

Conversación con Gemini

Exported: 15/6/2026 10:06:28

Jordi H.

Has dicho tengo un cluster de starrocks. hay operaciones que parecen inofensivas pero generan un gran stress en el cluster de backend. puedes indicarme alguna manera fácil de saber el nivel de stress de l backend? (alguna sentencia en la prompt del cliente de sql

Gemini

Entiendo perfectamente esa frustración. StarRocks es increíblemente rápido, pero esa misma potencia hace que una consulta mal planteada pueda "secuestrar" los recursos del cluster antes de que te des cuenta. Para monitorizar el estrés directamente desde tu cliente SQL sin recurrir a herramientas externas como Grafana, la mejor opción es consultar las **vistas de metadatos** del sistema. Aquí tienes las sentencias clave para medir el pulso de tu backend en tiempo r

1. El "Top" de Consultas (Carga Activa)

Esta es la forma más rápida de ver qué está pasando **ahora mismo**. Te permite identificar consultas que llevan demasiado tiempo ejecutándose o consumiendo mucha memoria. SQL

CODE

```
SELECT
query_id,
user,
host,
cpu_cost_seconds,
mem_used_bytes / 1024 / 1024 AS mem_used_mb,
definition
FROM information_schema.active_queries
ORDER BY cpu_cost_seconds DESC;
```

- En qué fijarse: Si ves `cpu_cost_seconds` muy altos o `mem_used_mb` que se acerca al límite de tus nodos, has encontrado al culpable

2. Estado de Salud de los Backends (BE)

Para saber si los nodos están sufriendo a nivel sistémico, consulta el estado de los Backends. SQL

CODE

```
SHOW BACKENDS\G
```

(Uso \G para que la salida sea vertical y más legible en terminal).

- Columnas críticas: `SystemDecommissioned`: Debe estar en false. `Alive`: Debe estar en true. Si ves un false, el nodo ha colapsado o se ha reiniciado por falta de memoria (OOM). `MemUsed` vs `MemLimit`: Si el uso de memoria está habitualmente por encima del 80%, cualquier consulta pesada estresará el cluste

3. Monitorización de Conexiones

A veces el estrés no es por una consulta pesada, sino por **demasiadas** consultas pequeñas que saturan el pool

de conexiones.SQ

CODE

```
SELECT user, host, count(*)
FROM information_schema.connection_id
GROUP BY user, host;
```

¿Cómo prevenir el estrés? (Tip de experto)

Si tienes operaciones que sabes que son peligrosas, puedes limitar su impacto antes de ejecutarlas usando variables de sesión. Esto evita que una sola "bala" tumbe todo el backend

- Limitar memoria por consulta: SET mem_limit = 8589934592; (ej. 8GB)
- Limitar tiempo de ejecución: SET query_timeout = 60; (60 segundos)

Nota: Si notas que el cluster se ralentiza de forma generalizada, revisa los procesos de **Compaction** (la limpieza interna de StarRocks) con SHOW PROC '/backends'; y mira la columna de TabletNum. Un número excesivo de tablets por nodo suele ser la raíz de muchos problemas de rendimiento. ¿Te gustaría que te ayude a crear una alerta específica o a analizar el plan de ejecución de alguna de esas consultas "inofensivas" para ver por qué estresan el sist

Jordi H.

Has dicho de momento el problema me lo he encontrado solamente en sentencias create materialized view. algun tip sobre el tema

Gemini

Es muy común. En StarRocks, las **Async Materialized Views (MV)** son especialmente potentes pero también "traicioneras", porque la creación inicial o el refresco actúan como una consulta INSERT INTO ... SELECT .. masiva que escanea tablas enteras. Aquí tienes varios tips específicos para domar el estrés cuando trabajas con M

1. Controla el paralelismo (El "freno de mano")

Por defecto, StarRocks intentará usar todos los recursos posibles para terminar la MV rápido. Puedes limitar cuántos hilos utiliza la consulta de la MV mediante variables de sesión justo antes de lanzarla:SQ

CODE

```
-- Reduce el paralelismo de la consulta (por defecto suele ser el número de cores)
SET pipeline_dop = 4;

-- Limita la memoria máxima que puede consumir esta operación específica
SET mem_limit = 17179869184; -- Ejemplo: 16GB
```

2. La trampa del "Refresh" inicial

Cuando haces un CREATE MATERIALIZED VIEW, si no especificas lo contrario, el sistema intenta calcularla inmediatamente. Si la tabla base es enorme, el estrés es instantáneo. **Tip:** Crea la MV sin datos y refresca manualmente en un momento de baja carga, o refresca solo una partición.S

CODE

```
CREATE MATERIALIZED VIEW mi_vista
REFRESH ASYNC
PROPERTIES (
  "initial_cache" = "false" -- No calcula los datos al momento de crearla
)
AS SELECT ... ;
```

3. Usa Particionamiento en la MV

Si tu tabla base está particionada (por ejemplo, por fecha), **asegúrate de que tu MV también lo esté**. Esto permite que StarRocks refresque solo los datos nuevos en lugar de re-escanear toda la historia

- Usa PARTITION BY en la definición de la MV.
- Esto reduce drásticamente el estrés en el backend durante los refrescos incrementales.

4. Monitoriza el estado del refresco

Si ya lanzaste la creación y quieres ver cómo va "por dentro" y cuánto está sufriendo el BE:SQL

CODE

```
-- Ver tareas de refresco activas
SELECT * FROM information_schema.task_runs
WHERE state = 'RUNNING';

-- Ver el progreso específico de la MV
SHOW MATERIALIZED VIEWS WHERE name = 'tu_nombre_de_vista'\G
```

5. Ojo con los Joins complejos

Si tu MV tiene Joins entre tablas muy grandes, StarRocks podría estar intentando hacer un **Shuffle Join** (moviendo millones de filas entre nodos del backend), lo que colapsa la red y la CPU

- Tip: Intenta que las tablas del Join compartan la misma distribución de claves (DISTRIBUTED BY) para que el Join sea "Colocated" (se resuelve localmente en cada nodo sin mover datos por la red)

6. Query Rewrite (Cuidado con las expectativas)

A veces el estrés viene porque el optimizador intenta usar una MV que aún se está construyendo o que es muy pesada. Puedes desactivar el reescrito de consultas temporalmente si ves que el planificador está tomando decisiones erróneas que estresan el backend:

CODE

```
SET enable_query_rewrite = false;
```

¿La MV que te está dando problemas hace algún Join o agregación muy pesada, o es sobre una tabla con

Jordi H.

Has dicho estoy usando la version 3.4 me puedes mostrar las queries que corresponden con esa version?

Para la versión **3.4 de StarRocks**, tienes acceso a vistas de sistema mucho más granulares. En esta versión se han mejorado las tablas de `information_schema` y `sys` para monitorizar el rendimiento. Aquí tienes las queries actualizadas y optimizadas específicamente para la versión 3.4 para detectar el estrés en el backend

1. Ver el "Stress" de CPU y Memoria por Nodo (BE)

En la 3.4, la tabla `sys.backends` es más cómoda para ver de un vistazo si algún nodo está al límite. SQL

CODE

```
SELECT
BackendId,
IP,
NumActiveTasks, -- Cuántas tareas está procesando ahora
MemUsedBytes / 1024/1024/1024 AS MemUsed_GB,
CpuUsedPercent, -- Porcentaje de CPU actual
MaxDiskUsedPct -- Si el disco se llena, el BE sufre mucho
FROM sys.backends;
```

2. Identificar el "Vampiro" de Recursos (Queries de MVs)

Cuando creas una MV, StarRocks lanza un **Task**. En la 3.4 puedes ver exactamente cuál está consumiendo más recursos en tiempo real: SQL

CODE

```
SELECT
query_id,
user,
start_time,
cpu_cost_seconds,
mem_used_bytes / 1024 / 1024 AS mem_mb,
(mem_used_bytes * 100 / execution_mem_limit) AS pct_mem_limit, -- % de la memoria asignada que ya usa
query_scope, -- Aquí verás si es una query de usuario o interna de la MV
definition
FROM information_schema.active_queries
WHERE definition LIKE '%CREATE MATERIALIZED VIEW%'
OR definition LIKE '%REFRESH MATERIALIZED VIEW%'
ORDER BY cpu_cost_seconds DESC;
```

3. Monitorizar el progreso de la Materialized View

Si has lanzado un `CREATE MATERIALIZED VIEW` y parece que se ha quedado "colgado" (generando estrés continuo), usa esto en la 3.4: SQL

CODE

```
SELECT
task_name,
job_id,
status,
progress, -- Porcentaje de avance
create_time,
error_message
FROM information_schema.task_runs
WHERE status = 'RUNNING'
ORDER BY create_time DESC;
```

4. Detectar "Write Quota" y Bloqueos (I/O Stress)

Las MVs escriben muchos datos. Si el backend está estresado por I/O, verás que las transacciones se acumulan:SQ

CODE

```
-- Ver si hay muchas transacciones de carga/escritura pendientes
SHOW PROC '/transactions';
```

Tips adicionales para StarRocks 3.4:

- **Spill to Disk:** En la versión 3.4, si una query (como el SELECT de una MV) se queda sin memoria, puede intentar hacer "spill" (volcar a disco). Esto salva al BE de un crash (OOM), pero dispara el estrés de I/O. Comprobación: Si ves que la CPU está alta pero la memoria está estable y el disco "vuela", es que está haciendo Spi
- **Resource Groups:** Si vas a crear MVs a menudo, en la 3.4 lo ideal es crear un Resource Group para limitar que las MVs no puedan usar más de, por ejemplo, el 40% de la CPU del cluster:SQ

CODE

```
CREATE RESOURCE GROUP mv_group
PROPERTIES (
  "cpu_core_limit" = "10",
  "mem_limit" = "20%"
);
-- Y asignar la creación de la MV a ese grupo.
```

¿Alguna de las MVs que estás creando tiene muchos Joins o usa funciones de ventana (OVER)? Esos son los dos motivos principales por los que el Backend de StarRocks suele sufrir picos de estrés en la 3.4
