Relational Model for Information and Monitoring 6/3/2001

Steve Fisher / RAL

<s.m.fisher@rl.ac.uk>

Warning

- These ideas are not fully worked out
- Prototypes have not yet been built

but

- It looks very promising
- It relates closely to work of Plale and Dinda (GIS)

and

- There are quite a number of places where solutions exist which I don't know about. *Please advise me*.

Messages

1. Information and monitoring should be treated together.
2. Should use a data model which can support arbitrary queries: Relational
3. This is largely consistent with the GGF performance architecture.
4. The system can be partitioned using a mixture of full RDBM systems and simple one table systems.

Information vs Monitoring

- From the user's point of view there there is little or no difference between "plain" information and monitoring information.
  - Arguments about rapidly and slowly changing data are unconvincing
  - Maybe you take some plain information (measurement or fact represented as a tuple) add a time stamp to the tuple and the information can now be stored for later analysis as monitoring information – so at most the difference is 1 field – the time stamp.
    - Time is the common element
  - It is also seems desirable to have a common interface to access data, whether it is *fresh* monitoring data or data from an archive.

Tuples

The short batch queue on the CSF system at RAL has 34 jobs on it.

add the time stamp → a tuple:

```
(RAL, CSF, SHORT, 34, 2001-5-02T16:07Z)
```

A set of such tuples could be stored in a table:

```
ComputingElementQueue(Site, Facility, Queue,
  Count, Time/Date)
```

Any structured data can be represented in tables in this manner.

Complex queries can be formulated with SQL.

A good data model

- The GGF performance architecture does not specify the protocol between the consumer and producer nor does it imply any data model.
- First choose a suitable data model, then select suitable protocols.
- The chosen data model must have the power to support all the queries we need to make.

Why not LDAP

- Relational database was offered by Codd, 30 years ago
  - solution to the inflexibility of hierarchical and network data bases.
- LDAP (hierarchical) is fine if you know the query in advance as you can build your database to answer that question very rapidly.
- For other questions, it could be very expensive as the LDAP query language cannot give results based on computation on two different objects in the structure.
- Consider for example the question which a scheduler asks when deciding which elements of the grid to use to run a job…

Registration of producers

- Following the GGF architecture, producer normally register themselves so that they can be found by consumers.
- Now if the RAL CSF has a producer of ComputingElementQueue information it can register itself:

  `ComputingElementQueue(Site=RAL, Facility=CSF)`

- This information could be stored at the level of RAL by an RDBMS with both a consumer interface and a producer interface. The producer would register itself as:

  `ComputingElementQueue(Site=RAL)`

- An RDBMS holding information on *all* ComputingElementQueues this would register itself as:

  ComputingElementQueue.

- Register the name of the table with the names of any attributes which are fixed and the values of those attributes.

Duplicate registrations

- If there is more than one producer offering the same data what should happen?
  - It could happen that two archives are set up to archive and offer the same data. Many events will be identical though not all because of different clean up strategies and because of losses where the consumer fails to keep up.
- To make this less likely the distinction between archives and producers only of fresh data should also be noted when registering
- Should probably prohibit duplicate registrations.

Use of existing Protocols

- LDAP, unlike SQL, has a defined wire protocol.
- We could:
  - adopt the solution used by MySQL which allows remote data bases to be accessed.
  - Or could use SQL embedded in XML as http(s) query
  - Or …

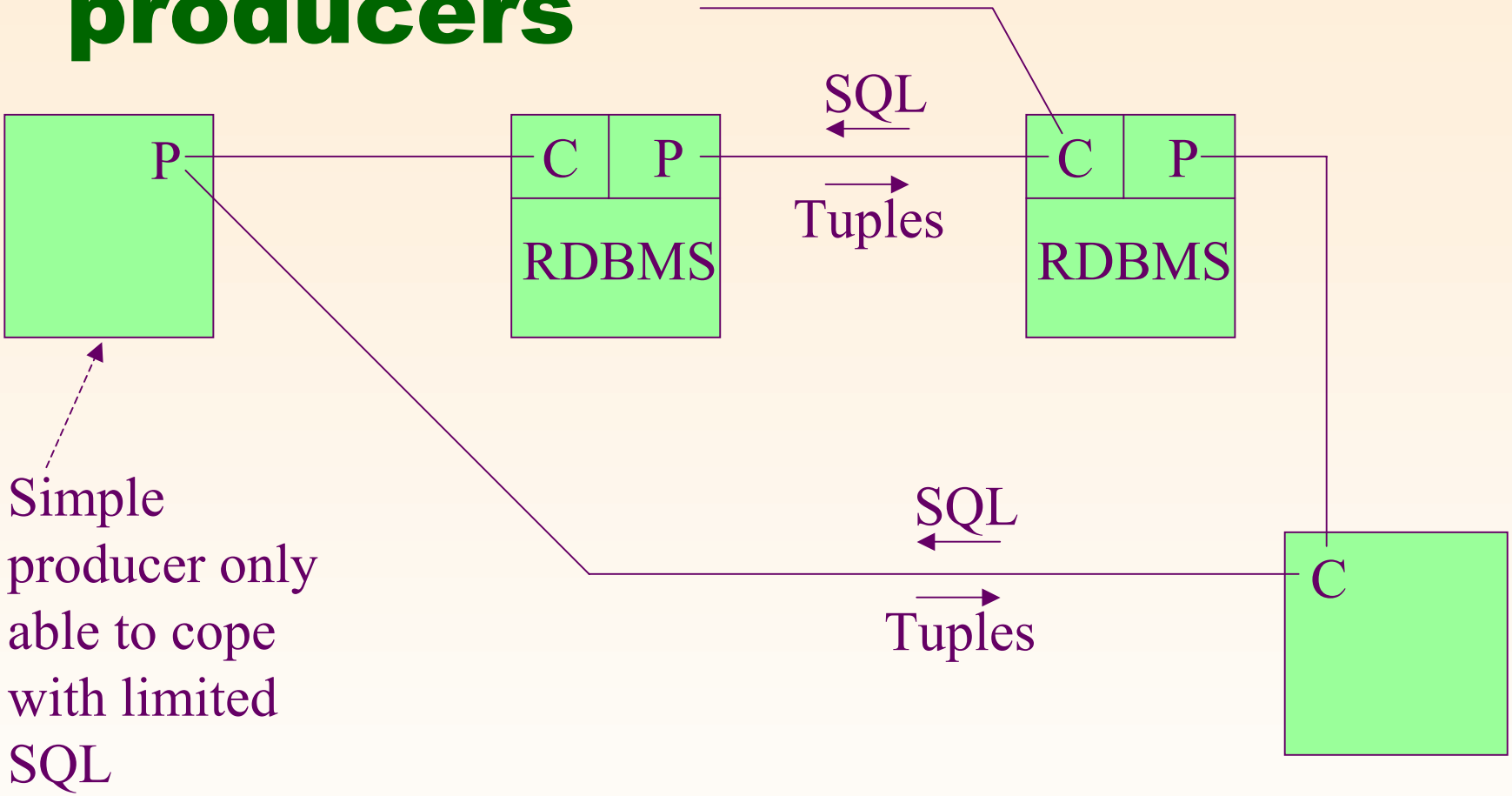
Protocols – single table

- For a producer offering rows from a single table
  - Process SQL statement:
    - Push - stream rows which match the query
    - Pull - return the latest row if it matches the query.
  - The SQL statement may of course only request certain columns of the table (fields within a row).
- SQL can be processed by some simple code

Protocols – more tables

- To bring the benefits of the relational model we want to be able to send queries which include joins to select information from two or more tables. The result of a an SQL SELECT statement is normally that of a dynamically created table.
- For aggregate functions (for example to compute an average), the full power of SQL would be needed. You transmit an SQL query and a dynamically constructed table comes back.
- Typically handled by an RDBMS.

Some consumers and producers



API - producer

- For the producer, for each table it produces it should register the table name and the identity and value of any fixed attributes. Then a producer simply has to announce a table name and the row(s) of a table.

API - consumer

- For the consumer API you send an SQL query and get back rows of a table or request that rows of a table are streamed to you. The client can analyse the query and based on the tables involved send the query to the right producer or producers.
  - Queries which can be processed by a single producer can be handled efficiently, but others will result in some operations being carried out by the client side.
- This suggests that there will be advantages in having Producer/Consumer/RDBMS units able to hold data which will often be joined.
  - In fact such a unit might be created automatically and then destroyed when it is no longer frequently used.

Time to live

- How to decide when to get rid of archived data
- Information may no longer be “up to date”, but if we are interested in historical data this is of no consequence.
- Source of data is no judge of its continued worth and so TTLs are of no value.
- Only the collector of data, who knows why he is collecting the data can devise a suitable strategy.

Surrogate keys

- Normally small integers are used as surrogate keys when designing data base schemas.
- A small integer is used as the primary key rather than some more natural string so that it can be referenced more compactly by other tables holding this integer.
- The allocation of these small integers would be difficult and it is suggested that this practice not be used here.

Schema

- The schema must be universally known.
  - This is a problem for application monitoring data where the schema could be very short lived.
  - Elegant solution is to ensure that the registration of new tables is easy to do.

Registration of schema and producers

- Each producer requires only a very small amount of registration information to be stored.
- A solution would be to have an RDBMS holding both the schema and the available producers.
- Duplicates itself over a number of RDBMS around the world – all of which are trying to become identical.
  - When you register, you use any one and the information spreads to all of them.
  - When you want information you just use any one.

Meta-schema

- Can have a table to describe the tables and one to describe the columns of the tables. Two other tables are needed for the registration of producers – see the paper.
- Add columns to indicate when each record was added (at least for the ProducerTable table).
  - The producers will periodically re-announce themselves and their record will be dropped from the tables when they are old if not refreshed.
- When a producer registers itself as a producer of a certain table, if the table is not known it can be added to the schema. If a Table is not used by any ProducerTable its definition can be removed.
  - Handles schema evolution.

Conflicts

- One problem which this will not solve is the case of two producers registering a table with the same name.
- Eventually the names will move around the system and will clash. In the same way if we wish to prevent a producer registering itself with the same information as an already registered producer this will not always work reliably.
- A solution to this problem would be that each copy of the schema/registry RDBMS knew about every other active one and so could synchronize important changes such as a new table definition.

Conclusion

- It appears beneficial to support a data model which can support arbitrary queries.
- It seems practical to introduce the relational model without any major impact upon the GGF performance architecture.
- The mechanism for partitioning and managing a distributed RDBMS outlined here seems practical.
- Will start prototyping soon to verify this!