Debugging for Shorter Downtimes

Presented by: ROB HAMEL, APRIL 13, 2010

hamel@pythian.com

Pythian

love your data

About Pythian

- Recognized Leader:

- Global industry-leader in database infrastructure services for Oracle, Oracle Applications, MySQL and SQL Server
- Currently work with 150 multinational companies such as Forbes.com, Fox Sports and Western Union to help manage their complex IT deployments

- Expertise:

- One of the world's largest concentrations of dedicated, full-time DBA expertise.

- Global Reach & Scalability:

- 24/7/365 global remote support for DBA and consulting, systems administration, special projects or emergency response

Who Am I?

Joined Pythian in Oct 2000 as DBA, Team Lead and currently Service Delivery Manager

DBA since 1993

Specialize in data architecture, performance tuning and scalability

Make my own maple syrup

Spend as much at my cottage as I can




Today's Agenda

What are we trying to achieve?

Why do you need to add your own instrumentation?

Default MySQL instrumentation

Typical troubleshooting checklist

Most common issues you will encounter

Simple instrumentation and how you use it

Gotcha's in 5.1.x

What Determines Effectiveness

How good you are at triage and getting to the real problem

You can achieve this with:

- Intelligent monitoring that just doesn't check thresholds or simply reporting if you encounter an actual error
- Good instrumentation that allows you to quickly determine what has changed from parameters to sudden or subtle changes in performance behaviour.

Why Add Your Own Instrumentation?

MySQL default instrumentation is inadequate

Your are not in control of everything

- Teams of DBAs, Developers and System Administrators
- 3rd party software patches or application upgrades
- MySQL patches and upgrades
- Configuration Management mistakes or weaknesses
- Operating Systems
- Hardware failure
- Sudden workload changes
- Malicious activities such as denial of services attacks
- Your memory, many servers, hard to keep track of changes

Lets face it its always the database's fault so you are guilty until proven innocent...so you better find out what happened ASAP

Default MySQL Instrumentation

Show global variables

Show global status

Status

Show innodb status \G

Error Log

Slow query log

Query profiler

Metadata on all database objects in the information schema

What's missing?

Typical Troubleshooting Checklist

Check the Error log

Check what is running (Show processlist)

Check Slow Query Log

Check the MySQL statistics (show global status)

Check OS logs and monitors

Check if something changed

- MySQL parameters
- DDL
- New queries
- Table growth
- New software release

Ask around to see if someone changed something

Common Issues

MySQL configuration discrepancies

DDL changes

Regularly scheduled jobs (cron/events)

New and/or changed queries

Workload changes

Lets break them down!!!

MySQL Configuration Discrepancies

Scenarios

- Changing current memory settings forgetting to change my.cnf
- Changes meant to a my.cnf on DB1 performed on DB2
- Applying MySQL patches or upgrades

Resolution

- Need to track historical changes
  - How far back do you need to go?
- Monitor for discrepancies

DDL Changes

Scenarios

- New tables
- DDL changes meant for DB1 applied to DB2
- Configuration Management
- 3rd party application database patches/upgrades

Resolution

- Need to track historical changes
  - How far back do you need to go?
  - What has been dropped is as important as what has been added

Regularly Scheduled Jobs

Scenarios

- Failures go unnoticed
- Don't run at all or removed from event scheduler or crontab
- Exceed prescribed run-time windows

Resolution

- Job Logging / tracking history
- Monitor for what is suppose to run rather then errors
- Monitor current executions against a baseline

New and/or Changed Queries

Scenarios

- Bad design
- Suboptimal Execution plans (Full table scans, bad index, File sorts)
- Locking

Resolution

- Let the DBAs validate the execution plans of all new queries
  - Its hard to fix a sub-optimal design after-the-fact
- Do proper testing with adequate data volumes and concurrency
- Track execution plans and workload of all queries historically
- Monitor key query performance against a baseline
  - Not every bad query is in the Slow-Query-Log
  - Not every query in the Slow-Query-Log is bad

Workload Changes

Scenarios

- Same query and an ever growing table
- Increase concurrency/execution of a query
- Database limitations
- Application configuration (connection pool, new workflow)
- Hardware failure or misconfiguration
  - Raid rebuilds, backups running, network, memory/cpu disappearing, etc

Resolution (You are not really fixing it but you can identify it)

- Track Database metrics (show global status) over time
- Track OS metrics (sar, vmstat, etc) over time
- Track query performance and workload over time
- Track object growth over time
- You are looking for pattern changes either sudden or incremental

Tracking Change Over Time

my.cnf

- Daily copy of the my.cnf file and keep the last 30 days
- Dump show global variables daily into a table and keep forever

DDL

- Mysqldump the structure only in a file and keep forever
- Copy the information schema into a table and keep it for X days

Problems with those approaches

- Large amount of data to sift through when you are troubleshooting wasting precious time
- Quantity of data to store especially with regards to DDL tracking. It can really add up

So how do we do it?

SCD Type-2 To The Rescue

Slowly Changing Dimension type-2

- Start with a baseline and only track changes
- Saves on space allowing you to keep data forever
- You can run it often for better granularity without consuming tons of space

What functionality will it give you

- What changed since X
- Historical change of parameter or table over time
- What was the configuration a given point in time
- What was removed in the last X days
- Current configuration or state of the DDL object
- And you can keep going

Consistent query structures no matter what object you are tracking

SCD-Type 2 – What Is Needed?

Identify

- Primary key (PK) of the original object
- Attributes to track

Additional attributes as meta-data

- Row_effective_date
- Row_end_date
- Current_flag

Develop a script or stored procedure to perform the work

Scheduled job through cron or event scheduler

Execute before and after planned maintenance activities

SCD-Type 2 - The Algorithm

For every object

- Compare against the current value in the SDC Type-2 via the PK
- If its new
  - Insert it SDC Type-2 setting the current_flag=1, row_effective_date=now() and row_end_date=max_date_in_the_future
- If its found but the attributes have changed
  - Update the SDC Type-2 setting the current_flag=0 and the row_end_date = now()
  - Insert a new row in SDC Type-2 with the new attributes setting the current_flag=1, row_effective_date=now() and row_end_date= max_date_in_the_future

For every current in SDC Type-2 not found in the object via the PK

- Update the SDC Type-2 setting the current_flag=0 and the row_end_date = now()

SCD-Type 2 - Example

Tracking show global variables over time

DDL of the SDC Type-2

```
Create table historical_global_variables (
variable_name varchar(100),
value varchar(1000),
row_effective_date datetime,
row_end_date  datetime,
current_flag tinyint);
```

- The real primary key is the original PK of the object + row_effective_date
- Obviously there is only one row per original object PK that has the current_flag set to 1

SCD-Type 2 - Intitial Population

Current Instance Values

variable_name	value
auto_increment_increment	1
auto_increment_offset	1
log	the.log
max_connections	80

Resulting SDC Type-2 Table

variable_name	value	row_effective_date	row_end_date	current_flag
auto_increment_increment	1	2010-04-01 10:00:00	2038-01-01 00:00:00	1
auto_increment_offset	1	2010-04-01 10:00:00	2038-01-01 00:00:00	1
log	the.log	2010-04-01 10:00:00	2038-01-01 00:00:00	1
max_connections	80	2010-04-01 10:00:00	2038-01-01 00:00:00	1

SCD-Type 2 - Change to 1 Variable

Previous Instance Values

variable_name	value
auto_increment_increment	1
auto_increment_offset	1
log	the.log
max_connections	80

Changed Instance Values

variable_name	value
auto_increment_increment	2
auto_increment_offset	1
log	the.log
max_connections	80

Resulting SDC Type-2 Table

variable_name	value	row_effective_date	row_end_date	current_flag
auto_increment_increment	1	2010-04-01 10:00:00	**2010-04-01 10:05:00**	**0**
auto_increment_offset	1	2010-04-01 10:00:00	2038-01-01 00:00:00	1
log	the.log	2010-04-01 10:00:00	2038-01-01 00:00:00	1
max_connections	80	2010-04-01 10:00:00	2038-01-01 00:00:00	1
auto_increment_increment	**2**	**2010-04-01 10:05:00**	**2038-01-01 00:00:00**	**1**

SCD-Type 2 - Deprecated Variable

Previous Instance Values

variable_name	value
auto_increment_increment	1
auto_increment_offset	1
log	**the.log**
max_connections	80

Changed Instance Values

variable_name	value
auto_increment_increment	2
auto_increment_offset	1
general-log	**node1.log**
max_connections	80

Resulting SDC Type-2 Table

variable_name	value	row_effective_date	row_end_date	current_flag
auto_increment_increment	1	2010-04-01 10:00:00	2010-04-01 10:05:00	0
auto_increment_offset	1	2010-04-01 10:00:00	2038-01-01 00:00:00	1
log	the.log	2010-04-01 10:00:00	**2010-04-01 10:05:00**	**0**
max_connections	80	2010-04-01 10:00:00	2038-01-01 00:00:00	1
auto_increment_increment	2	2010-04-01 10:05:00	2038-01-01 00:00:00	1
general-log	**node1.log**	**2010-04-01 10:10:00**	**2038-01-01 00:00:00**	**1**

SCD-Type 2 - Queries

Current state of values

```
Select variable_name, value
from historical_global_variables
where current_flag=1
```

Historical changes for auto_increment_increment

```
Select value, row_effective_date, row_end_date
from historical_global_variables
where variable_name='auto_increment_increment'
order by row_effective_date
```

Show configuration as of 2010-04-10 10:05:00

```
Select variable_name, value
from historical_global_variables
where '2010-04-10 10:05:00' between row_effective_date and row_end_date
order by variable_name
```

SCD-Type 2 - Queries

Show what's new since the 2010-04-10 10:05:00 maintenance window

```
Select variable_name, value
from historical_global_variables
where row_effective_date >='2010-04-10 10:05:00'  and current_flag=1
order by variable_name;
```

Show what's been deprecated/removed since the 2010-04-10 10:05:00 maintenance window

```
Select variable_name, value
from historical_global_variables
where row_end_date >='2010-04-10 10:05:00'
and variable_name not in
(select variable_name from historical_global_variables where current_flag=1)
```

Job Logging for Stored Procedures

Metadata

- job_master
- job_logs
- job_config

Functions and procedures

- add_job
- add_step
- update_step
- complete_job

How to use it

- Monitoring
- Troubleshooting

Job Logging - Metadata

```
create table job_master (
job_id bigint auto_increment primary key,
Job_name varchar(100),
timestamp timestamp ,
user_name varchar(30),
completion_status varchar(10),
Completion_timestamp timestamp)
engine=myisam;
```

```
create table job_logs (
job_id bigint,
step_id bigint auto_increment primary key,
action varchar(200),
start_time timestamp,
end_time timestamp,
elapsed_time int(11),
status varchar(10),
status_message varchar(80))
engine=myisam;
```

```
create table job_config (
job_name varchar(100) primary key,
min_successes int,
max_failures int,
baseline_exec_time bigint
);
```

Note that min_successes and max_failures have a daily granularity

Job Logging - Procedures

```
create function job_logging_add_job
    (p_job_name varchar(100))
    returns bigint

create function job_logging_add_step
    (p_job_id bigint,
    p_action varchar(100))
    returns bigint

create procedure job_logging_upd_step
    (p_job_id bigint,
    p_step_id bigint,
    p_status varchar(10),
    p_status_message varchar(80))

create procedure job_logging_complete_job
```

Job Logging - Example

```
create procedure sample_proc

begin
      declare v_error int default 1;
declare continue handler for sqlwarning, sqlexception,not found
begin
    call job_logging_upd_step(@job_id,@step_id,'error','state unavailable');

   end;

   set @job_name = 'sample_proc';
   set @job_id = job_logging_add_job(@job_name);
   set @s = 'update table x set col1=2 where id=3';
   set @step_id = job_logging_add_step(@job_id, @s);
   prepare stmt1 from @s;
   execute stmt1;
   deallocate prepare stmt1;
   call job_logging_upd_step(@job_id,@step_id,'ok','');
   ....
   call job_logging_complete_job(@job_id);

end;
```

Job Logging - Monitoring Usage

Identify which jobs ran successfully as often as they should have or didn't run at all by ensuring that actual_successes >= job_config.min_successes

Identify which jobs failed more than allowed by ensuring that actual_failures > job_config.max_failures

Identify which jobs that exceed the prescribe execution time by ensuring that (completion_timestamp – timestamp) > job_config.baseline_exec_time

Show how things went yesterday or over the weekend using select job_name, status, count(*) from job_masters where timestamp >= now() – interval '3' day group by job_name, status

Job Logging - Troubleshooting Usages

Performance of a job over time by querying the job_masters table for a specific job order by timestamp

View a complete job execution by querying job_logs order by step_id where job_id = ? (‘?’ is the instance of a job that you are interested in).

Compare a particular job step by querying job_logs where action = ‘?’ and job_id in (select job_id from job_masters where job_name=‘?’ and timestamp >= ‘?’)

Query Metrics Over Time

Track key query/process performance (executions, rows read, rows returned, execution plans, etc)

Monitor against baseline

Best done at the application level or through web logs

Track the end-user experience not just what happens at the database level

The only KPIs that really matter

Database and OS Metrics

It doesn't matter how you collected it

- Cacti or any other 3rd party
- Database
- File

Capture database and OS metrics as well as database object sizes with identical granularities

- Show global status
- Select owner, table_name, data_size, index_size from information_schema.tables
- sar/vmstat output

Looking for pattern changes

Can monitor against a baseline

Event Scheduler Considerations

Runs like cron so a job can lock on itself and kill the system if frequently executed

Add explicit locks in the your procedures to mitigate

Sample Code:

```
declare v_lock_results int;
declare v_lock_string varchar(100)  default concat (database(), v_job_name,
'_lock');
....

set v_lock=get_lock(v_lock_string,5);
If v_lock_results=0 then
      # you didn't get the lock exit gracefully

else

      #execute the rest of the procedure
set v_lock_results = release_lock(v_lock_string);
```

Partitioning Considerations

Locking

- MySQL will lock all partitions for a single insert/update/delete or partition operation
- Very bad for concurrency if you think to use hash or list partitioning is in other database platforms
- Executed DML in batches if you can

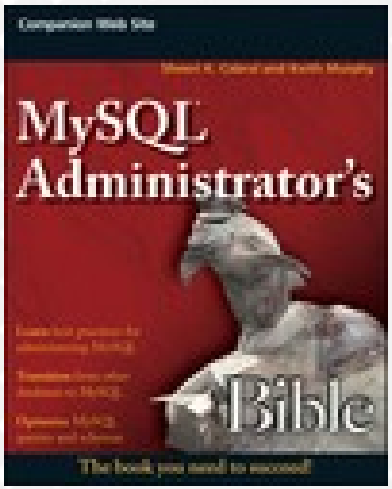
Partition Maintenance for Range Partitioning

- The most important task is to ensure that the maintenance is occurring and monitor for it by inserting a row in the future then removing it.
- Remember you only have 1024 partitions per table
- Never use the maxvalue partition since it gives you a false sense of security.

Thank You.

Win a signed copy of Sheeri's book.

Leave your business card and you could win a book. We'll invite you to read our blog posts, follow us on twitter, and join our next webinars.



Thank You

Questions, Comments, Feedback?

Rob Hamel

hamel@pythian.com

Read: www.pythian.com/news/author/hamel

Ask me about saving 15% on our Adoption Accelerator
for MySQL Offering
while at MySQL Conference 2010!

mysql@pythian.com