Data Formats and Databases

Linda Woodard
Consultant
Cornell CAC

Workshop: Data Analysis on Ranger, January 19, 2012
How will you store your data?

• Binary data is compact but not portable
  – Machine readable only
  – Byte-order issues: big endian (IBM) vs. little endian (Intel)

• Formatted text is portable but not compact
  – Need to know all the details of formatting to read the data
  – 1 byte of ASCII text stores only a single decimal digit (~3 bits)
  – Compression can help, but is slow and often impractical for large files

• Need to consider how data will be used
  – Is portability an issue?
  – Will your favorite analysis tools be able to read the data?
  – Are there storage constraints?
Data Preservation and Discovery

• NSF requires a data management plan with all grant proposals
  Metadata
  Formats used
  Data location
  Discovery and access plans
  https://confluence.cornell.edu/display/rdmsgweb/Home

• Large Research Projects
  Personnel
  Long time horizons
  Distant collaborators

• Scientific data formats address some of these issues…
<table>
<thead>
<tr>
<th>Data Format</th>
<th>Academic Discipline</th>
<th>Parallel I/O</th>
<th>Software Interfaces</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDF5</td>
<td>2D and higher dimensional data</td>
<td>yes</td>
<td>C, C++, Fortran, Java, Python, Perl, IDL, Matlab, Mathematica</td>
<td>developed at NCSA</td>
</tr>
<tr>
<td>NetCDF</td>
<td>Earth Sciences</td>
<td>yes</td>
<td>C, C++, Fortran, Java, Python, Perl, Ruby, IDL, R, Matlab, ArcGIS</td>
<td>developed at UCAR</td>
</tr>
<tr>
<td>FITS</td>
<td>Astrophysics</td>
<td>no</td>
<td>C, C++, Fortran, Java, Python, Perl, IDL, R, Matlab, Mathematica</td>
<td>developed at NASA</td>
</tr>
<tr>
<td>Silo</td>
<td>General Visualization</td>
<td>yes</td>
<td>VisIt</td>
<td>developed at LLNL</td>
</tr>
</tbody>
</table>
Scientific Data Formats: HDF5

- Versatile data model that can represent complex data objects and metadata
- Portable file format with no limit on the number or size of data objects
- Open software library that runs on platforms from laptops to massively parallel systems
- Integrated performance features that optimize access time and storage space
- Tools and applications for managing, manipulating, viewing, and analyzing the data in the collection

Source: www.hdfgroup.org/hdf5
HDF5 Features

- *Headers* include extensive metadata (datatypes, dimensionality, storage layout); files are self-documenting

- *Virtual file layer* provides flexible storage and transfer capabilities: Standard (Posix), Parallel, and Network I/O file drivers

- *Compression & chunking* increase access and storage efficiency

- *Datatype transformations* can be performed during I/O operations

- *Subsetting* reduces transferred data volume & improves access speed during I/O operations

*Source: www.hdfgroup.org/hdf5*
Scientific Data Formats: netCDF

• Similar to HDF5; newest version uses the HDF5 format

• Used extensively in the Earth Sciences community for time varying geospatial data; most data from NOAA is in netCDF format

• NetCDF has good tools for geo-gridded data

  – *Ferret* ([http://ferret.wrc.noaa.gov/Ferret/](http://ferret.wrc.noaa.gov/Ferret/)) offers a Mathematica-like approach to analysis. Variables and expressions may be defined interactively; calculations may be applied over arbitrarily shaped regions; geophysical formatting is built in.

netCDF Features

- **Self-Describing**—files include information about the data they contain
- **Portable**—endian problems handled automatically
- **Direct-access**—subsets of a larger dataset can be accessed without reading through all the preceding data
- **Appendable**—data may be appended to a properly structured netCDF file without copying the dataset or redefining its structure
- **Shareable**—one writer and multiple readers may simultaneously access the same netCDF file
- **Archivable**—netCDF will always be backwards compatible

Source: [http://www.unidata.ucar.edu/software/netcdf](http://www.unidata.ucar.edu/software/netcdf)
Scientific Data Formats: Silo

- Silo is a library for reading and writing scientific data to binary disk files
- Silo supports point meshes, structured and unstructured meshes in 2D and 3D
- Two layers
  - API with Fortran, C, and Python interfaces
  - I/O driver (HDF5 is one of these drivers)
- Primary file format for VisIt

https://wci.llnl.gov/codes/silo/index.html
Why Parallel I/O is important

- P0 may become bottleneck
- System memory may be exceeded on P0
Why Parallel I/O is important – part 2

- Possible to achieve good performance
- May require post-processing
- More work for applications, programmers
Why Parallel I/O is important – part 3

- HDF5, netCDF and Silo can take the place of a parallel I/O library - they’ve linked the parallel I/O library for you

- Variant: only P1 and P2 act as parallel writers; they gather data from P0 and P3 respectively (chunking)
Scientific Data Formats: Scientific Databases

• What is a database?
  from Wikipedia—an organized collection of data

• When to use a database
  Data hierarchy more complicated than space/time dimensions

• Database added value
  Built-in data integrity checks
    Management of row duplication
    Enforcement of data ranges and types

  Enforces planning about the data to be stored
    Data types (integer, decimal, datetime), scale, precision
    Missing data (null values)

  Scalability
Scientific Databases vs. Hierarchical Data Formats

• Academic Disciplines  
  All with any kind of hierarchical data

• Parallel I/O  
  Available from many commercial vendors and open sources

• Software interfaces to SQL databases  
  C, C++, C#, Fortran, Java, Python, Perl, Ruby, IDL, R, Matlab, ArcGIS, Excel

• Advanced query capabilities  
  Fine grained ability to extract subsets of the data efficiently
Relational Database Software

- Enterprise-class relational database systems
  - Oracle
  - Microsoft SQL Server
  - MySQL
  - PostgreSQL

- Small, light-weight relational database systems
  - SQLite (C based)
  - SmallSQL (Java based)
  - Apache Derby (Java based)
  - Gadfly (Python based)
Real life example

• A Facebook application that allows people to show their arXiv.org papers on their Facebook profile page

• The application needs to store information about papers so that it can extract these papers based on queries about authors

• Conceptually we have a couple of objects we want to connect:
  – Authors (Facebook ID, arXiv info, etc.)
  – Papers (title, abstract, journal reference, etc.)

• Two approaches to this problem
Approach one – Delimited flat file

- Add a row to the table for every unique paper an author has written

- Search the table for all rows that have the appropriate ID

<table>
<thead>
<tr>
<th>Facebook ID</th>
<th>Paper ID</th>
<th>Paper Title</th>
<th>Paper Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>2341234</td>
<td><a href="http://arxiv.org/abs/5234">http://arxiv.org/abs/5234</a></td>
<td>Particle Pleasantry</td>
<td>CH Foo</td>
</tr>
<tr>
<td>2341234</td>
<td><a href="http://arxiv.org/abs/3234">http://arxiv.org/abs/3234</a></td>
<td>Reading is neat</td>
<td>CH Foo, RG Fields</td>
</tr>
<tr>
<td>1234123</td>
<td><a href="http://arxiv.org/abs/3234">http://arxiv.org/abs/3234</a></td>
<td>Reading is neat</td>
<td>CH Foo, RG Fields</td>
</tr>
</tbody>
</table>

- Benefits:
  - Easy to add new entries (depending on sorting); harder with more entries
  - Simple to code the read/write functions

- Problems:
  - A row is duplicated for every author of a paper (e.g., Reading is neat)
  - To match a given Facebook ID, a linear scan of the entire file is required
Approach two – Relational database

• Define *two* tables
  Users/Authors table and Papers table

• Link them together by the paperID
  *PaperID* in Papers is a *Primary Key*; it uniquely identifies a row
  *PaperID* in Users is a *Foreign Key*; it points to a row in another table

• Benefits
  Fast retrieval of papers for an author
  Easy to add fields to the Users and Papers tables

• Problems
  Database management
More relationships

- We can create relationships that are much more finely-grained
  - Make the Users table more general
  - Create a separate Authors table and a Reviewers table for reviews
Relational databases

- Relational databases are based on the relational model--data is expressed by a set of binary relationships.

Flat files would replicate columns of metadata for each row.

The replication gets worse when the metadata is hierarchical.

![Database Diagram]

Sensor Type
- SensorTypeID
- SensorTypeDescription
- SensorValue

Sensors
- SensorID
- SensorTypeID
- SensorName

Sensor Data
- SensorID
- SensorLocationID
- ReadTime
- Measurement

Sensor Location
- SensorLocationID
- XCoordinate
- YCoordinate
- Online
Flat file or database?

- Flat files are useful for
  - Small datasets
  - Static dumping of data

- Databases are useful for
  - Evolving data
  - Data where searching/querying is important/complex
  - Expressing relationships that are not captured in a row-based table

- Other factors to consider:
  - Database overhead
  - Expectations about sharing data
Interacting with a database – SQL

• **SQL – Structured Query Language**

  A programming language designed for the creation, management, modification and retrieval of data from a database

  All databases speak SQL, though many also provide non-standard extensions

  Using a database requires a basic knowledge of SQL

  Designing a database requires extensive knowledge of SQL

• **PL/SQL and SQL/PSM**

  Database extensions for creating stored procedures
SQL language – Select, Where

Select & Where control what subset of data to obtain from the database

- Retrieve sensor locations
  
  ```
  SELECT XCoordinate, YCoordinate
  FROM SensorLocation
  ```

- Retrieve sensor data for a one sensor
  
  ```
  SELECT * FROM SensorData
  WHERE SensorID = 200
  ```

- Retrieve sensor data for a one time period
  
  ```
  SELECT * FROM SensorData
  WHERE ReadTime = '1/1/2012'
  ```
SQL language – Join

Use Join to retrieve data from more than one table

Retrieve sensor data with locations

```sql
SELECT SensorID, ReadTime, Measurement, Xcoordinate, YCoordinate
FROM SensorData SD
INNER JOIN SensorLocation SL
ON SD.SensorLocationID = SL.SensorLocationID
```

Retrieve sensor data with sensor types

```sql
SELECT SD.SensorID, ReadTime, Measurement, SensorValue
FROM SensorData SD
INNER JOIN Sensors S
ON SD.SensorID = S.SensorsID
INNER JOIN SensorType ST
ON S.SensorTypeID = ST.SensorTypeID
```
SQL language – insert, update, commit

New rows can be inserted and existing rows can be updated

Insert a new sensor

```
INSERT INTO SensorType (SensorTypeID, SensorTypeDescription, SensorValue)
Values(1001,'Heat Sensor',20)
```

```
INSERT INTO Sensors
Values(89019, 1001, 'Cayuga')
```

Update the sensor data

```
UPDATE Sensors
SET SensorName = 'Cayuga Lake'
WHERE SensorID=89019
```

Commit the transactions

```
COMMIT
```
SQL language – delete, order, etc.

Dropping rows from a table mirrors SELECTing

- Delete data for a particular day
  ```sql
  DELETE FROM SensorData
  WHERE ReadTime = '1/1/2011'
  ```

- No assumptions can be made about the order that rows are retrieved
  - Sort rows by location and within location by time
    ```sql
    SELECT * FROM SensorData
    ORDER BY SensorLocationID, ReadTime
    ```
Using SQL in application code

- Most programming languages have SQL interfaces which are software modules that provide a connection to the database and a cursor.

```python
import MySQLdb
conn = MySQLdb.connect(host="h", user="u", passwd="p321", db="test")
cursor = conn.cursor()
cursor.execute("SELECT * FROM SensorData WHERE SensorID = 1234")
row = cursor.fetchone()
cursor.execute ("SELECT * FROM SensorLocations")
row = cursor.fetchall()
cursor.close()
conn.close()
```

- Note: Many interfaces have a special executeQuery function which returns an iterable to retrieve rows (res->next())
SQL language assessment

• **Benefits:**
  – SQL is a relatively simple language
  
  – Interfaces exist from many programming languages to every type of database; all reasonable databases support SQL; therefore SQL is a ubiquitous choice
  
  – Lines of code can be drastically reduced by taking advantage of powerful SQL commands for searching and retrieving objects from a database

• **Problems:**
  – SQL queries can be amazingly inefficient; there are tools for optimization
  
  – Another language to learn
Object-relational mapping

- Object-relational mapping (also ORM and O/R mapping) converts data between a database and an object-oriented programming language.

- An ORM tool lets you create and use a database within a standard OO programming paradigm:
  - Database tables are created from class definitions
  - SQL queries are basically written for you by the tool.

- The ORM tools also allow you direct SQL access where optimized queries are needed.
OR mapping

Script to create database tables

Begin;
CREATE TABLE ‘SensorType’ (  
   ‘SensorTypeID’ integer NOT NULL PRIMARY KEY,  
   ‘SensorTypeDescription’ varchar(100) NOT NULL,  
   ‘SensorValue’ integer NOT NULL);

CREATE TABLE ‘Sensors’ (  
   ‘SensorID’ integer NOT NULL PRIMARY KEY,  
   ‘SensorTypeID’ integer NOT NULL,  
   ‘SensorName’ varchar(100) NOT NULL);

CREATE TABLE ‘SensorLocation’ (  
   ‘SensorLocationID’ integer NOT NULL PRIMARY KEY,  
   ‘XCoordinate’ integer NOT NULL,  
   ‘YCoordinate’ integer NOT NULL,  
   ‘Online’ integer NOT NULL;)

CREATE TABLE ‘SensorData’ (  
   ‘SensorID’ integer NOT NULL,  
   ‘SensorLocationID’ integer NOT NULL,  
   ‘RealTime’ datetime NOT NULL,  
   ‘Measurement’ integer NOT NULL;)

ALTER TABLE ‘SensorData’ ADD CONSTRAINT SensorID_refs_id
FOREIGN KEY (‘SensorID’ REFERENCES ‘Sensors’ (‘SensorID’));

COMMIT
OR mapping – data structure

The structure of the data is specified using classes and member variables.

Null-ability, default values, primary and foreign keys are easily specified.

Specify the primary key (otherwise one is generated).

This means varchar(100)

class SensorType (Model):
    SensorTypeID = models.IntegerField (primary_key = True)
    SensorTypeDescription = models.CharField(max_length=100, null = False)
    SensorValue = models.IntegerField (null=False)
OR mapping – programming

The notation for dealing with an OR-mapped version is relatively simple but has several important features:

- Transactions/sessions are managed by the mapper.
- Type checking is enforced by the language rather than at runtime by SQL.
- Changing data tables means changing the class structure.
Summary – databases

• Databases can be an effective way to improve your ability to share and manage your data.

• Databases and database technologies are increasingly embedded in a variety of systems and technology stacks to support easy use of these systems are increasingly omnipresent.

• Database languages and tools can help reduce the amount of code you manage in your projects.