DBOS Project

A Team of 20 people from MIT, Stanford, UW-Madison, Lincoln Labs, Google, VMWare, BCG, and Sigma Computing

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)

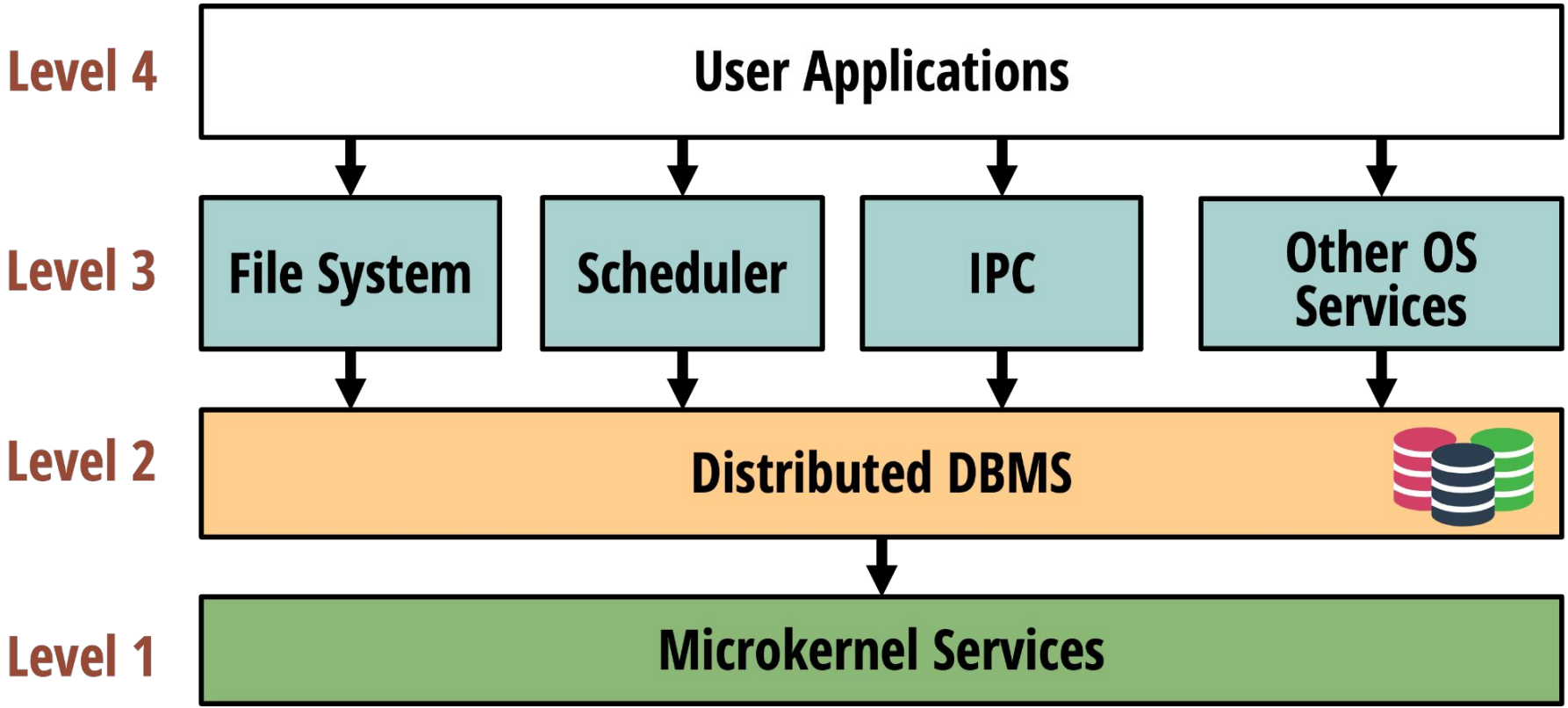
- This makes managing OS state a DBMS problem!

The Context

- And new RM implementations have gotten a lot faster
  - Main memory
  - Different concurrency control
  - Latch-free
  - Deterministic execution
  - compilation

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: a high performance, OLTP, multi-node, multi-core distributed DBMS
  - VoltDB for now
- Level 1: microkernel (interrupt handlers, raw device support, basic byte movement)


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

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

Summary of Our Point of View

- **Old mantra**
  - Everything is a file

- **New mantra**
  - Everything is a table (we don’t see any advantage to a more complex data model)

- **All OS state in the DBMS in tables!!!!!**

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

- We implemented
  - A file system
  - IPC
  - Multiple schedulers

- On top of VoltDB

- Running on MIT Supercloud and on GCP

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

DBOS Results (VLDB ‘22)

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

A comment on 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

This Encouraged Us to Move to Phase 2

- Provenance support
- Java serverless environment on top of what we have (runs now – subject of next talk)
- We have the microkernel left (where do we go from here – subject of Kostis talk)

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

And

- 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

Outreach

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

Implications

- 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

DBOS Conclusion

- We 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

Looking forward to your feedback!

https://dbos-project.github.io