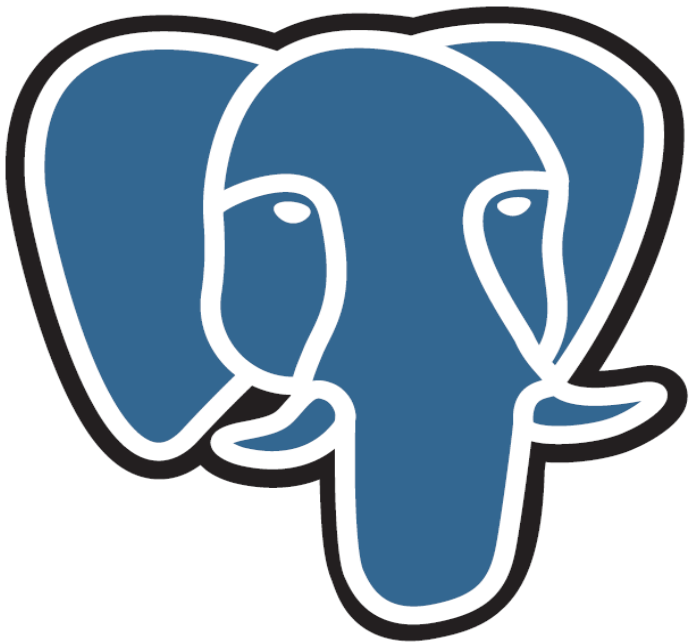
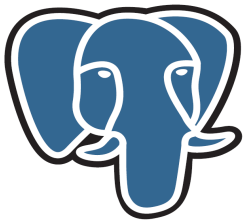


PostgreSQL & MySQL

Analisis FODA



Quienes somos?

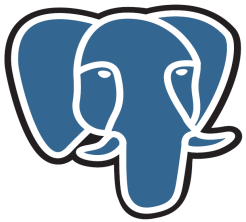
Guido Barosio
- Gerente de Ingeniaría de Sistemas & PostgreSQL Nerd

Emanuel Calvo Franco
- MySQL, PostgreSQL & Oracle DBA

Mariano Reingart
- Desarrollador PostgreSQL & Python

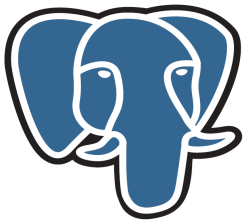
Grupo de Usuarios PostgreSQL Argentina

www.arpug.com.ar



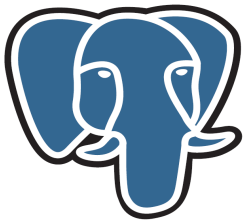
Breve Historia

PostgreSQL	MySQL
Est. 1985 Postgres + SQL 1995	Est.1994
Universidad de California, proyecto Ingres, Michael Stonebreaker (propulsor Bases de datos Objeto Relacionales) PostgreSQL es una comunidad	TcX, evolución de UNIREG, Michael “Monty” Widenius Posteriormente MySQL AB Sun Oracle



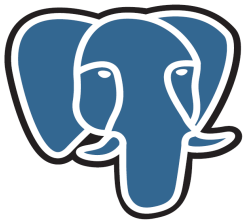
Sobre Licencias

PostgreSQL	MySQL
BSD Libre incluso para usos comerciales	Licencia Dual GPL para software libre Comercial ($$$) MySQL Enterprise dde USD 599 por año por servidor



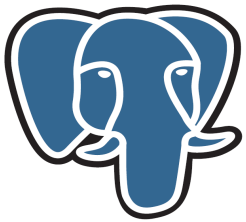
Algunas comparaciones

Cumple ANSI SQL	Cumple algunas partes de ANSI SQL
Sub-selects nativo	Sub-selects a partir de version 4.1 (*)
Transacciones ACID nativo (MVCC)	Transacciones ACID nativo con InnoDB/DBD *
Nivel de Aislamiento: • READ UNCOMMITTED* • READ COMMITTED • REPEATABLE-READ* • SERIALIZABLE	Nivel de Aislamiento: • REPEATABLE-READ • SERIALIZABLE • READ UNCOMMITTED • READ COMMITTED
Procedimientos nativo (PL)	Procedimientos a partir de version 5.1



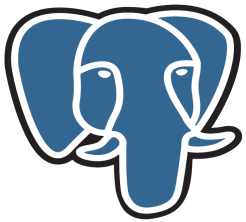
Algunas comparaciones

Tipos de datos Básicos, Avanzados y Objetos	Tipos de datos Básicos
Geométricos, Arrays, (nativo) etc., y del Usuario	Solo Números, Strings y Fechas/Horas
Secuencias (extra transacción)	Campos AutoIncrement (único, indizado, positivo)
Indices BTREE, HASH, GiST, GIN (y del usuario) bitmap, fulltext* parciales y expresiones	Indices BTREE*, HASH, RTREE SPATIAL fulltext (solo MyISAM) prefijos
Pl/PgSql (portable), Pl/Perl, Pl/Python, ...	Lenguaje Procedural SQL propio



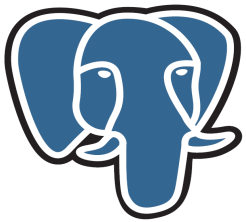
Mas comparaciones

Triggers nativo	Triggers nativo a partir de version 5.1
Full joins y vistas	Full joins y vistas a partir de 5.x (*)
Cursores nativo	Cursores read only
Point-in-Time Recovery WAL (anticipado)	Point-in-Time Recovery Binary Log*
fork (procesos)	Threads (hilos)
Roles	-



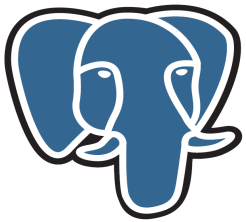
Mas comparaciones

Señales LISTEN y NOTIFY (mensajes/señales)	
Optimizador (avanzado)	Optimizador (básico)
(wtf? postgresql no daña los datos)	REPARE TABLE (reparar tablas ante crashs)
PREPARE (planeamiento)	PREPARE (solo parseo)
RULES (reglas)	-
Operadores y Casts	(no programables)



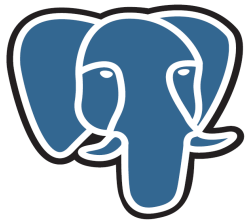
Motores de Almacenamiento

PostgreSQL	MySQL
Postmaster :)	ISAM (obsoleta) MyISAM (default) InnoDB NDB (disable) BDB CSV ARCHIVE MEMORY FEDERATED Maria (v.6 alpha) BerkeleyDB (obsoleta)



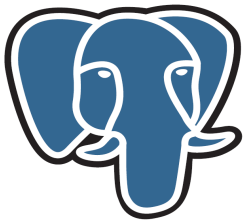
Analisis FODA - Fortalezas

PostgreSQL	MySQL
Integridad PITR - WAL **Concurrencia** multiples usuarios - escritura MVCC **Funcionalidad** tipos (OO), reglas, lenguajes embebidos	**Escalabilidad** replicacion, clustering **Flexibilidad** Multiples Motores de Almacenamiento **Velocidad** sin integridad / concurrencia



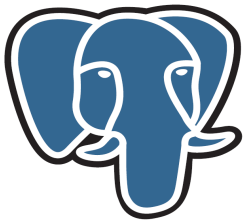
Analisis FODA - Oportunidades

PostgreSQL	MySQL
MySql -> Sun -> Oracle: forks: MariaDB, Drizzle, ... **Estable y Sólido** (MySQL 5.1 != bug free)	**Nuevos Motores de Almacenamiento** (Maria/Falcon?) **Gran aceptación por parte del mercado** (11 millones de instalaciones)



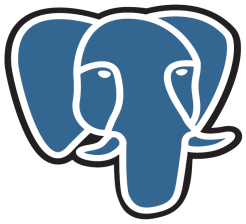
Analisis FODA - Debilidades

PostgreSQL	MySQL
Soporte Comercial Ausencia de replicacion nativa (roadmap 8.5) Ausencia de cluster nativo	(Modelo de Arquitectura) **SQL rudimentario** **Planner Basico** **MyISAM no soporta FKs** (roadmap 5.2)



Analisis FODA - Amenazas

PostgreSQL	MySQL
MariaDb ? **FireBird ?** **Oracle ?**	**Oracle compra InnoDB Oy** **Sun Compra MySQL** **Oracle compra SUN** **...**



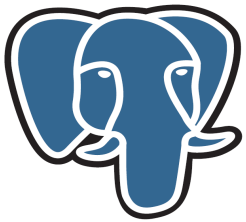
Links relacionados

Why PostgreSQL Instead of MySQL: Comparing Reliability and Speed in 2007

(wiki de PostgreSQL)

http://monty-says.blogspot.com/2008/11/oops-we-did-it-again-mysql-51-released.html

(blogs de uno de los fundadores de MySQL sobre los errores de la versión 5.1)



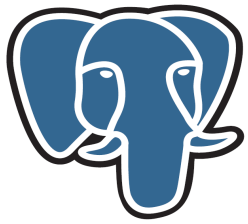
Benchmark

El Mito:

MySQL máz veloz que Postgres

La Realidad:

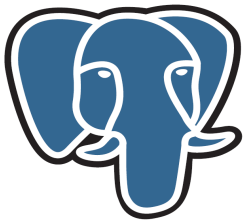
Todo Depende...



Benchmark

Entorno:

- **Hardware: EEEPC 1000HA ("Netbook")**
  - **CPU: Intel Atom N270 1.600 GHz**
  - **Memoria: 1GB, Disco: 160GB 5400 RPM**
- **Software:**
  - **Sistema Operativo: DEBIAN GNU/Linux 5.0 (Lenny - Estable)**
  - **Bases de datos:**
    - **PostgreSQL 8.3.7 (estable en Debian)**
    - **MySQL 5.0.51a-24 (estable en Debian)**
  - **Lenguaje de Programación: Python 2.5.2**
  - **Conectores:**
    - **PsycoPg2 2.0.7**
    - **MySQLdb 1.2.2**



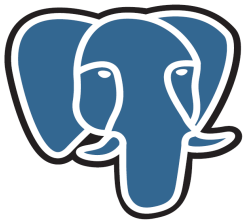
Benchmark

Ejemplo: aplicación simple con varios hilos en Python

Esquema PostgreSQL:
CREATE TABLE prueba (id SERIAL PRIMARY KEY, texto TEXT, flotante FLOAT, entero INTEGER, fecha TIMESTAMP DEFAULT now())

Esqumea MySQL:
CREATE TABLE prueba (id INTEGER AUTO_INCREMENT PRIMARY KEY, texto TEXT, flotante FLOAT, entero INTEGER, fecha TIMESTAMP DEFAULT now()) ENGINE=INNODB;

Tablas simples, con el mismo nivel de funcionalidad

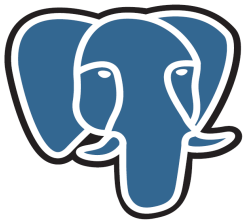


Benchmark

Conexión:

- **myconnect = lambda: MySQLdb.connect(db=" benchmark", user="root", passwd="m" ,host=" localhost")**
- **pgconnect = lambda: psycopg2.connect (database="benchmark", user="postgres", password="m", host="localhost")**

Ambos por TCP/IP

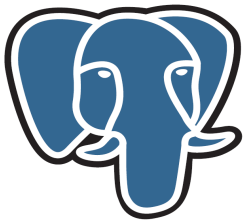


Benchmark

Hilo de medición (benchmark):

```
class BenchmarkSelect(Thread):
    "Hilo para timing de SELECT"
    def run(self):
        cn = self.connect()
        cur = cn.cursor()
        cur.execute("SELECT * FROM prueba")
        for row in cur:
            #print self.nro, row
            pass
        cn.close()
```

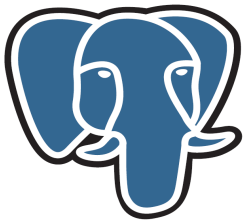
Ambos usan el mismo, interfaz DbApi 2.0 de Python



Benchmark

Inicialización: insertar 10.000 registros

```
    cn = connect()
    cur = cn.cursor()
    cur.execute("DELETE FROM prueba")
    for x in range(10000):
        cur.execute("INSERT INTO prueba (texto,
entero, flotante) VALUES (%s,%s,%s)", ("hola %
s" % x, int(x), float(x)))
```



Benchmark

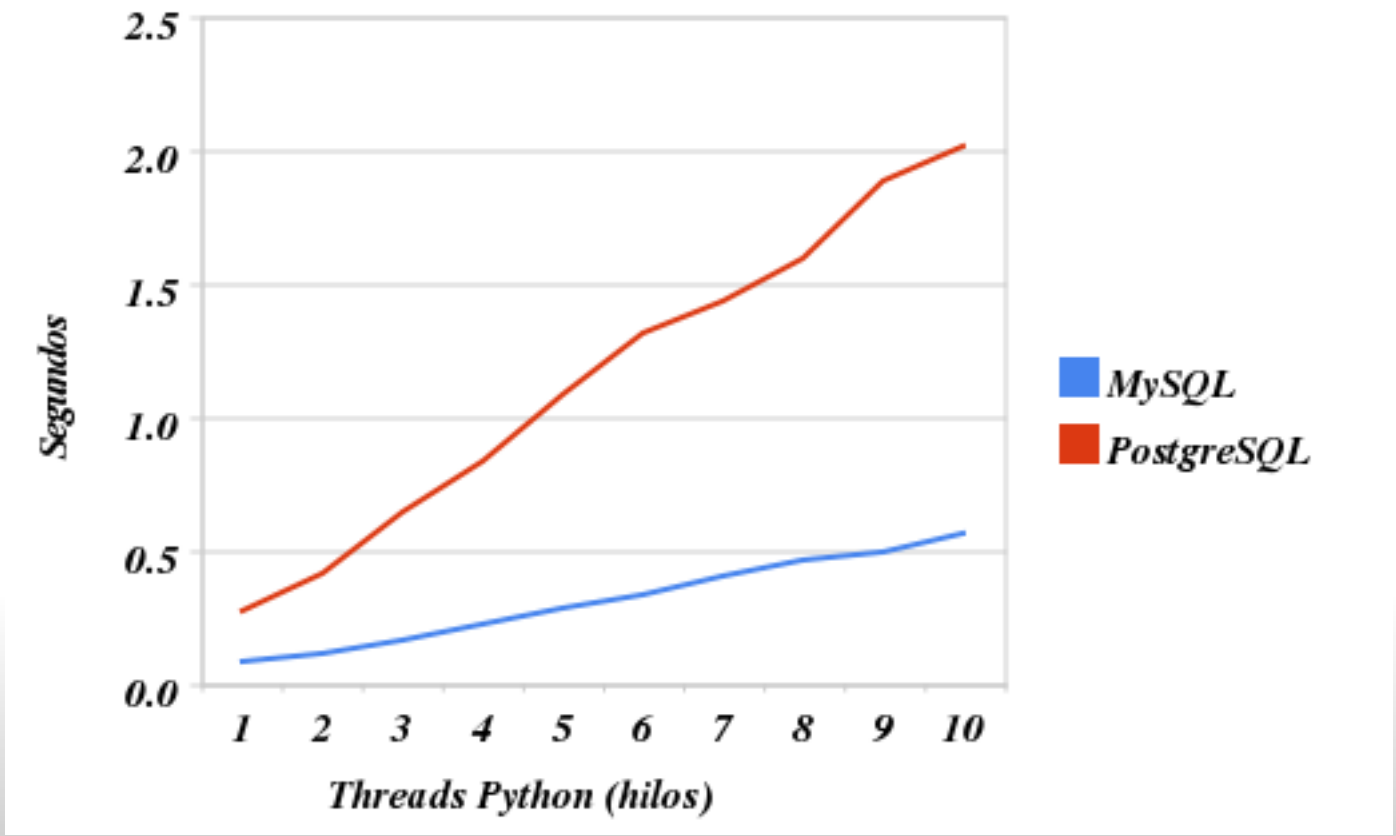
Función de medición: inicar n hilos y esperarlos

```
def bench(cant ,connect, benchmark):
    threads = []
    for i in range(cant):
        thread = benchmark(connect, i)    # creo el
thread
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()      # espero que termine
```

Nota: los threads en python dependen del GIL (su granularidad es por "instrucción python" y no

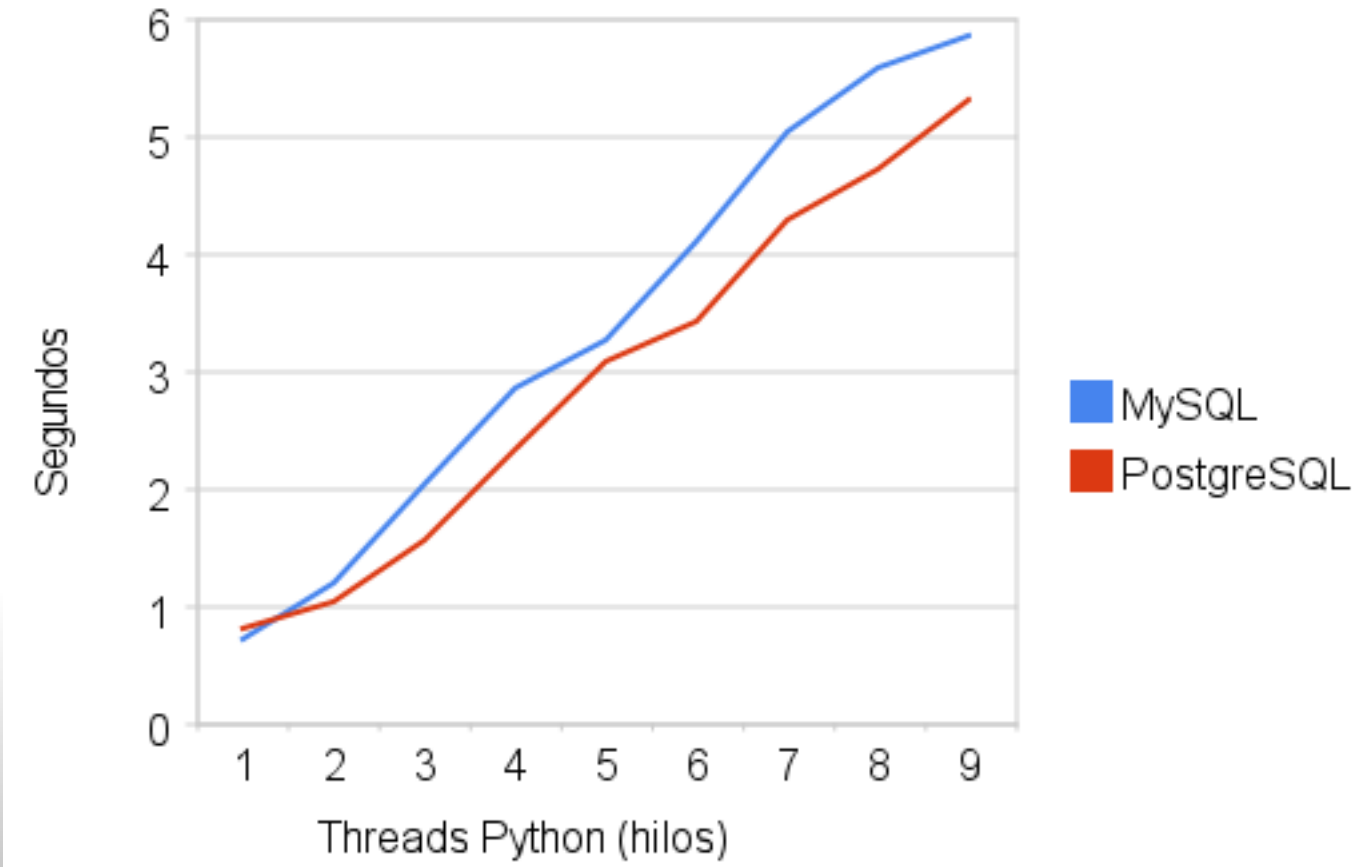
Benchmark

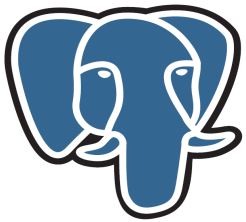
Resultados base de datos "trivial" (1000 registros, 10 hilos, poca carga):



Benchmark

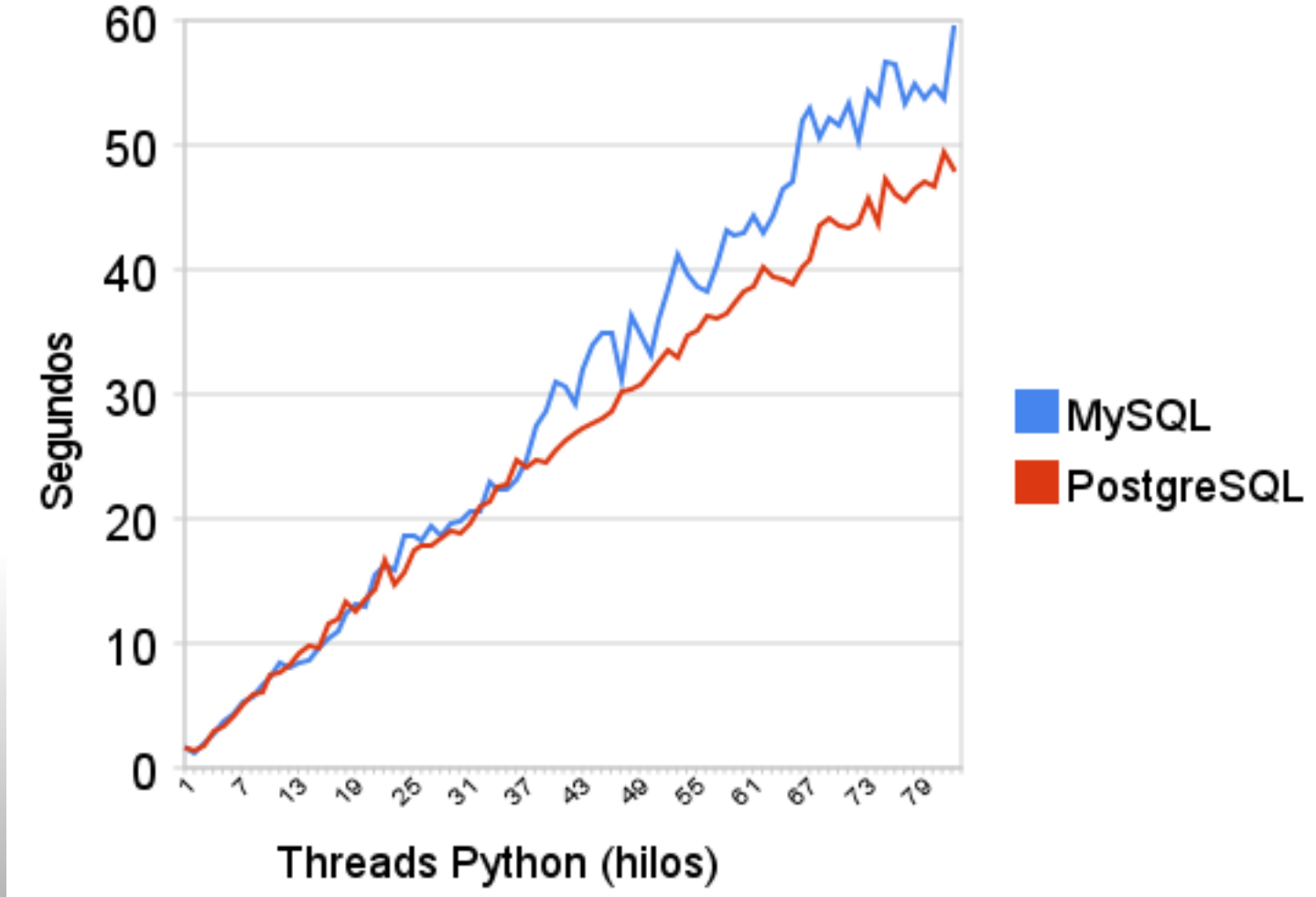
Resultados base de datos menos trivial (10.000 registros):





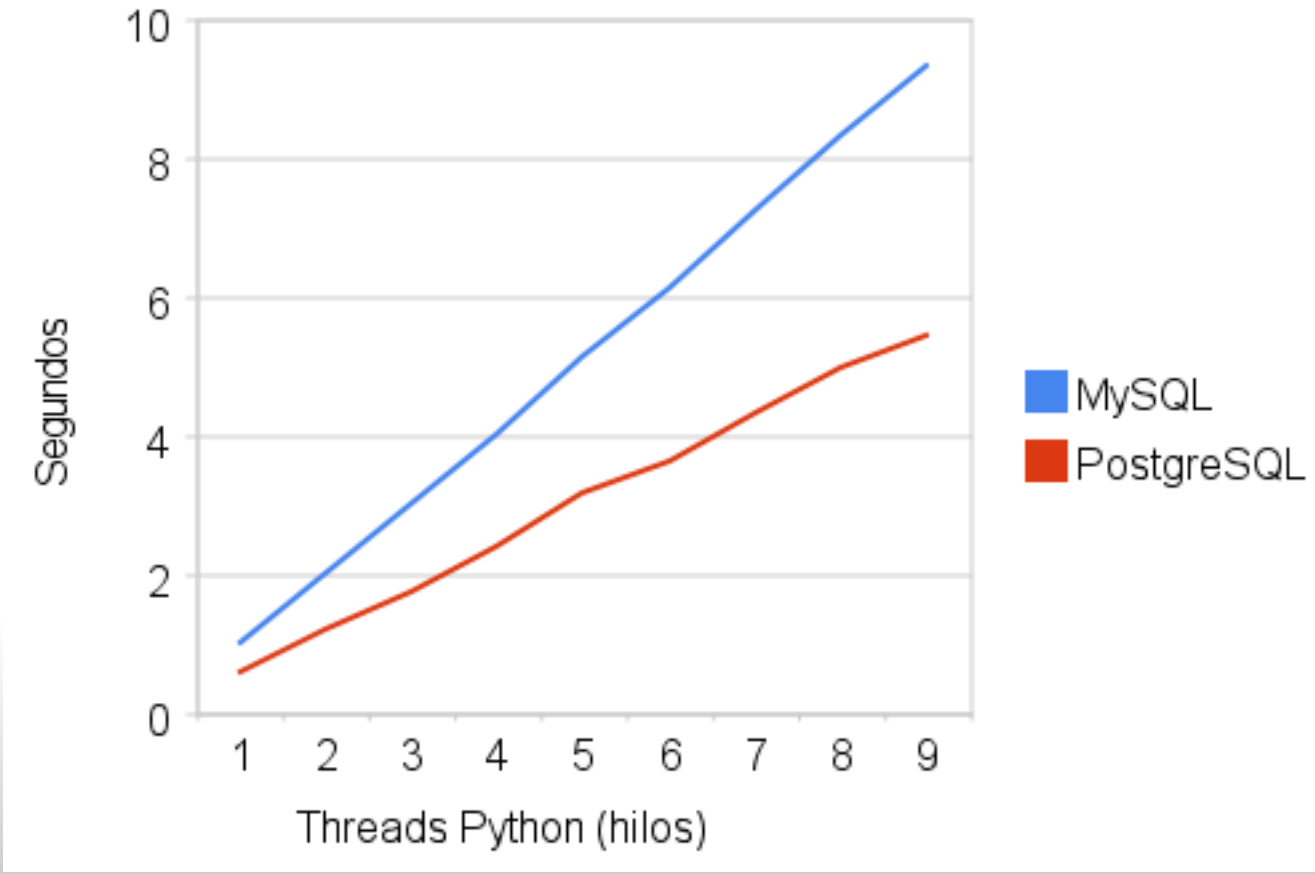
Benchmark

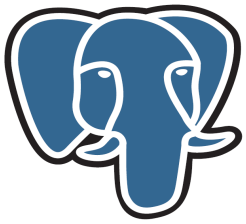
Resultados base de datos menos trivial, con aprox. 100 hilos



Benchmark

Resultados base de datos menos trivial, con aprox. 10 hilos, en WINDOWS:



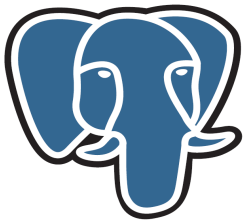


Benchmark

Algunas conclusiones (para esta aplicación):

- PostgreSQL y MySQL pueden tener rendimientos similares para bases de datos "no triviales"
- PostgreSQL puede ser incluso más velóz en general (para bases de datos "no triviales")
- PostgreSQL puede ser significativamente más veloz a medida que la concurrencia se incrementa (nº de hilos o "clientes"), para bases de datos "no triviales"

YMMV



Benchmark

Observaciones:

- Solo analizamos un SELECT trivial, sin joins, indices, funciones, etc. (donde PostgreSQL tendría que estar mejor preparado por su optimizador más avanzado)
- No analizamos concurrencia de INSERT, UPDATE y DELETE (donde PostgreSQL aprovecha MVCC)
- No optimizamos ni PostgreSQL ni MySQL
- No optimizamos el Sistema Operativo

- Y, obviamente no analizamos caracteristicas que son posibles de hacer solo en PostgreSQL :)

Preguntas?

Muchas gracias!

Guido Barosio

Emanuel Calvo Franco

Mariano Reingart