2ndQuadrant

Professional PostgreSQL

Writing a parallel and distributed tool for backing up a multi-terabyte data warehouse

Marco Nenciarini
<marco@2ndQuadrant.com>
http://www.2ndQuadrant.com/

EUROPYTHON 2011
FLORENCE, JUNE 20-26

PostgreSQL

License

- Creative Commons:
  - Attribution-NonCommercial-ShareAlike 3.0
  - You are free:
    - to Share — to copy, distribute and transmit the work
    - to Remix — to adapt the work
  - Under the following conditions:
    - Attribution: You must attribute the work in the manner specified by the author or licensor
    - Non-Commercial: You may not use this work for commercial purposes
    - Share Alike: If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one

Source: http://creativecommons.org/licenses/by-nc-sa/3.0/

About me

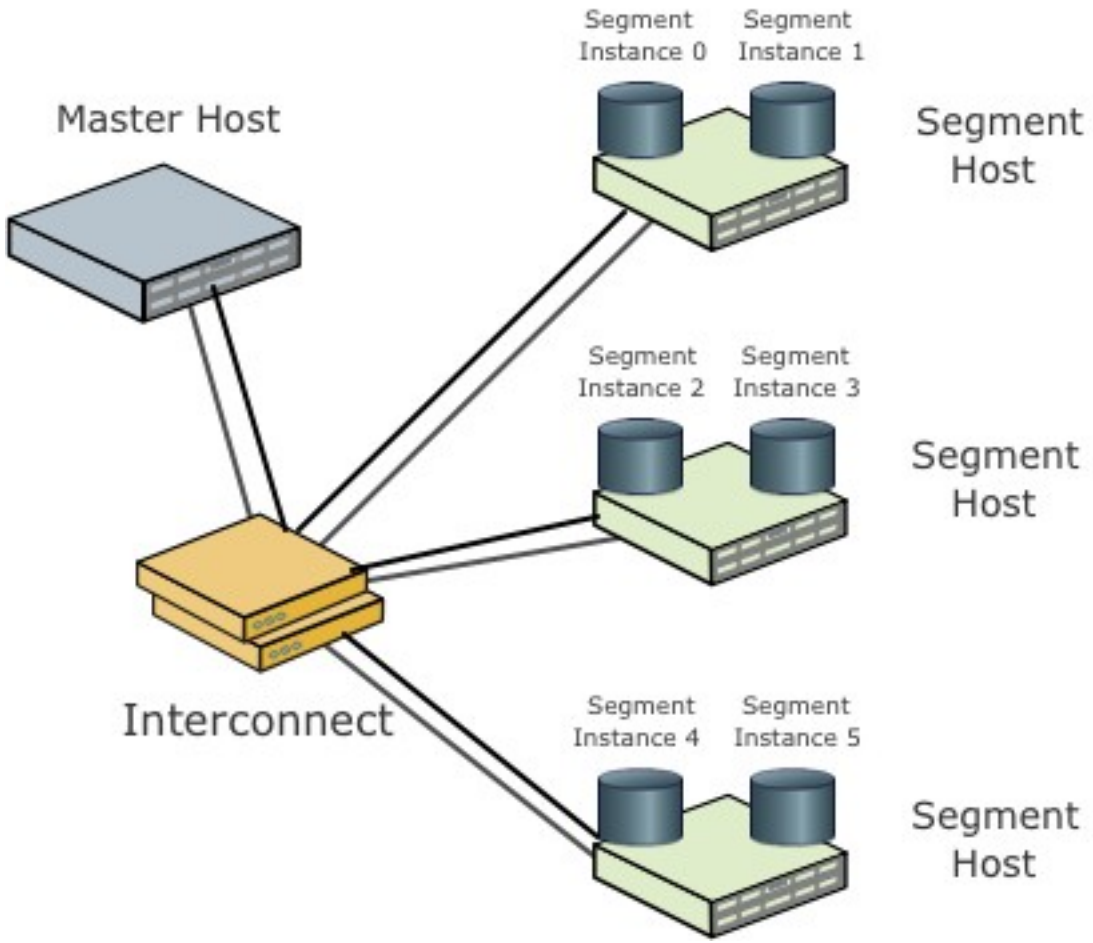
- Linux system administrator for over 10 years
- Debian developer since 2001
- Co-founder of Italian PostgreSQL Users Group (ITPUG)
- PostgreSQL Consultant @ 2ndQuadrant.com

Table of contents

Greenplum Database

- Fork of PostgreSQL (based on 8.2)
- MPP = Massively Parallel Processing
  - Multiple units of parallelism working on the same task
    - Parallel Database Operations
    - Parallel CPU Processing
  - Greenplum Units of Parallelism are “Segments”
- “Shared Nothing” Architecture
  - Segments only operate on their portion of the data
  - Segments are self-sufficient
    - Dedicated CPU Processes
    - Dedicated storage that is only accessible by the segment
- MapReduce implementation for non structured analysis

Greenplum Database



EMC/Greenplum

- In 2010 EMC Corporation acquired Greenplum
- Free Community Edition
  - Research and Development
  - Data Scientist
  - Commercial use (Single node, limits on number of CPUs)
  - More information at http://community.greenplum.com

Context

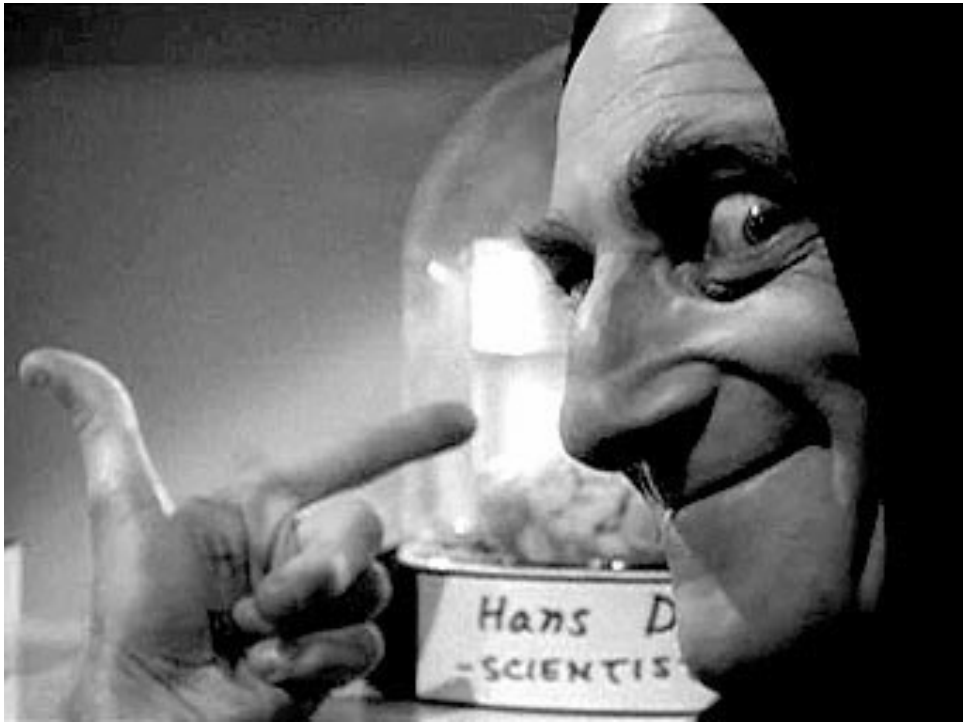
- Backup of a Greenplum powered data warehouse
- Very large customer
- About 100 TB of data
  - Increasing every day
- Over 10k tables
- Many hosts involved
  - One master
  - Multiple segments (on multiple hosts)
  - Multiple backup servers
- Each host has multiple NICs
  - Multiple networks

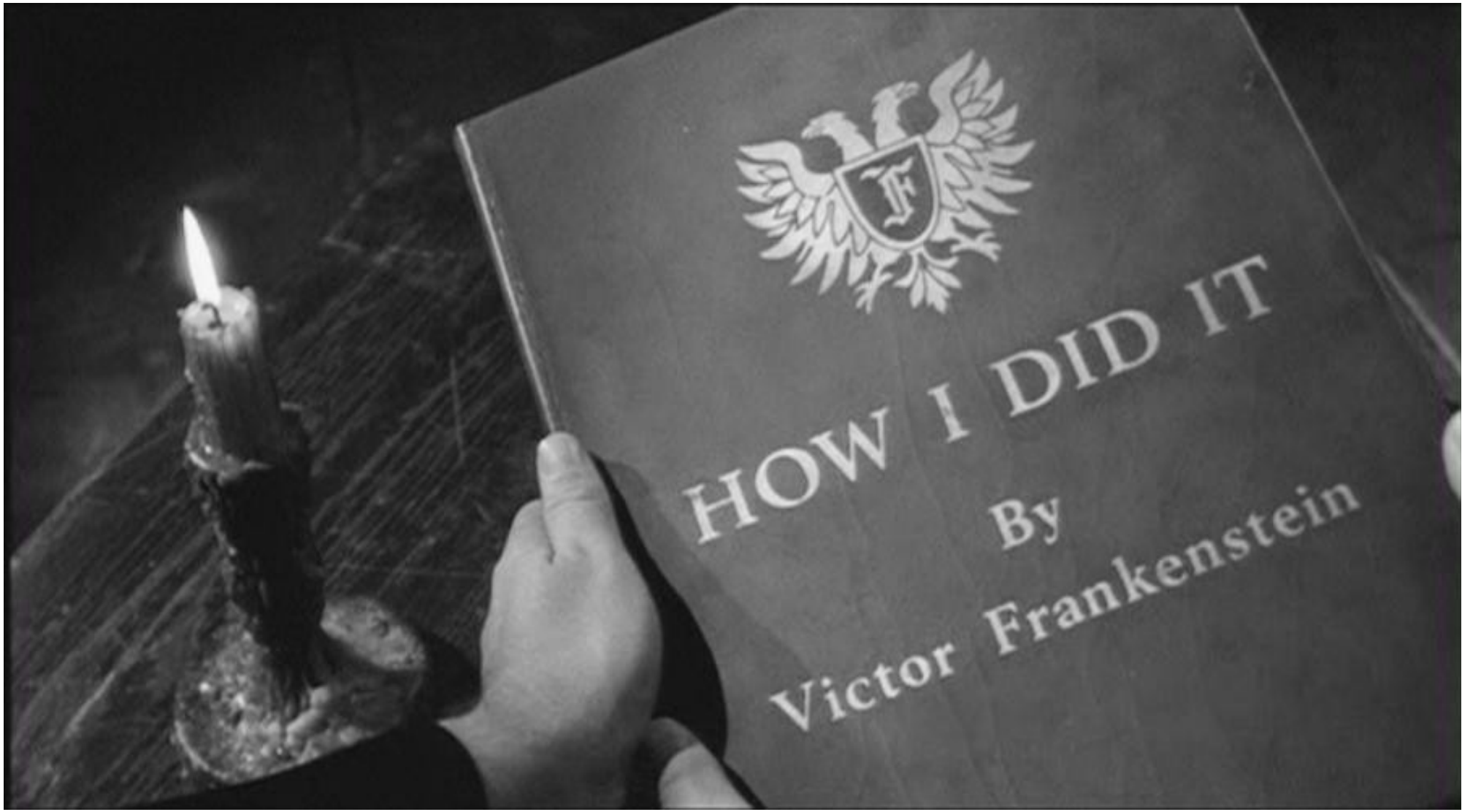
The problem and its main requirements

- **Daily** full logical backup
  - Manage tables individually
  - Incremental
- High performance
  - Backup operation
  - Restore operation
- Scalable
- Backup metadata database (in PostgreSQL)
  - Monitoring and analysis

Main issue

- **Lots of data**
- 100TBs per day or:
  - ~ 9.7Gb/s
- or, if you prefer:
  - *~ 1 DVD every 4 sec*
  - *~ 10 Gigabit eth links*
- Fortunately:
  - GP compresses data
  - *Just 1 DVD every 8 seconds!*





Our idea

- 24-hour time constraint
- Parallel and distributed operations:
  - Maximise hardware resources usage:
    - CPU
    - **Disk**
    - **Network**

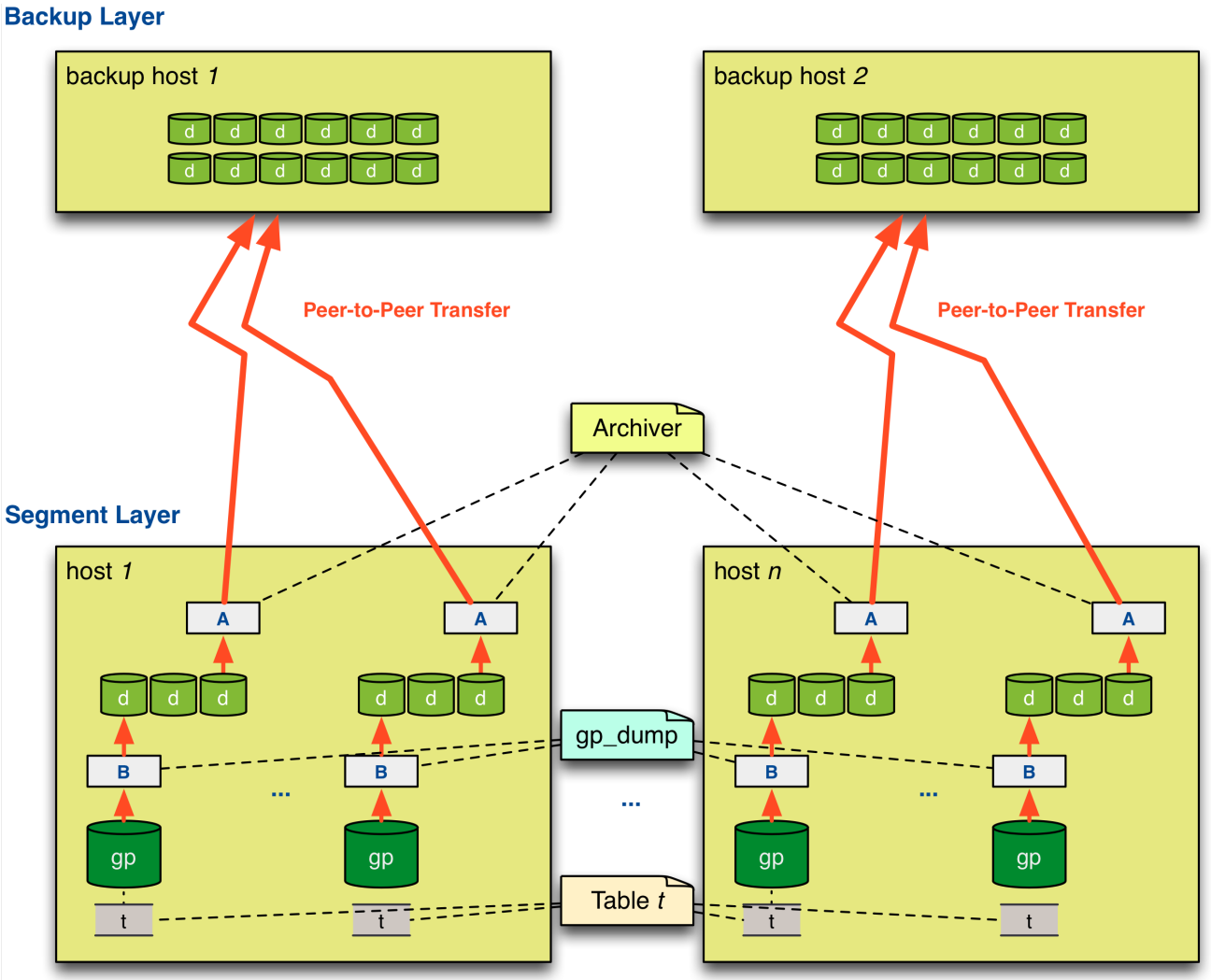
A new project arises

- Codename: “Greenback”

Greenback's architecture

- One centralised master daemon (Manager)
- One distributed agent per Greenplum node
- Peer-to-Peer transfer between Greenplum nodes and backup servers
- Command line utility to interact with the Manager

Greenback's architecture overview



Python

- Possible candidate languages for the implementation:
  - Shell scripting
  - Java
  - C++
  - Python
- The final choice was Python:
  - Extensive and powerful standard library
    - Networking
    - Parallel processing (thread/process)
    - System utilities
  - Fast prototyping and deployment
  - Rapid Application Development
  - …

Python requirements

- Python 2.4 only on backup servers
  - No additional modules
- Python 2.6 on nodes (shipped with Greenplum)
  - Only a few modules are available (paramiko, *pg8000*)
- Python 2.6 on the Master
  - Same modules as the nodes
  - **Psycopg2** available!

Greenback Node Agents

- PYthon Remote Objects (PYRO)
  - Written in 100% pure Python
  - Small, simple and extremely portable
  - Runs wherever Python runs
    - Requires TCP/IP networking
  - Dynamic Proxies - no need for additional tools or classes
  - Transparent remote attribute access
- Dual status reporting
  - Logging and status update in the Postgres database
  - Callbacks to the master through PYRO

Greenback Node Agents

- PYthon Remote Objects (PYRO)
  - Written in 100% pure Python
  - Small, simple and extremely portable
  - Runs wherever Python runs
    - Requires TCP/IP networking
  - Dynamic Proxies - no need for additional tools or classes
  - Transparent remote attribute access
- Dual status reporting
  - Logging and status update in the Postgres database
  - Callbacks to the master through PYRO
- Self deployment via SSH and *tarfile* module

Greenback v1.0alpha

- First attempt of parallel backup
- File transfer through multiple SFTP channels (SSH)
  - Paramiko
- No requirement whatsoever on the backup nodes
  - Apart from SSH :-)

It did not quite work ...

Greenback v1.0alpha failure

- Encryption overhead
  - Not necessary in an extremely secure local network
    - We assume that a security breach in Greenback communications would be the last of the client's concerns …
- Paramiko's performance issues with Python shipped with Greenplum (*requirement*)
  - Paramiko is a great library, but its main goal is not extreme performance

Peer-to-Peer server for backup

- Runs on backup nodes
- Uses the Standard Python Library
  - **SocketServer.TCPServer** with ThreadingMixIn
    - Random port
    - Simple authentication using a security token
    - No data encryption (CPU intensive)
    - Very simple protocol
      - Needs to be extremely fast (remember the 1 day constraint!)
- No need to install the application on backup nodes
  - Launch Python via ssh
  - Load all the required modules from stdin
  - Launch and control the remote server

Sending the code remotely 1/2

```
pack = StringIO()
for module in manifest:
    if os.sep in module:
        dir = os.path.dirname(module)
        while dir != '' and dir != os.sep:
            parent = os.path.join(dir , '__init__.py')
            if parent not in manifest:
                try:
                    data = bundle(parent)
                except:
                    break
                else:
                    pack.write(data)
            dir = os.path.dirname(dir)
    pack.write(bundle(module))
pack.write('%s\n' % main)
```

Sending the code remotely 2/2

```
...
    remote_code = 'import sys; exec compile(sys.stdin.read(%d),
                                        "bootstrap.py", "exec")' % len(bootstrap_code)
    if not host:
        argv = [python, '-c', remote_code]
    else:
        cmd = "%s -c '%s'" % (python, remote_code)
        argv = ['ssh', host, '--', cmd]
    p = subprocess.Popen(argv, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, close_fds=True)
    p.stdin.write(bootstrap_code)
    p.stdin.write(pack.getvalue())
    p.stdin.flush()
...
```

The boostrap code

```
while True:
    name = sys.stdin.readline().strip()
    if name.endswith('.py'):
        len = int(sys.stdin.readline())
        data = zlib.decompress(sys.stdin.read(len))
        modname = name[:-3].replace(os.sep,'.')
        if modname.endswith('.__init__'):
            modname, _ = modname.rsplit('.', 1)
        mod = types.ModuleType(modname)
        exec compile(data, name, "exec") in mod.__dict__

        sys.modules[modname] = mod
    else:
        break
```

It could work!

Greenback v1.1alpha

- It is our current version
- Saturation of the available bandwidth

Conclusions

- Greenback is an under development project
- As of today we are able to meet the full backup requirements
- Next step will be focused on:
  - Incremental backup and backup management
  - Restore operations
- We are thinking of porting the application to PL/Proxy distributed databases in PostgreSQL
- Confidence: Python will help us achieve these goals

Questions?

Thank you

- You can contact Marco Nenciarini via email at marco@2ndQuadrant.com
- For more information on our professional services on PostgreSQL and Greenplum, visit our website http://www.2ndQuadrant.com/
- See you next year!