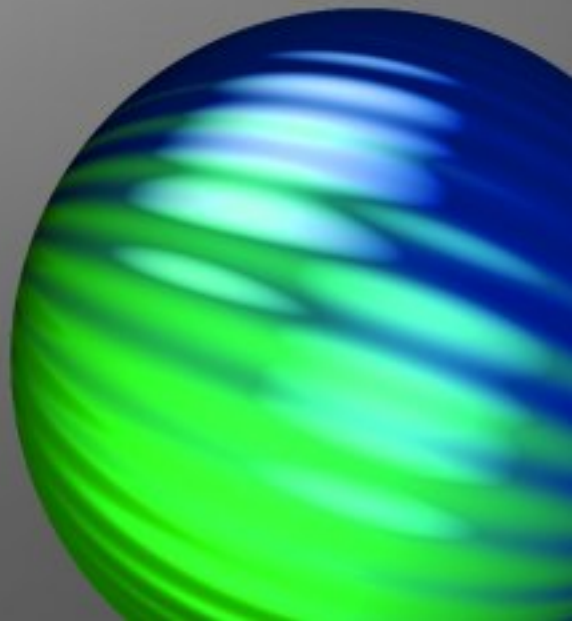
Billion Tables Project (BTP)

Álvaro Hernández Tortosa <aht@Nosys.es>

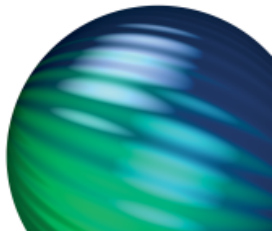
José Luis Tallón <jltallon@Nosys.es>



Who I am

- Álvaro Hernández Tortosa <aht@Nosys.es>
- CTO @ NOSYS
- What we do @NOSYS:
  - Training, consulting and development in PostgreSQL (and Java)
  - EnterpriseDB partners
  - Java training. Javaspeciaslits.eu: Java Master Course
  - AWS partners. Training and architecting in AWS

- Twitter: @ahachete
- LinkedIn: http://es.linkedin.com/in/alvarohernandeztortosa/



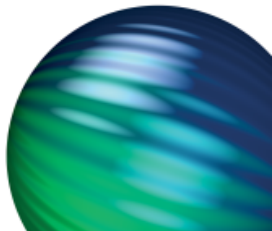
What is a "large" database?

- Single-node databases of up to TBs / dozens TBs. Billions / trillions of records

- Multi-node databases, virtually unlimited. Reportedly hundreds of TBs, PBs

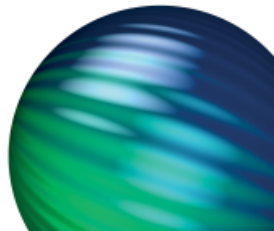
- This talk is not about Big Data. It's just about Big ~~Data~~

- Indeed, we're talking here about Big **Meta**Data (and the world's worst data/metadata ratio ever)

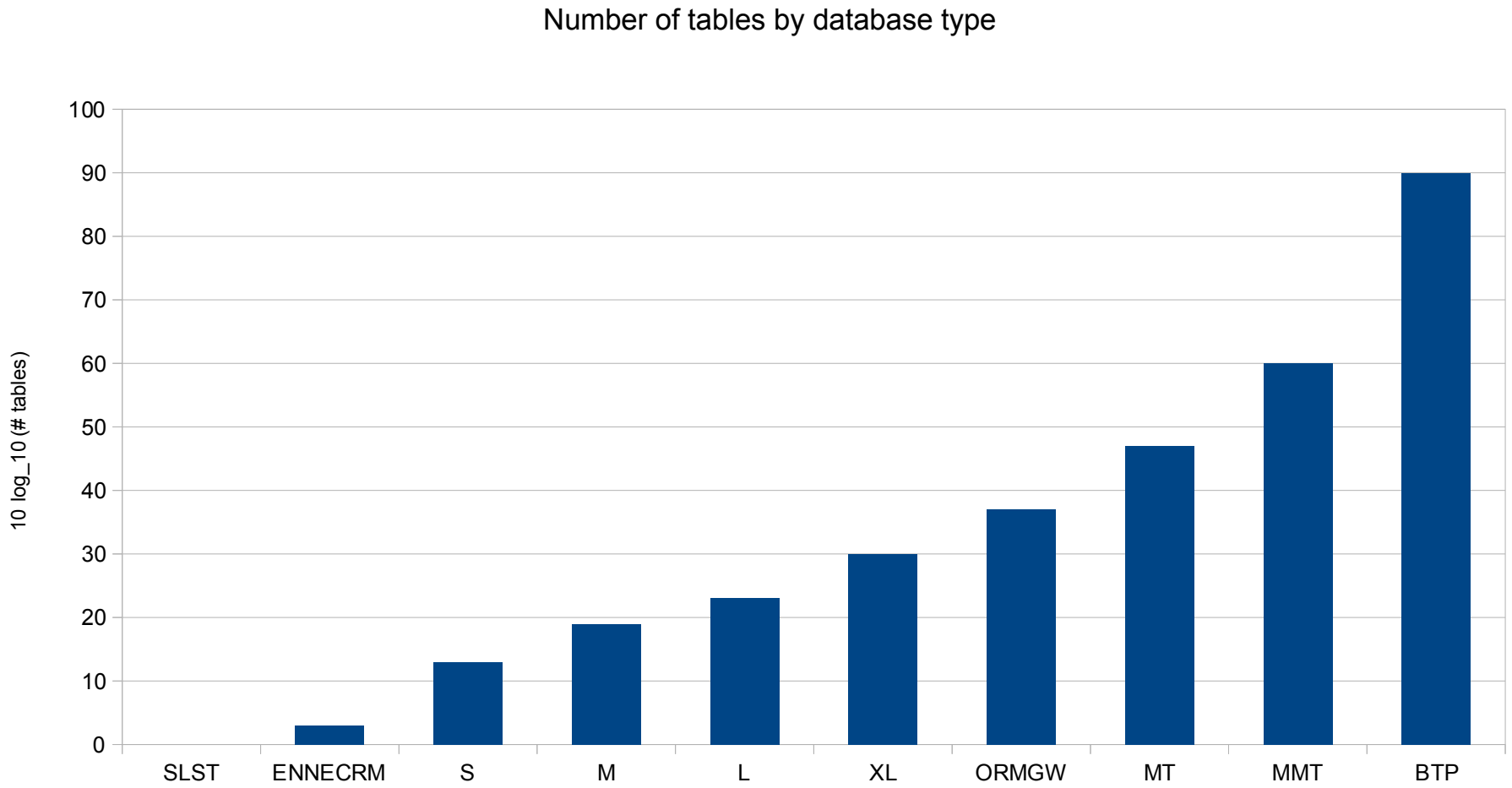


Database "types" (by number of tables)

Database		# Tables
SLST	Schema-Less-Like, Single-Table	1
EDNECRM	Extremely De-Normalized Enterprise CRM	2
S	Small	20
M	Medium	80
L	Large	200
XL	Extra Large	1,000
ORMGW	ORMs Gone Wild	5,000
MT	Multi-Tenancy	50,000
MMT	Massive Multi-Tenancy	1,000,000
BTP	Billion Tables Project	1,000,000,000

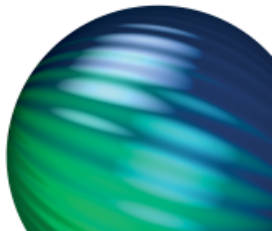


Database “types” (II)



Theoretical PostgreSQL limits

Feature	Limit
# attributes / table	250-1600 (depending on attribute types)
Max size / attribute	1GB
Max size / row	1.6 TB
Max # rows / table	unlimited
Max size / table	32 TB
Max # tables / database	unlimited
Max size / database	unlimited



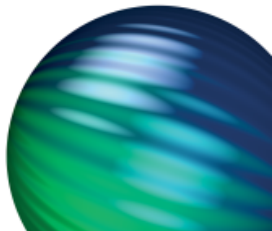
Where it all started...

- 2002, mail to `pgsql-admin@postgresql.org`:

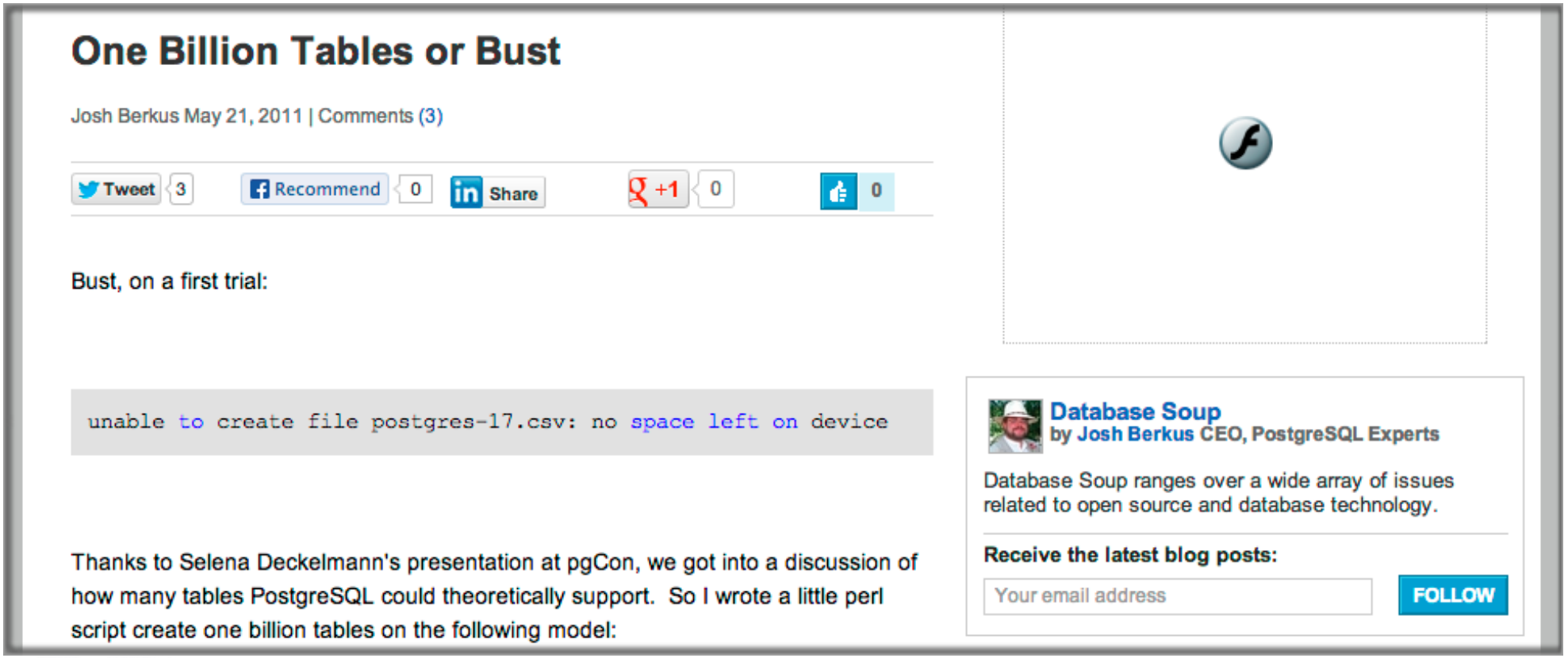
"*I'm guessing that the maximum number of tables is related to how much can be stored in the pg_ tables […]. So, based on that, the maximum number of rows is unlimited and the maximum size for a table is 64 TB. So realistically, you would need an enormous number (trillions) of tables to exceed that limit*"

Simon Cawley

http://www.postgresql.org/message-id/53386E0C47E7D41194BB0002B325C997747F2B@NTEX60



Where it all started... (II)



One Billion Tables or Bust

Josh Berkus May 21, 2011 | Comments (3)

Tweet 3 Recommend 0 Share +1 0 0

Bust, on a first trial:

```
unable to create file postgres-17.csv: no space left on device
```

Thanks to Selena Deckelmann's presentation at pgCon, we got into a discussion of how many tables PostgreSQL could theoretically support. So I wrote a little perl script create one billion tables on the following model:

Database Soup
by Josh Berkus CEO, PostgreSQL Experts

Database Soup ranges over a wide array of issues related to open source and database technology.

Receive the latest blog posts:

Your email address FOLLOW

http://it.toolbox.com/blogs/database-soup/one-billion-tables-or-bust-46270

May 21th, 2011

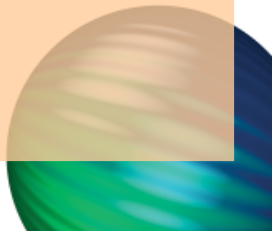
So... why do it?

- To prove PostgreSQL has no limits on the # of tables
- To stress PostgreSQL in an unusual way
- To test a new server before going to production

Official reasons

- To beat Josh Berkus, creating tables faster than him ;)
- "Mine is bigger than yours" (database)
- Because **we can**

In reality...



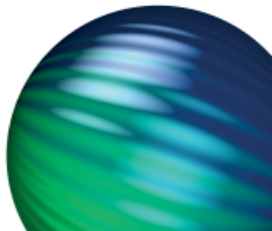
Re-defining "tps"

Wikipedia (http://en.wikipedia.org/wiki/Transactions_per_second):

"*Transactions Per Second refers to the number of atomic actions performed by certain entity per second*"

From now on, for this presentation, it simply is:

"*tables per second*"



First attempts (2011)

- Josh Berkus

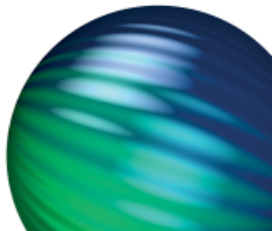
(http://it.toolbox.com/blogs/database-soup/one-billion-tables-or-bust-46270):
3M tables, 83 tps. Server crashed (out of disk). Serial + text

- Jan Urbanski

(http://it.toolbox.com/blogs/database-soup/one-billion-tables-part-2-46349):
4.6M tables, 1K tps. Server crashed (inodes). Int + text

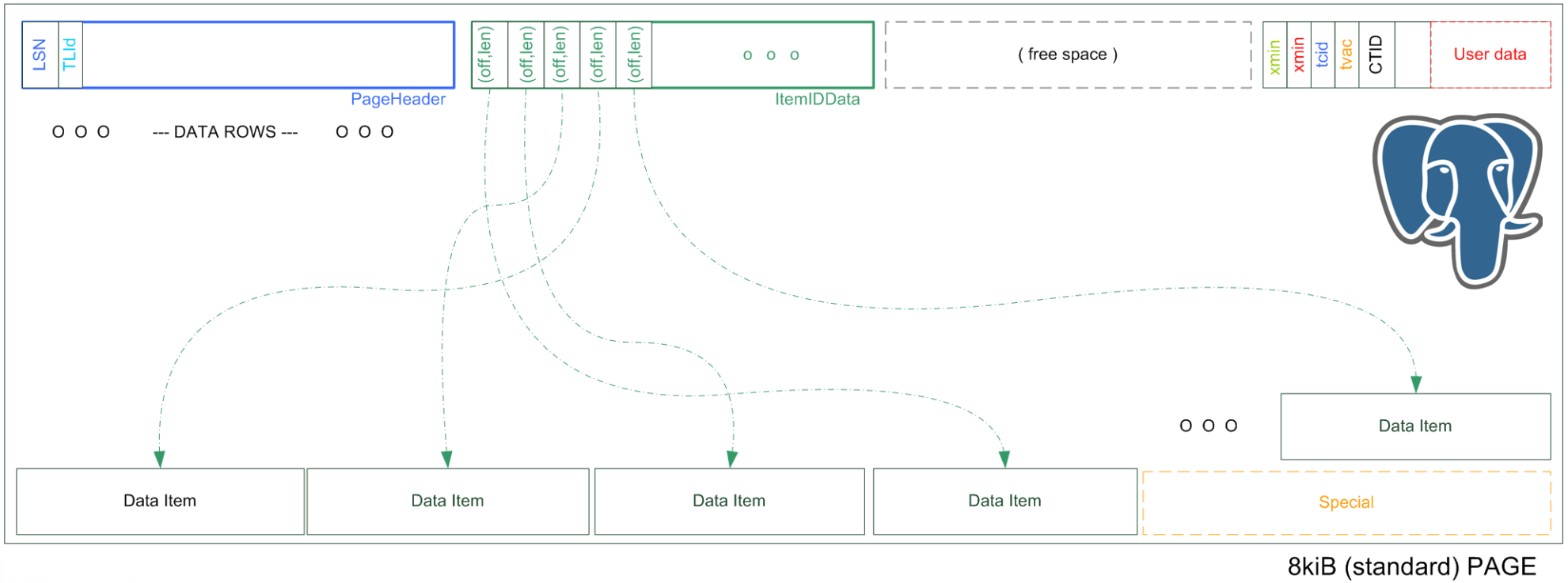
- $SELF

(http://it.toolbox.com/blogs/database-soup/one-billion-tables-part-2-46349):
10M tables, 2K2 tps. Stopped. Single int column
100M tables, 1K5 tps. Stopped. Single int column



First problems: running out of storage

- `pg_class` storage



- Filesystem storage

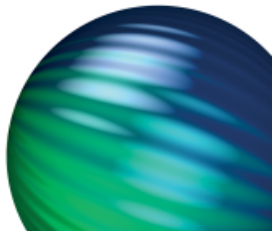
100M tables. How to get there?

- We need RAM:

```
Out of memory: kill process 4143 (postgres) score
235387 or a child
Killed process 4146 (postgres)
```

- Use a FS capable of handling a large # of files: `reiserfs`

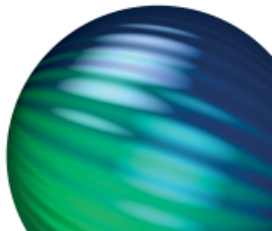
- Table creation strategy:
  - Don't use a pre-created CSV or .sql file
  - Don't use a driver over TCP/IP
  - Best solution: feed SQL commands via `stdin` with psql over unix domain sockets



100M tables. How to get there? (II)

Tune `postgresql.conf`:

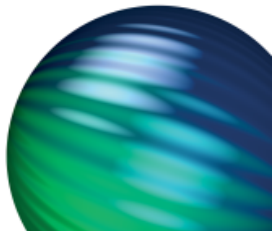
```
fsync = off
synchronous_commit = off
full_page_writes = off
wal_buffers = 256MB
autovacuum = off
max_locks_per_transaction = 10000
shared_buffers = 16384MB
checkpoint_segments = 128
```



100M tables. How to get there? (III)

Server setup:

- Intel Core 2 CPU
- 4GB RAM
- 3X 1TB SATA 7K2 rpm, RAID 0
- Reiserfs
- Ubuntu 10.04
- PostgreSQL 9.0

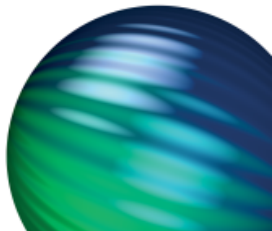


100M tables. The script

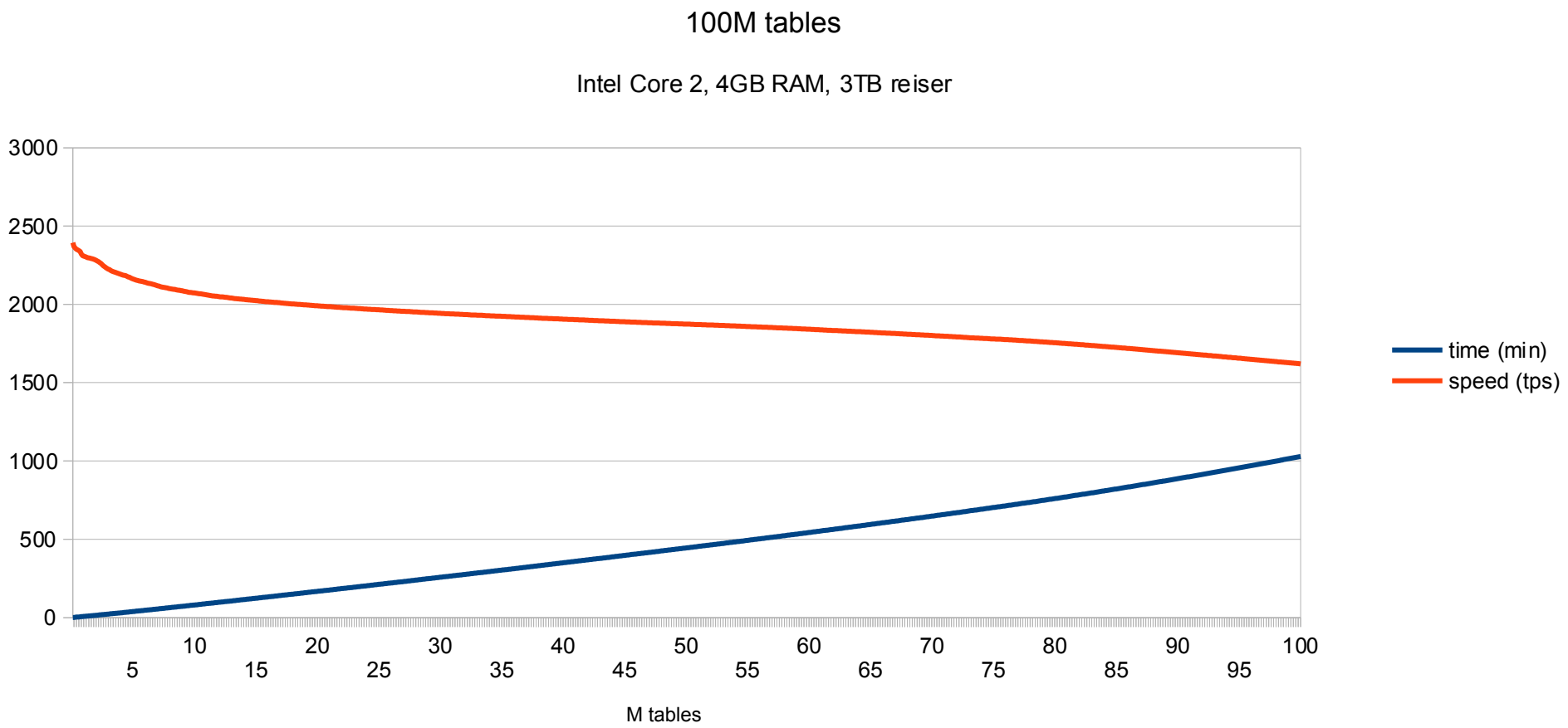
```
def iteration(table_nr, n_tables):
        psql = subprocess.Popen(shlex.split(PSQL_COMMAND), stdin=subprocess.PIPE, stdout=subprocess.PIPE)
        start = time.time()
        for i in range(table_nr, table_nr + n_tables):
                psql.stdin.write('CREATE TABLE tab_%09d (i integer);' % i)
        psql.stdin.write('CHECKPOINT;\n')
        psql.stdin.flush()
        psql.stdin.close()
        psql.wait()
        return time.time() - start

def main():
        n_tables_iteration = N_TABLES / N_ITERATIONS
        next_table_nr = 0
        logger = log(LOGFILE)
        start = time.time()
        for i in range(0, N_ITERATIONS):
                disk_usage = get_disk_usage()
                duration = iteration(next_table_nr, n_tables_iteration)
                next_table_nr = next_table_nr + n_tables_iteration
                logger.log(next_table_nr, duration, disk_usage, get_disk_usage(), get_mem_stats())
        logger.close()
        print 'Total ellapsed time: %f seconds\n' % (time.time() - start)

if __name__ == '__main__':
        main()
```



100M tables. The results



Disk usage: 257GB

The road to 1B tables. Your worst enemies

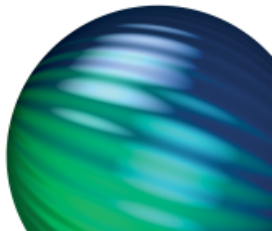
- Autovacuum

(but wasn't it `autovacuum = off` ?)

```
autovacuum_freeze_max_age = 2000000000
# maximum XID age before forced vacuum
```

- `updatedb`

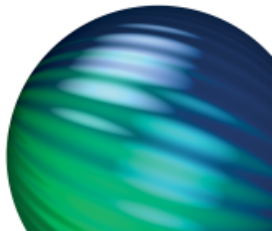
(who the hell enables it by default???????)



The road to 1B tables. Storage

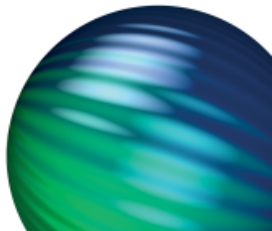
- Separate base from tables dir
- Create a tablespace (or more –see later) in a reiserfs partition (we named it "`/data`")
- Best performance achieved with base on xfs ("`/bigdata`")

  Large appends, works as a "normal" database
- WAL records on RAM (tmpfs with swap to avoid overruns, "`/xlog`")

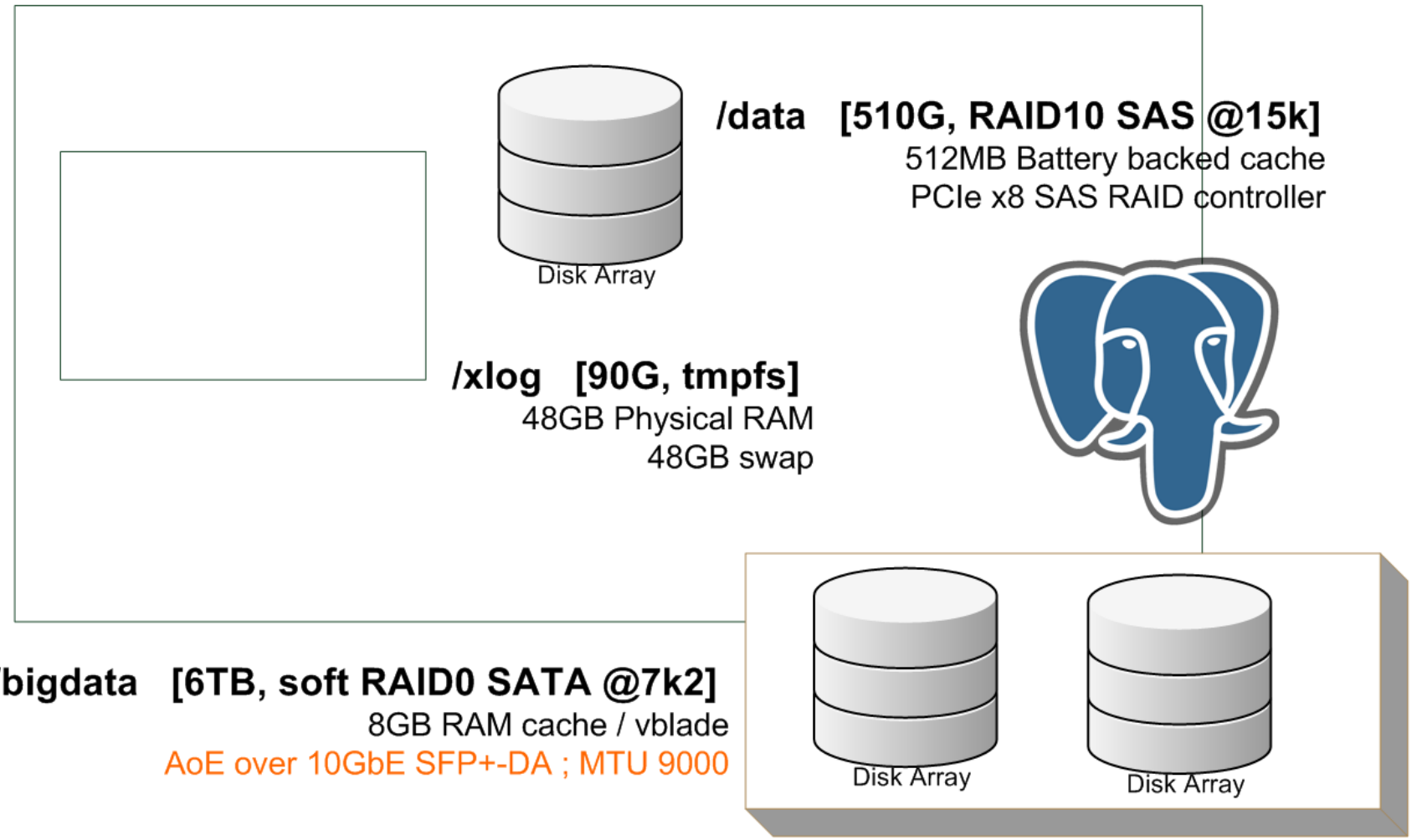


The road to 1B tables. A larger pizza

- 2X Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz (16 cores, 32 threads)
- 48GB RAM
- Modern SO and postgres:
  - Debian wheezy (kernel 3.2.41)
  - PostgreSQL 9.2.4
- Just 6 seconds to "`make -j16`" postgresql src

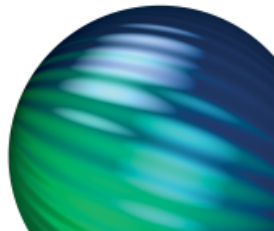


The road to 1B tables. Storage (II)



The road to 1B tables. Tablespaces

- Except for reiserfs, any fs degrades very fast with # files
- Even reiserfs degrades after several millions
- Solution: create as many tablespaces as desired (even in the same, reiserfs fs)
- For the 1B run, we used 1000 tablespaces for optimal performance



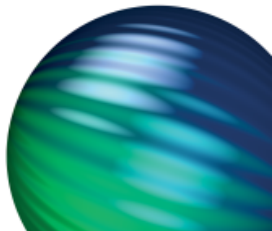
The road to 1B tables. Concurrency

- Table creation is not disk-limited: avg disk throughtput was < 5MB/s on the 100M tables test

- There are two main limits:
  - CPU speed (backends rise to 100% if run alone)
  - Contention

- To improve performance, we launched several processes in background

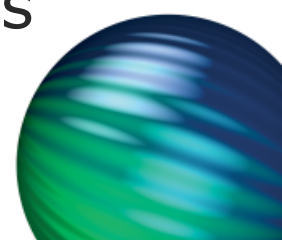
- 16 processes proved to be the sweet spot



The road to 1B tables. Concurrency (II)

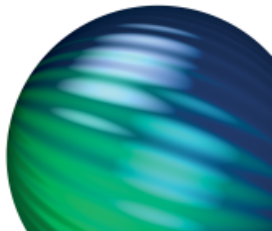
- With multiple processes, we cannot have each process log its own set of log data (really difficult to merge, no status/progress snapshot)

- We run another process to log the data:
  - The logger process has the PID of every worker
  - When the logger wants to log data, sends SIGUSR1 to workers
  - The logger waits for input in a fifo identified by worker PID
  - The worker writes the actual number of tables and whether it already finished

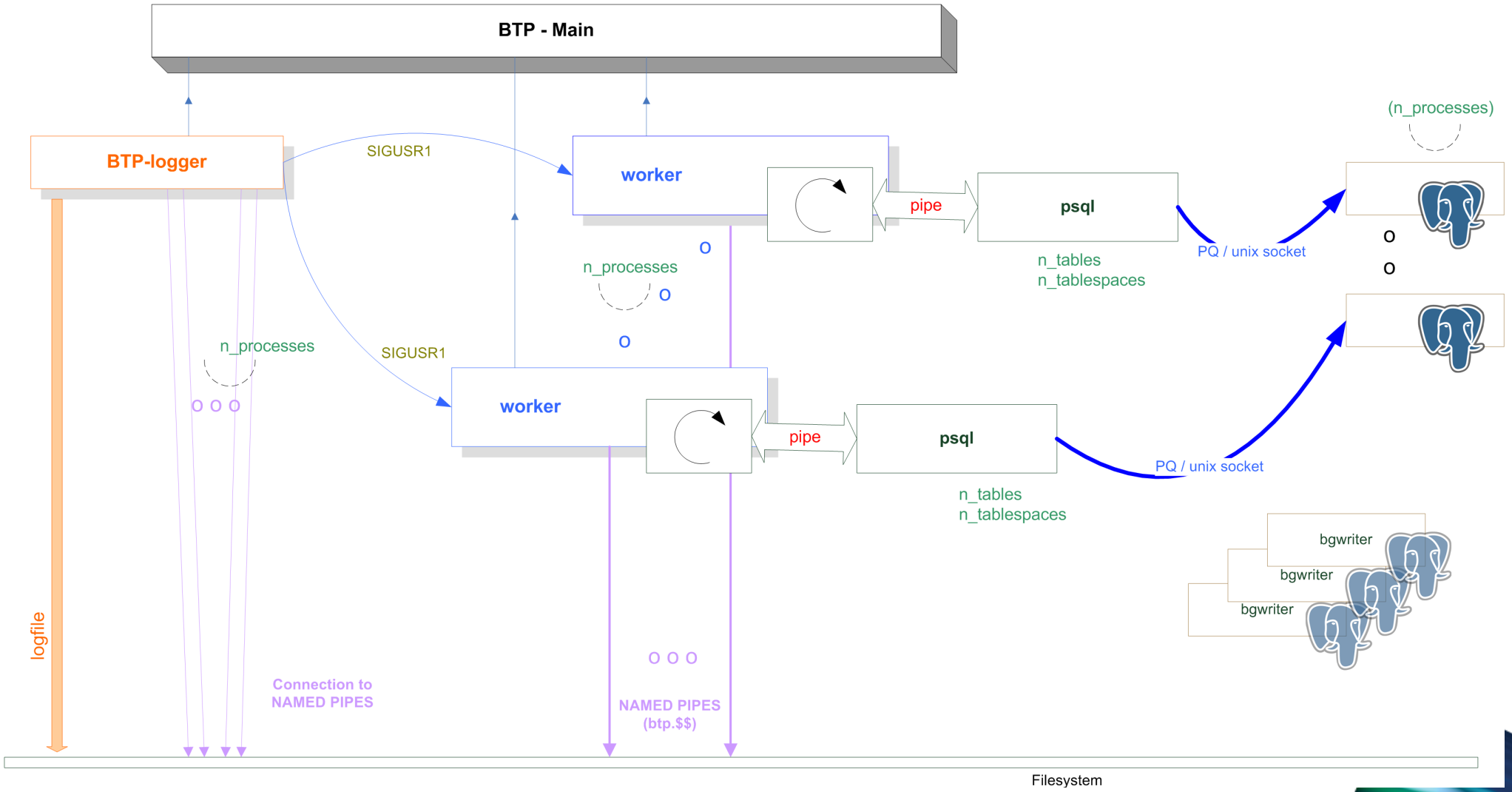


The road to 1B tables. The source code

- Worker is a python script:
  - Divides the number of tables (assigned to the worker) in iterations
  - For each iteration, spawns a psql and feeds `CREATE TABLE … TABLESPACE …` statements via stdin
  - When signaled USR1, writes # tables to fifo
  - Exits when signaled TERM (by logger process)
  - Iterations run in its own thread
- Logger is a shell script. When signaled USR1, logs data
- Main is a shell script. Launches all processes and signals logger when to log (every 10s)



The road to 1B tables. The source code (II)



btp-main.sh

```
function usage() {
        echo $1 >&2
        echo "Usage: $0 <n_tables> <n_processes> <n_tables_per_psql> <n_tablespaces>"
        exit 1
}

[ $# -eq 4 ] || usage "Wrong number of arguments"

n_tables=$1
n_processes=$2
n_tables_per_psql=$3
n_tablespaces=$4
[ $n_tablespaces -ge $n_processes ] || usage "The number of tablespaces should be equal or greater than the number of proces
ses"
[ $(( $n_tables % ($n_tables_per_psql * $n_processes) )) -eq 0 ] || usage "The number of tables must be a multiple of n_tabl
es_per_psql * n_processes"

tables_per_process=$(( $n_tables / $n_processes ))
table_offset=0
procs=""
for i in `seq 1 $n_processes`
do
        ./btp-process.py $i $table_offset $tables_per_process $n_tables_per_psql $n_tablespaces &
        procs="$procs $!"
        table_offset=$(( $table_offset + $tables_per_process ))
done

./btp-logger.sh $procs &
logger_proc=$!

while true
do
        sleep 10s
        kill -USR1 $logger_proc 2> /dev/null
        [ 1 -eq $? ] && break
done
```

btp-process.py

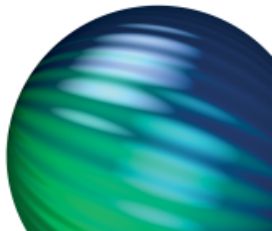
```
def sigUSR1handler(signum, frame):
        global fifo
        fd = open(fifo, 'w')
        global created_tables
        global finished
        fd.write('%s %d\n' % (finished, created_tables))
        fd.close()
def sigTERMhandler(signum, frame):
        global exit
        exit = True
def iteration(table_nr, n_tables, tablespace):
        psql = subprocess.Popen(shlex.split(PSQL_COMMAND), stdin=subprocess.PIPE, stdout=subprocess.PIPE)
        for i in range(table_nr, table_nr + n_tables):
                psql.stdin.write('CREATE TABLE _%08x (%s) TABLESPACE %s;' % (i, TABLE_ATTRIBUTES_DEF, tablespace))
        psql.stdin.close()
        psql.wait()
        global created_tables
        created_tables += n_tables
def subprocess_thread(n_tables, table_start, n_tablespaces, n_tables_per_psql, process_n):
        n_iterations = n_tables / n_tables_per_psql
        next_table_nr = table_start
        for i in range(0, n_iterations):
                iteration(next_table_nr, n_tables_per_psql, '_%03x' % ((i + process_n) % n_tablespaces))
                next_table_nr = next_table_nr + n_tables / n_iterations
        global finished
        finished = True
def main():
        # args validation. Removed for screenshot

        global fifo
        fifo = '%s-%d' % (FIFO_PATH_PREFIX, os.getpid())
        os.mkfifo(fifo)
        signal.signal(signal.SIGUSR1, sigUSR1handler)
        signal.signal(signal.SIGTERM, sigTERMhandler)
        try:
                thread.start_new_thread(subprocess_thread, (n_tables, table_start, n_tablespaces, n_tables_per_psql, process
_n))
        except:
                pass
```

btp-logger.sh

```
n_pids=`echo $* |awk 'END {print NF}'`          # aka $# (but works even with shift)

declare -A workers
function count_tables {
        total=0
        for pid in $pids
        do
                kill -USR1 $pid
                fifo="${FIFO_BASE_PATH}-${pid}"
                read finished n_tables < $fifo
                if [ "True" = "$finished" ]
                then
                        workers[$pid]=42
                fi
                total=$(( $total + $n_tables ))
        done
        eval "$1=$total"
}
function handleUSR1 {
        log
}
function log {
        # removed for screenshot
}
trap handleUSR1 SIGUSR1
while true
do
        sleep 2s
        if [ $n_pids -eq ${#workers[*]} ]
        then
                log
                sleep 5s
                for pid in $pids
                do
                        kill -TERM $pid
                done
                exit 0
        fi
done
```



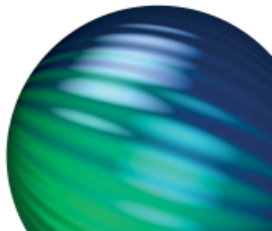
1B tables. So, did it work?

```
$ time ./btp-main.sh 1000000000 16 50000 1000
real    2022m19.961s
user    240m7.044s
sys     165m25.336s
```

(aka 33h 42m 20s)

- Avg: 8242tps

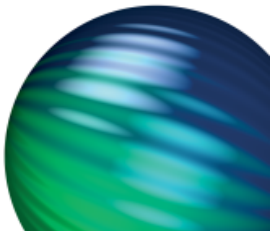
```
btp=# SELECT txid_current();
 txid_current
--------------
   1000001685
```



1B tables. So, did it work? (II)

```
$ echo -e '\\timing on\nSELECT count(*) FROM
pg_class' |psql btp
   count
------------
 1000000288
Time: 9221642.102 ms

$ df -h /data /bigdata /var/tmp
Filesystem                Size  Used Avail Use% Mounted on
/dev/mapper/vgMain-data   500G   97G  404G  20% /data
/dev/etherd/e15.0         5.5T  2.6T  3.0T  46% /bigdata
tmpfs                      90G  4.1G   86G   5% /var/tmp
```

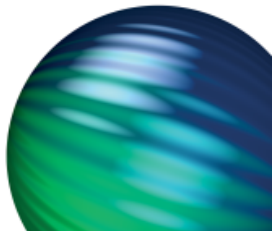


1B tables. So, did it work? (III)

```
btp=# SELECT relname, heap_blks_read, heap_blks_hit,
idx_blks_read, idx_blks_hit FROM pg_statio_all_tables WHERE
relname IN ('pg_tablespace', 'pg_database', 'pg_shdepend');
```

relname	heap_blks_read	heap_blks_hit	idx_blks_read	idx_blks_hit
pg_tablespace	35	6226009368	13	6794
pg_database	3	63015	12	105017
pg_shdepend	1	1000001001	5	1001537778

```
btp=# INSERT INTO _3ade68b1 VALUES (2), (3);
Time: 20.673 ms
btp=# SELECT * FROM _3ade68b1 LIMIT 1;
[...]
Time: 0.207 ms
```

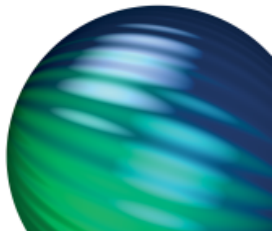


1B tables. How long does a "\dt" take?

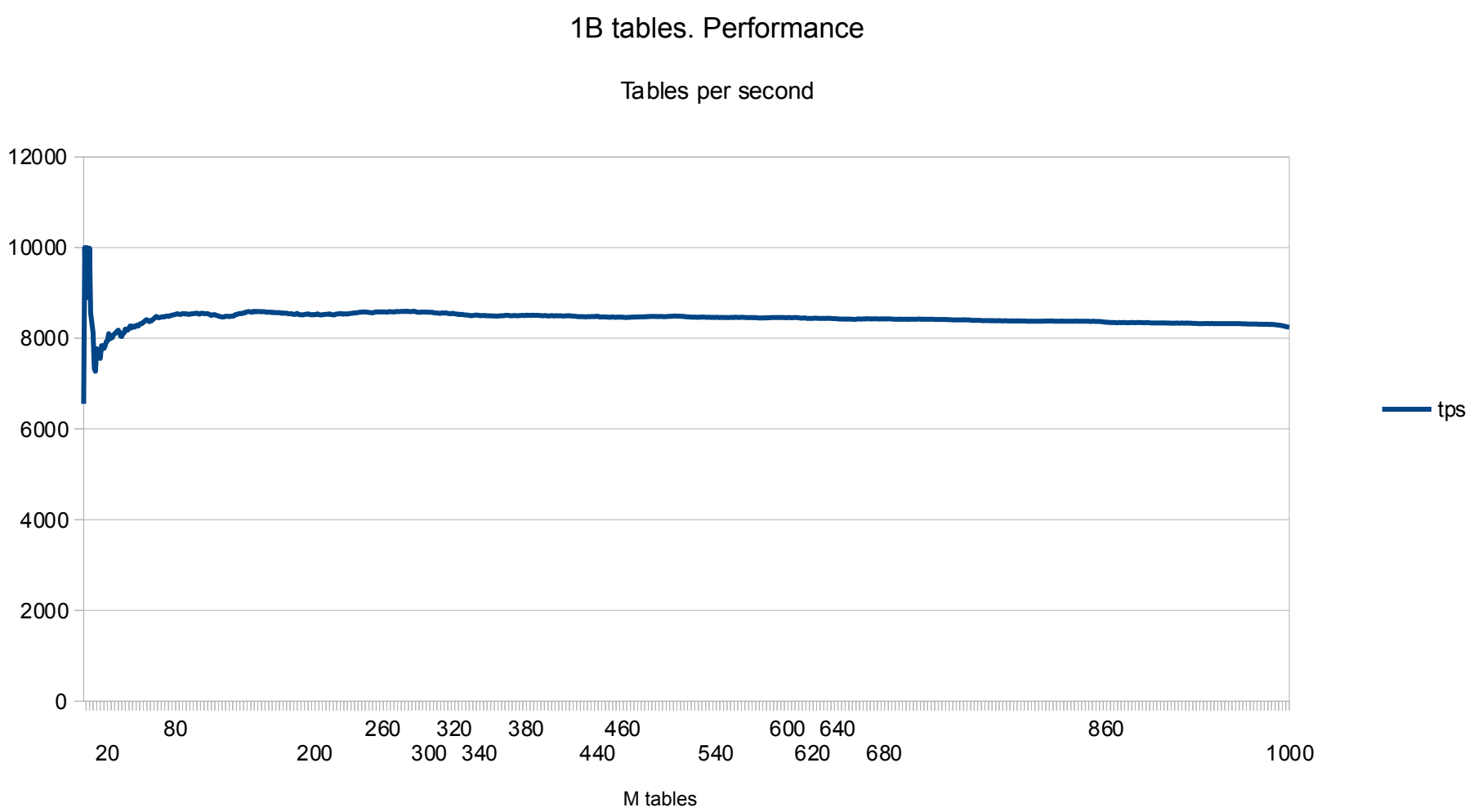
```
$ time ./postgresql-9.2.4/bin/psql btp -c "\dt" > tables
```

```
ERROR:  canceling statement due to user request
real    2993m51.710s
user    0m0.000s
sys     0m0.000s
```

cancelled by `pg_cancel_backend()`

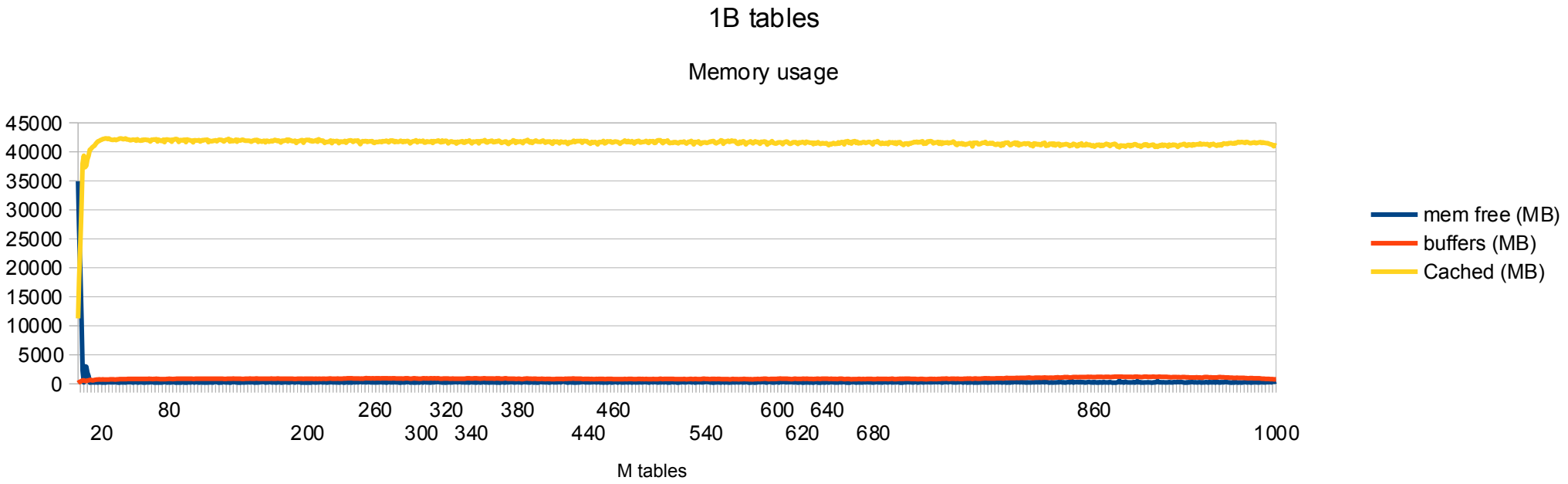


1B tables. Performance



Peak: 10Ktps

1B tables. Performance (II)



Avg backends load: 57%

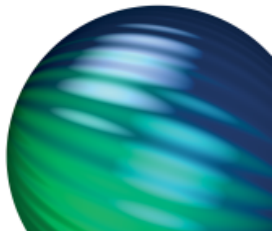
Avg system load: 11.7

1B tables. Make the db durable again

- Stop server. Move `pg_xlog` to disk
- Tune `postgresql.conf`:

```
fsync = on
synchronous_commit = on
full_page_writes = on
autovacuum = off
```

- Restart server. Enjoy ;)

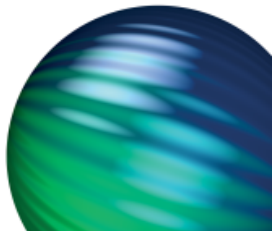


Acknowledgements

- Josh Berkus (and Selena Deckelmann, Jan Urbanski and Álvaro Herrara) who seem responsible for this crazy idea

- Big, big thanks to José Luis Tallón:
  - For bringing in the server and fine-tunning it
  - For co-authoring, co-working, co-architecting, co-programming and co-enjoying this project

- PgCon organization and sponsors :)



Billion Tables Project (BTP)

Álvaro Hernández Tortosa <aht@Nosys.es>

José Luis Tallón <jltallon@Nosys.es>