Mediating Better Answers

Talk by: Andy Cooke

Collaborators: Alasdair Gray, Lisha Ma, and Werner Nutt

Heriot-Watt University

Current Situation

- *All Insertables can stream*
  - Continuous queries get *complete answers* more often ☺
  - *Easier mediation* as more chance of complete republishers being available. ☺
  - Republishers are *always complete* ☺

Current Situation

- Some answers to queries are *adventurous*:
  - e.g. 3 publishers, full views: LP, LP, LRP
    - “latest” consumer chooses the closest, not the most complete   => incomplete/ wrong answer
  - e.g. 2 publishers: SP, LP with full view
    - LP isn’t complete anymore (neither is the answer)
  - e.g. LP with partial view
    - consumers can only use LPs with full views
    - empty set is returned

Problem: user can’t find out that R-GMA was adventurous ☹

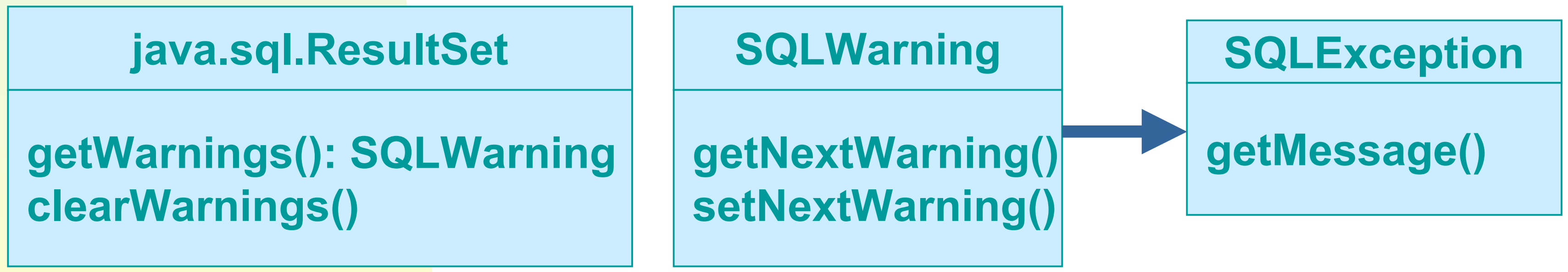
What’s next?... better answers!

- Next Steps:
  - RGMAWarnings
  - Improve answers to one-time queries

- Future Steps:
  - Improve answers to continuous queries
  - Republisher Hierarchies
  - Support More Queries?

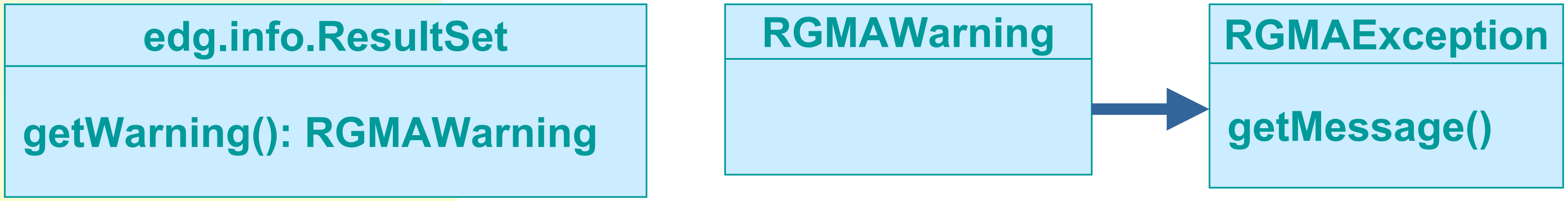
RGMAWarnings *about Answer Quality!*

- **java.sql.SQLWarnings** can be retrieved from **Connection**, **Statement** or **ResultSet** objects:
  - "Provides information on db access warnings"
  - "Silently chained to the object" (Java API)



RGMAWarnings *about Answer Quality!*

- **RGMAWarnings** could be attached to **ResultSets**.
  - e.g. “answer might be incomplete: …”,
    “answer might be wrong: …”
  - *Is chaining needed? (I don’t think so)*
  - care needed to ensure *backwards compatibility*
  - need to design *useful messages*
  - need to identify *all cases* where answer might be *incomplete*

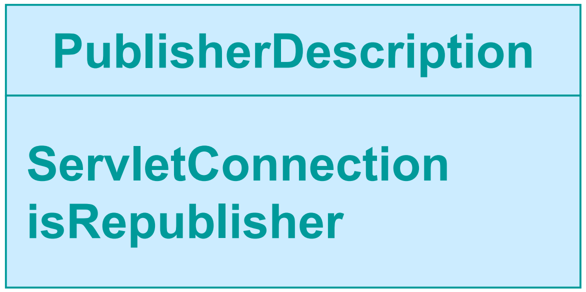


Improving Answers to One-Time Queries

- Opportunity now to return *better answers*, as *all insertables stream*.
- Users can now be informed of quality, with the help of RGMAWarnings.
- Strategy:
  - Always try to use complete publishers that have full views
  - Otherwise, merge answers from incomplete publishers …*may still get safe answer!*

Example 1: PublisherDescriptions

- Problem:
  - Consumers get ServletConnections to relevant publishers from the registry
  - Would like to identify the republishers, but can’t!
- Solution:



  - *wrap* **isRepublisher** flag plus **ServletConnection** inside a **PublisherDescription**

Example 1: PublisherDescriptions

- e.g.: LP, LP, LRP registered (all with “full” views)
  - Currently: consumer queries the closest publisher
  - Using PublisherDescriptions:
    - Consumer identifies one complete LRP, and two incomplete LPs.
    - So query the LRP, and get a complete answer
- In future, PublisherDescriptions could hold other useful information, e.g. views, retention periods.

Example 2: No Complete Publishers

- Can safe answers be returned even when no complete publishers are available?
- e.g. Query two LPs (full views), and merge…

  How safe is the answer? It depends…

  - e.g. aggregation => "answer might be wrong"
  - e.g. join => "answer might be incomplete"
  - e.g. simple selection => no warning needed ☺
- Can extend to cases where LPs are not full.
- Question: *is there a use case for this?*

Example 3: Producer Completeness

- Problem:
  - A producer is complete if there are no other producers with overlapping views.
  - Consumer needs more information from registry
- Solution:

RelevantPublisherInfo
boolean: otherTypesRegistered Vector: publisherDescriptions

```
RelevantPublisherInfo info = registry.registerOneTimeQuery()
```

  - wrap **`otherTypesRegistered`** flag plus descriptions into **`RelevantPublisherInfo`**
  - *notify* Consumer if situation changes

Example 3: Producer Completeness

- *e.g. 2 producers registered: SP and full LP*
  - Consumer discovers that LP is incomplete as otherTypesRegistered is true.
  - Query LP, and set RGMAWarning, if the answer might be incomplete or wrong.

- *Using producers for answering one-time queries is tricky!*

Example 4: Partial Views

- When can queries be answered by publishers with partial views?
  - If query condition implies view condition, e.g.
    - query: “select * from cpuLoad where site = ‘RAL’”,
    - view: “where site = ‘RAL’”
  - If producer’s database maintain foreign keys for the attributes in the join condition,
    - *so things that logically belong together are stored together*
  - Some conditions exist for aggregate queries

Conclusions: One-Time Queries

- Complete Publishers with full views have all the tuples needed for a complete answer.
- Consumer needs to work out completeness:
  - send RelevantPublisherInfo to Consumer, which contains PublisherDescriptions
  - Notify consumer when situation changes.
- Safe answers can still be returned, even when Publishers don’t have all the tuples.
  - RGMAWarnings if *incomplete* or *adventurous!*

Improving Answers to Continuous Queries

- Can Continuous Consumers use republishers?

  need to avoid duplicates and tuple loss...

- Problem1: Need to figure out how to alter plans:
  - when publisher drops out
  - when publisher becomes available
- Problem 2: Transition from old plan to new plan
  - use retention periods
  - views
  - plus snapshot table

  to avoid duplicates/ loss

Example: Republisher Drops Out

- Scenario: 3 SPs, one “full” LRP registered
  - Consumer streams from LRP, as it is complete.
  - Backup plan: stream from 3 SPs.

What if the LRP stops responding?

- Idea 1: when registry calls removeProducer(), switch plans.

...but tuples might be lost if retention period is too short!

Example: Republisher Drops Out

- Idea 2: Consumer waits almost as long as the *smallest retention period*, before switching plan.

  no tuples are lost…

  - retention periods should be registered.
  - alter API so that retention period can’t be changed or set to zero – otherwise this won’t work!

  … but duplicates could be received!

Example: Republisher Drops Out

- Idea 3: Keep a latest snapshot when switching plan,
  - from registered views of each producer, can work out when to stop looking for duplicates

    … duplicates avoided!

  - consumers need to keep a latest snapshot.
  - consumers need to know registered views of producers.

    won’t work if producer views overlap!

Example: Republisher becomes available

- Scenario: 3 SPs
  - Consumer merges streams from each SP.

What if a republisher becomes available?

- Idea : Use a latest snapshot table.
  - Start streaming from RP and stop SP streams.
  - During transition, use table, plus views, to know when to stop filtering for duplicates.

… duplicates avoided!

Conclusions: Continuous Queries

- Continuous queries could use republishers…
  - more efficient use of network bandwidth ☺
  - evolving plans as registry changes is hard ☹
- Tentative solution:
  - use retention periods to avoid tuple loss
  - use views/snapshot tables to avoid duplicates?
  - Alter API to avoid changing retention periods.
  - producer views shouldn’t overlap.

…stepping stone towards supporting hierarchies

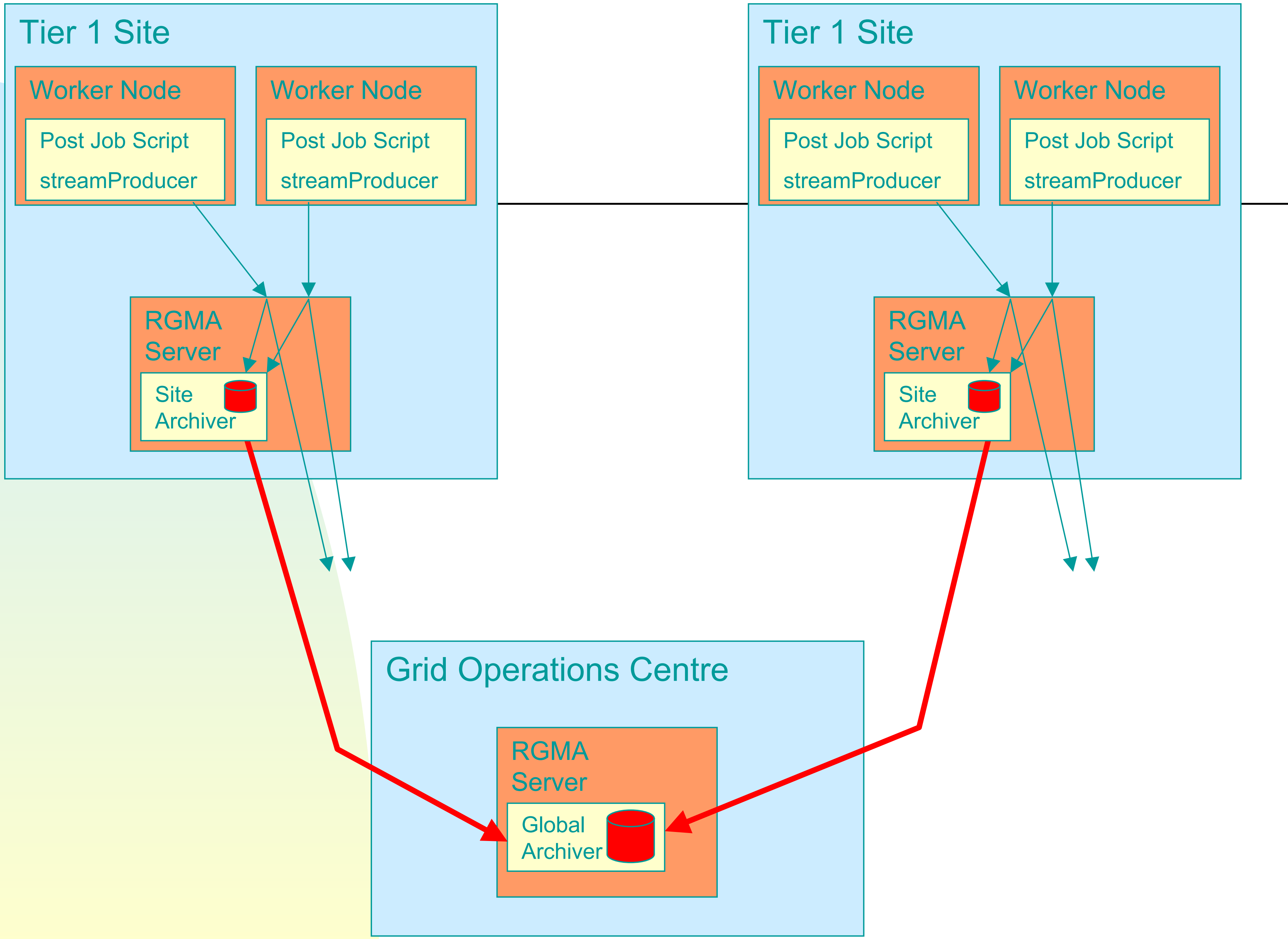
Republisher Hierarchies

Republisher Hierarchies may help to:

- Reduce network traffic
- Improve the max republishing rate
  - as less threads!
- Share load across publishers
  - as more choice for consumers.

Republisher Hierarchies for LCG

- *LCG would like to collate info about jobs that ran into a central db.*
  - System should recover if a site goes down temporarily, without loss of tuples.
- Short-term:
  - hard-wire a hierarchy (site RPs, global RP)
  - some code changes are needed.
- Longer-term:
  - automatically configure hierarchies.


Tier 1 Site
Worker Node
Post Job Script
streamProducer
Worker Node
Post Job Script
streamProducer
RGMA Server
Site Archiver
Tier 1 Site
Worker Node
Post Job Script
streamProducer
Worker Node
Post Job Script
streamProducer
RGMA Server
Site Archiver
Grid Operations Centre
RGMA Server
Global Archiver

Short Term: Hard-wire a Hierarchy

- Currently: Insertable has responsibility for keeping socket channel alive:
  - if channel found to be dead, then on next insert, new channel is created.
- Code change: if DBP's buffer fills up, then
  - note the date/time of next tuple to send
  - when connection re-created, pose db query to retrieve outstanding tuples, and send these

Longer Term: Dynamic Hierarchies

- Dynamic hierarchies would:
  - sense when new sites came on-line
  - recover if any site archivers went down.
- The problem is much tougher!
  - a logic puzzle: figuring out automatically which is the most efficient hierarchy, and adapting this as publishers come & go
  - protocols needed that avoid tuple loss/duplicates as plans change (see earlier)

What's next?... better answers!

- Next Steps:
  - RGMAWarnings
  - Improve answers to one-time queries
- Future Steps:
  - Improve answers to continuous queries
  - Republisher Hierarchies
  - Support More Queries?