачи, обладают теми же недостатками, что и реляционные. Если говорить об объектно-ориентированных системах, то дать однозначный ответ на вопрос об их пригодности не представляется возможным. Существует ряд свободно распространяемых систем, но они по разным причинам не подходят для решения поставленной задачи. Есть и развитые коммерческие продукты, однако они слишком слабо распространены, и судить об их пригодности в рамках поставленной задачи возможно только по их описанию.

Таким образом, наиболее приемлемым вариантом из всех можно считать создание специализированной системы. Сложность такого решения высока, ведь требуется реализация множества базовых механизмов, присущих всякой СУБД. Но, несмотря на это, оно позволяет использовать алгоритмы и структуры данных, оптимизированные для хранения именно временных рядов и с учетом характера поступления данных.

InfinityHistoryServer 3.0. История в деталях

ОБЩЕЕ ОПИСАНИЕ АРХИТЕКТУРЫ INFINITYHISTORYSERVER 3.0

В качестве решения задачи, рассматриваемой в настоящей статье, и с учетом описанных выше условий, был разработан продукт InfinityHistoryServer 3.0. Его создание, прежде всего, было обусловлено тем, что для его предшественников - InfinityHistoryServer 2.0 и InfinityHistoryServer 2.1, - использовавших для работы с хранилищем реляционную СУБД Interbase, характерна низкая производительность, особенно заметная при увеличении объема хранимых данных. Сразу следует заметить, что данная линейка продуктов создавалась в первую очередь для использования в сфере промышленной автоматизации, поэтому при проектировании InfinityHistoryServer 3.0 большое внимание уделялось минимизации вероятности потери регистрируемых данных в результате возможных сбоев компонентов системы или разрывов связи.

В результате, InfinityHistoryServer 3.0 был построен на абсолютно иных принципах. Во-первых, была реализована собственная специализированная СУБД, ориентированная на хранение данных временных рядов. Во-вторых, были полностью пересмотрены механизмы сбора данных.

В общем виде архитектура системы представлена на рис. 1. Она состоит из следующих компонентов:

- коллектор - компонент, выполняющий сбор данных от источников;
- сервер - компонент, непосредственно управляющий хранилищем и предоставляющий данные клиентским приложениям.

С одним сервером может быть связано несколько коллекторов, каждый из которых может выполнять сбор данных от нескольких источников. Все указанные компоненты системы (включая источники данных и клиентские приложения) могут располагаться как на одном компьютере, так и раздельно.

Рассмотрим более детально связи между компонентами системы. Коллекторы сохраняют данные, полученные от источников, в файловый буфер. За счет этого значительно повышается надежность сбора данных, поскольку коллекторы могут сохранять их в буфер даже при отсутствии связи с сервером. Буфер не является временным, поэтому даже если произошла перезагрузка коллектора (штатная или в результате сбоя), в нем остаются все данные, еще не переданные на сторону сервера. Более того, наличие буфера также исключает и потери данных в случаях пиков интенсивности их поступления за счет сглаживания нагрузки на сервер. Устройство файлового буфера обеспечивает максимальную производительность операций записи и чтения и практически не требует ресурсов компьютера, что позволяет устанавливать коллекторы совместно с источниками данных - на том же компьютере. Испытания показали, что коллектор способен сохранять до полутора миллионов (1 500 000) записей в секунду, если говорить об оперативных технологических данных (не строкового типа). Больший размер файлового буфера обеспечивает более длительную работу коллектора в автономном режиме. Например, если в среднем за месяц накапливается порядка ста миллионов (100 000 000) записей (если говорить опять же об оперативных технологических данных), то буфера размером 2 Гб будет достаточно на 1 месяц.

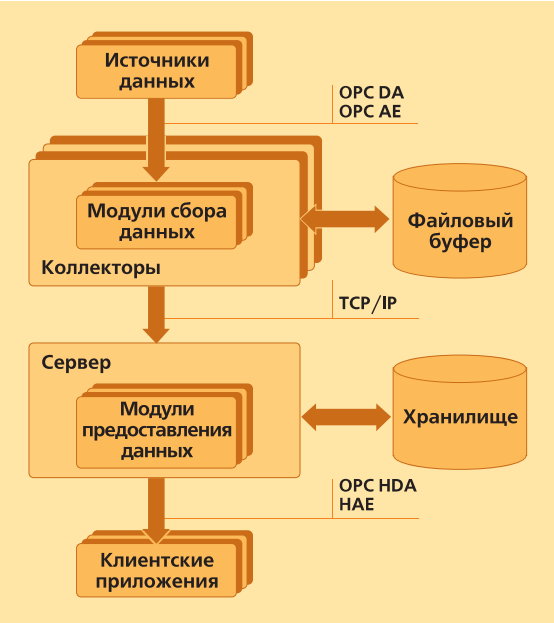


Рис 1. Архитектура системы

Сервер непосредственно управляет хранилищем и предоставляет данные клиентским приложениям, т. е. выполняет функции СУБД.

Коллектор и сервер построены по модульному принципу. Так, коллектор включает в себя модули, обеспечивающие сбор данных от источников различных типов. Точно так же и сервер включает в себя модули, предоставляющие данные по различным интерфейсам. Устройство же хранилищ (и файлового буфера на стороне коллектора, и основного хранилища) является общим для данных, полученных от источников различных типов. В настоящее время поддерживается сбор данных от источников, поддерживающих интерфейсы OPC DA и OPC AE, и, соответственно, сервер предоставляет данные по интерфейсам OPC HDA и HAE (Historical Alarms & Events, см. об этом далее).

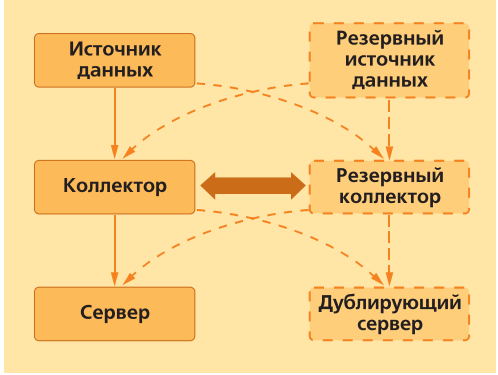
РЕЗЕРВИРОВАНИЕ В INFINITYHISTORY-SERVER 3.0

В InfinityHistoryServer 3.0 поддержано резервирование:

- соединений с источниками данных, с которых осуществляется сбор;
- коллекторов;
- серверов.

На рис. 2 видно, что коллектор может выполнять сбор данных от резервируемых источников. Более того, для коллектора может быть определен резервный коллектор, который будет выполнять сбор данных от тех же источников. Резервируемые коллекторы постоянно обмениваются друг с другом информацией о том, какие из источников данных доступны для них в данный момент, на

Рис 2. Сбор данных о резервируемых источниках



основании чего формируют маркер активности, определяющий, по каким источникам каждый из коллекторов должен выполнять сбор. Это необходимо для исключения дублирования сохраняемой информации. Если один из коллекторов выходит из строя, оставшийся активирует сбор данных по всем доступным для него источникам. Дублирование же самих серверов означает, что коллекторы передают данные не одному, а всем дублируемым серверам.

Такая схема резервирования, в случае ее полного использования, сводит к минимуму вероятность потери данных.

УСТРОЙСТВО ХРАНИЛИЩА

То, как устроено хранилище, проиллюстрировано на рис. 3. Оно состоит из трех частей:

- служебные файлы - файлы, используемые для внутренних нужд СУБД;
- файлы фронтальных данных - файлы, используемые для временного хранения поступающих данных;
- файлы данных.

Файлы данных разбиваются по времени в соответствии с заданным пользователем параметром, с точностью до часа. Так пользователь может указать разбиение на 8-часовые, 24-часовые (суточные), недельные и т.п. файлы. Также указываются параметры, определяющие длительности хранения данных:

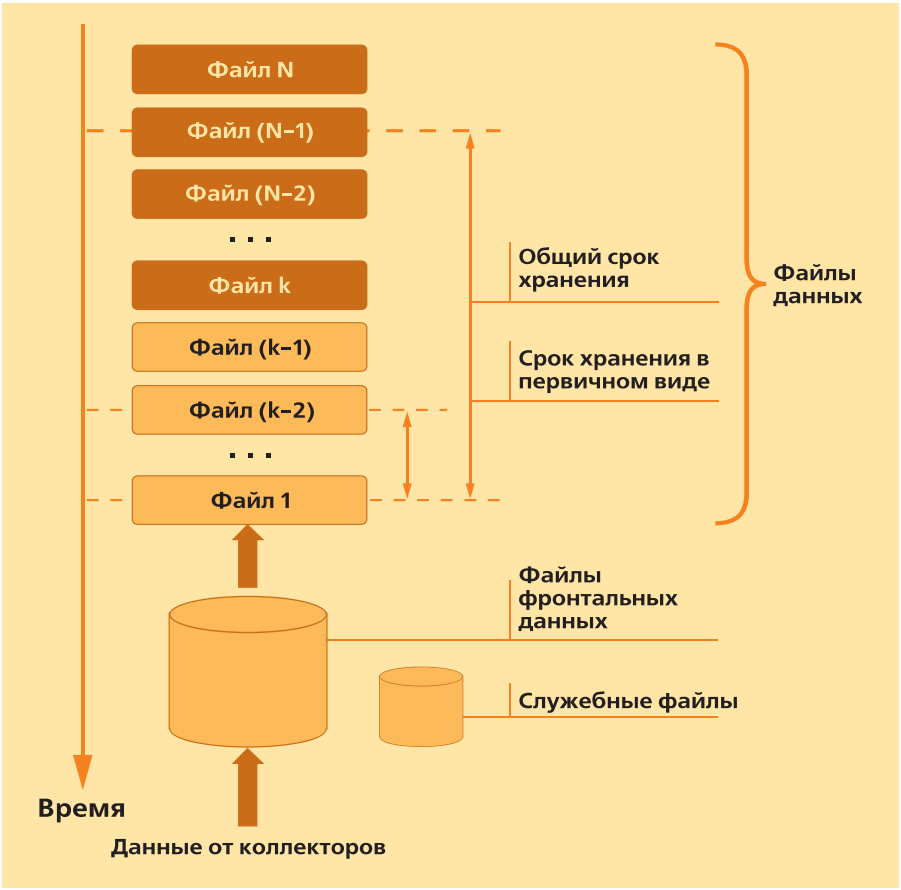
- длительность хранения в первичном виде;
- общая длительность хранения.

В соответствии с этими параметрами файлы данных разделяются на первичные и архивные. Первичные файлы допускают вставку записей (например, записей, полученных «задним числом»). Затем, по истечении срока хранения в первичном виде, они преобразуются в архивные. Архивные файлы оптимизированы для выполнения только операций чтения и не допускают вставок. Первичные и архивные файлы идентичны с точки зрения клиентских приложений, выполняющих чтение. Иными словами, такое разбиение является прозрачным для клиентов. В соответствии с параметром, определяющим общий срок хранения данных, выполняется процедура очистки базы: те файлы, для которых истек общий срок хранения, удаляются.

Процедура архивации и очистки запускается сервером автоматически при обнаружении такой необходимости. Так, на рис. 3 показано, что для первичного файла под номером (k - 2) истек срок хранения в первичном виде, и он будет преобразован в архивный; для файла же под номером N истек общий срок хранения, и он будет удален.

Сервер обеспечивает автоматическое обнаружение нарушений целостности и восстановление базы данных в случае сбоев.

Рис 3. Устройство хранилища



КОНФИГУРИРОВАНИЕ

Конфигурирование InfinityHistoryServer 3.0 осуществляется с помощью приложения «Конфигуратор», которое может быть установлено совместно с одним из коллекторов или серверов, либо отдельно от них.

Процесс конфигурирования начинается с задания расположения компонентов системы - коллекторов и серверов. После этого пользователь определяет, с каких источников и какую информацию коллекторы должны собирать. При этом элементы адресных пространств источников данных связываются с элементами адресного пространства InfinityHistoryServer 3.0, которые затем становятся доступными по интерфейсам OPC HDA и HAE.

Для облегчения процедуры конфигурирования реализована функция импорта адресного пространства источника данных.

Вся конфигурация системы хранится рядом с «Конфигуратором». Применение внесенных изменений осуществляется по запросу пользователя. При этом «Конфигуратор» передает обновления серверам, а затем сами серверы рассылают коллекторам соответствующие им части конфигурации. Таким образом, пока администратор выполняет редактирование, система продолжает работать на последней сохраненной версии конфигурации до того момента, пока администратор не решит применить обновления.

На рис. 4 показано главное окно «Конфигуратора». В нем показана конфигурация, в которой имеется коллектор с именем PlantCollector, собирающий данные от источников DAServer и AEServer. В области «Расположение серверов» указано, на каких компьютерах установлены дублируемые серверы. В области справа отображено дерево, представляющее адресное InfinityHistoryServer 3.0.

InfinityHistoryServer 3.0. История в деталях

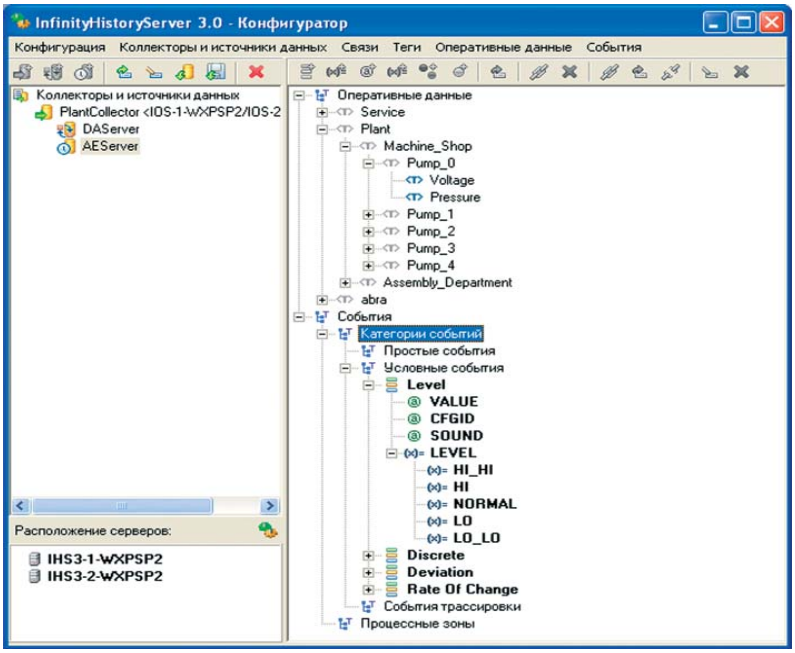


Рис 4. Главное окно конфигуратора

МЕТКА ВРЕМЕНИ, ПОДДЕРЖИВАЕМЫЕ ТИПЫ ДАННЫХ

InfinityHistoryServer 3.0 поддерживает метку времени с точностью 100 наносекунд. При этом важно, что по умолчанию данные сохраняются с меткой времени, полученной от источников данных, что способствует минимальному искажению информации о наблюдаемом процессе. На случай, когда нет гарантии того, что источник не искажает метку времени, существует опция, предписывающая коллектору сохранять данные с его собственным временем. Данная опция может быть настроена в отдельности для каждого из параметров, по которым ведется история, так как источник может являться посредником между коллектором и другими источниками.

Для хранения истории значений поддерживаются следующие типы:

- булевские значения;

мзнаковые и беззнаковые целые числа размером 1, 2, 4 и 8 байт;

- вещественные числа размером 4 и 8 байт;
- дата/время;
- строки;
- массивы произвольной вложенности, состоящие из значений перечисленных выше типов.

При этом параметр, по которому ведется история, не связывается жестко с каким-либо из типов. Значения сохраняются в том виде, в каком они получаются от источника. Например, если по некоторому параметру сначала было получено значение типа int, а затем значение типа double, то они так и сохранятся - система не будет преобразовывать их к другому типу.

InfinityHistoryServer 3.0 способен сохранять строки неограниченной длины. При этом для их хранения используется формат Unicode.

Что касается хранения истории событий, то каждая запись здесь представляет структуру, содержащую информацию о событии: источник события, категория, важность, текст сообщения, список атрибутов и т. д. Структура состоит из полей, которые могут иметь значения перечисленных выше типов.

ПОКАЗАТЕЛИ ПРОИЗВОДИТЕЛЬНОСТИ

Показатели производительности подобных систем принято оценивать количеством записей, сохраненных или прочитанных за секунду, причем предполагается, что записи имеют приблизительно один и тот же размер. Поскольку InfinityHistoryServer 3.0 может хранить записи самой разной длины, имеет смысл привести результаты тестов, проведенных на данных скалярных типов, перечисленных в предыдущем пункте. В среднем размер записей, соответствующих этим типам, составляет порядка 18 байт. Сразу нужно заметить, что производительность, выраженная в байтах, прочитанных или записанных за секунду, при увеличении среднего размера записи, возрастает. Поэтому следует считать, что производительность записи и чтения строк и структур, выраженная в байтах в секунду, будет не хуже производительности при работе с простыми скалярными типами.

Итак, как говорилось ранее, коллектор способен сохранять записи с производительностью до 1,5 млн (1 500 000) записей в секунду. Этого достаточно для того, чтобы выдержать значительные пики интенсивности поступающих данных. Сервер же, по результатам испытаний, продемонстрировал способность выполнять запись с производительностью до 200 тыс. (200 000) записей в секунду. При полной загрузке этот показатель варьировался в пределах от 70 до 150 тыс. (70 000-150 000). Производительность чтения данных, не находящихся в кэше, колебалась в пределах от 50 до 250 тыс. (50 000-250 000) записей в секунду. Если же читаемые данные обнаруживались в кэше, то запрос мог исполняться с производительностью до 1.3 млн (1 300 000) записей в секунду.

Испытания проводились на компьютере с двухядерным процессором Pentium-4 3.2 GHz и жестким диском на интерфейсе SATA-II со скоростью обращения 7200 rpm.

НАПРАВЛЕНИЯ ДАЛЬНЕЙШЕГО РАЗВИТИЯ

В настоящее время ведется реализация поддержки языка SQL. Это придаст InfinityHistoryServer 3.0 максимальную доступность хранимых им данных.

Среди других намеченных направлений развития можно выделить следующие:

- сохранение устаревших данных на CD/DVD;
- хранение произвольных записей, структура которых определяется пользователем.

ЗАКЛЮЧЕНИЕ

С помощью InfinityHistoryServer 3.0 можно создавать производственные архивы всего предприятия. Демонстрируемые им характеристики производительности позволяют быстрее и больше обращаться к хранимым историческим данным, что способствует значительному расширению возможностей проведения ретроспективного анализа и, соответственно, большему обеспечению управленческого и технического персонала информацией для принятия решений.