

Real-time reporting at 10,000 inserts per second

Wesley Biggs

CTO

25 October 2011 - Percona Live



Agenda



-
1. Who we are, what we do, and (maybe) why we do it
 2. Solution architecture and evolution
 3. Top 5 tips and tricks
 4. The next 10,000... and beyond

What we do



Adfonic is a London-based startup launched in 2009. We power display advertising on all types of mobile devices.

- More than 8,000 mobile web and app publishers
- Over 3,000 ad campaigns each month
- >15 billion user interactions per month

Peak load October 2011: > 10,000 event insertions per second
ACID requirements - events must all be counted toward revenue
(CPC and CPM ad campaigns)

Self-service management and reporting tools w/real-time data

Who am I



Co-founder and CTO

Software Architect

Systems Architect

Java Developer

IT procurement

Tech hiring and HR

R&D lead

Makes a mean cup of tea

... (ahem) DBA?

We made a decision early on to use best-of-breed free and open source technologies where possible.

- CentOS Linux
- Java 6
- MySQL (currently 5.1)
- Apache ActiveMQ
- Pentaho Mondrian ROLAP

Follow the KISS principle

Remember, for every problem there exists a solution which is both simple and wrong

Avoid technology sprawl: 5 tools that each produce a 10% gain are rarely worth putting together.

The vision

Record events from the adserver (requests, impressions, clicks) and make them available for real-time viewing via the web site, broken down by device, location, mobile operator, etc.

“Real-time” reporting means data is available to be queried via the web-based self-service interface within seconds.



...if only it were that simple!

Hidden requirements



Auditability - keep 3 months of full data set around (already in the 6+ TB range) → Required

Aggregation - quickly became apparent we couldn't query on the whole raw dataset

Sharding - separate reporting data for internal vs. external use

Metadata replication - must be able to join against names, etc. at the reporting tier

24/7 availability, performance and all that rot

The reality

Lots of endpoint nodes (in multiple geographic locations)
Post-processing of events to help latency on the endpoints
MySQL Cluster for write scalability and H/A
Query performance via aggregate tables and ROLAP



Relational OLAP and Mondrian



Open source tool

Uses the MDX language

No ETL required for “data warehouse” (vs. BusinessObjects)

Looks for very particularly named aggregate tables

Drawbacks: Slow startup - performs count(*) on each aggregate table

Workaround: Hacked Mondrian to call an approx_count() function that uses a lookaside table in MySQL

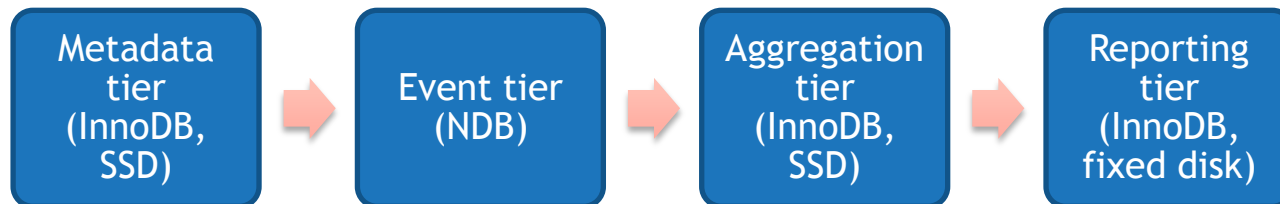
Issues complex queries that can tax the query planner

Tip 0: Use replication. Also, use replication.

It's seriously fast, even as a data copy mechanism

Use row-based replication (5.1+)

The replication chain can span multiple engine types and performance configurations



Tip 1: Use hash indexes on NDB



Factor of 10x or more write performance on cluster

```
CREATE TABLE AD_EVENT_LOG (  
  ID BIGINT PRIMARY KEY USING HASH ...  
) ENGINE=ndbcluster;
```

You can still use autoincrement (MySQL Cluster handles this well and lets you control contiguous ID blocks)

Replication slaves using InnoDB can revert to BTree

Tip 2: Use events, not cron jobs



Lifecycle tied to the MySQL server

Works just like a stored procedure

Easy management (alter event foo [disable|enable])

Write events with global mutexes and 1s repeat to accomplish “constantly running” processes, e.g. aggregation

```
create event E_RUN_AGGREGATES on schedule every 1
second
do
BEGIN
    IF GET_LOCK('ADFONIC_AGG_LOCK',1) THEN
        { do some work and know we're alone }
        SELECT RELEASE_LOCK('ADFONIC_AGG_LOCK');
    END IF;
END
```

Tip 3: Use the ENUM datatype



Enormous data savings over CHAR or VARCHAR

Stored as an integer, so easily indexed

No impact on client libraries (these still assume VARCHAR)

Don't try this at home: InnoDB DDL files can be edited if you ever need to associate existing ENUM values with different names

```
AD_ACTION ENUM
('UNFILLED_REQUEST', 'IMPRESSION', 'CLICK', 'INST
ALL', 'AD_SERVED', 'CONVERSION', 'BID_FAILED')
NOT NULL
```

Tip 4: Use `sql_log_bin` to diversify



Turning off binary logging at the right time allows the preservation of different configurations at different stages of replication.

Separate data replication from access patterns

Control partitioning on a per-tier basis

Manage indexes for different performance criteria

```
CREATE PROCEDURE update_daily_partitions()  
BEGIN  
SET sql_log_bin=0;  
...  
ALTER TABLE foo DROP PARTITION yesterday;  
ALTER TABLE foo ADD PARTITION (... tomorrow ...);  
...  
SET sql_log_bin=1;  
END
```

Tip 5: Avoid insert...select with replication



Aggregation use case: read from replicated event table, write to aggregate tables

```
INSERT INTO foo SELECT ... FROM bar [ON  
DUPLICATE KEY UPDATE...]
```

Great for compact SQL, but locks `bar` as well as `foo` unless you can live with `READ_COMMITTED` isolation. In our case no, so that means replication is blocked during the select.

Instead, replace with cursors over the selects (work in progress).

Tip 6: Normalize event-stream data



Normalized data keeps the event-level data set small
BUT read-compare-update-insert semantics are slow

Example: User-agent strings - large and unwieldy, but constantly changing

Solution: Treat MySQL Cluster as “write only”; only reads are via replication

Implication: The post-processor must retain the foreign key database in memory and know when to insert. Use message queue to synchronize multiple post-processors.

Wish list/challenges/needs



Canonical (& fast) estimated row count for InnoDB
Improve DDL replication between different storage engines
Transactional TRUNCATE TABLE - for use with aggregation
Easier multithreaded transactions - aggregation again
HOWTO for HA replication to cluster
Smarter locking on INSERT...SELECT
Pinnable query plans (or at least more transparency)
Easy offlining of date-based partitions (hot backup may address)

The next 10,000 (and beyond)



Sharding and horizontal scaling are ultimately the only tools at your disposal

Scale the cluster horizontally - just add hardware

Parallelize aggregation - multiple concurrent “insert/selects” for performance

Sharded replication to customer-centric reporting databases - requires some duplication of data (advertiser x publisher)

Federation/spider topology

Early vs. late aggregation (not all needs to be real-time)

More hacking on Mondrian to make MySQL-friendly: forced indexes, etc.

Thank you. Questions?



Contact me:

Wes Biggs

Email wes.biggs@adfonic.com

Twitter [@wbiggs](https://twitter.com/wbiggs)

We're hiring!

London-based DBA position available immediately

See me for details or visit <http://adfonic.com>

Email jobs@adfonic.com