From Requirements to Sharding and Partitioning - and everything in between

Raj Thukral

Percona Live: MySQL Conference 2012

Pythian
love your data

Why Pythian

Recognized Leader:

- Global industry-leader in remote database administration services and consulting for Oracle, Oracle Applications, MySQL and SQL Server
- Work with over 165 multinational companies such as Forbes.com, Fox Sports, Nordion and Western Union to help manage their complex IT deployments

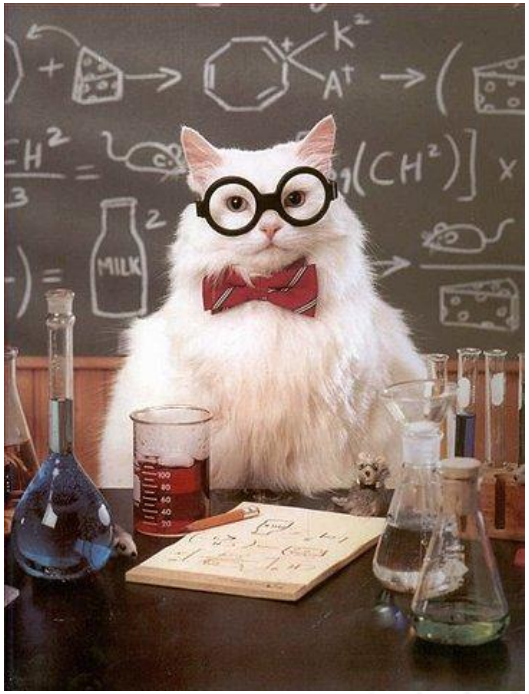
Expertise:

- One of the world's largest concentrations of dedicated, full-time DBA expertise. Employ 7 Oracle ACEs/ACE Directors
- Hold 7 Specializations under Oracle Platinum Partner program, including Oracle Exadata, Oracle GoldenGate & Oracle RAC

Global Reach & Scalability:

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

About Myself



- Hands-on Consultant, DBA, Sysadmin
- Consulting on Database Architecture, Data Modelling, Performance, Security, Disaster Recovery, High Availability
- Experience with Oracle and MySQL
- Over 8 years at Pythian
- Even longer working with MySQL
- email: thukral@pythian.com

A brief Outline

- a real-world design scenario
  - It has all the classic project stages
- Requirements
- Architecture
- Development
- Testing
- Validation
- Acceptance

Not really!

Here are the requirements

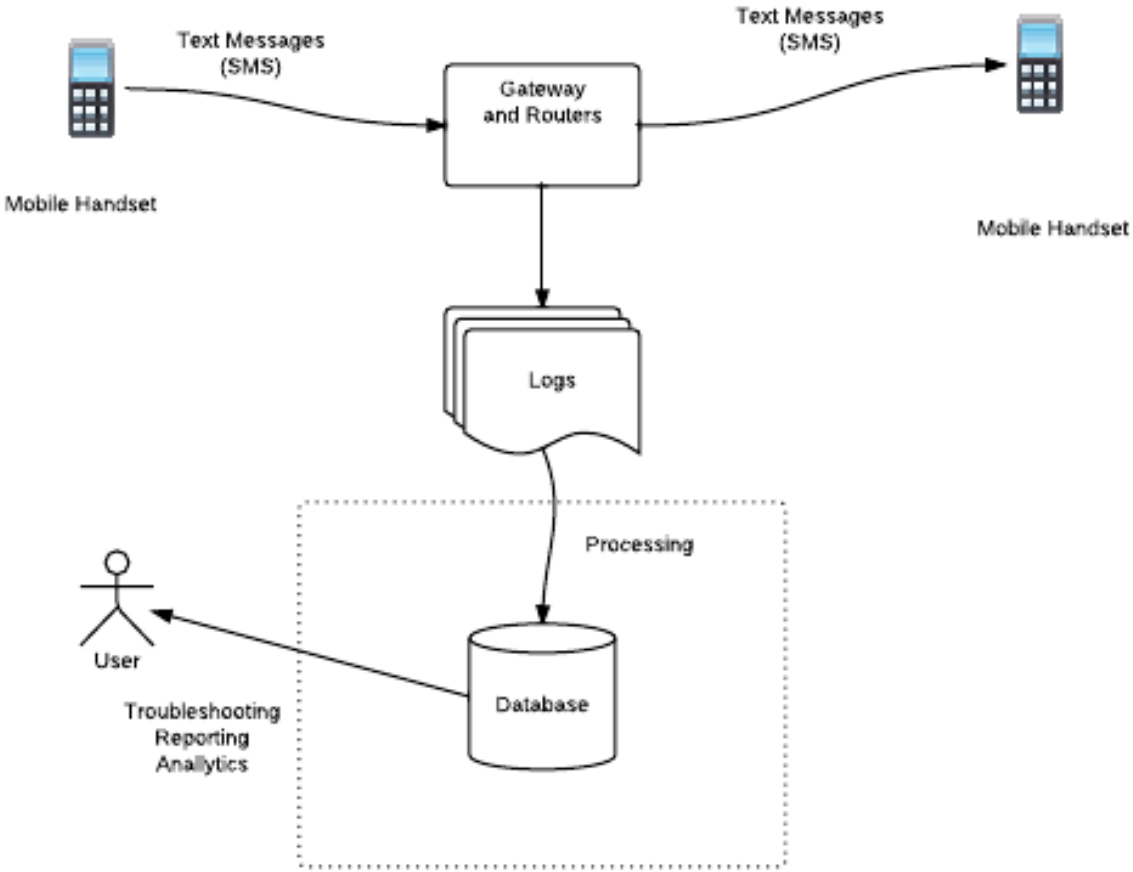
“We have an application that can currently handle about 2000 events/ sec and chokes at 10,000 events/sec, we need it to handle 60,000 events/sec”

Now what?

- Understand the current application and context
- Understand current pain points
- Understand the motivation behind the redesign
- Drill into requirements
- 3 day white-boarding session

First things first

What is the “product”



What is it used for?

- User Support. Or.. I haven't been getting my text messages. Or.. You've billed me for more messages than I sent!
- Spam detection
- Bulk message details
- Router Troubleshooting
- Capacity assessment (bottlenecks)
- Identifying major message and revenue sources
- Reporting: Analytics on demographics, app message usage (eg weather), market identification
- And many more uses

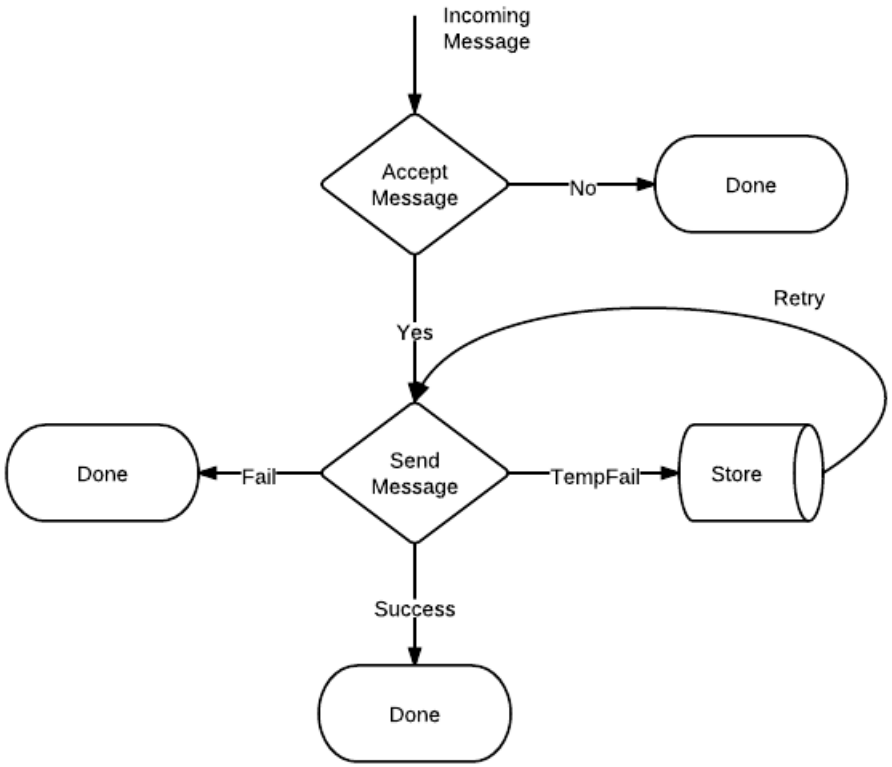
What is it NOT used for?

- Billing. Billing reports are sent directly from the routers
- Real-time data requirements (Data is delayed by at least 15 minutes)
- Router monitoring (not a monitoring dashboard)
- Real-time Capacity assessment
- License compliance

In other words, if the solution goes down, it doesn't stop the messages from flowing

Drilling into the details

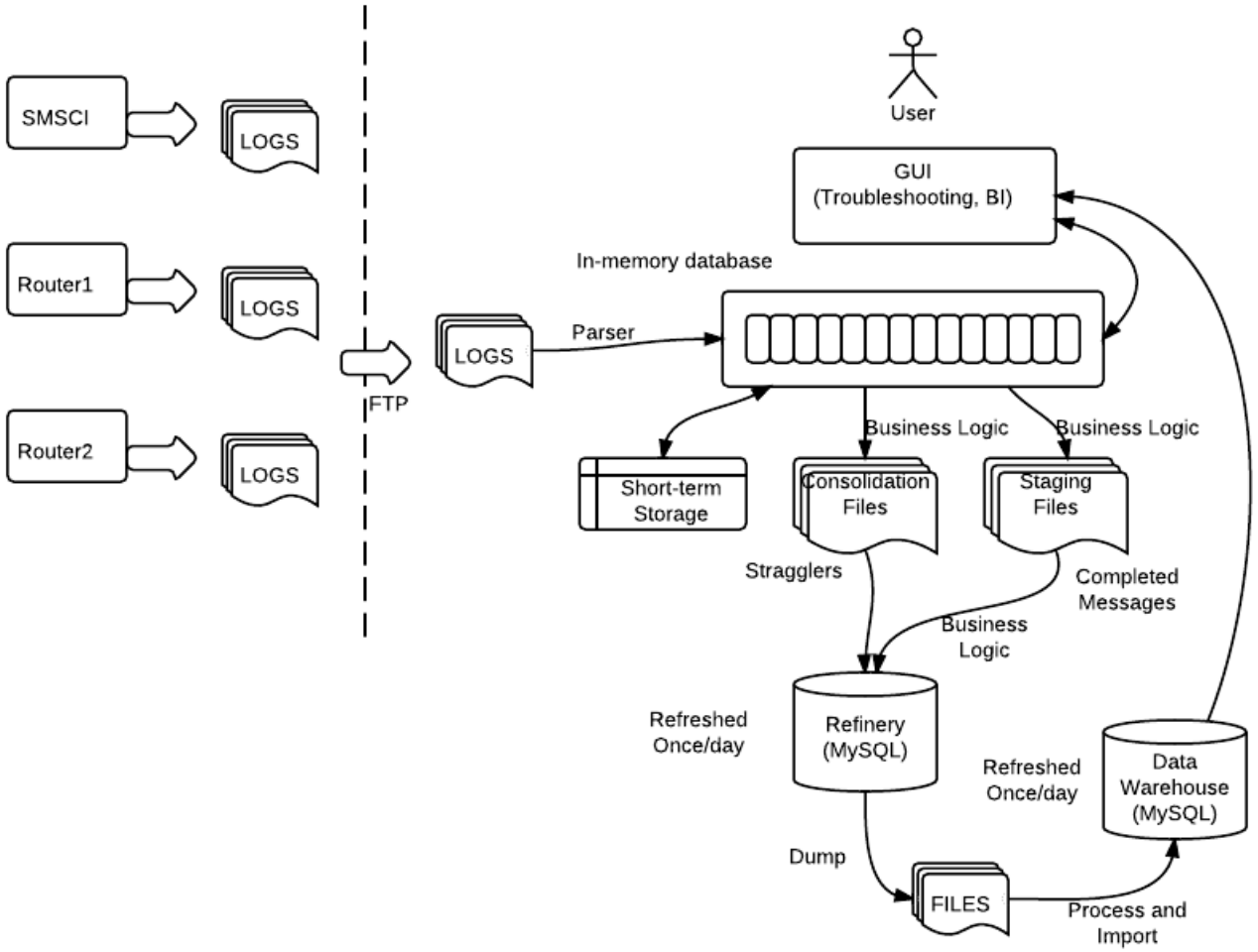
Message Flow



Details

- Messages are identified uniquely by the sender, recipient and timestamp Router also adds a semi-unique 64-bit number, cycled every few months
- An “event” is generated whenever a message goes though a gateway or router. events are recorded in log files for each device. These log files are processed by the product
- events for the same message need to be "correlated" in order to generate a full pictures of the message. Messages may take 14 days or longer to be delivered and events may not come in the order they were generated
- events come in fast and furious. each sms message generates at least 3 events (Message Accepted, Message Routed, Message Sent) and sometimes even 7 or more if there are retries.

The current state of affairs



Functions

- Log files from routers and SMSC's are rotated, copied and processed every 5 minutes. Custom "Parser" extracts information from log files
- In-memory database written as c memory structures to hold message info until message is "complete."
- apply business logic rules (written in c) to messages to update billing information, find out if messages are complete or expired and removed (written to staging db)
- dump memory database + staging database into refinery nightly, aggregate and transform into reporting database
- troubleshooting queries need to query 'live' data from memory as well as staging database to build complete picture
- reporting queries also go against live data as well to be more up-to-date

Issues

- does not scale (memory requirements)
- needs a full lock to update memory structures (partitioning is a weak way out)
- power failure means lengthy recovery as memory structure is loaded from staging and then all log files since dump replayed (can take hours to recover)
- Business logic changes often making product unmaintainable
- Troubleshooting and reporting queries slow down processing
- Applying business logic to messages (to mark them complete and remove from the memory index) painful and slow

Now the real requirements!

- system should be scalable and cost-effective from 100messages/sec (3000 events/sec) all the way to 20,000messages/sec (60,000 events/sec)
- Use 'commodity' hardware
- Use MySQL
- Use off-the-shelf BI (eg Jaspersoft) and ETL tools (eg Talend)
- System recovery time should be < 1h in case of power failure or system crash
- Troubleshooting tool should have access to data from routers with less than 15m delay
- Reporting database should be up to date within a few hours (not 1 day+ behind)

- Business logic should be 'easy' to customize -change should not require an extensive application rewrite
- Reports should be easy to customize as well
- Troubleshooting should be fast across a wide range of queries
- Troubleshooting should have the ability to drill down to raw events if required

Broad design drivers

- Overall data size: 300 bytes/message or event x 60,000=18000,000bytes/sec or 18M/sec or 1G/hour or 24G/day. Plus overhead – we are looking at approximately 1.5TB/Month of RAW data. Need to think about data archival and retention policy

- Speed of inserts - can we keep up with 60,000 inserts/second? How many can we do on 'commodity' hardware?

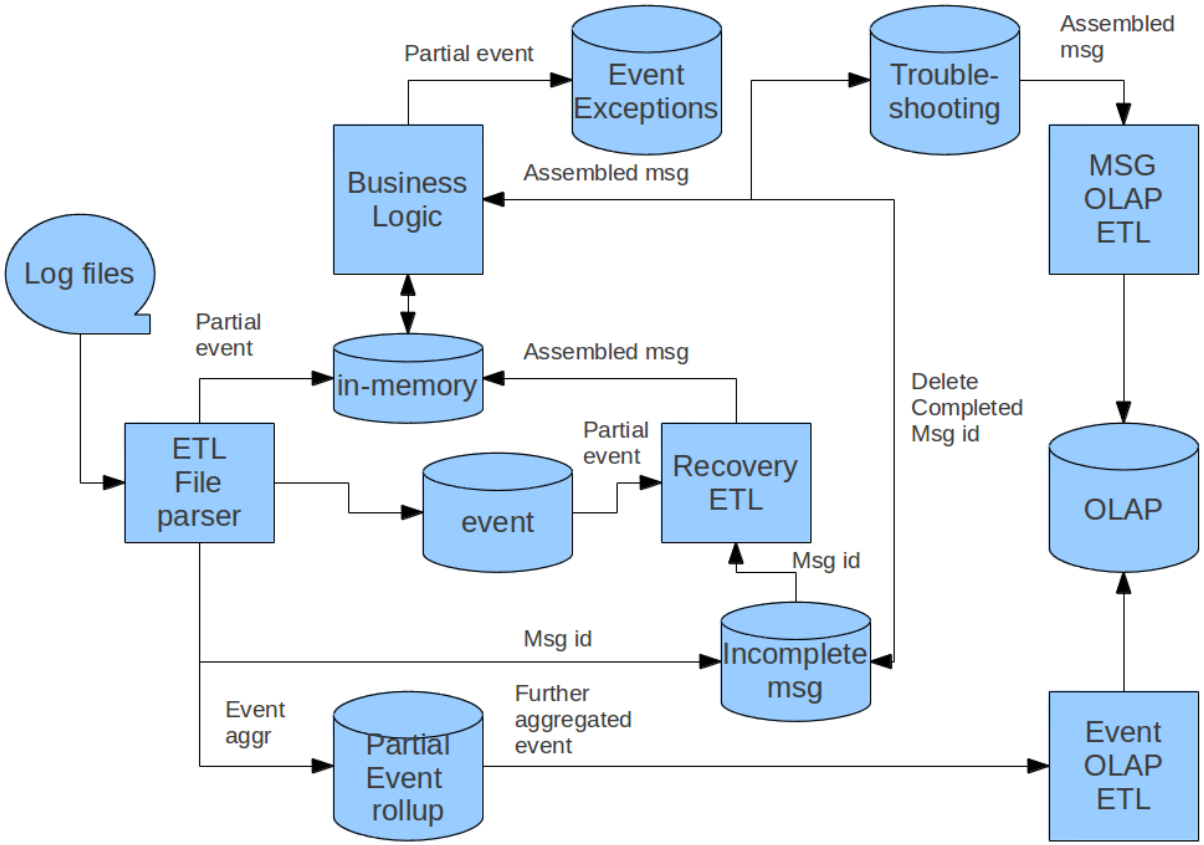
- Troubleshooting queries are not well-defined – need to think about optimizing for most common queries while keeping others not too slow to run

- Business logic needs to be easy to change – data model can not reflect business logic decisions, they need to be outside the db

Broad design decisions

- Partitioning is a must for 2 of the requirements:
  - Partitioning helps with data retention – far easier to drop old partitions than delete data at 60,000/sec
  - Partitioning helps with speed of queries if partition elimination is possible
  - Partitioning by date seems best option if queries can be limited by date range – most queries will be on recent data
- Speed of inserts:
  - Our tests on "average" sized servers with 15k disks shows we could achieve about 18,000 inserts/second. 60,000 would require at least 4x that
  - "sharding" became a given – but events need to be correlated for final message – how do we do that?
- Business logic updates:
  - Updates are expensive, try to keep database insert-only if possible

Draft architecture Version 0.1



Limitations

- Complicated, does not allow for easy scalability
- Lots of work at the etl layer
- Business logic hard to implement
- Huge memory requirements for in-memory database
- Complicated recovery for in-memory database
- Adding sharding further complicates the picture since events for a particular message will need to go to the same shard every time to allow the business logic to aggregate them

In other words, we had replicated the original architecture using off-the-shelf tools, but we had also replicated the problems

More rethinking

- Separate out reporting from troubleshooting
- Limit troubleshooting to 15 days of data for faster searching
- Decide on level of granularity for reporting (aggregates)
- Decide on how to shard (based on hash of message_id)
- Still need to apply business-logic which means updates which are slow!
- No clear picture on how to keep business logic “clean” from the database

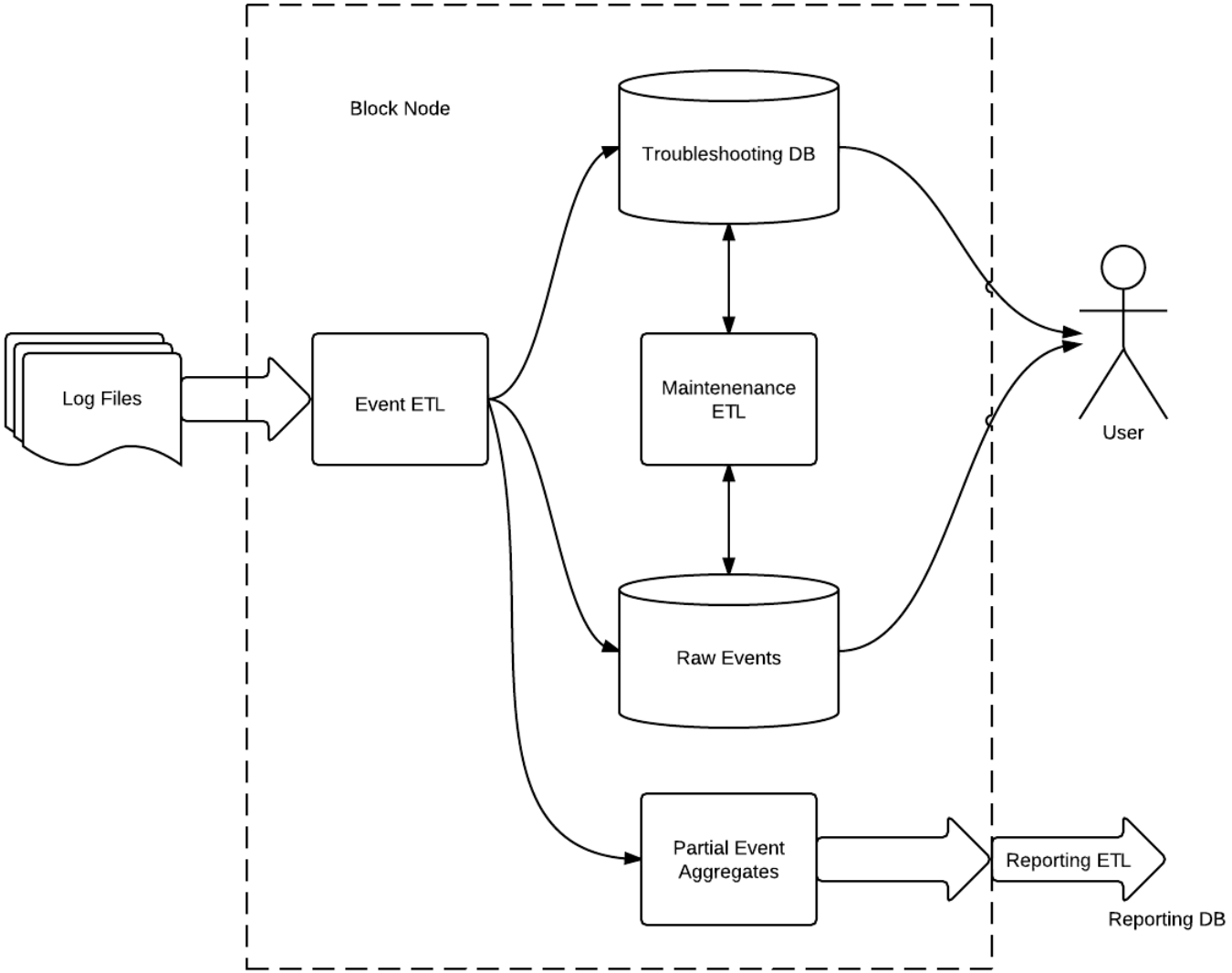
The Aha! moment

- Routers know the message the event is related to since can correlate events to tie back to a specific message
- routers apply business logic to events already - billing is done directly via routers and not through reporting
- The aha moment: why duplicate effort!
- lets focus on getting more information from the actual routers
- let router apply business logic (it already does!)

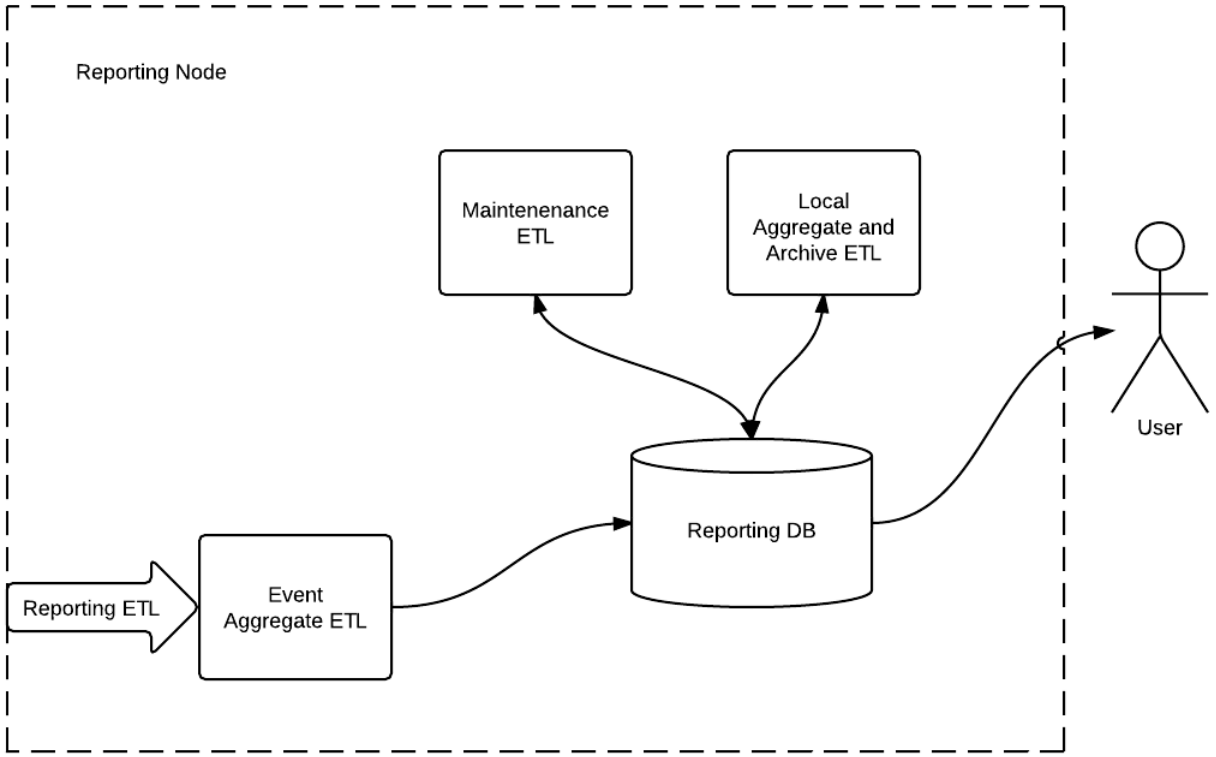
And how it was implemented

- Have routers tag ‘boundary’ events, required for troubleshooting
- ‘boundary event' tagged events have full information about message
- messages can now go to any shard since only ‘final event’ messages are required and can be pulled from any shard
- Troubleshooting database can be insert-only since no business logic needs to be applied
- processing can continue even if a shard is down - remaining shards 'share' the load
- solution can be scaled up or down easily by adding/removing nodes

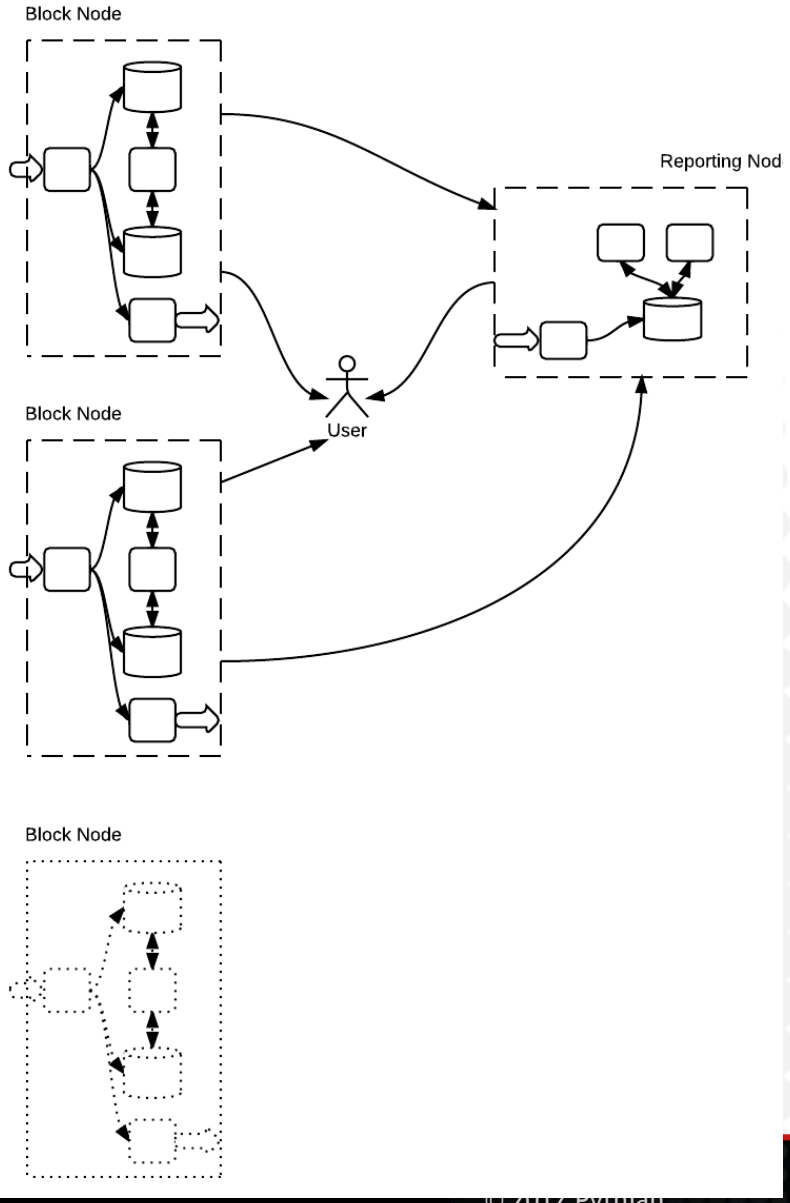
Final Architecture - TS



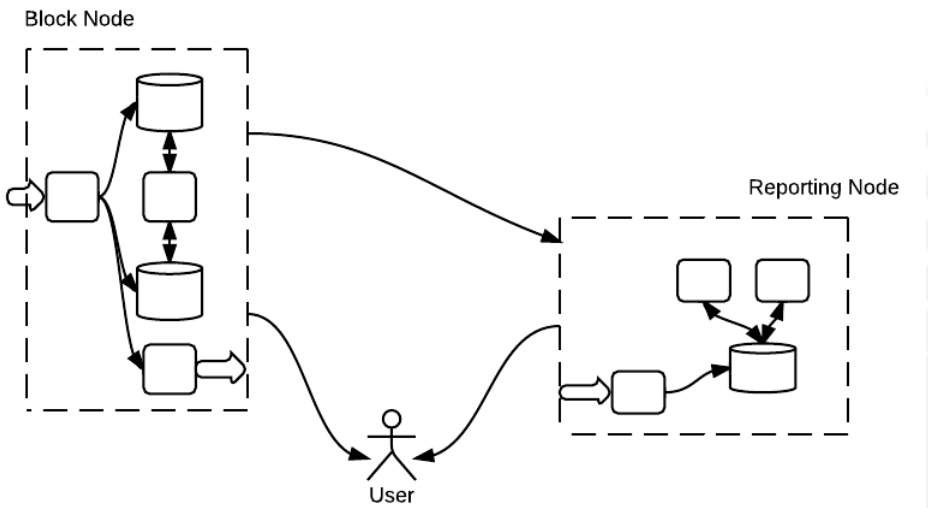
Final Architecture - Reporting



Final Architecture - Scaling



Final Architecture (small)



Final Architecture

- Broken up into 2 distinct parts: Block Nodes (troubleshooting) and Reporting
- Multiple block nodes possible, but only 1 reporting node
- Log files processed by multiple ETL processes.
- Each ETL process writes all raw events to the ‘event’ database on one node (insert-only)
- Each ETL process writes all ‘final state’ events to the troubleshooting database on one node (insert-only)

Final Architecture

- ETL process also does partial aggregation and writes to staging DB
- Staging DB further aggregated and unloaded into Reporting db by an ETL process
- Separate ETL process on reporting to generate daily/weekly/monthly aggregates periodically
- Tables maintained by a partition-maintenance job to limit data in troubleshooting database
- Reporting tool queries all shards in parallel and combines data

Practical problems

- Troubleshooting queries were not pre-determined. We could not optimize for all use cases
- Partitioning by date helped with quick partition elimination to keep queries relatively fast and also allowed for easy data maintenance by dropping old partitions
- But: Partitioning brought down speed of inserts down by 10x (1000 partitions)

Practical problems

Why?

- MySQL does not eliminate partitions on insert or update – all partitions are locked
- Partition locking is slow and expensive
- We could not work without partitions or deleting old data would have been very expensive – as expensive as the inserts (one delete per insert)
- Solution? Micro-batches to reduce overhead of locking

Results

Solution in use at over a dozen implementations now

Customization is easy and fast

Scalable to meet all requirements

Better reporting and troubleshooting support

Final thoughts

- Question Everything. Take nothing for granted.
- Test Everything. Take nothing for granted

Thank you and Q&A

To contact us…


sales@pythian.com


1-877-PYTHIAN

To follow us…


http://www.pythian.com/news/


http://www.facebook.com/pages/The-Pythian-Group/163902527671


@pythian


@pythianjobs


http://www.linkedin.com/company/pythian