HANDLING LARGE SCALE SPATIAL DATA FOR FAST AND SMALL SIZE JOBS

Costantino Soru <costantino.soru@crs4.it>

Introduction

- ✔ We aim at bulding a common interface to access huge amounts of spatial data from small size and fast jobs for a web CWE
- ✔ Data are produced by many VO around a large grid
- ✔ We need to keep the highest level of usability on the web portal
- ✔ We have to make all those data available to the website with an almost zero latency
- ✔ We are looking for the fastest way to interact with the grid infrastructure

The EnviroGRIDS Project

Building capacity for a Black Sea catchment observation and assessment system supporting sustainable development

www.envirogrids.net

- ✔ 1.6~ Million Km²
- ✔ 160~ Million people
- ✔ More than 10 nations
- ✔ 12 main rivers including Danube, Don and Dnieper
- ✔ Multi model analysis
- ✔ Hundreds of simulations

CRS4 and EnviroGRIDS

CRS4 is involved in two tasks:

- ✔ **Development of a Web portal for SWAT (presentation and analysis)**
- ✔ SWAT model gridification

Objective: **Development of decision support system on the Web with highly interactive tools optimized for the report mechanism.**

Problem: **Define a reliable workflow to run fast and small size jobs with almost zero latency on huge amounts of geographically distributed data.**

Solution: **Data uniformation and centralization through dedicated acquisition procedures *(preprocessing).***

The Soil and Water Assessment Tool

SWAT is a river basin scale model developed to quantify the impact of land management practices in large, complex watersheds

SWAT setup and calibration can be done with the ArcView GIS extension, ***AVSWAT*** or ***ArcSWAT***, which handle shapes and rasters

SWAT input is stored in multiple **dbf tables** created by the AVSWAT interface

SWAT results are stored in tabular text files located in standard location: the ***txtinout*** directory

Scenario

Hundreds of SWAT simulations are done by several *Virtual Organizations* around the grid; resulting data are shared with all the other partners.

Our portal must represent all those data making them available to web users in an organic framework, using interactive tools

LONG TIME JOBS

SHORT TIME JOBS

Interactive and navigable maps rendering

Database queries

USABILITY

The more these tasks are fast, the more portal results usable

Get input for these jobs as fast as possible

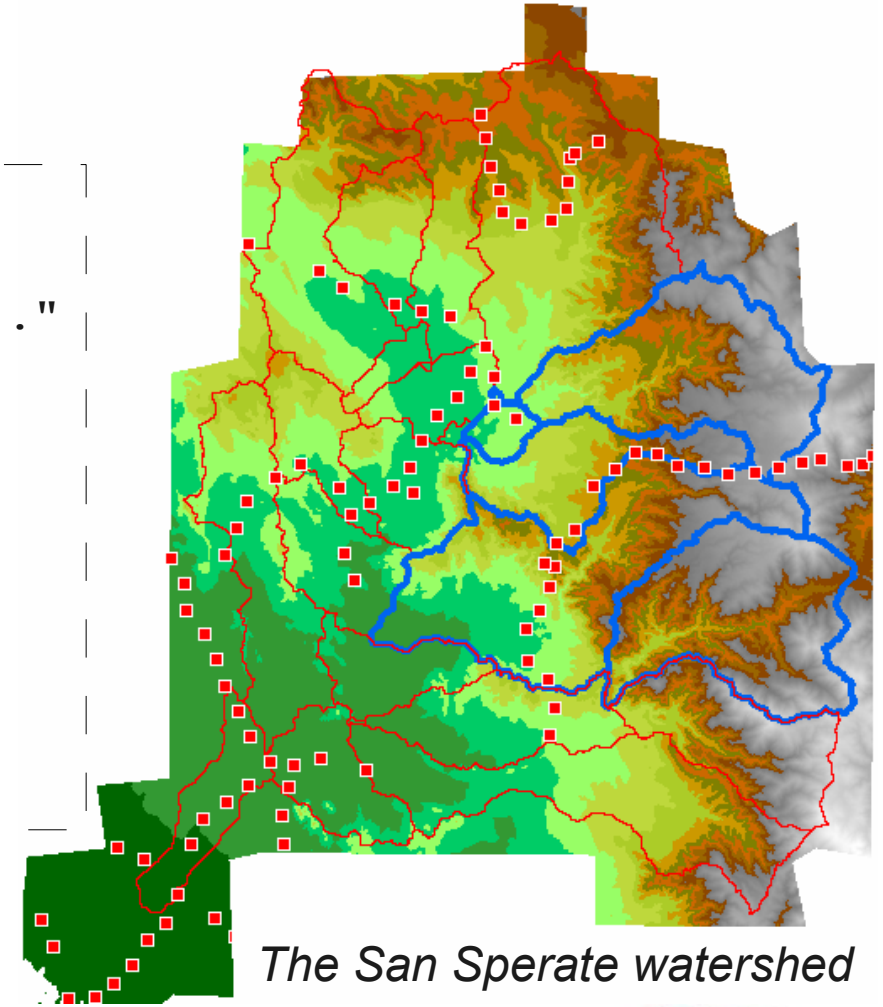
Low Latency Jobs: Mapserver

Maps are navigable, rendering speed influence the end user work

Accessing multiple data sources at the same time with mapserver

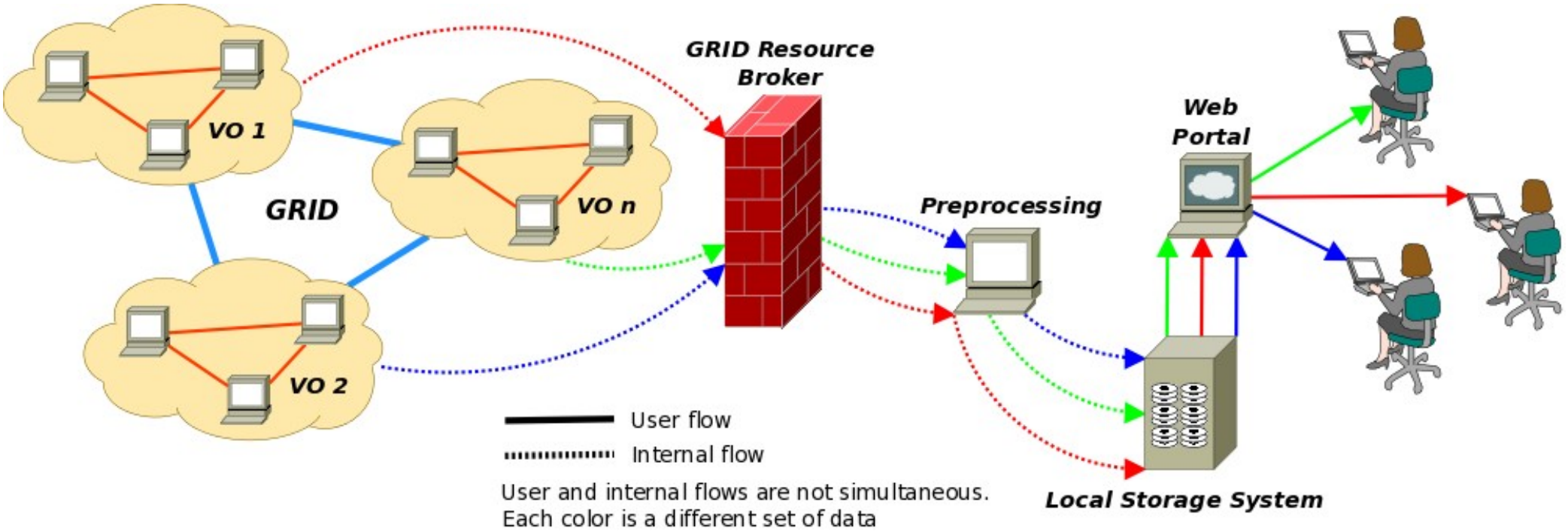
```
TYPE POLYGON  # PostGIS database
CONNECTIONTYPE POSTGIS
CONNECTION "host=localhost dbname=bashyt..."
DATA "the_geom FROM v_shp_sub_y"
...
TYPE RASTER       # Raster PNG file
DATA "/tmp/image.png"
PROCESSING "BANDS=1"
CLASSITEM "[pixel]"
...
TYPE POINT    # ESRI Shapefile
DATA roads
```

Mapfile snippet for the image on the right



The San Sperate watershed

How to Minimize the Overhead



We want to reduce the impact of data flow lag for the end user

Data are cached locally in the same system where the portal lives

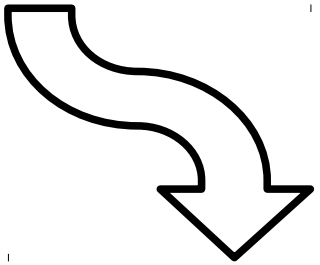
Preprocessing tasks are done during system deployment and upgrade operations only

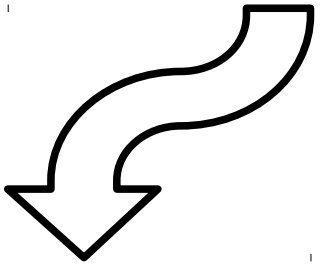
User requests never go across the GRID Broker

Why a Preprocessing Phase?

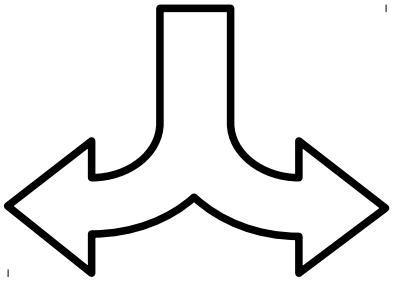
Mixed Formats
Data produced by and for the SWAT model are stored in several distinct well-known formats

Portal tools like maps and charts rendering must share access procedures and data





RESOURCES UNIFORMATION

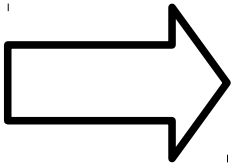


DATA FORMAT ***PROCEDURE***

Data Digestion

We need to:

- ✔ Store data in a structured manner
- ✔ Keep consistency between data
- ✔ Centralize the repository
- ✔ Standardize its access method



RELATIONAL DATABASE

...and:

- ✔ *Provide **automatic procedures** to get raw data from the grid and import them into our database system*

We aim at a **problem generalization**, where not only the SWAT model is involved, but where each numerical model or environment assessment tool used in the project has its own **preprocessing data driver**.

SWAT Data Digestion

What SWAT does:

- ✔ Gets input from dbf tables and text files
- ✔ Puts output on several different text files
- ✔ Generates raster files

The SWAT data and files format are well-known; we always know what kind of information is needed and how to handle it

What we do:

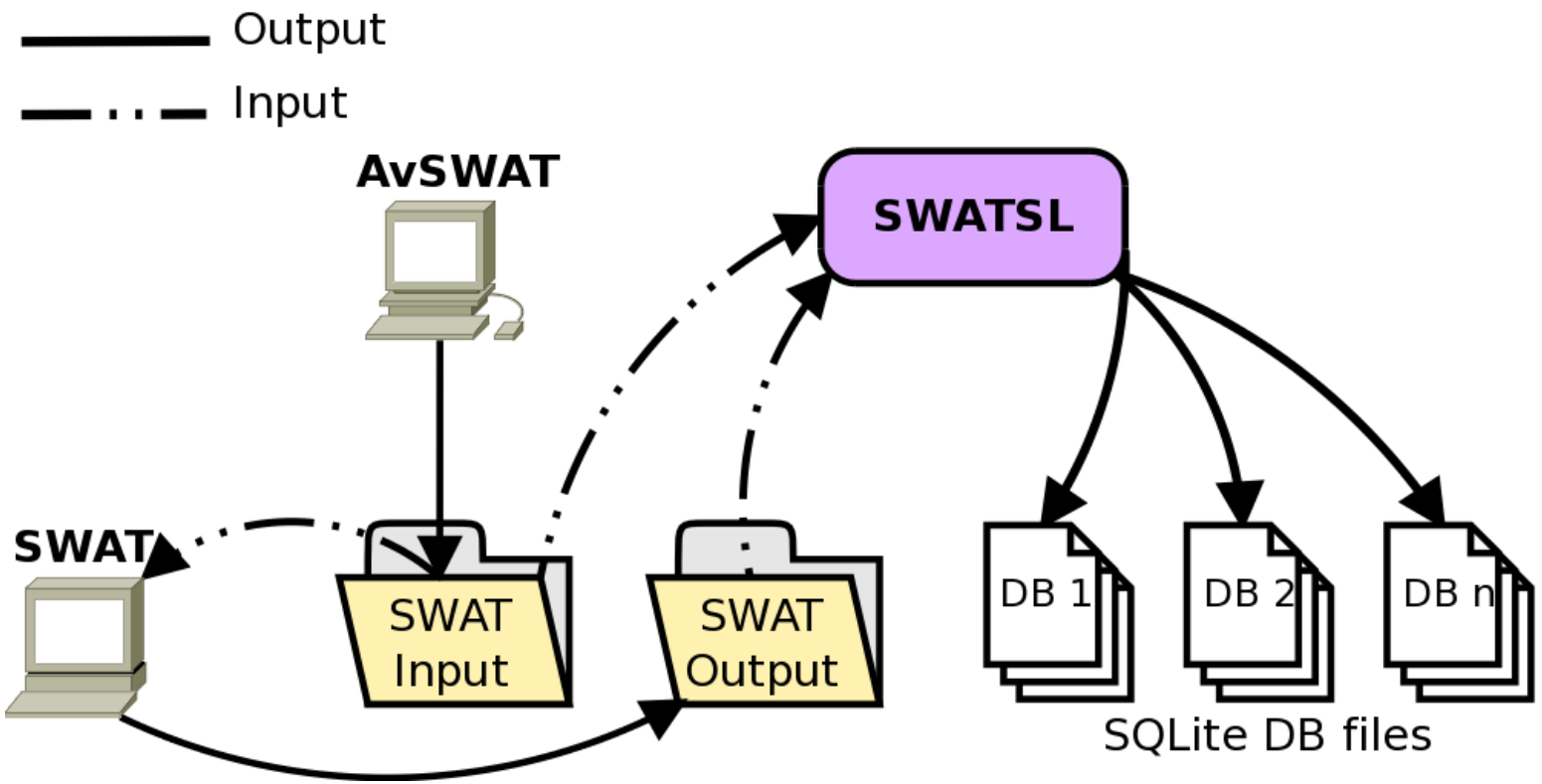
- ✔ Import dbf input tables into db
- ✔ Parse and store text SWAT outputs
- ✔ Put raster data into db... when possible
- ✔ Import shapefiles of interesting areas

SWAT Data Digestion

SWATSL builds one or more SQLite database file and populate them with SWAT simulations and inputs

SWATSL is a C++ library that hide the complexity of the SWAT files architecture providing a uniform dataset for the user

After SWATSL has done its work data can be accessed using common SQL queries



Why SQLite?

What it is

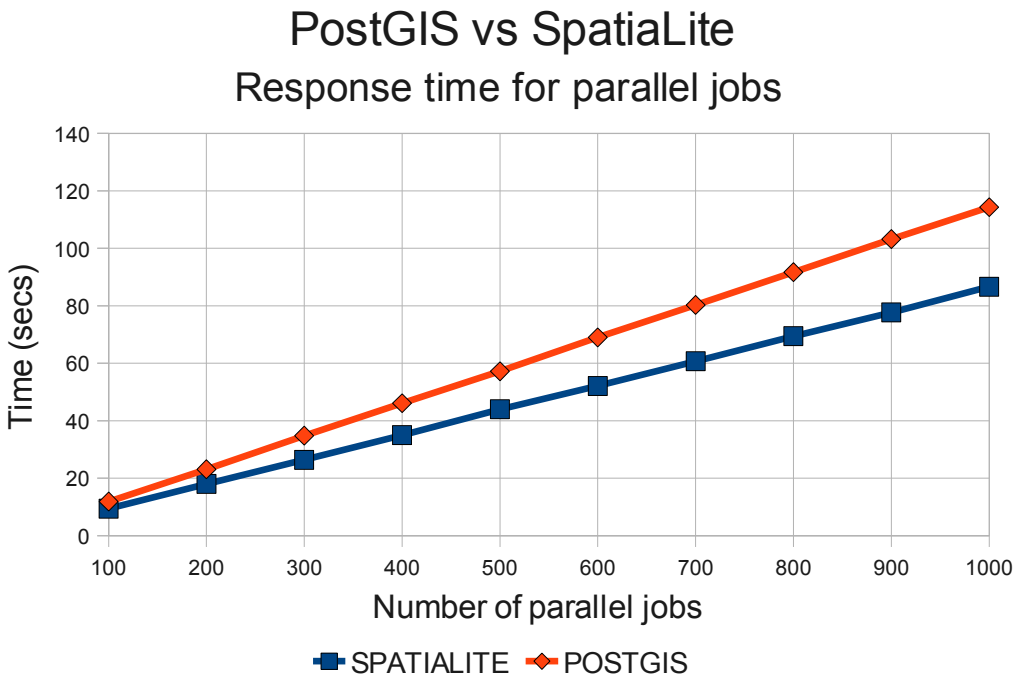
- ✔ **SQLite** is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine
- ✔ The **SpatiaLite** extension enables SQLite to support spatial data too, in a way conformant to OpenGis specifications

Why we are using it

- ✔ Understand how and if this emerging solution can face problems alredy solved using the well-known state-of-the-art PostGIS tool
- ✔ Test its practical usability and complexity
- ✔ It opens several scenarios for scalable storage solutions

PostgreSQL vs SQLite: Load Test

- ✔ Parallel jobs on a single computing node with 24 cores and 16GB of RAM
- ✔ Simple SQL query *SELECT * FROM Table* through shell DB client
- ✔ Default PostgreSQL configuration altered to allow 1000 simultaneous connections



JOBS	SPATIALITE	POSTGIS
100	9,4	11,9
200	18	23,1
300	26,4	34,8
400	35	46,1
500	44	57,2
600	52,1	69
700	60,7	80,3
800	69,4	91,7
900	77,7	103,2
1000	86,6	114,3

(Values in seconds)

PostgreSQL vs SQLite: Mapserver

SQLite gives a bit better average speed to mapserver rendering

```
$ time mapserv -nh "QUERY_STRING=map=test.map&mode=map" > /dev/null
```

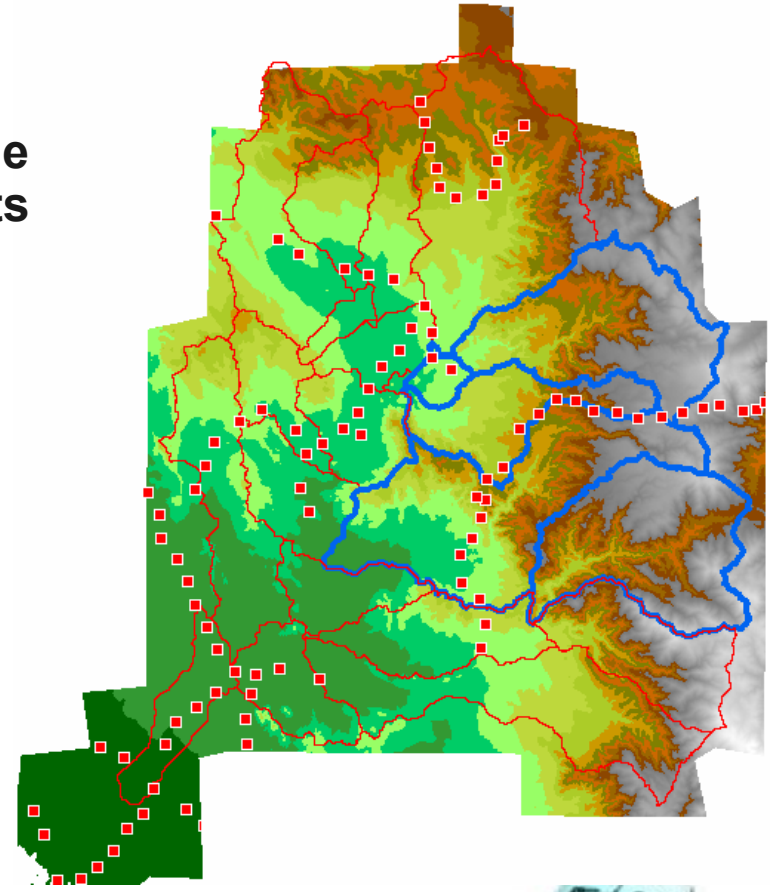
Running the above command using the mapfile for the image on the right we obtain the following time results

PostGIS

```
real    0m0.800s
```

SpatiaLite

```
real    0m0.730s
```



Conclusions and Future Works

- ✔ SQLite and its spatial extension SpatiaLite seem to be a very interesting geo-database solution for small projects or testing purposes
- ✔ We are still experimenting the limits of the SQLite technology
- ✔ Test real cases
- ✔ Define a highly scalable storage system for the local cache repository
- ✔ Provide an efficient method to call preprocessing methods when new data are available inside the grid