Arenadata™ Database

Версия - v6.13.0-arenadata12

Архитектура кластера ADB

Оглавление

В документе описываются основные компоненты Arenadata DB и архитектура системы. Раздел рекомендуется к прочтению перед переходом к непосредственной установке системы.

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io

Глава 1

Введение

Arenadata DB (**ADB**) хранит и обрабатывает большие объемы данных, распределяя нагрузку на все сервера в кластере. Логически база данных в **ADB** представляет собой массив отдельных инстансов баз данных **PostgreSQL**, расположенных на серверах кластера. Мастер (инстанс PostgreSQL, расположенный на мастер-сервере) координирует рабочую нагрузку на другие инстансы (сегменты), расположеные на серверах-сегментах, которые занимаются обработкой и хранением данных. Сегменты обмениваются данными друг с другом и с мастером через интерконнект при помощи одной или нескольких сетей. Каждый основной сегмент в кластере может иметь сегмент-зеркало, постоянно синхронизированный со своим основным сегментом и расположенный на другом сегмент-сервере. Сегмент-зеркало хранит те же самые данные, что и его парный основной, и включается в работу в случае, если основной сегмент по какой-то причине работать не может.

ADB – это программное решение, аппаратное обеспечение и программное обеспечение базы данных не связаны. **ADB** работает на любых x86-совместимых серверах, удовлетворяющих минимальным системным требованиям.

Поскольку база данных распределена между несколькими машинами, правильный выбор и конфигурация оборудования имеет важное значение для достижения наилучшей производительности.

Important: Важно помнить, что, как и любое MPP-решение, ADB работает со скоростью самого медленного из своих сегментов, поэтому все сервера-сегменты в кластере должны иметь одинаковые характеристики

Глава 2

Мастер

Мастер – управляющий инстанс **PostgreSQL** в кластере. Мастер является точкой входа в систему базы данных **ADB**. Он принимает клиентские соединения и обрабатывает команды SQL, которые передают ему пользователи системы. Пользователи подключаются к **ADB** через мастера с помощью совместимой с **PostgreSQL** клиентской программы, например, **psql** или любое ODBC-совместимое приложение.

Мастер содержит системный каталог (набор системных таблиц, содержащих метаданные о **ADB**), однако мастер не содержит никаких пользовательских данных. Данные хранятся только на сегментах. Мастер аутентифицирует клиентские соединения, обрабатывает входящие команды SQL, распределяет рабочую нагрузку между сегментами, координирует результаты, возвращаемые каждым сегментом, и представляет конечные результаты для клиентской программы.

Поскольку мастер не содержит никаких пользовательских данных, он имеет небольшую нагрузку на диск. При этом мастер нуждается в быстром, выделенном ЦПУ для загрузки данных, обработки соединений и планирования запросов.

2.1 Запасной мастер

При необходимости можно развернуть резервную копию (или зеркало) главного мастера. Запасной мастер находится в режиме ожидания и берет на себя функции мастера в случае, если основной хост становится неработоспособным. Резервный мастер можно развернуть на назначенном хосте или на одном из хостов сегмента.

Резервный мастер содержит все необходимые для работы данные благодаря синхронизации между основным и запасным инстансами. Если основной мастер выключается из работы, процесс репликации журнала завершается, и администратор может инициализировать резервный мастер заново. Во время работы резервного мастера реплицированные журналы используются для восстановления состояния основного мастера.

Поскольку мастер не содержит никаких пользовательских данных, необходимо синхронизировать только таблицы системного каталога. Когда данные таблицы обновляются, изменения автоматически копируются на резервный мастер, и поэтому он всегда синхронизирован с основным.

Глава 3

Сегменты

3.1 Основные сегменты

Сегменты хранят данные и обрабатывают большинство запросов. Пользовательские таблицы и их индексы распределяются по доступным сегментам в **ADB**.

Сегменты – это инстансы **PostgreSQL**. Пользователи не взаимодействуют напрямую с сегментами в **ADB**, а делают это через мастера. Количество сегментов на сервере-сегменте определяется количеством потоков ЦПУ или доступной памятью. Количество сегментов на сервере-сегменте задаётся при инициализации базы и является очень важным параметром при работе **ADB**.

3.2 Избыточные сегменты

При инициализации базы данных **ADB** можно настроить зеркальные сегменты (сегмент-зеркало, mirror), которые принимают на себя нагрузку в случае, если основной сегмент становится недоступен.

Important: Использование ADB без настроенного механизма зеркалирования возможно, но крайне не рекомендуется. При этом зеркальный и основной сегменты должны находиться на разных хостах

Существует две схемы создания зеркальных сегментов. При конфигурации по умолчанию все зеркальные сегменты одного хоста помещаются на другой. При втором варианте для каждого сегмента основного хоста зеркала распределяются на остальных хостах. Данный метод требует, чтобы хостов в системе было больше, чем сегментов на одном хосте. На хостах с несколькими сетевыми интерфейсами первичный и зеркальный сегменты распределяются поровну между интерфейсами.

3.3 Восстановление сегментов

При включенном зеркалировании система автоматически переключается на зеркальный сегмент, если основная копия становится недоступной. В случае если сегмент или хост выходит из строя, **ADB** может оставаться в рабочем состоянии при условии, что все части данных доступны на оставшихся активных сегментах.

Если мастер не может подключиться к сегменту, он отмечает в каталоге, что данный сегмент недействителен. Сегмент остается недействительным и не работает до тех пор, пока администратор не возвращает его онлайн. Процесс восстановления копирует пропущенные во время нерабочего состояния сегмента изменения.

При неподключенном зеркалировании в случае, если сегмент становится недействительным, система автоматически отключается. При этом администратору необходимо восстановить все сегменты прежде, чем продолжатся операции.

3.4 Пример конфигурации сегмента и хоста

Независимо от выбранной аппаратной платформы, узел обработки данных **ADB** обычно конфигурируется в соответствии с последующим описанием в данном разделе.

Important: Производительность базы данных соответствует скорости самого медленного сервера. Рекомендуется, чтобы все хосты имели идентичные аппаратные ресурсы и конфигурации

Хосты сегментов должны быть предназначены только для операций с базой данных **ADB** – не следует размещать другое ПО на тех же серверах. Для получения максимальной производительности необходимо, чтобы **ADB** не конкурировала с другими приложениями или сетевыми ресурсами.

Количество эффективных процессоров на хосте является основой для определения количества сегментов на хост.

В зависимости от выбранной аппаратной платформы конфигурации **RAID** предлагают различные уровни мощности и производительность.

Глава 4

Интерконнект

Интерконнект представляет собой сеть (или несколько сетей), предназначенную для взаимодействия мастера и сегментов между собой. Когда пользователь подключается к базе данных и запускает SQL запрос, на каждом из сегментов создаются процессы для обработки данного запроса. Интерконнект является как связью сегментов внутри одного хоста, так и между сегментами на разных серверах.

В качестве интерконнекта крайне желательно использовать одно или несколько 10-гигабитных Ethernet-подключений. По умолчанию интерконнект базы данных **ADB** использует протокол **UDP** (**User Datagram Protocol**) с управлением потока данных для отправки сообщений по сети. Программное обеспечение **ADB** выполняет дополнительную проверку пакетов, не выполненную **UDP**, поэтому надежность передачи эквивалентна **TCP** (**Transmission Control Protocol**), а производительность и масштабируемость значительно превосходят показатели **TCP**.

4.1 Избыточность сетей интерконнекта

Избыточность соединения может быть достигнута путем развертывания двух коммутаторов *10 Gigabit Ethernet* на сети и дополнительным 10-гигабитным подключением к мастер-серверу и сегмент-серверам.

4.2 Конфигурация сетевого интерфейса

Сегмент-сервер обычно имеет несколько сетевых интерфейсов, предназначенных для трафика межсетевого соединения **ADB**. Мастер-сервер обычно имеет вспомогательные внешние сетевые интерфейсы в дополнение к интерфейсам, используемым для внутреннего трафика.

В зависимости от количества доступных интерфейсов необходимо распределить межсетевой трафик интерконнекта по доступным интерфейсам. Это достигается путем назначения сегментов конкретному сетевому интерфейсу и равномерным распределением сегментов между интерфейсами.

Для этого создаются отдельные адреса хостов для каждого сетевого интерфейса. Например, если хост имеет четыре сетевых интерфейса, тогда у него будет четыре соответствующих адреса хоста, каждый из которых будет представлять один или несколько экземпляров первичного сегмента. Файл */etc/hosts* необходимо настроить так, чтобы он содержал не только имя хоста каждой машины, а также все адреса интерфейсов для всех узлов базы данных **ADB** (мастер, резервный мастер, сегменты и хосты **ETL** в случае, если используется **gpfdist**).

4.3 Настройка коммутатора

При использовании нескольких коммутаторов *10 Gigabit Ethernet*, необходимо равномерно разделить количество подсетей между каждым коммутатором.

Для примера конфигурации при наличии двух коммутаторов сетевые платы 1 и 2 на каждом хосте используют коммутатор 1, а сетевые платы 3 и 4 используют коммутатор 2. Для мастер-сервера имя хоста, связанное с первой сетевой платой (и, следовательно, с использованием коммутатора 1), является именем мастер-сегмента. Поэтому при активации резервного мастера, он должен быть подсоединён к сетевому интерфейсу, который использует коммутатор, не подключенный к основному мастеру.

Глава 5

Хосты ETL для загрузки данных

ADB поддерживает быструю параллельную загрузку данных с помощью функции внешних таблиц. Используя внешние таблицы в сочетании с параллельным файловым сервером (**gpfdist**), удаётся получить максимальное распараллеливание и наивысшую пропускную способность.

Одним из преимуществ использования программы файлового сервера **gpfdist** является то, что он гарантирует, что все сегменты в системе базы данных полностью используются при чтении данных таблицы из внешних файлов. Программа **gpfdist** может обслуживать данные в объектах сегмента со средней скоростью около *350 МБ/с* для файлов с разделителями текста и *200 МБ/с* для файлов в формате *CSV*. В связи с этим, следует рассмотреть следующие параметры при запуске **gpfdist**, чтобы максимизировать пропускную способность сети систем **ETL**:

- Если сервер **ETL** настроен с несколькими сетевыми платами, необходимо запустить один объект **gpfdist** на хосте **ETL** и затем задать местоположение внешней таблицы так, чтобы имя хоста каждой сетевой платы было объявлено в соответствующей части параметра *LOCATION*. Это позволяет сетевому трафику между кластером **ADB** и сервером **ETL** одновременно использовать все сетевые платы.
- Запустить несколько объектов **gpfdist** на хосте **ETL** и разделить файлы внешних данных одинаково на каждом объекте. Например, если есть система **ETL** с двумя сетевыми платами (**NIC**), то можно запустить два экземпляра *gpfdist* на этом аппарате, чтобы максимизировать производительность загрузки. Это достигается путем равномерного деления данных между двумя программами **gpfdist**.

Глава 6

Мониторинг производительности ADB

ADB включает в себя специальную базу данных управления и мониторинга системой, называемую *gpperfmon*. Когда эта база данных включена, агенты на каждом сервере кластера собирают показатели системы и данные о статусе запроса. С регулярными интервалами (обычно каждые 15 секунд) мастер-агент запрашивает данные от агентов сегмента и обновляет базу данных *gpperfmon*.

Пользователи могут сделать запрос в базу данных *gpperfmon* для просмотра собранных данных.

Глава 7

Управление ресурсами СУБД

Ресурсные группы перекладывают разграничение ресурсов на ядро ОС **Linux** с его механизмом *cgroups*. По факту, SQL-команды управления ресурсными группами являются обертками над виртуальной файловой системой */sys/fs/cgroup* хостов кластера **ADB**. Из всех возможных ресурсов **ADB** умеет управлять через ресурсные группы только памятью и процессором с помощью подкаталогов *cgroups*: memory, cpu и cpuset. При этом ресурсы процессора можно распределять как по полосе загрузки в процентах (cpu), так и по ядрам между различными группами (cpuset).

Логика управления ресурсами напоминает конструкцию из ящиков, вложенных друг в друга (Рис.7.1.).

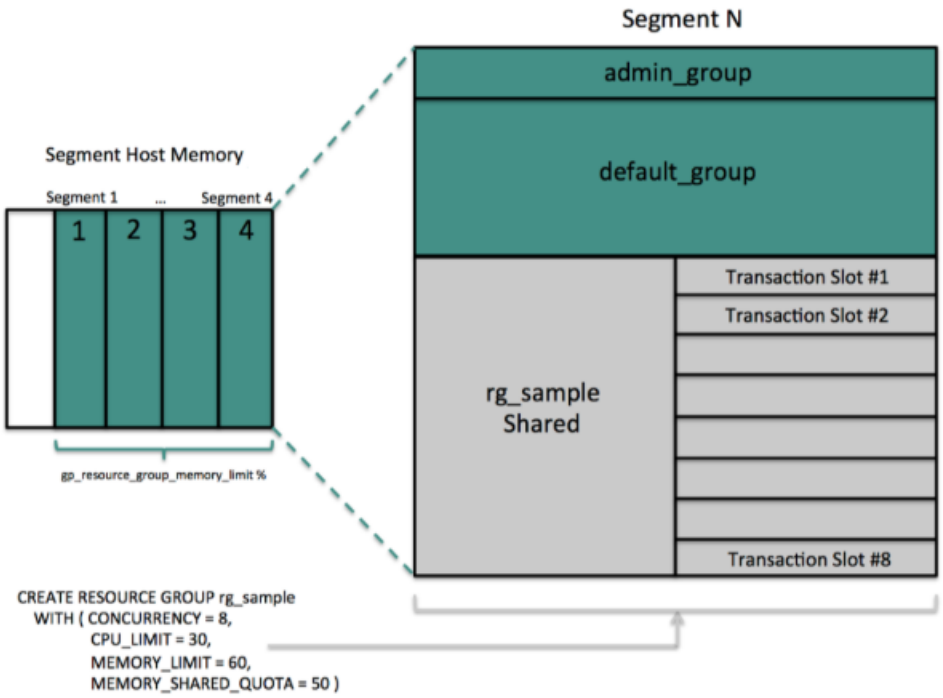

Рис.7.1.: Логика управления ресурсами

7.1 Управления памятью

Управление памятью делится по уровням разграничения.

Первый уровень разграничения ресурсов памяти определяется параметром *gp_resource_group_memory_limit*. Параметр указывает ядру ОС, сколько процентов памяти на каждом хосте принадлежит запущенным процессам **ADB**. Указанная память делится поровну между всеми сегментами на хосте. На рисунке в блоке "Segment Host Memory" показана вся память хоста, где зеленым отмечена память под **ADB**, поровну разделенная между четырьмя сегментами.

Important: При установке значения параметра важно оставить память для самой ОС. При этом процессы ADB не могут получить больше памяти, чем задано. Так же никакие другие внешние процессы не имеют возможности забрать память из зарезервированной для ADB

Следующий уровень вложенности – память, выделенная отдельному сегменту. В "ящик" памяти блока "Segment N" на рисунке вкладываются "ящики" поменьше, каждый из которых является отдельной ресурсной группой. На картинке приведено три ресурсные группы – *admin_group*, *default_group* и *rg_sample*.

Important: Процент памяти ресурсной группы задается параметром *memory_limit*, который всегда меньше или равен *100%*. Сумма *memory_limit* по всем ресурсным группам так же не должна превышать *100%* (в ином случае ADB выдает ошибку)

Параметр *memory_limit* гарантирует, что ресурсная группа получает свой процент памяти. В случае если сумма параметров *memory_limit* по всем ресурсным группам менее *100%* (т.е. в наличии имеется свободная память), то в момент выполнения тяжелых запросов любая ресурсная группа может позаимствовать данную свободную память.

В примере на картинке ресурсной группе *rg_sample* выделено *60%* памяти от доступной сегменту, у которого *25%* памяти (так как память делится поровну между сегментами) от *gp_resource_group_memory_limit*, составляющей *80%* от всей памяти хоста (5 равных секций в блоке "Segment Host Memory").

Для предотвращения активной утилизации памяти тяжелыми запросами можно указать параметр *memory_spill_ratio*, который не позволяет запросу использовать более указанного процента памяти от доступного ресурсной группе. В результате, если планировщик видит, что запрос должен потребить более *memory_spill_ratio* процентов памяти в ресурсной группе, то его данные сбрасываются на диск. При этом статистика должна быть актуальной во избежание ошибки планировщика (иначе запрос отменяется из-за нехватки памяти). Интересная особенность в том, что *memory_spill_ratio* можно менять внутри сессии.

7.2 Управление ЦПУ

Существует два взаимодополняющих друг друга сценария управления ресурсами ЦПУ через ресурсные группы: по ядрам или по полосе пропускания.

Если обрабатываются долгие тяжелые запросы в ресурсной группе малым количеством одновременных потоков, а в других ресурсных группах выполняется много более легковесных запросов, то на ядрах процессора, обслуживающих тяжелые запросы, происходят частые переключения контекста. Это сильно замедляет производительность и для такой ресурсной группы можно зарезервировать определенные ядра ЦПУ с помощью *cpuset*. В результате только данная ресурсная группа использует указанные ядра (их номера должны присутствовать на всех хостах). Для остальных ресурсных групп данные ядра недоступны даже в случае их простаивания – поэтому следует резервировать ядра только в крайних случаях и в минимальном количестве. Так же следует понимать, что планировщик ОС может использовать ядра и для других программ, а не только для СУБД **ADB**.

Возможно выделять полосу загрузки ЦПУ. Существует параметр *gp_resource_group_cpu_limit*, указывающий максимальный процент ресурсов ЦПУ, который может потребить **ADB** со всеми ресурсными группами внутри. Он используется в первую очередь для того, чтобы защитить саму ОС и другие программы на хосте при высоких нагрузках **ADB**.

Important: Рекомендуется не выставлять параметр *gp_resource_group_cpu_limit* выше значения *0.9* (*90%*)

Для управления полосой загрузки ЦПУ для ресурсной группы используется параметр *cpu_rate_limit*. При этом сумма *cpu_rate_limit* по всем ресурсным группам не должна превышать *100%*. Если в других ресурсных группах нет выделенных ядер (*cpuset*), то *cpu_rate_limit* показывает доступный данной ресурсной группе процент от присущих базе данных ресурсов ЦПУ на хосте (*gp_resource_group_cpu_limit*). В случае если для части ресурсных групп используются выделенные ядра, то доступные ресурсы определяются как минимальное из двух значений:

- Незарезервированные ядра ЦПУ / все ядра ЦПУ;
- *gp_resource_group_cpu_limit*.

7.3 Пример

Для примера используется профиль нагрузки на кластер, который единовременно загружен:

- 50 “легкими” запросами SQL1, извлекающими единственную строку по первичному ключу;
- 20 “средними” запросами SQL2, содержащими 2-3 соединения и возвращающими несколько десятков строк;
- 2 “тяжелыми” запросам SQL3, считающими витрины со множеством соединений и по большим объемам данных (минимизация времени выполнения критична);
- 1 “сверхтяжелый” запрос SQL4, загружающий данные из ETL системы через *gpfdist*.

Тогда можно использовать разделение ресурсов в кластере, представленное в таблице.

Таблица7.1.: Пример разделения ресурсов в кластере

Ресурсная группа	Запросы	concurrency	cpu rate limit	cpuset	memory limit	memory shared quota	memory spill ratio
g1	SQL1	50	35	–	20	20	20
g2	SQL2	20	25	–	15	10	20
g3	SQL3	2	–	1-16	40	0	40
g4	SQL4	1	30	–	15	0	20
default_group	–	20	5	–	5	50	20
admin_group	–	10	5	–	5	50	20

Для “тяжелых” запросов SQL3 с перестройкой витрин используются выделенные ядра *1-16* на каждом хосте (из *72* физических в наличии), что минимизирует переключения контекста на них и сильно увеличивает скорость запроса под высококонкурентной нагрузкой. Для “сверхтяжелого” запроса SQL4 нет необходимости в выделенных ядрах, так как при загрузке данных из ETL переключение контекста не оказывает сильного влияния на производительность. А правильная стратегия с выделенными ядрами под ресурсную группу заключается в использовании данного решения только тогда, когда не остается других вариантов (выделенные ядра ЦПУ “невидимы” для остальных ресурсных групп).