What to do with Scientific Data?

by

Michael Stonebraker
Outline

- Science data – what it looks like
- Hardware options for deployment
- Software options
  - RDBMS
  - Wrappers on RDBMS
  - SciDB
O(100) petabytes

Courtesy of LSST. Used with permission.
Courtesy of LSST. Used with permission.
LSST Data

- Raw imagery
  - 2-D arrays of telescope readings
- “Cooked” into observations
  - Image intensity algorithm (data clustering)
- Spatial data
- Further cooked into “trajectories”
  - Similarity query
  - Constrained by maximum distance
Example LSST Queries

- Recook raw imagery with my algorithm
- Find all observations in a spatial region
- Find all trajectories that intersect a cylinder in time
Snow Cover in the Sierras

MODIS, 3/7/04
- band 2
- band 6
- band 3

0 25 50 100 150 200 Kilometers

© Source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/fairuse.
Satellite Imagery

- **Raw data**
  - Array of pixels precessing around the earth
  - Spherical co-ordinates

- **Cooked into images**
  - Typically “best” pixel over a time window
  - i.e. image is a composite of several passes

- **Further cooked into various other things**
  - E.g. polygons of constant snow cover
Example Queries

- Recook raw data
  - Using a different composition algorithm
- Retrieve cooked imagery in a time cylinder
- Retrieve imagery which is changing at a large rate
Chemical Plant Data

- Plant is a directed graph of plumbing
- Sensors at various places (1/sec observations)
- Directed graph of time series
- To optimize output plant runs “near the edge”
- And fails every once in a while – down for a week
Chemical Plant Data

◆ Record all data

\{(time, sensor-1, \ldots\, sensor-5000)\}

◆ Look for “interesting events – i.e. sensor values out of whack”

◆ Cluster events near each other in 5000 dimension space

◆ Idea is to identify “near-failure modes”
General Model

sensors → Derived data

Cooking Algorithm(s) (pipeline)
Traditional Wisdom

◆ Cooking pipeline outside DBMS
◆ Derived data loaded into DBMS for subsequent querying
Problems with This Approach

- Easy to lose track of the raw data
- Cannot query the raw data
- Recooking is painful in application logic – might be easier in a DBMS (stay tuned)
- Provenance (meta data about the data) is often not captured
  - E.g. cooking parameters
  - E.g. sensor calibration
My preference

- Load the raw data into a DBMS
- Cooking pipeline is a collection of user-defined functions (DBMS extensions)
- Activated by triggers or a workflow management system
- ALL data captured in a common system!!!
Deployment Options

- Supercomputer/mainframe
- Individual project “silos”
- Internal grid (cloud behind the firewall)
- External cloud (e.g. Amazon EC20)
Deployment Options

- Supercomputer/main frame
  - ($$$$$)
- Individual project “silos”
  - Probably what you do now....
  - Every silo has a system administrator and a DBA (expensive)
  - Generally results in poor sharing of data
Deployment Options

- Internal grid (cloud behind the firewall)
  - Mimic what Google/Amazon/Yahoo/et.al do
  - Other report huge savings in DBA/SE costs
  - Does not require you buy VMware
  - Requires a software stack that can enforce service guarantees
Deployment Options

◆ External cloud (e.g. EC2)

◆ Amazon can “stand up” a node wildly cheaper than Exxon – economies of scale from 10K nodes to 500K nodes

◆ Security/company policy issues will be an issue

◆ Amazon pricing will be an issue

◆ Likely to be the cheapest in the long run
What DBMS to Use?

- RDBMS (e.g. Oracle)
  - Pretty hopeless on raw data
    - Simulating arrays on top of tables likely to cost a factor of 10-100
  - Not pretty on time series data
    - Find me a sensor reading whose average value over the last 3 days is within 1% of the average value over the adjoining 5 sensors
What DBMS to Use?

- RDBMS (e.g. Oracle)
  - Spatial data may (or may not) be ok
  - Cylinder queries will probably not work well
  - 2-D rectangular regions will probably be ok
  - Look carefully at spatial indexing support (usually R-trees)
RDBMS Summary

◆ Wrong data model
  ◆ Arrays not tables
◆ Wrong operations
  ◆ Regrid not join
◆ Missing features
  ◆ Versions, no-overwrite, provenance, support for uncertain data, …
But your mileage may vary......

- SQLServer working well for Sloan Skyserver data base
- See paper in CIDR 2009 by Jose Blakeley
How to Do Analytics (e.g. clustering)

- Suck out the data
- Convert to array format
- Pass to MatLab, R, SAS, ...
- Compute
- Return answer to DBMS
Bad News

◆ Painful
◆ Slow
◆ Many analysis platforms are main memory only
RDBMS Summary

◆ Issues not likely to get fixed any time soon
◆ Science is small compared to business data processing
Wrapper on Top of RDBMS -- MonetDB

- Arrays simulated on top of tables
- Layer above RDBMS will replace SQL with something friendlier to science
- But will not fix performance problems!!

Bandaid solution......
RasDaMan Solution

◆ An array is a blob
  ◆ or array is cut into chunks and stored as a collection of blobs
◆ Array DBMS is in user-code outside DBMS
  ◆ Uses RDBMS as a reliable (but slow) file system
◆ Grid support looks especially slow
My Proposal -- SciDB

◆ Build a commercial-quality array DBMS from the ground up.
SciDB Data Model

- Nested multidimensional arrays
  - Augmented with co-ordinate systems (floating point dimensions)
- Ragged arrays
- Array values are a tuple of values and arrays
Data Storage

• Optimized for both dense and sparse array data
  □ Different data storage, compression, and access

• Arrays are “chunked” (in multiple dimensions)

• Chunks are partitioned across a collection of nodes

• Chunks have ‘overlap’ to support neighborhood operations

• Replication provides efficiency and back-up

• Fast access to data sliced along any dimension
  □ Without materialized views
CREATE ARRAY Test_Array
< A: integer NULLS,
  B: double,
  C: USER_DEFINED_TYPE >
[I=0:99999,1000, 10, J=0:99999,1000, 10 ]
PARTITION OVER ( Node1, Node2, Node3 )
USING block_cyclic();

attribute names
A, B, C

index names
I, J

chunk size
1000

overlap
10
Array Query Language (AQL)

- Array data management (e.g. filter, aggregate, join, etc.)
- Stat operations (multiply, QR factor, etc.)
  - Parallel, disk-oriented
- User-defined operators (Postgres-style)
- Interface to external stat packages (e.g. R)
Array Query Language (AQL)

```
SELECT Geo-Mean (T.B)
FROM Test:Array T
WHERE
  T.I BETWEEN :C1 AND :C2
AND T.J BETWEEN :C3 AND :C4
AND T.A = 10
GROUP BY T.I;
```

- User-defined aggregate on an attribute B in array T
- Subsample
- Filter
- Group-by

So far as SELECT / FROM / WHERE / GROUP BY queries are concerned, there is little logical difference between AQL and SQL
Matrix Multiply

CREATE ARRAY TS_Data < A1:int32, B1:double >
 [ I=0:99999,1000,0, J=0:3999,100,0 ]

Select  multiply (TS_data.A1, test_array.B)

- Smaller of the two arrays is replicated at all nodes
  - Scatter-gather
- Each node does its “core” of the bigger array with the replicated smaller one
- Produces a distributed answer
Architecture

- Shared nothing cluster
  - 10’s-1000’s of nodes
  - Commodity hardware
  - TCP/IP between nodes
  - Linear scale-up

- Each node has a processor and storage

- Queries refer to arrays as if not distributed

- Query planner optimizes queries for efficient data access & processing

- Query plan runs on a node’s local executor & storage manager

- Runtime supervisor coordinates execution

Application Layer
  - Java, C++, whatever...

Language Specific UI
  - Doesn’t require JDBC, ODBC

Runtime Supervisor
  - AQL an extension of SQL
  - Also supports UDFs

Query Interface and Parser

Plan Generator

Node 1
  - Local Executor
  - Storage Manager

Node 2

Node 3

Application Layer

Server Layer

Storage Layer
Other Features
Which Science Guys Want
(These could be in RDBMS, but Aren’t)

◆ Uncertainty
  ◆ Data has error bars
  ◆ Which must be carried along in the computation
    (interval arithmetic)
Other Features

◆ Time travel
  ◆ Don’t fix errors by overwrite
  ◆ I.e. keep all of the data
◆ Named versions
  ◆ Recalibration usually handled this way
Other Features

◆ Provenance (lineage)
  ◆ What calibration generated the data
  ◆ What was the “cooking” algorithm
  ◆ In general – repeatability of data derivation