DBOS: DBMS-oriented Operating System

Athinagoras Skiadopoulos, Qian Li, Peter Kraft, Kostis Kaffes, Daniel Hong, Shana Mathew, David Bestor, Michael J. Cafarella, Vijay Gadepally, Goetz Graefe, Jeremy Kepner, Christos Kozyrakis, Tim Kraska, Michael Stonebraker, Lalith Suresh, Matei Zaharia

(VLDB'22)

Presented by Sepehr Jalalian
Discussion led by:Sidhartha Agarwal
CPSC-504 Prof. Rachel Pottinger

The Context

Current system software dates from the 1980's

- Unix/Linux
- TCP/IP is just as elderly

The Context

In the last 40-ish years, the OS state to be managed

- processors, memory, storage
- Tasks, messages, files etc

- Has gotten massively larger (think 10^6 bigger)
- This makes managing OS state a DBMS problem!

The Context

And new resource management implementations have gotten a lot faster

- Main memory
- Different concurrency control
- Latch-free
- Deterministic execution
- Compilation

More Motivation

- Cloud infrastructure
- Serverless computing model
- Heterogeneous hardware
- Provenance

➢ All leading to a more complicated OS state management problem

Discussion 1

What does Linux do well? Think along the lines of

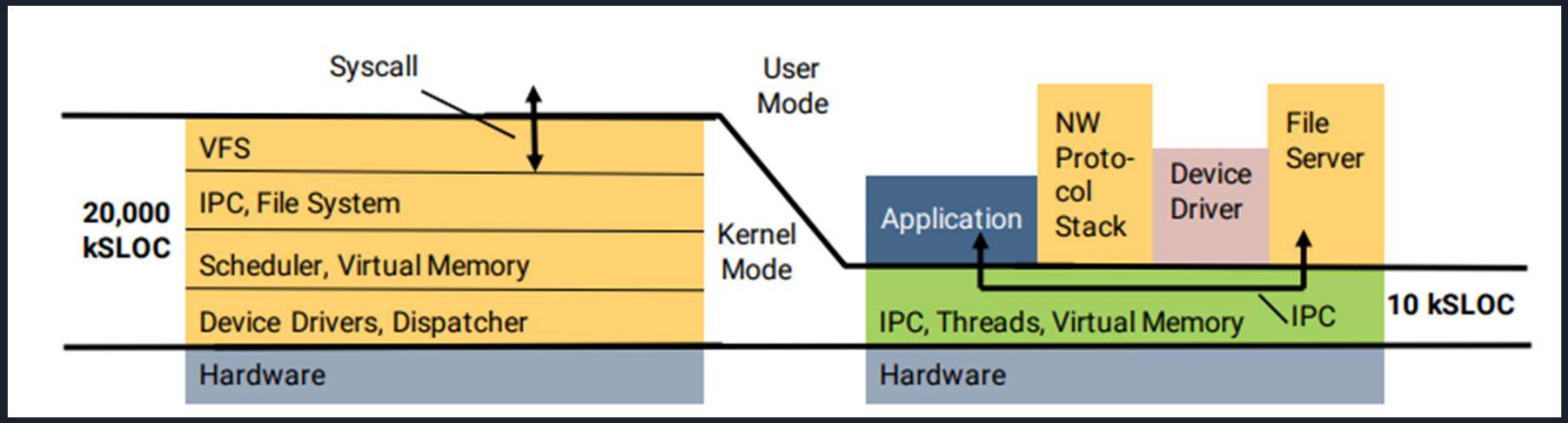
- How has hardware changed since linux was conceived?
- How has user applications changed since then?
- And for how many users those assumptions still hold?

(Discuss in groups of 4)

Some OS Preliminaries

- Processes and Virtualization
- Resource management
- Inter Process Communication (IPC)
- TCP/IP
- Kernel space and user space

Microkernels VS. Monolithic Kernels

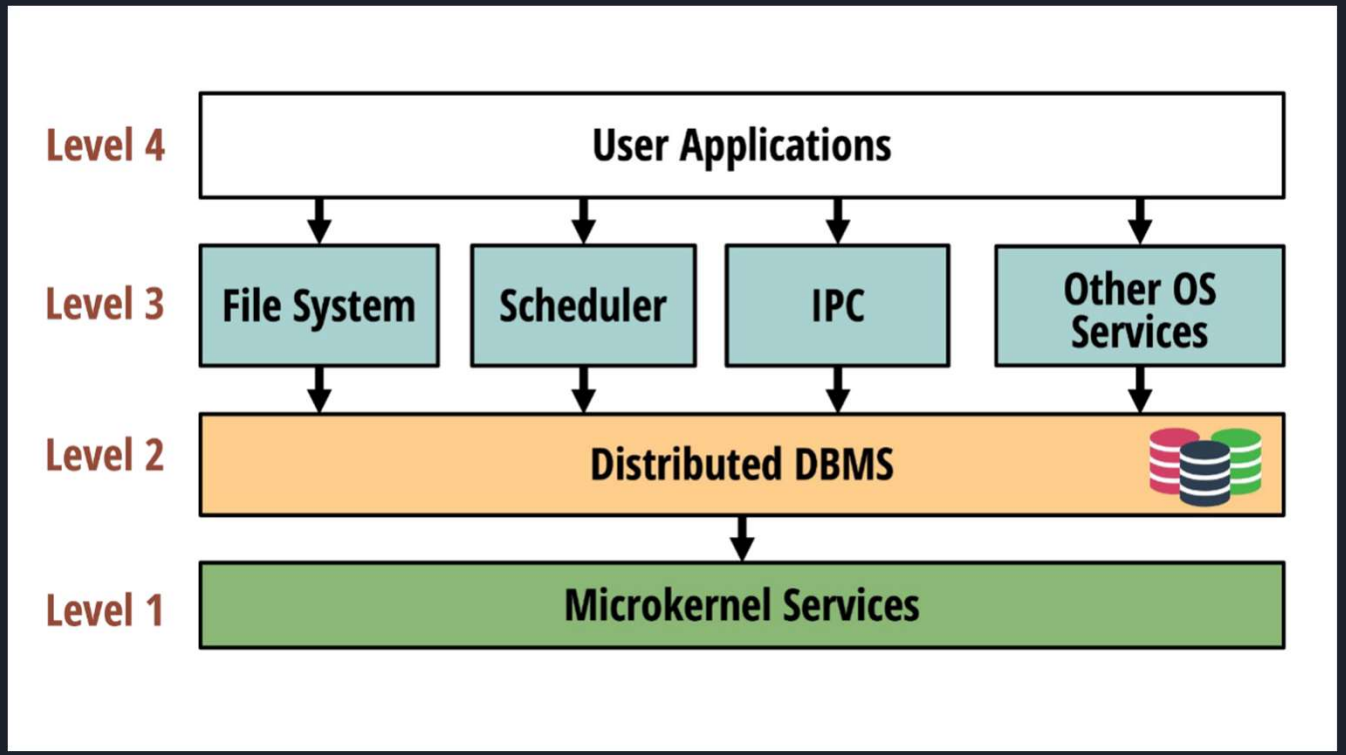


https://sel4.systems/About/seL4-whitepaper.pdf

Time to Rethink OSs from the Bare Metal on Up

- Level 4: User programs
- Level 3: OS support routines (mostly written in SQL)
- Level 2: High performance, OLTP, multi-node, multi-core distributed DBMS
  - VoltDB for now
- Level 1: microkernel (interrupt handlers, raw device support, basic byte movement)

DBOS Stack


Level 4
User Applications
Level 3
File System
Scheduler
IPC
Other OS Services
Level 2
Distributed DBMS
Level 1
Microkernel Services

Implementation Phase 1: Straw

Implement key OS services:

- IPC
- File Systems
- Task Scheduling

Using Linux as the level 1, and RDBMS as level 2

To Address the key doubt:

Can DBOS be performant?

Implementation: Wood

- Support for end-to-end distributed applications
  - A DBOS serverless framework

- Propose: Show that OS functions can be readily and compactly coded in SQL and the implementations work well in a real system.

Implementation: Brick

- Create a viable computing environment with DBOS
  - Utilize a microkernel (or write one) for the level 1
  - Port the serverless environment on top of that
- Will need support from industry!

Implementing the straw: VoltDB

Main memory DBMS

- Partitioned over many nodes
- SQL
- Transactional (ACID)
- High availability (replicas, failover)
- Very fast!
  - Millions of TPS on a modest cluster
- One example of a single-threaded, run-to-completion, main memory, deterministic, implementation

New Point of View

Old mantra

- Everything is a file

New mantra

- Everything is a table (they don’t see any advantage to a more complex data model)
- All OS state in the DBMS in tables!

Example: Scheduling

State Tables:

Task (p_key, task_id, worker_id, other_fields)

Worker (p_key, worker_id, unused_capacity)

Example: 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,...);
}
```

Implementing a FIFO scheduler with SQL and user defined functions

Basic Issue

- Analytics are much better/easier
- Monitoring is much better/easier
- Multi-node OS – no need for a separate cluster manager
- Transactions for all OS state
- But can this be fast enough????
  - Is a main memory distributed DBMS (VoltDB) fast enough?
    - Yes!

DBOS results - phase 1

- Implemented
  - A file system
  - IPC
  - Multiple schedulers
- On top of VoltDB
- Running on MIT Supercloud and on GCP, and a prototype deployment at Boston Consulting Group.

Message System

- A message table
  - `Message (sender, receiver, payload)`
- Partitioned on receiver
- Sending a message: SQL insert
- Reading a message: SQL query followed by a delete

Discussion 2

[**Jingxuan**] How the tradeoff between simplicity and performance would be different in OS compared to the database systems?

For example, since databases are closer to end-users, we expect them to be easy to use. Is simplicity that important in OS? Can it outweigh the importance of performance?

(discuss in pairs)

Discussion 2

[**Jianhao, paraphrased**] Authors claim that all of OS state can be thought of as a table. But is relational model the right data model for OS data?

We have seen other data models in this course. Would any of those be more fitting?

(discuss in pairs)

DBOS Results

- Scheduling
  - Is competitive
- File system
  - Is competitive with Linux FS and with Lustre
- IPC
  - Is competitive with gRPC
  - Loses to TCP/IP (but ...)

Experience with VoltDB

- Unbeatable on single partition xacts
- Run-to-completion as a stored procedure – single threaded
- Command logging/asynchronous checkpoints for power failure
- Active-active replication for HA
- But lays an egg on multi-partition xacts
- Basically locks a whole partition
- DBOS is slightly inconvenienced by making everything single-partitioned
- Xinjing Zhou (Lotus -- VLDB ‘22) fixed the problem

The Fix

- Idea #1: Phase-Switching
  - MP phase: group execution and group distributed commit.
  - SP phase: run-to-completion
- Idea #1 For MP, divide partitions into granules,
- Lock the granules
- NO_WAIT for deadlock prevention
- Lotus beats all comers (e.g. Aria (VLDB'20) and Calvin (SIGMOD'12))
- Better SP performance – the same or better MP performance

Moving to phase 2

- Provenance support
- Java serverless environment on top of what they have
  - Apiary
- What is needed from the micro-kernel
  - The OS side of things

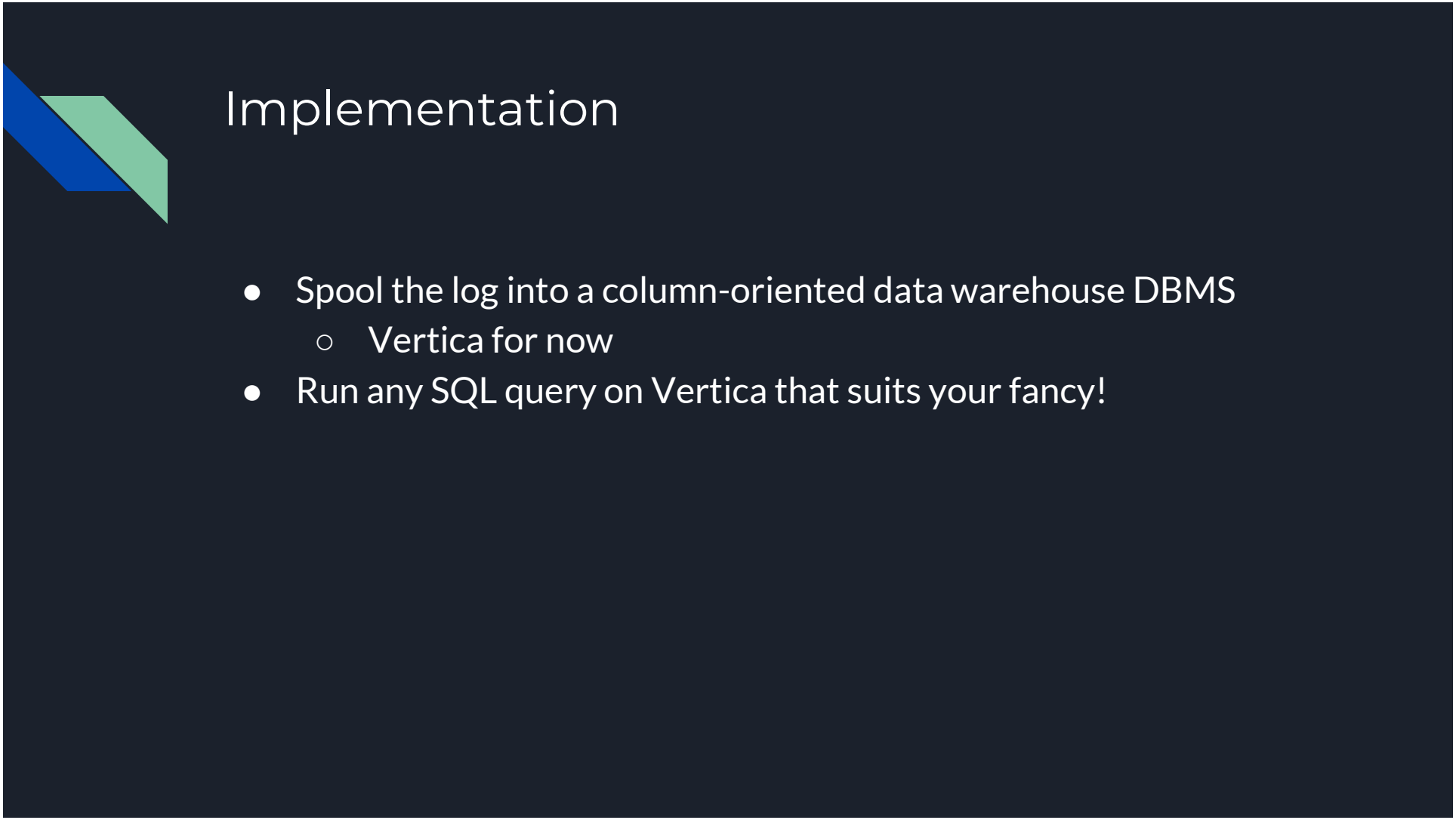
Updates

Provenance

- Keep a record of everything that has happened to a
  - DBMS record
  - File (which is collection of DBMS records)
  - Message (again a DBMS record)
- This is DBMS log processing
  - Well understood

More

- Applications can store “interesting state” in VoltDB tables
- Automatic provenance for such state
- App level monitoring – with very little effort


Implementation
Spool the log into a column-oriented data warehouse DBMS
Vertica for now
Run any SQL query on Vertica that suits your fancy!

Example Queries

- Rank suspicious objects
  - Ranks tables based on average daily visits to check for anomalies in data access patterns
- User connectivity
  - In the case of a compromised user, find all occurrences of other users interacting with the compromised user

Results

- Ingest faster (and easier) than Splunk

- Queries wildly easier to write and faster

Implications of DBOS

- Render Linux/Kubernetes obsolete
- Design paradigm inspired by DBOS
- All state in the DBMS (including application state)
  - Running on the cloud
  - In a serverless environment
  - With complete provenance
  - And much better security than now

Discussion 3

[Jason] The authors make a good case that current operating systems are out of date for current usage. We've seen many operating systems (and orhestration systems like k8s) come and go on both the server and desktop and more recently on mobile devices and cloud platforms. Is it somewhat necessary to "rethink the OS" at key points or perhaps better to expand on established systems?

(discuss in groups of 3)

Future: Exploring Use Cases

- Working with Mars Candy and John Deere on security/monitoring
  - Discover attacks quickly (SQL or ML model)
  - Restore DBOS state from provenance (seconds)

Discussion 4

This is a regular research track paper. How does this differ from other regular research track papers that we've read (that are from at least 1995 on)?

- In what ways is it similar? In what way is it different?
- Why do you think that it got in when it is obviously not what we would currently expect from database papers?

(Let’s do the 1, 2, 3, 4 thing)

Discussion 5

IPC: Inter Process Communication, send small machines between processes running on the same OS.

TCP: Send arbitrary sized messaged between two processes connected via a network.

[**Sid**] They compared IPC to TCP.

- Is that really fair?
- If not, why do you think they got away with it?
- If yes, explain why?

DBOS Conclusion

- They believe the next generation OS should be DBMS-oriented.
- DBOS design principles:
  - Distributed, cloud native
  - All state in the DBMS (including application state)
  - With complete provenance
  - And much better security than now
- For the curious:
  - dbos-project.github.io