



Calculating the future

How to model PostgreSQL performance

Dmitrii Dolgov

14-03-2024

- Why is it relevant for you?
- Back of the envelope calculations
- Approximation
- Simulation

Why bother?

Benchmarking instead?

Benchmarking instead?

→ Resource intensive

Benchmarking instead?

- Resource intensive
- Hard to get full coverage

Benchmarking instead?

- Resource intensive
- Hard to get full coverage
- Requires cross validation

Benchmarking instead?

- Resource intensive
- Hard to get full coverage
- Requires cross validation

Enhance benchmarking!

It's easy

It's easy

→ just bump `max_wal_size`?

It's easy

- just bump `max_wal_size`?
- just increase `shared_buffers`?

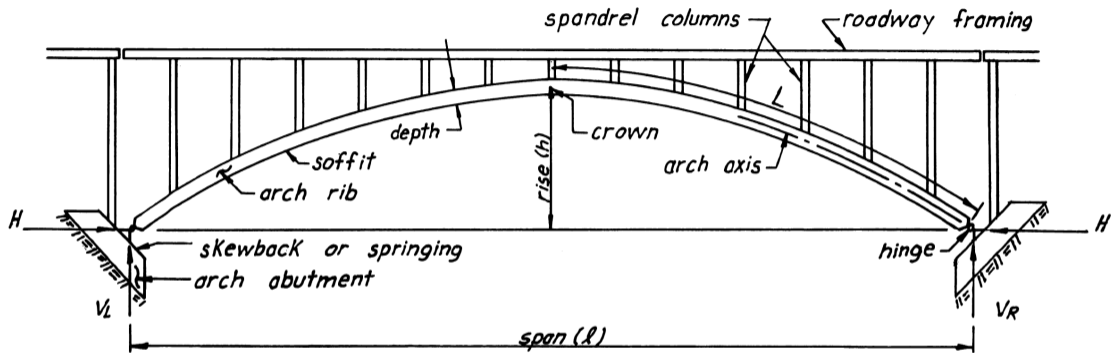
It's easy

- just bump `max_wal_size`?
- just increase `shared_buffers`?
- just configure `autovacuum`?

It's easy

- just bump `max_wal_size`?
- just increase `shared_buffers`?
- just configure `autovacuum`?

Well...



Douglas A. Nettleton, John S. Torkelson Department of Transportation, Federal Highway Administration, Office of Engineering, Bridge Division



OSM



Target of the experiment


```
create table test(a int);  
create index on test(a);
```

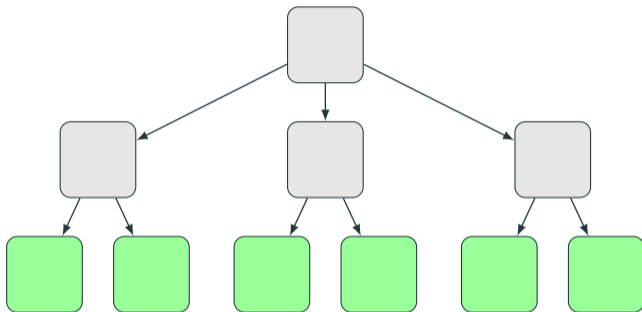
```
create table test(a int);  
create index on test(a);
```

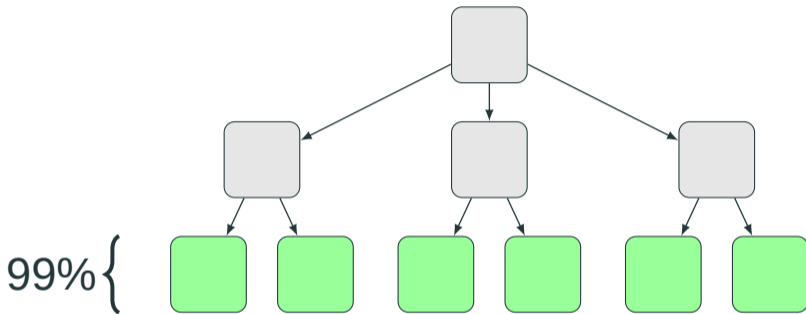
Back of the envelope calculations

Assuming we know the schema,
how to approximate space usage?

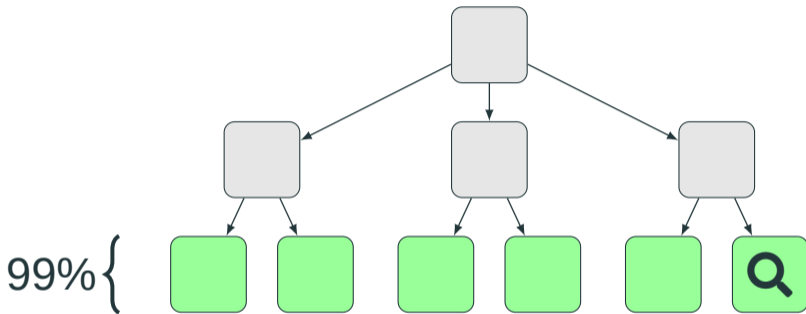
```
create table test(a int);  
create index on test(a);
```

```
create table test(a int);  
create index on test(a)  
    with (fillfactor = 100);
```





Goetz Graefe. "Modern B-Tree Techniques." Foundations and Trends in Databases 3.4 (2010) 203-402



Goetz Graefe. "Modern B-Tree Techniques." Foundations and Trends in Databases 3.4 (2010) 203-402

8192 / 0000

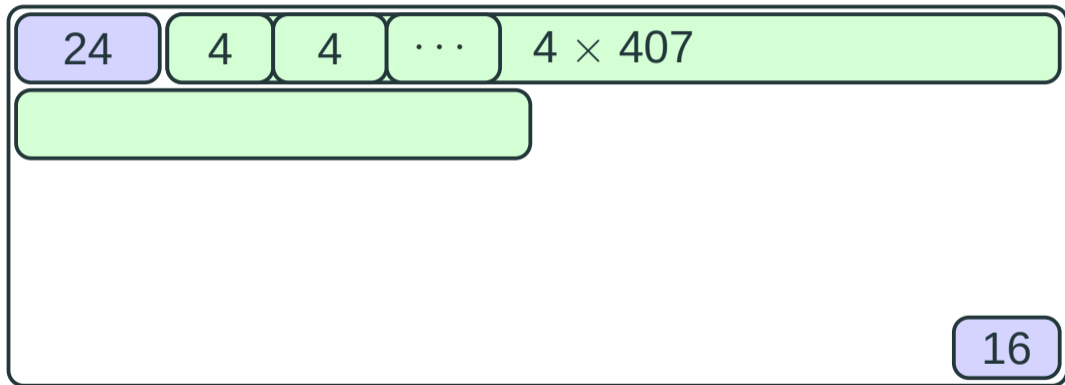


8192 / 0040

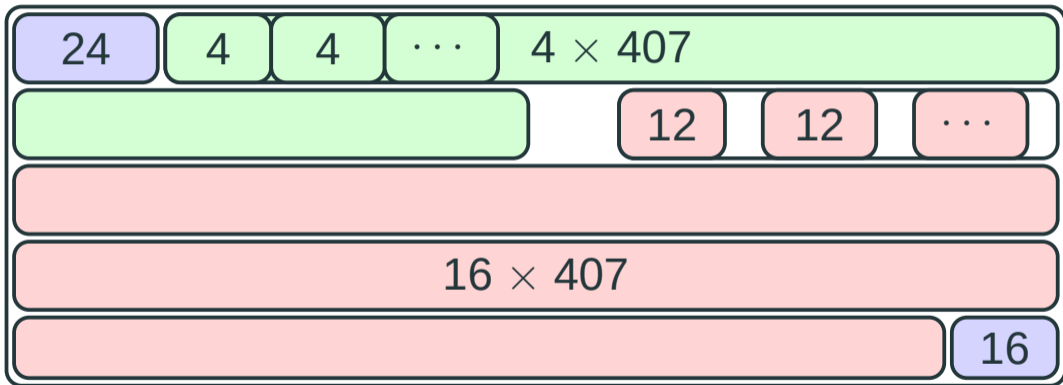
24

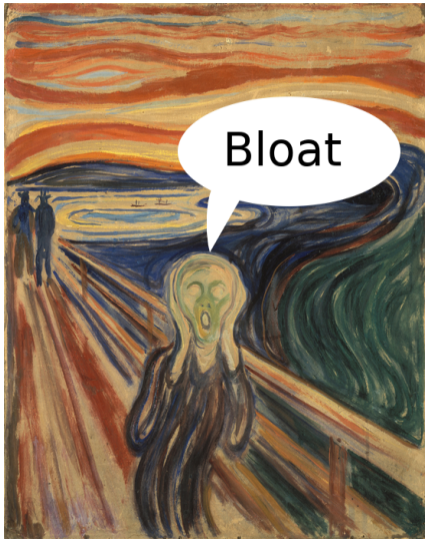
16

8192 / 1668



8192 / 8180





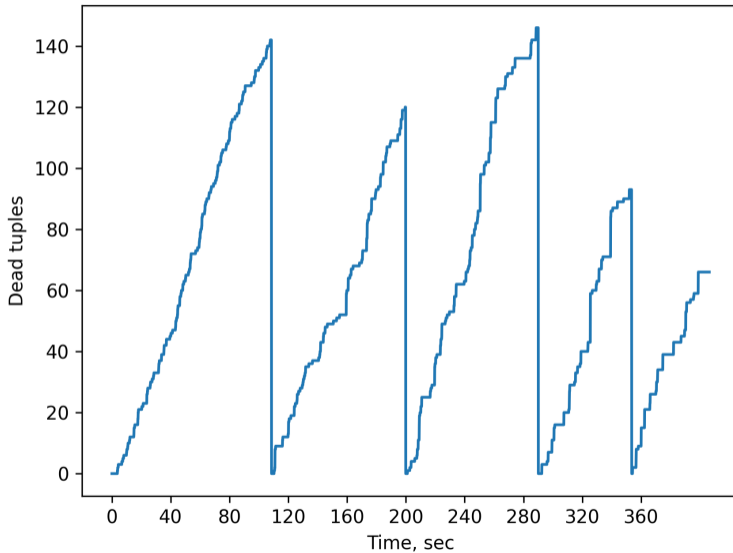
Edvard Munch – The Scream (about a bloated index)

Assuming we know the workload,
how to approximate bloat?

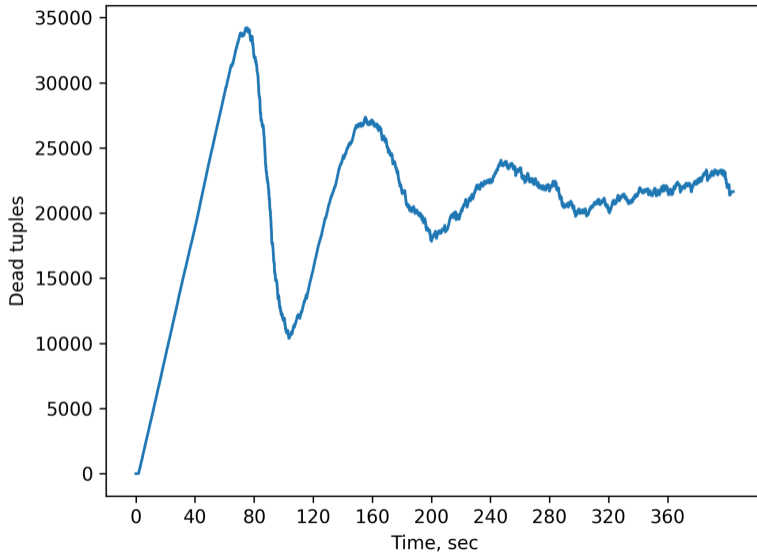
```
-- -M prepared --rate=max-rate
\set aid random(0, N)
\set bid random(0, N)

-- a pre-populated table
update test set a = :aid
      where a = :bid;
```


B-Tree page, uniform point update



B-Tree, uniform point update





Portrait de l'artiste sous les traits d'un moqueur

Approximation

Assuming we know the workload,
how to approximate amount of IO?

```
create table test(a int);  
create index on test(a)  
    with (fillfactor = 100);
```

```
create unlogged table test(a int);  
create index on test(a)  
    with (fillfactor = 100);
```

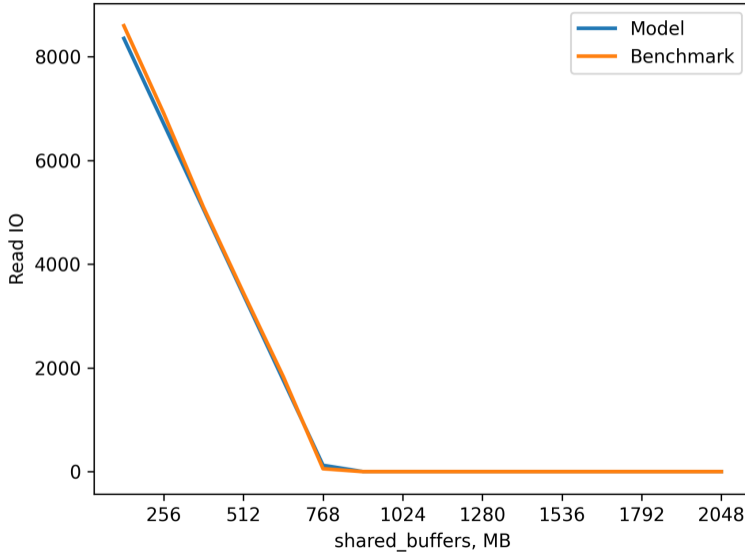
```
create unlogged table test(a int);
create index on test(a)
    with (fillfactor = 100);

# autovacuum = off
# *_flush_after = 0
# etc
```



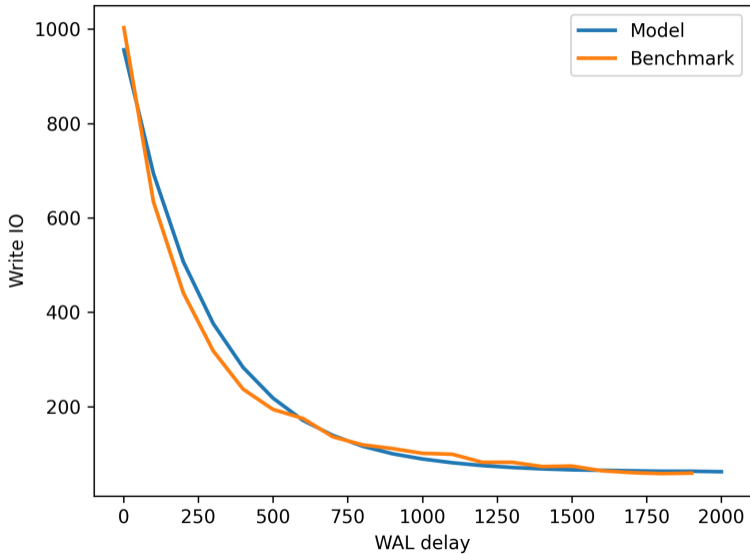
```
-- -M prepared --rate=max-rate  
\set aid random(0, N)  
  
-- a pre-populated table  
select * from test where a = :aid;
```

B-Tree, uniform read



```
-- -M prepared --rate=max-rate  
\set aid random(0, N)  
  
-- an empty table  
insert into test values(:aid);
```

B-Tree, uniform insert



$$IO = Q \cdot M_{press} \sum_{l=1}^L \frac{N_l}{N} + Q_w \cdot W_g + \frac{Q_w}{I}$$

$$IO = Q \cdot M_{press} \sum_{l=1}^L \frac{N_l}{N} + Q_w \cdot W_g + \frac{Q_w}{I}$$

Diagram illustrating the components of IO (Input/Output) with annotations:

- Read** (Red arrow) points to the first term: $Q \cdot M_{press} \sum_{l=1}^L \frac{N_l}{N}$
- Split** (Blue arrow) points to the third term: $\frac{Q_w}{I}$
- Insert** (Green arrow) points to the second term: $Q_w \cdot W_g$

Modeling the Linux page cache for accurate simulation of data-intensive applications

Hoang-Dung Do*, Valérie Hayot-Sasson*, Rafael Ferreira da Silva[‡], Christopher Steele[§], Henri Casanova[†], Tristan Glatard*

*Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada

[†]Department of Information and Computer Sciences, University of Hawai‘i at Mānoa, USA

[‡]Information Sciences Institute, University of Southern California, Marina Del Rey, CA, USA

[§]Department of Psychology, Concordia University, Montreal, Canada

Abstract—The emergence of Big Data in recent years has resulted in a growing need for efficient data processing solutions. While infrastructures with sufficient compute power are available, the I/O bottleneck remains. The Linux page cache is an efficient approach to reduce I/O overheads, but few experimental studies of its interactions with Big Data applications exist, partly due to limitations of real-world experiments. Simulation is a popular approach to address these issues, however, existing simulation frameworks do not simulate page caching fully, or even at all. As a result, simulation-based performance studies of data-intensive applications lead to inaccurate results.

In this paper, we propose an I/O simulation model that includes the key features of the Linux page cache. We have implemented this model as part of the WRENCH workflow simulation framework, which itself builds on the popular Sim-Grid distributed systems simulation framework. Our model and its implementation enable the simulation of both single-threaded and multithreaded applications, and of both writeback and writethrough caches for local or network-based filesystems. We evaluate the accuracy of our model in different conditions, including sequential and concurrent applications, as well as local and remote I/Os. We find that our page cache model reduces the simulation error by up to an order of magnitude when compared to state-of-the-art, cacheless simulations.

I. INTRODUCTION

The Linux page cache plays an important role in reducing filesystem data transfer times. With the page cache, previously read data can be re-read directly from memory, and written

type of hardware/software stacks are best suited to different application classes, as well as understanding the limitations of current algorithms, designs and technologies. Unfortunately, performance studies relying on real-world experiments on compute platforms face several difficulties (high operational costs, labor-intensive experimental setups, shared platforms with dynamic loads that hinder reproducibility of results) and shortcomings (experiments are limited to the available platform/software configurations, which precludes the exploration of hypothetical scenarios). Simulations address these concerns by providing models and abstractions for the performance of computer hardware, such as CPU, network and storage. As a result, simulations provide a cost-effective, fast, easy and reproducible way to evaluate application performance on arbitrary platform configurations. It thus comes as no surprise that a large number of simulation frameworks have been developed and used for research and development [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13].

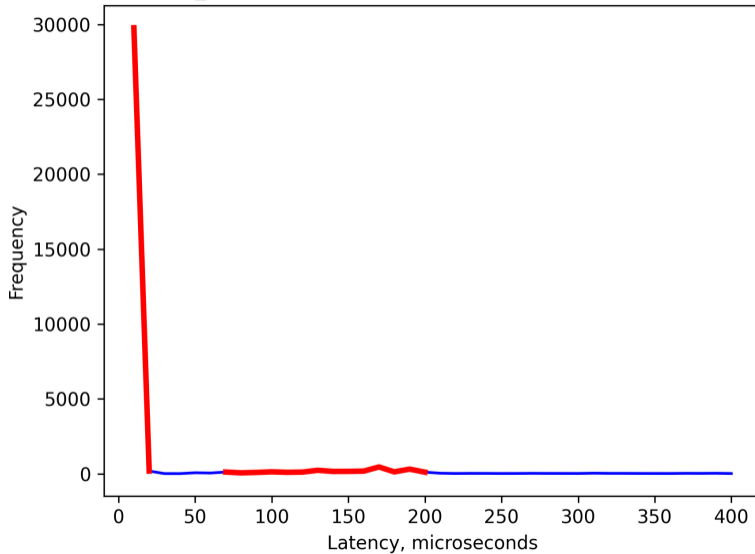
Page caching is an ubiquitous technique for mitigating the I/O bottleneck. As such, it is necessary to model it when simulating data-intensive applications. While existing simulation frameworks of parallel and distributed computing systems capture many relevant features of hardware/software stacks, they lack the ability to simulate page cache with enough details to capture key features such as dirty data and cache eviction policies [5], [6]. Some simulators, such as the

Simulation

Assuming we know the workload,
how to approximate query latency?

```
data Event = PqGetByte TxLatency
           | GetCachedPlan TxLatency
           | BtGetTuple TxLatency
           | BtInsert TxLatency
           | HeapPagePrune TxLatency
           | HeapUpdate TxLatency
           | CommitTx TxLatency
           | SocketFlush TxLatency
```

pq_getbyte latency distribution, uniform update



Summary

- Predicting the future is possible!
- Be aware of limitations
- Reduce large system to small parts
- Combine with benchmarking and profiling

Questions?

 @erthalion@fosstodon.org

 ddolgov at redhat dot com