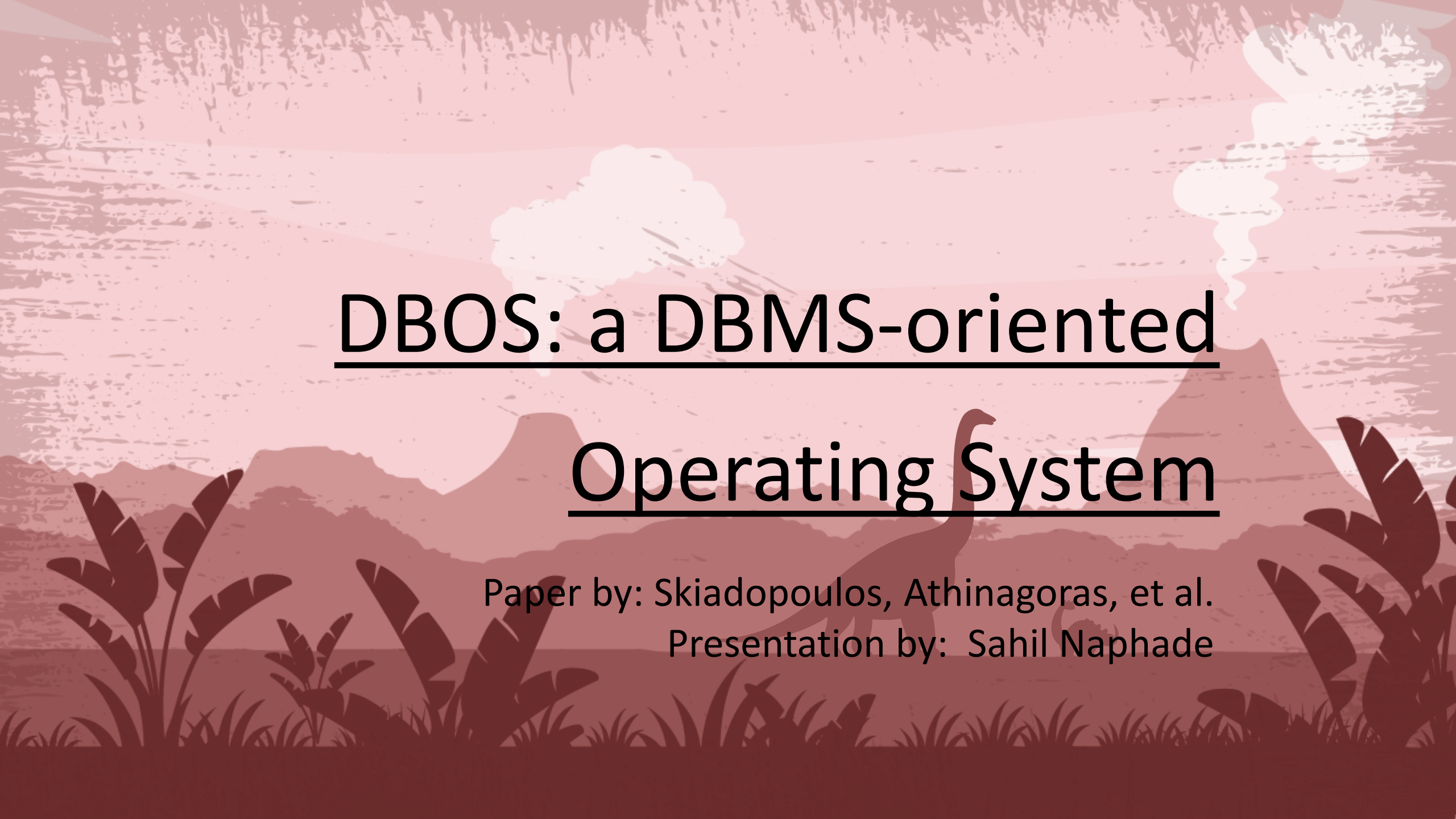


DBOS: a DBMS-oriented Operating System

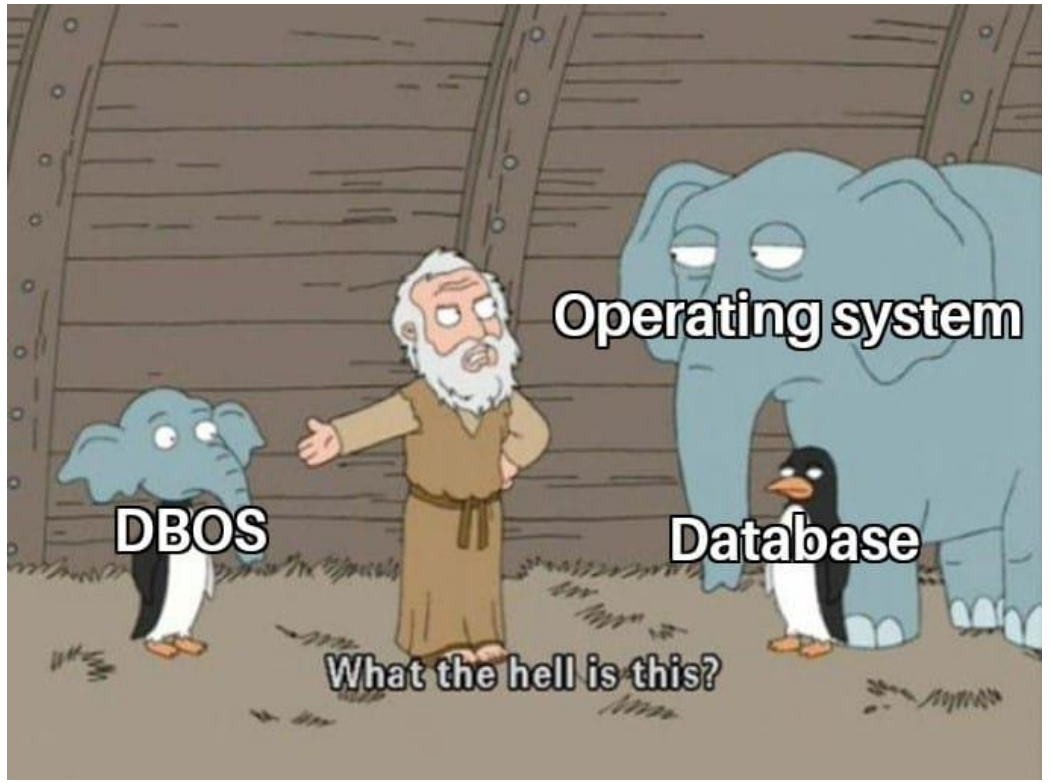
Paper by: Skiadopoulos, Athinagoras, et al.
Presentation by: Sahil Naphade

Inspiration

Challenges with current OS:

- Scale
- Cloud proliferation
- Parallel computation
- Heterogenous HW
- New applications
- New programming model
- Age
- Provenance

Original Idea: 2020
Partial Execution: 2022



Proposal (2020)

- A new OS with a data-centric architecture
  - All states: Data structures -> DB tables (everything-is-a-file -> everything-is-a-table)
  - State transitions -> use transactions
  - All operations performed as queries
  - Leverage DBMS for all of possible capabilities
  - Separate data from computations
  - OS states represented as uniform data model

What are the benefits?

- Performance Optimization
- Security
- Virtualization + Containerization
- Geographic distribution
- (Sophisticated) file management
- Better scheduling
- Improved state management

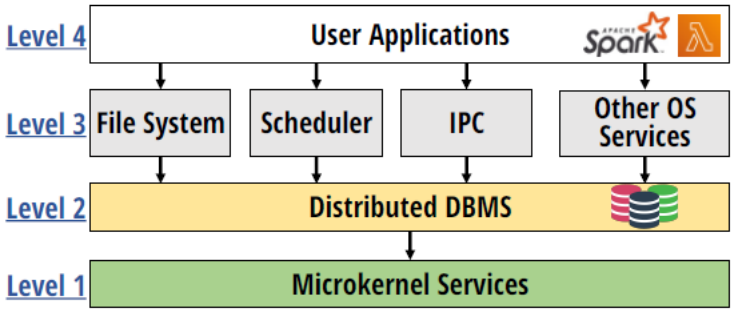
DBOS stack

Layer 4: User space
- Distributed applications
- Serverless model

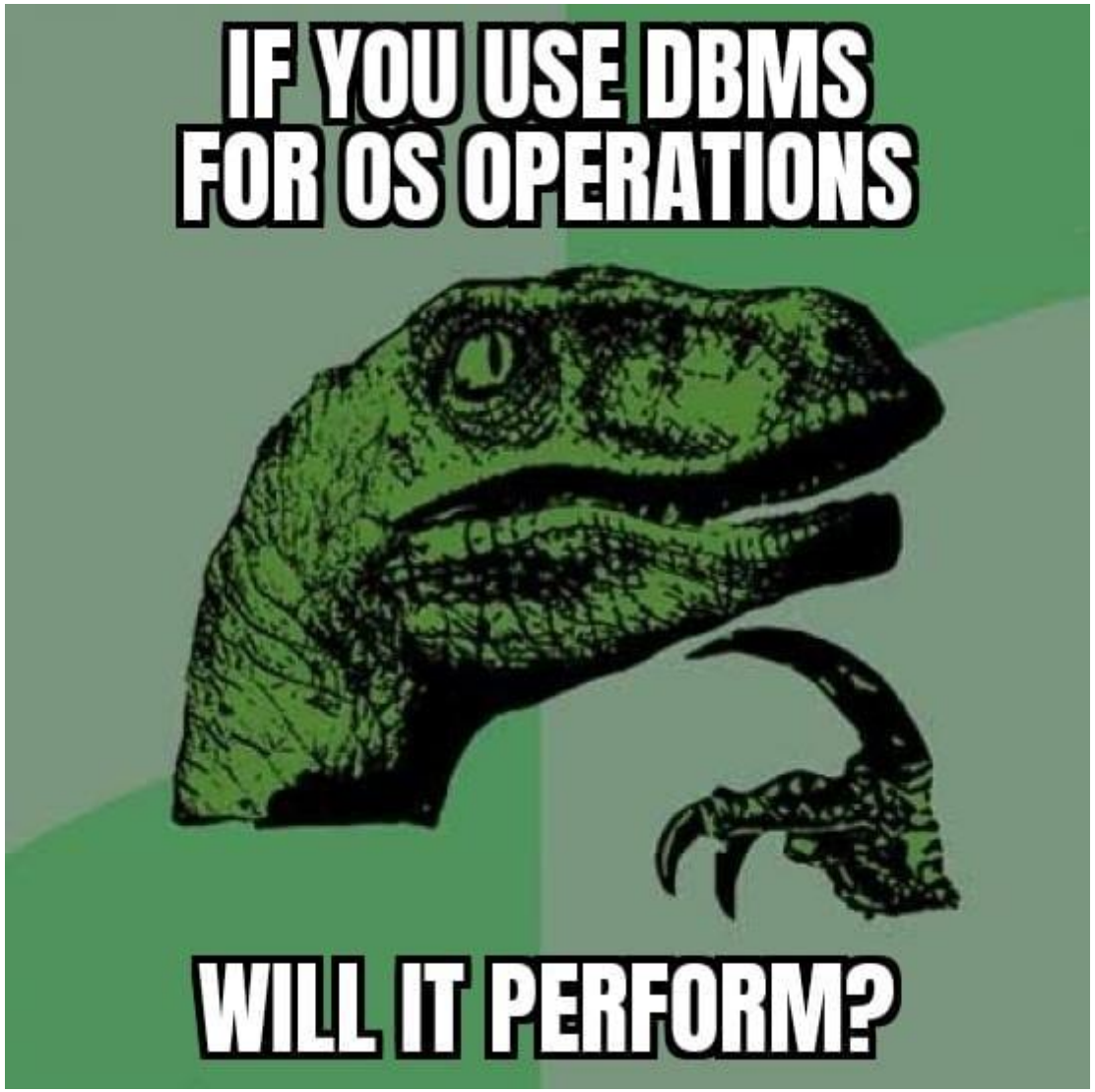
Layer 3: OS Functionality
- Task scheduling, Distributed FS, IPC

Layer 2: DBMS
- High-performance, multi-node, main-memory T-DB
- NoSQL can also be used
- Manages own memory

Layer 1: Microkernel
- No sophisticated Memory Mgmt



Img. Credit: DBOS (VLDB-2022)
https://doi.org/10.14778/3485450.3485454


IF YOU USE DBMS
FOR OS OPERATIONS
WILL IT PERFORM?

Implementation time!

Prototype in 3 stages

1. **Straw**

Possible to provide reasonable performance for 3 operations?

a. Task scheduling
b. Providing a Filesystem
c. Supporting IPC

Building the prototype with

Layer 1: Linux

Layer 2: RDBMS (VoltDB)

Layer 3: Coding by hand

Layer 4: Test programs

Image credits: Google search

DBMS Straw

- Why VoltDB?
  - Parallel, high-performance, multi-node, transactional (+ One more reason)
  - Tables are hashed on a user-specified key across nodes
  - Serializability and transactional failover
  - User-defined DBMS procedures, which are compiled and optimized
- As expected, highest performance is obtained when
  - Data in a Single partition
  - User task + data partition -> On the same node
- A task and worker

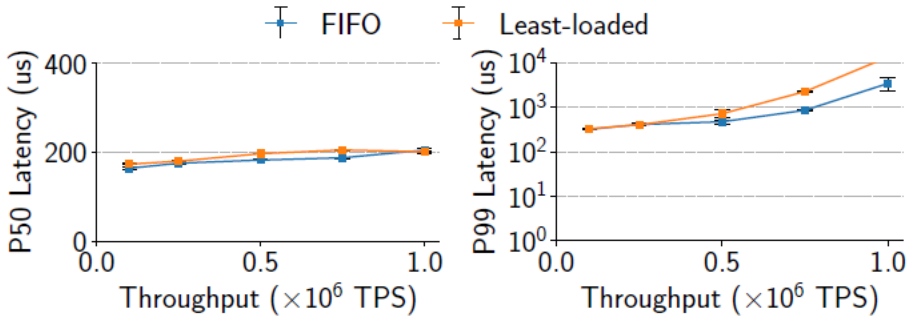
  Task (p_key#, task_id, worker_id, other_fields)

  Worker (p_key#, worked_id, unused_capacity)

DBMS Straw - Scheduling

```
schedule_simple(P, TID) {
  select worker_id,unused_capacity from Worker
    where unused_capacity>0 and p_key=P
    limit 1;
  if worker_id not None:
    WID = worker_id[0];
    UC = unused_capacity[0];
    update Worker set
      unused_capacity=UC - 1
      where worker_id=WID and p_key=P;
    insert into Task (P, TID, WID,...);
}
```

Figure 2: Simple FIFO scheduler.



(a) Median latency **(b) P99 latency**

Figure 5: Performance of schedulers.

```
schedule_locality(HP, TID) {
  ... // Same code as FIFO scheduler
  if worker_id not None:
  ... // Same code as FIFO scheduler
  else:
    insert into Task (HP, TID, null, ...);
}
```

Figure 3: Locality-aware scheduler.

```
schedule_least_loaded(P, TID) {
  select worker_id,unused_capacity from Worker
    where unused_capacity>0 and p_key=P
    order by unused_capacity desc
    limit 1;
  ... // Same code as FIFO scheduler
}
```

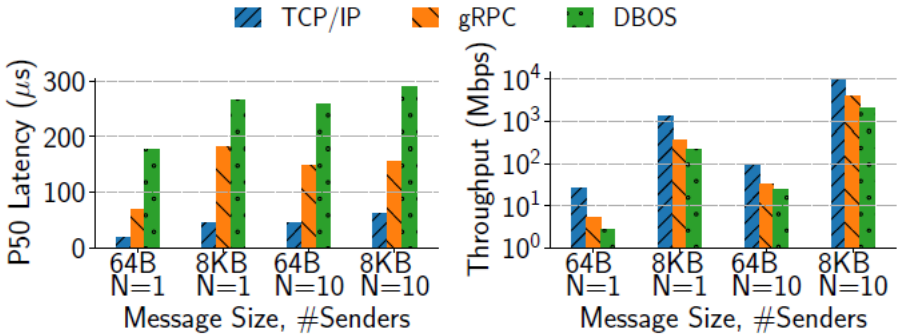
Figure 4: Least-loaded scheduler.

DBMS Straw - IPC

- Compare against TCP/IP and gRPC
- Ping-pong benchmark
- A message:

Message(sender_id, receiver_id#, message_id, data)

- Replicated *Message* table
- In-order delivery with message_id, exactly-once



(a) Median latency (b) Throughput

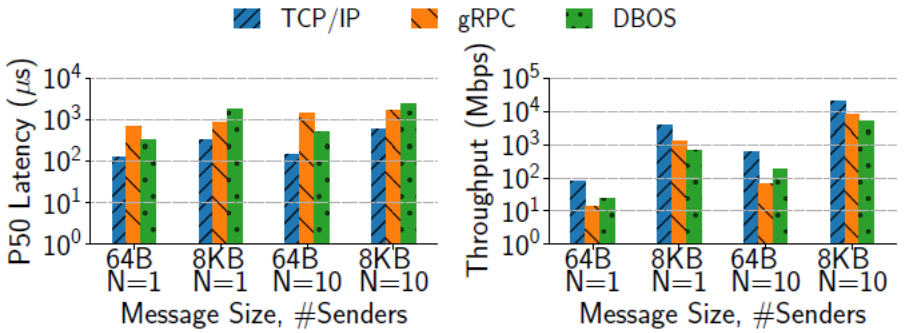
Figure 6: Performance of ping-pong benchmark.

- Limitations:
- Periodic Polling -> Mitigate with Triggers (VoltDB does not support!, But Postgres does)

DBMS Straw – IPC Performance

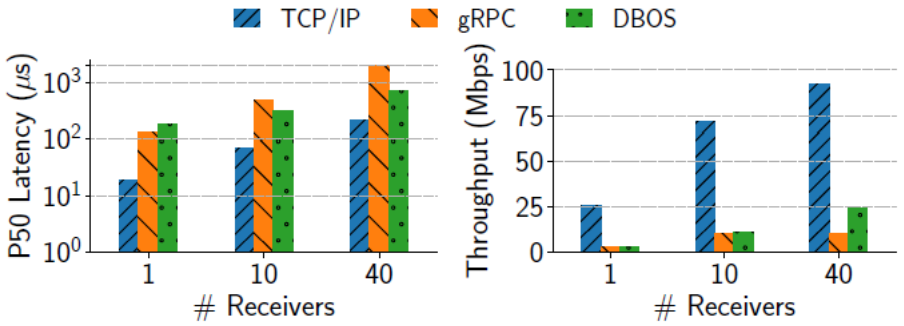
- DBOS achieves 24%–49% lower throughput and 1.3 – 2.5× higher media latency compared to gRPC
- DBOS achieves 4–9.5× lower performance than TCP/IP.

- Can be further optimized!
  - VoltDB uses TCP/IP as msg substrate
  - Next -> Run bare-bones data transport
  - Next-> Eliminate polling in DBOS
  - Still competitive enough! (Against gRPC)



(a) Median latency (b) Throughput

Figure 7: Performance of ping[20]-pong[20] benchmark.



(a) Median latency (b) Throughput

Figure 8: Performance of multicasting benchmark.

DBMS Straw – Filesystems

- Transactional and multi-node filesystem
- Two filesystems supported
  1. Stores data for the user on a single partition – partitioned on user_name
  2. Partitioned on block_no
- No need of “open” and “close”

DBMS Straw – Filesystems Perf

(a) Same File (b) Different Files

Figure 10: Throughput for 40 clients that read 20 blocks of 2KB each from (a) the same and (b) different files.

Table 1: Performance of file operations.

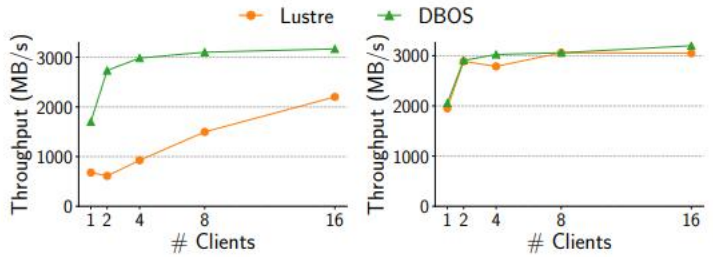
Operation	FS	Avg Latency (μs)	Max Ops/sec ($\times 10^3$)
Create File	Ext4	656.78 (±8.99)	31.39 (±0.56)
	DBOS	67.48 (±6.98)	303.85 (± 1.33)
Delete File	Ext4	654.04 (±8.99)	30.53 (± 0.61)
	DBOS	65.58 (±7.1)	302.30 (±2)

Table 2: Performance and LoC of conditional size aggregate.

Language	Time (msec)	Lines of Code
C++	9.90	98
SQL	0.65	2

No need of directory traversal – only single insert. Same for delete!

DBMS Straw – Filesystems Perf



(a) Read (b) Write

Figure 11: Throughput for 1 to 16 remote clients, co-located in a single node. Each client (a) reads and (b) writes parallel files of 8KB blocks.

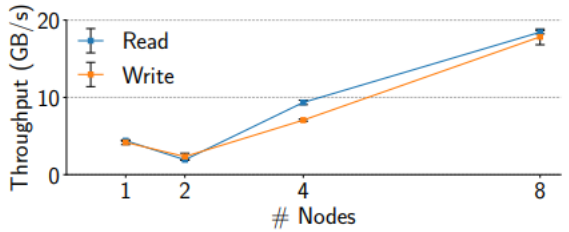


Figure 12: Throughput scalability with increasing DB nodes.

Implementation time!

Prototype Stages

2. Wood

Can OS functions be readily and compactly coded in SQL?

Can FS, Scheduling and IPC implementation work well?

Installing prototype in Linux is User space

Processes -> collection of short-running tasks assembled in a graph

Currently On-going!

Image credits: Google search

Implementation time!

Prototype Stages

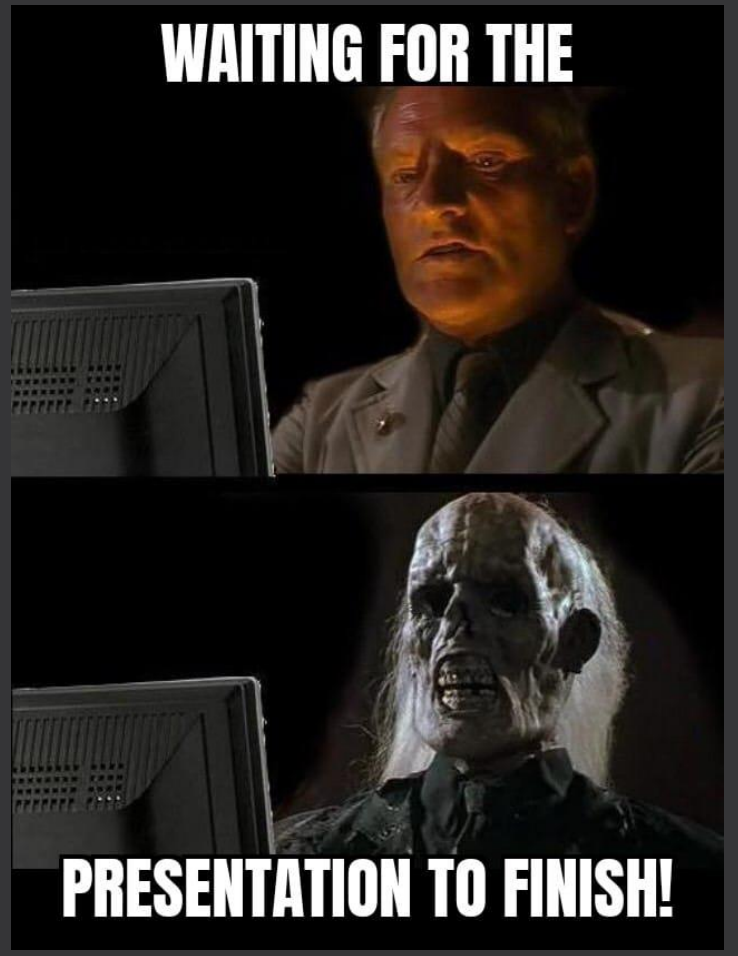
3. Brick

Beg, borrow, steal, or implement a micro-kernel

Not yet started!

Image credits: Google search

Thoughts and questions



Thank you!