High Performance Data Management with PyTables & Family

Francesc Alted
falted@carabos.com

September 3th, 2004

SciPy '04 Conference @ CalTech, Pasadena, California

Outline

- Who are we?
- What is PyTables and why does it exist?
- ViTables (interactive demonstration)
- What’s new in PyTables 0.9
- Reliability and performance in 0.9
- CSTables (PyTables goes Client-Server)
- Final remarks

Who are we?

- Cárabos is the company committed to PyTables development and deployment
- We have years of experience in designing software solutions for handling extremely large data sets
- What we offer:
  - Commercial support for PyTables & companion products
  - PyTables-based applications
  - Consulting services for managing complex data environments

Motivation

- Many scientific applications need to save and read very large quantities of data. Analysing this data effectively is a challenge.
- Computers are powerful enough to deal with very large data sets. But, can people handle that and not expire in the attempt?
- Requirements of a data management application:
  - Analysis is an iterative process: interactivity
  - Reading the data over and over: efficiency
  - Solid and flexible framework that would allow the user to provide a clear structure to his data
  - Easy management
- PyTables is a Python package designed with these requirements in mind!

What does PyTables offer?

- Interactivity
  - The user can take immediate action based on previous feedback
  - This greatly accelerates the process of data mining
- Efficiency
  - Very important when interactivity is an issue
- Hierarchical structure
  - Break your data into smaller, related chunks
  - Intuitive way to categorize data
- Object-oriented interface
  - Datasets become objects that can be easily manipulated
  - In a hierarchical structure, objects facilitate data browsing

When PyTables can be useful

Situations where one has to store and efficiently retrieve very large amounts of data. Some examples:

- Scientific applications
  - Meteorology
  - Astronomy
  - Genetics
  - Medicine (Physiological sensors, ...)
- Industrial applications
  - Data acquisition of sensors
  - Real time monitoring
- Data acquisition from IT applications
  - Tracing data from routers
  - System log monitoring
  - Security alerts (Firewalls, IDS, ...)

What’s behind PyTables

PyTables relies on powerful software to achieve its goals:

- Python -- Everyone here knows that (2.2 version needed because generators are heavily used)
- HDF5 -- General purpose library and file format for storing scientific data
- numarray -- Next generation of the well-known Numerical Python package
- Pyrex -- Tool to make Python extensions with a Python-like syntax

ViTables (visualize PyTables datafiles)

Easy-to-use graphical user interface for viewing data (and metadata) in PyTables files

- It uses PyTables and the excellent Qt graphic library, so it is available on most platforms (ViTables runs on most Unix, Linux, MacOSX and Windows systems)

- Can be used as well as a starting point for creating PyTables-based graphical applications (prototyping)

ViTables main features

- Visualizes the object tree graphically
- Can open an independent view for each dataset
- Offers information about the metadata present in nodes
- Can open several files simultaneously
- Can move or copy datasets or complete sub-hierarchies from one group to another, even between different files
- Great browseable documentation
- As a PyTables companion, it deals very well with extremely large datasets (tables exceeding one billion rows)

ViTables interactive demonstration

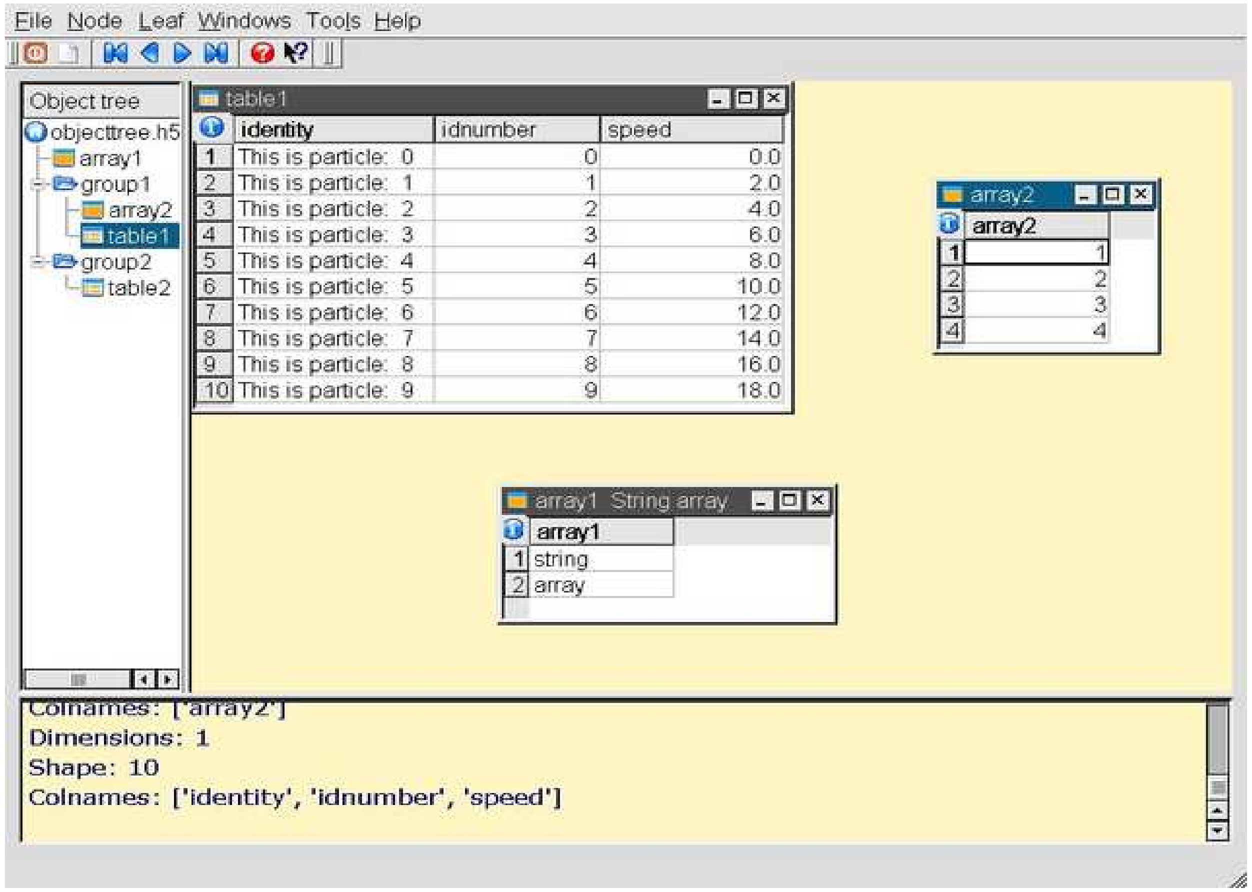


table1

	identity	idnumber	speed
1	This is particle: 0	0	0.0
2	This is particle: 1	1	2.0
3	This is particle: 2	2	4.0
4	This is particle: 3	3	6.0
5	This is particle: 4	4	8.0
6	This is particle: 5	5	10.0
7	This is particle: 6	6	12.0
8	This is particle: 7	7	14.0
9	This is particle: 8	8	16.0
10	This is particle: 9	9	18.0

array2

	array2
1	1
2	2
3	3
4	4

array1 String array

	array1
1	string
2	array

```
Colnames: ['array2']
Dimensions: 1
Shape: 10
Colnames: ['identity', 'idnumber', 'speed']
```

Future plans for ViTables

- Implement graphical row modification & deletion in tables and arrays
- Create an automated test suite
- Improve stability and reliability
- Improve the documentation
- Create a binary installer for Windows (MacOSX?) platforms
- Release the version 1.0 (most likely dual-licensed) by the end 2004

What’s new in PyTables 0.9?

- Table values can be modified (yes, finally :)
- In-kernel selections
- Indexed selections
- Improved speed for large row sizes in Table objects

Modification of values in tables

In PyTables 0.9 two ways of modifying values in Table objects have been introduced

- Row modification

```
rows = numarray.records.array([[457,’db1’,1.2],[6,’de2’,1.3]],
                              formats="i4,a3,f8")
table.modifyRows(start=1, stop=4, step=2, rows=rows)
table[1:4:2] = rows  # shortcut
```

- Column modification

```
table.modifyColumns(start=1, step=2, columns=[[2,3,4]], names=["col1"])
table.cols.col1[1:7:2] = [2,3,4]  # shortcut
# Modify several columns at time:
columns = [["aaa","bbb","ccc"], [1.2, .1, .3]]
table.modifyColumns(start=1, columns=columns, names=["col2", "col3"])
```

In-kernel & Indexed selections

- In-kernel selections:
  - Like regular selections, but more efficient
  - Condition is passed "as is" to the PyTables C extension
  - Can be between 2 to 5 times faster than regular selections
- Indexed selections:
  - An index is created in the same data file
  - Can be 5 to 500 times faster than traditional selects (but slower under some situations!)
- All scalar types are supported (string, ints, floats and booleans)
- Limitation: you can only pass conditions on a single column

In-kernel & Indexed selections syntax

- SQL syntax

```
e = cursor.execute(select sum(col1) from table where 3 < col2 <= 20)
```

- Traditional search in PyTables:

```
# Compute the sum of the column "col1" values that pass the selection
e = sum([row['col1'] for row in table if 3 < row['col2'] <= 20])
# Using generator expressions to save memory (you need Python 2.4!)
e = sum(row['col1'] for row in table if 3 < row['col2'] <= 20)
```

- In-kernel selection:

```
e = sum(row['col1'] for row in table.where(3<table.cols.col2<=20))
```

- Indexed selection:

```
table.cols.col2.createIndex()  # Create the index
e = sum(row['col1'] for row in table.where(3<table.cols.col2<=20))
```

- Mixed selections:

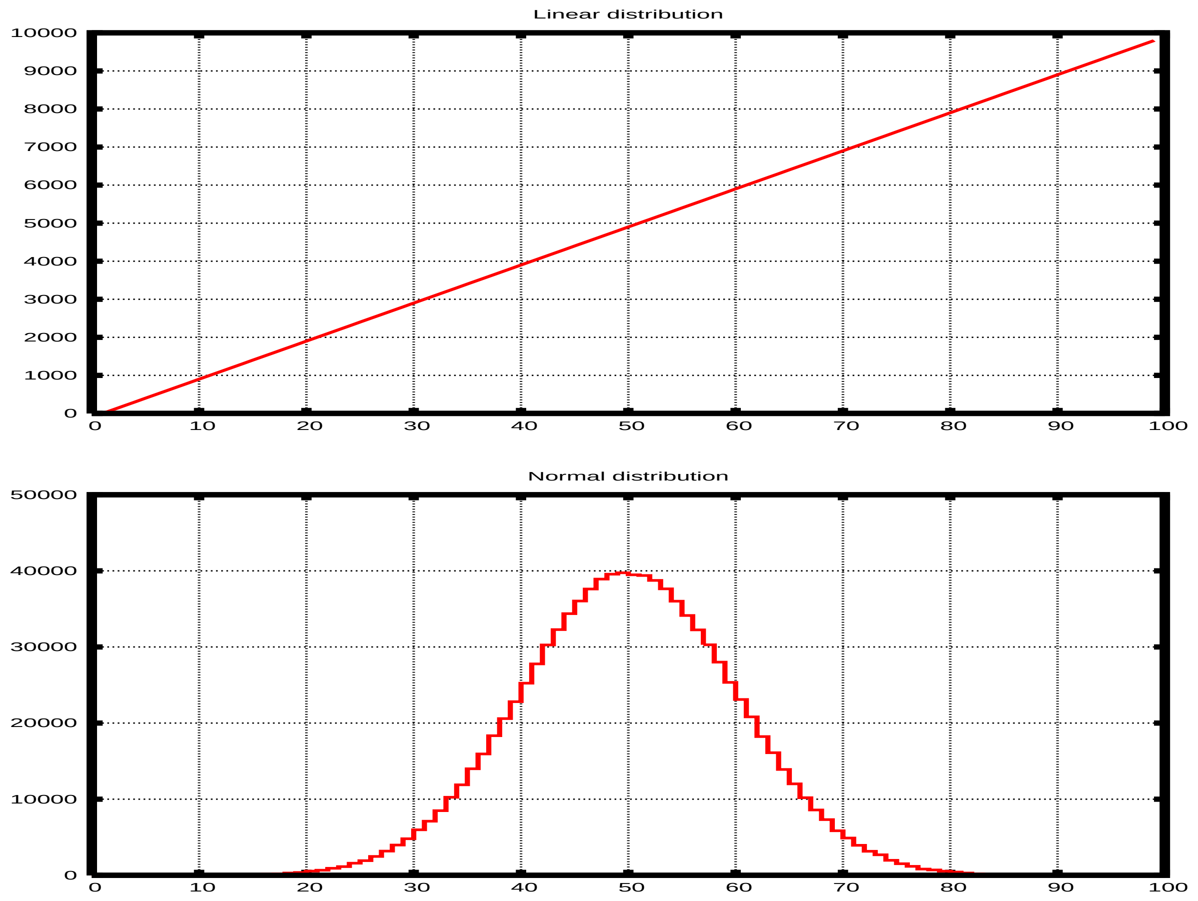
```
e = sum(row['col1'] for row in table.where(3<table.cols.col2<=20)
                    if row['col3'] == 2 and row['col1'] > 4)
```

Always pass the more restrictive selections to the "where" method!

How fast is fast?

- Several benchmarks have been conducted in order to know if PyTables is competitive with existing tools to save data persistently
- Comparisons have been made with SQLite (a fast relational database)
- The benchmarks tested writing and selecting table data under a series of conditions
- Two basic parameters were changed in each test to comparatively measure I/O performance:
  - The number of rows in the table
  - The selection method (regular iterator, in-kernel and indexed)

Dataset election

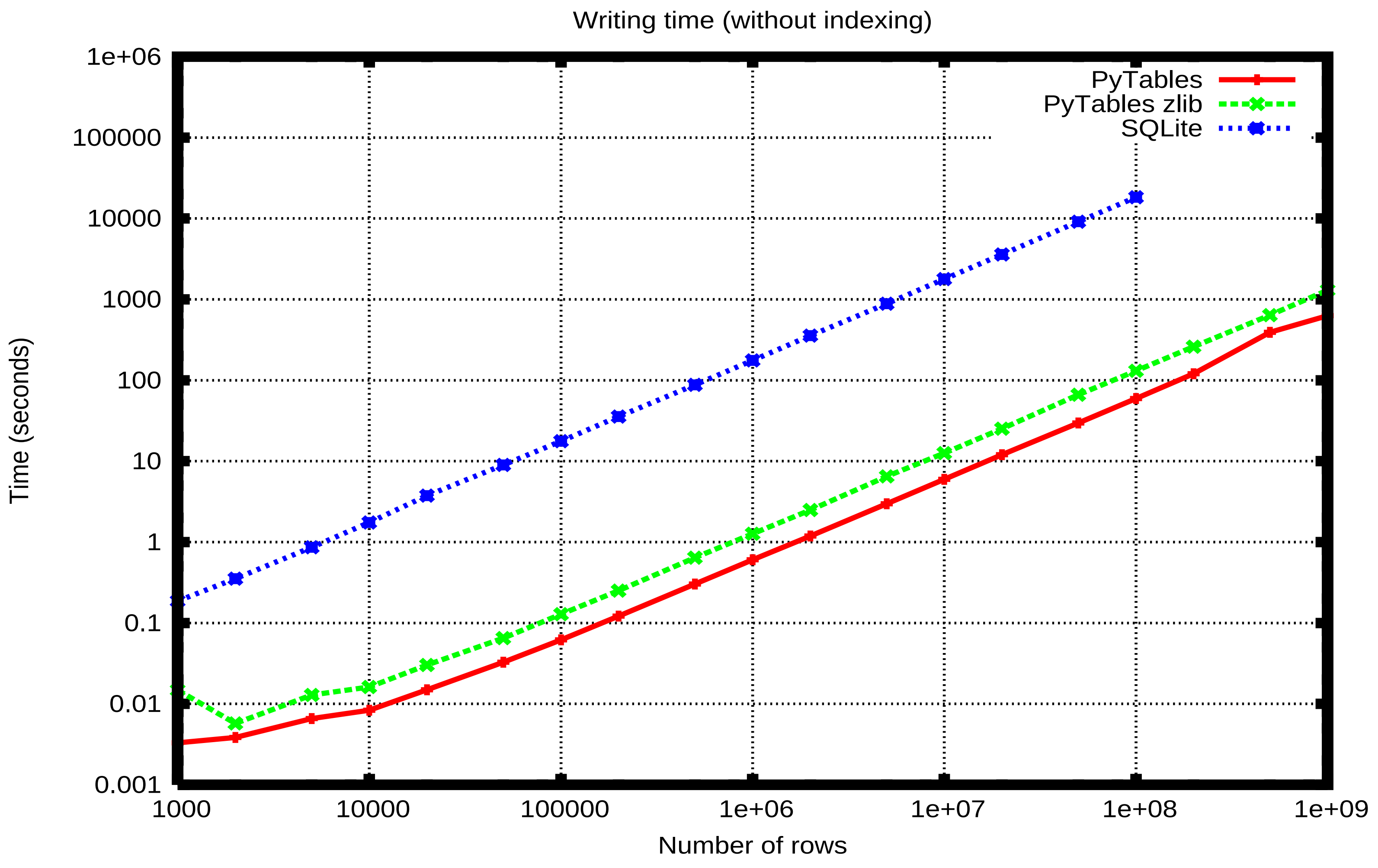


The normal distribution has been chosen because it should be more like "real life" data

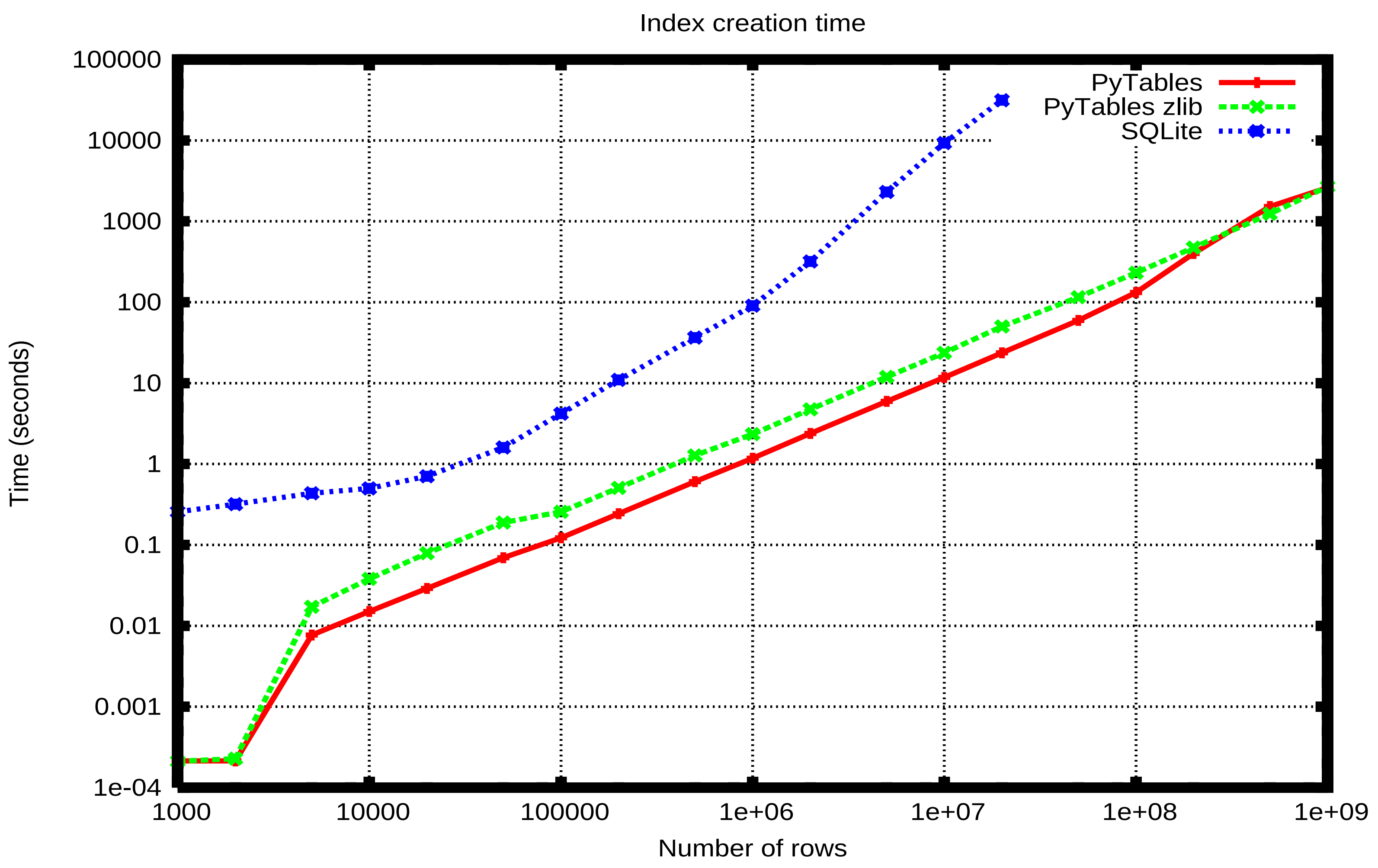
Benchmark platform description

- AMD Opteron @ 1.6 GHz and 8 GB RAM
- IDE disk @ 7200 RPM
- PyTables 0.9 (beta)
- Python 2.3.3
- HDF5 1.6.2
- numarray 1.0
- SuSe GNU/Linux 8.0 (Enterprise)
- Linux Kernel 2.4.21
- GCC 3.2.2 compiler
- SQLite 2.8.14
- PySQLite 0.5

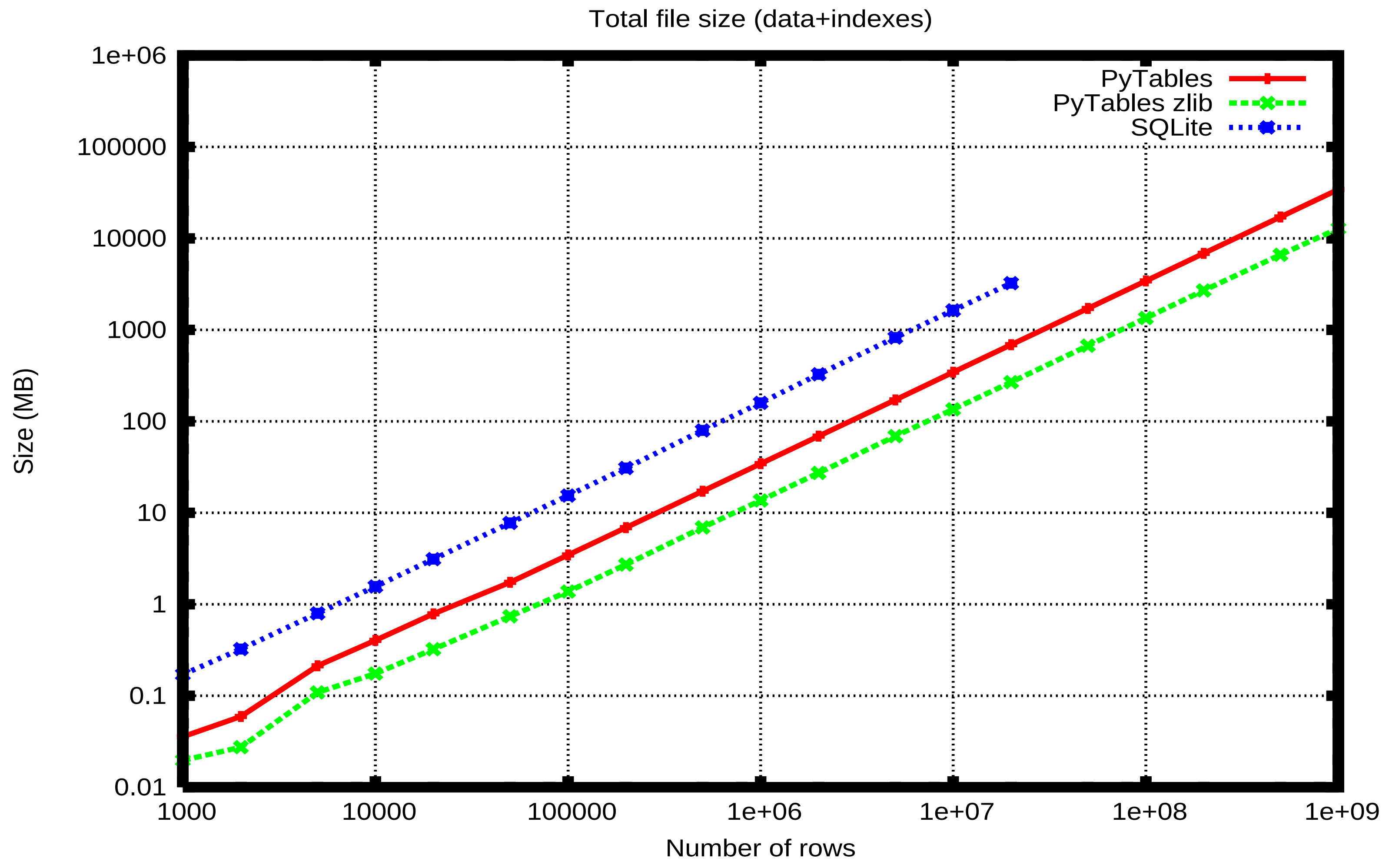
PyTables vs SQLite (time to write entries)



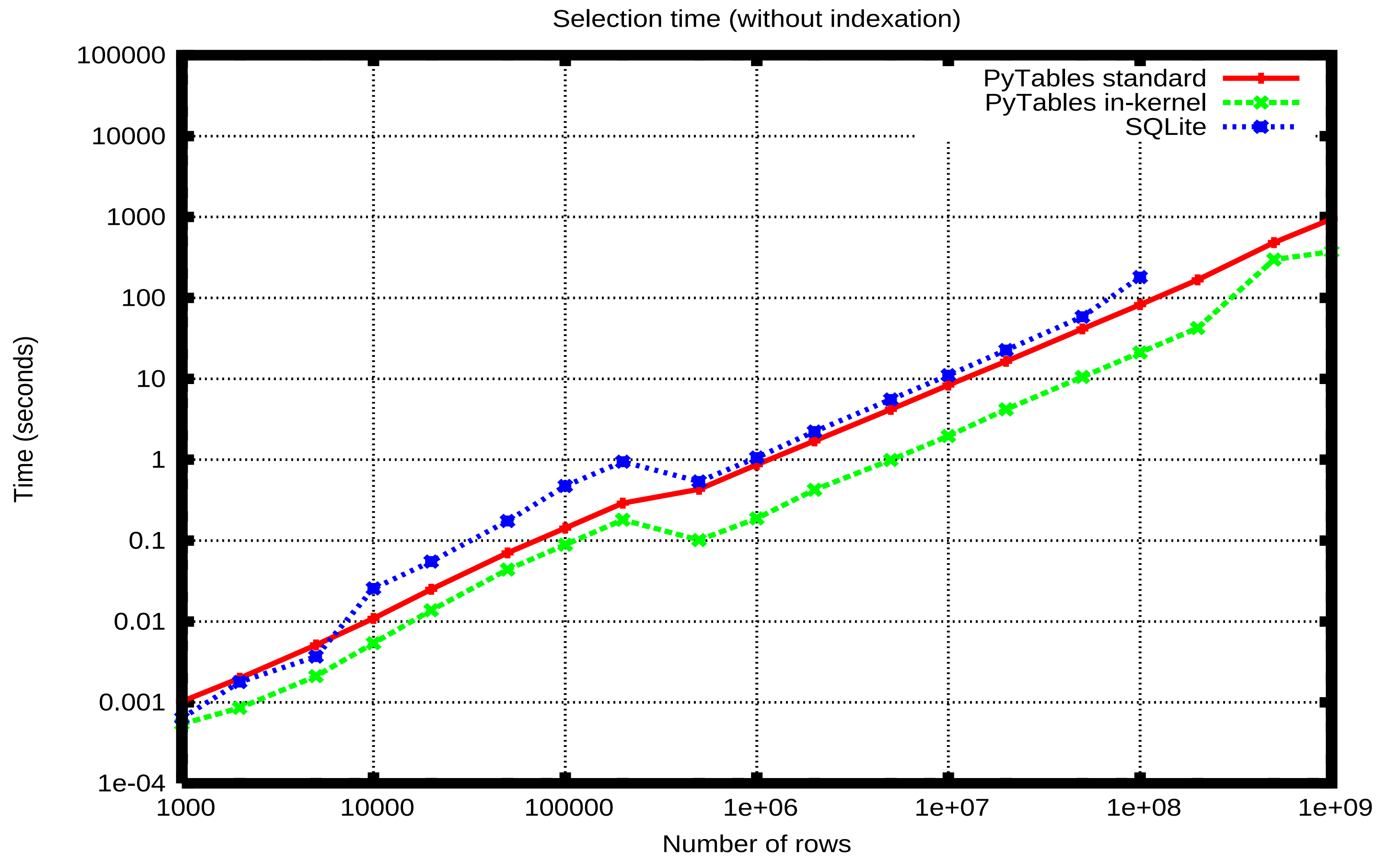
PyTables vs SQLite (time to create index)



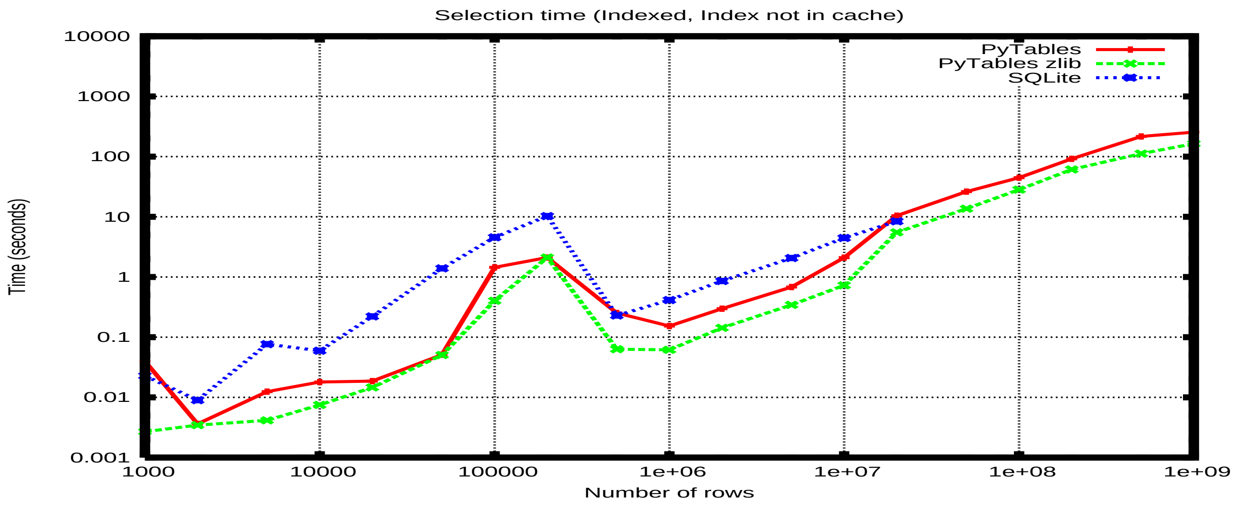
PyTables vs SQLite (disk usage w/index)

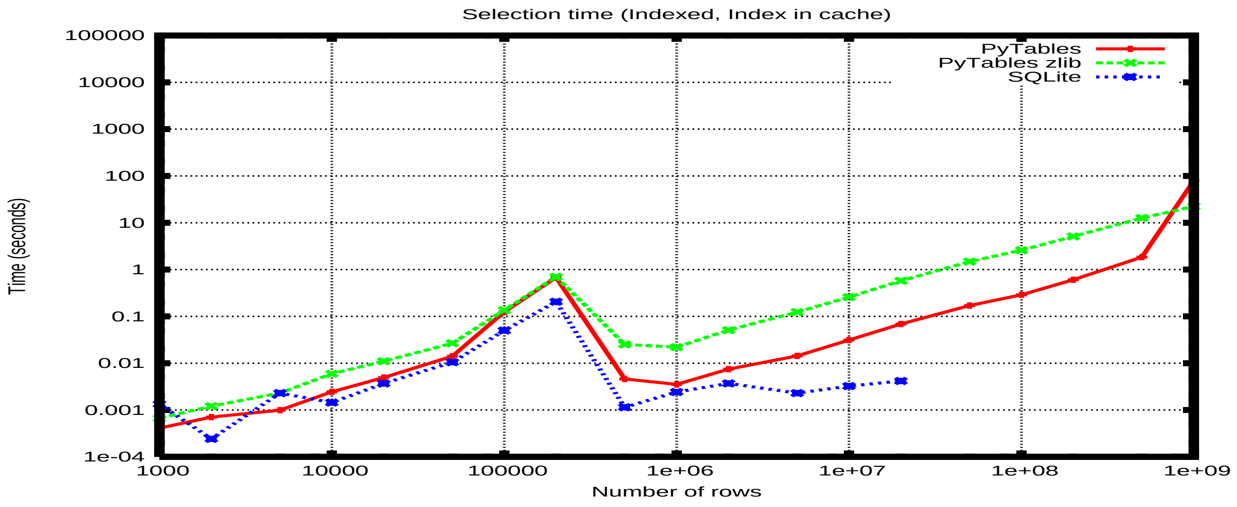


PyTables vs SQLite (non-indexed search)

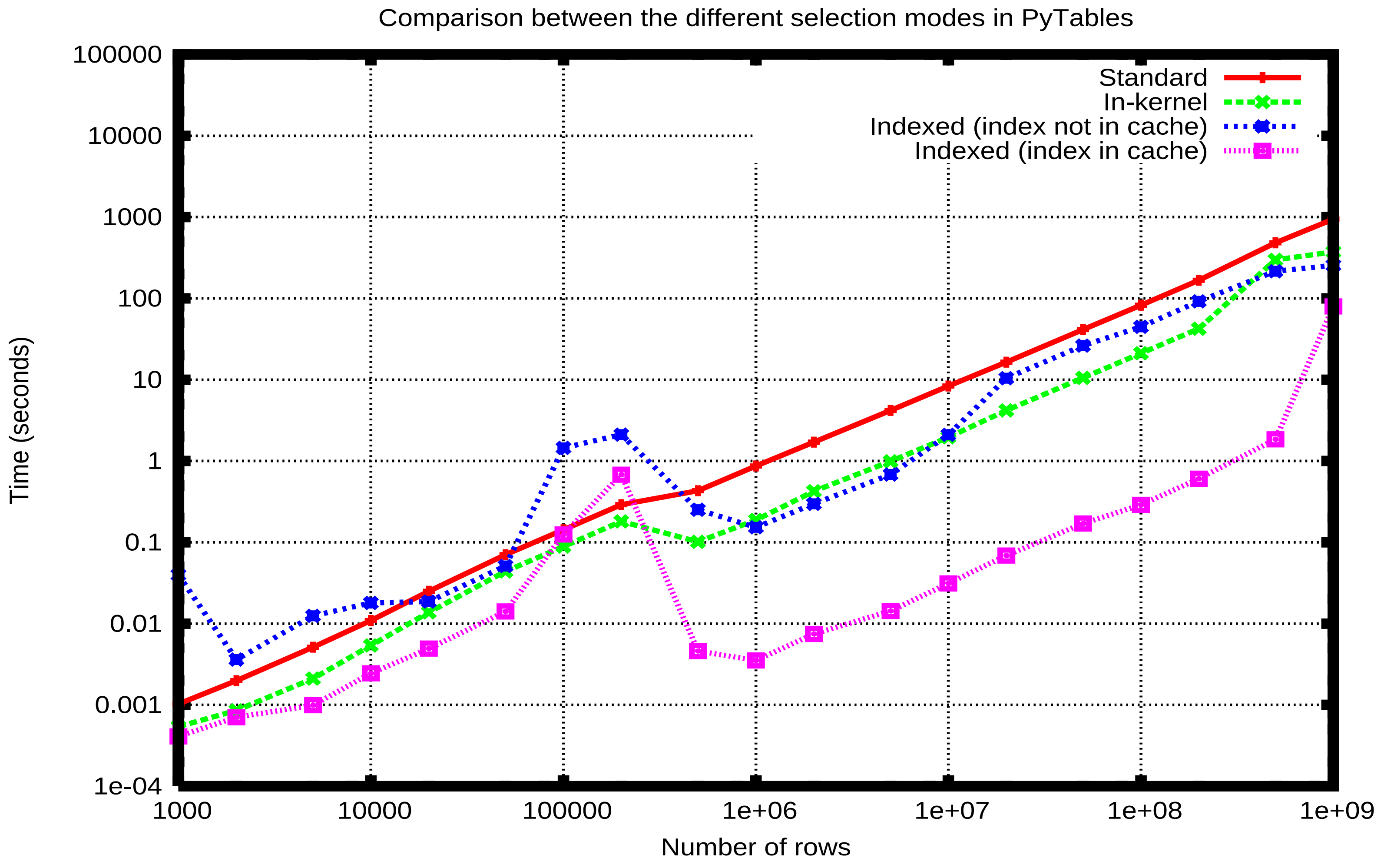


PyTables vs SQLite (indexed search)





PyTables selection modes comparison



Conclusions from benchmark

- Writing
  - PyTables tipically write more than 100 times faster than SQLite
  - SQLite files occupies 3 to 5 times more space than PyTables; if compression is used, these ratio can double
  - SQLite time indexing cost for very large tables is prohibitive; however, PyTables keeps this cost relatively small
- Reading
  - In-kernel selections can be up to 5 times faster than standard and up to 10 times faster than SQLite (for very large tables)
  - Indexed selections can be up to 500 times faster than standard
  - When the index is in-cache, SQLite can be up to 10 times faster than PyTables for moderately large table sizes (but does not scales well)

PyTables indexing can be applied to much larger tables than SQLite

Reliability

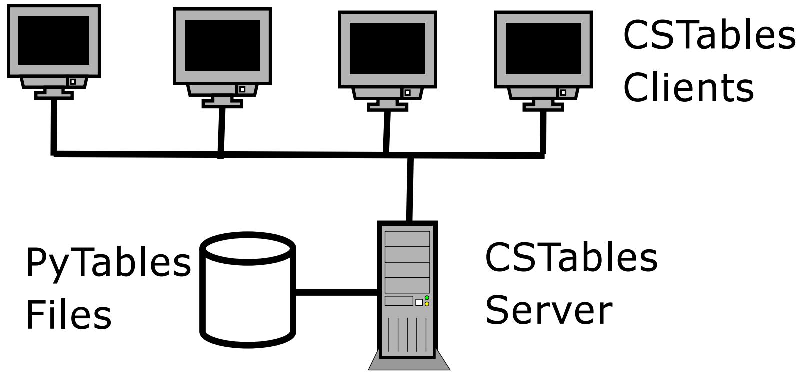
- Developing software suited for production environments is a very important design goal
- Test suite based on UnitTest, the standard Python unit testing framework
- A lot of effort has been put into making the test suite as complete a possible
- More than 2000 tests units (represents more than 12000 lines of pure code) are now incorporated; more will be added and quality will only improve as PyTables evolves

PyTables limitations and plans for future

- Object elements can not be related to other elements (i.e. no references support)
- Table elements cannot be of variable length
- Deleting rows in tables is slow
- Modifying elements in *Array objects is not yet supported
- Optimization of some corner cases

CSTables: PyTables goes Client-Server

- CSTables is the client-server implementation of PyTables
- Provides the possibility of using PyTables remotely and concurrently
- Uses the Twisted (www.twistedmatrix.org) package’s excellent networking capabilities



CSTables usage

- The API is much the same as the PyTables API
  - PyTables API:

```
import tables
fileh=tables.openFile("file.h5")
print fileh.root.table.cols.col1[:]
fileh.close()
```

  - CSTables API:

```
import client, commontables
c=client.TablesClient()
root=c.connect()
gltables=root.getTablesApplicationRoot()
fileh=gltables.openFile("file.h5")
print fileh.root.table.cols.col1[:]
fileh.close()
```

- Some (very few) instructions added to control:
  - Server execution parameters
  - Concurrent mode execution

CSTables client cache

- CSTables caches some of the metadata of the PyTables object tree
- When a client makes a change to the metadata, this change is pushed to the other clients' caches
- Easy to control which attributes should be cached and which shouldn't
- Object is to cache primarily read-only attributes
  - Caveat emptor: Caching attributes that are updated very often generates more traffic than attributes that are not cached at all

CSTables concurrency issues

- CSTables does not provide threading or ayncronous features yet
- So, how it deals with several requests at a time?
- Large data read and write requests are splitted into smaller chunks
- This considerably improves server response time
- In addition, CSTables provides a lock mechanism that allows applications to explicitly put a lock on a node or on an entire subtree
- The locks offer different blocking access modes: READ, WRITE and ALL

CSTables status & availability

- The main features are already implemented and working (95% of PyTables tests already pass)
- Focus now is on checking & debugging possible errors, as well as providing a 100% compatibility with PyTables API
- This will allow to have a client version of ViTables very easily
- A public release is scheduled for the end of this year
- Future directions: Java interface, SOAP support, threading, asynchronous communications

Final remarks

- PyTables & family allows you to process your data interactively and quickly
- Its powerful writing, reading and selection features makes an interpreted language like Python powerful enough to get maximum performance
- The development of ViTables & CSTables has given an unexpected momentum to the PyTables project. Together they form an excellent tool suite to deal with extremely large amounts of data

PyTables is for real work!

- It has been production-ready for over a year. Its API is fairly stable; < 1.0 release numbering only shows that all the desired functionalty is not yet implemented

- It comes with complete documentation both in doc strings format as well as a high quality 90 pages user's manual in PDF and HTML formats.

- Download the latest version (0.9 will be released in September) and use it for free from:

http://pytables.sourceforge.net

Credits

Carabos Crew:

- Vicent Mas (author of ViTables)
- Andreu Alted (author of CSTables)
- Ivan Vilata (for many, many corrections to this talk and User’s Manual)
- Pablo Boronat (for providing general support)

Special Thanks:

- Scott Prater (co-author of the PyTables manual)
- Fernando Pérez (for providing the excellent iPython that is just great to create this presentation graphs)
- Travis Oliphant and Eric Jones (who offered me the opportunity to give this talk)

Thank you!

Questions?

cárabos

SOLUCIONS INFORMÀTIQUES

www.carabos.com