DBOS: a DBMS-oriented operating system

Seminar: Moderne Datenbanksysteme für maschinelles Lernen und Wissensentdeckung (Summer Term 2023)

Huseyn Mammadli
University of Bamberg

ABSTRACT

Current operating systems are intricate systems that weren't created for the computer settings of today. The scalability, heterogeneity, availability, and security concerns in the present cloud and parallel computing systems are made tougher for them as a result. In order to solve these issues, is suggested a completely new OS design built on a data-centric architecture: all operating system state should be consistently represented as database tables, and actions on this state should be performed via queries from otherwise stateless activities. With this approach, it is simple to grow and evolve the OS without restructuring the entire system, examine and troubleshoot system state, update components without causing a service interruption, use machine learning to manage choices, and add advanced security features. It is addressed how a database operating system (DBOS) might enhance the programmability and performance of many of the most crucial applications used today, and a strategy for creating a DBOS proof of concept is suggested.

In paper is demonstrated how a database operating system (DBOS) may execute scheduling, file management, and inter-process communication at a level of performance comparable to that of current systems. Implementing OS services as ordinary database queries, while implementing low-latency transactions and high availability only once, can also deliver noticeably improved analytics while drastically reducing code complexity.

ACM Reference Format:
Huseyn Mammadli. 2023. DBOS: a DBMS-oriented operating system: Seminar: Moderne Datenbanksysteme für maschinelles Lernen und Wissensentdeckung (Summer Term 2023). In *Proceedings of Seminar*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Began with operating systems that have been in operation for almost 50 years. The Unix operating system was developed in the 1970s and Linux was first developed in the 1990s. While the design and capabilities of the machines on which it runs have evolved significantly, the fundamental architecture of the unix lineage of operating systems, as well as other operating systems like windows, has not changed in all that time. Not single node operating systems or desktop operating systems, but rather operating systems that run on massive computer clusters or in data centers are being discussed here.

It should be mentioned that the Unix concept of having everything in a file was appropriate at the time. However, it doesn't actually scale to these huge applications operating on sizable computer clusters. Since they rely heavily on network connectivity and mostly employ RPC, they have a lot of state to maintain across this big distributed system. It appears that under that load, Unix is starting to squeak and is suggested that, because there is so much state to handle in these enormous, distributed contemporary systems, the state management problem is a big data issue in and of itself. Hardware is becoming massively parallel and heterogeneous. These "sea changes" make it imperative to rethink the architecture of system software, which is also the topic of this paper[1].

2 RELATED WORK

Data Base Operating System, or DBOS, is the name of the new system, which results in a radically different operating system stack.

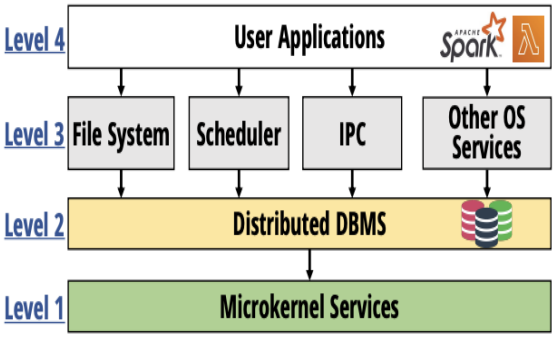


Figure 1: Proposed DBOS stack. Level 1 is the bottom layer

Level 4: User Applications

At the highest level level four is user applications. These are big distributed applications, such as online search, machine learning and other functions. The programming paradigm is comparable to the serverless architecture, which is used by many modern commercial clouds. Consequently, the entire program is divided up into a number of smaller sub jobs, each of which runs only when necessary. In order to do serverless computing, his top level operating system offers abstractions.

Seminar,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The most significant aspect of this stack's benefit is that all the state that must be managed in order to execute these massive apps is maintained in a database. This contains elements like scheduling, file storage, and inter-process communication. All of it is supported by a contemporary database with SQL querying capabilities. This makes tasks like monitoring and debugging considerably simpler to have access into the operational health of these systems.All of these things are fairly simple to achieve because they are able to quickly query the database for the application status[2].

Level 3: OS Functionality

Going down one,now at level three. Basic OS features include things like task scheduling, file systems, and inter-process communication.

Every operating system service operates using a combination of user-defined functions and SQL, and all of these services rely on a unified, global view of the operating system state represented by DBMS tables.

As a result, supporting cross-cutting activities is made simple for services.A contemporary database serves as the foundation for all of that.

Level 2: The DBMS

The distributed database system, which is level two of the architecture, is its center.A utilizing a multi-node main memory transactional database with this core component, which would be extremely high performance is suggested. The database must have an effective sql query engine.since that will serve as the foundation for most of the querying and state management in this system.

Level 1: Microkernel Services

And lastly, right close to the hardware at the lowest level. A number of microkernel services are available. This is what supplies the very fundamental stuff that the distributed database system requires to function and abstracts the hardware.This includes elements like as the raw device drivers, interrupts, and so on. The system comprises a set of backend nodes that assume one or more 'roles' in the cluster[3].

3 METHODE

- The answer to the question "Doesn't a full-fledged database need a lot of high-level operating system abstractions in order to run" is no, since, like most commercial database systems, this actually avoid many of the file system and other fundamental high level abstractions. Directly managing the data on the drive and using the RAW disk. Therefore, a fully functional modern database may run with very minimal assistance from the operating system.Consequently, this architecture is quite realistic.

- "How would some typical system tasks performed in such an operating system?" Everything focuses on tables and SQL queries, much like it would expect in a database, which makes it really fascinating.

Therefore, the first step to capture the current state of the task in a table, just as one would in a database.This is a table and has its fundamental structure.

```
Task (p_key, task_id, worker_id, other_fields)
```

Then the code that would be used to create a task scheduler, that chooses which task to schedule when. This is FIFO task scheduler pseudocode.hence it can be seen that the work was chosen using a SQL SELECT query. The task's characteristics are continued in some pseudocode after that.

```
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.

Finally, there is another table, which called the worker table, where tasks might be run. A SQL UPDATE statement was then used to update the worker table with the job that was chosen to run.

```
Worker (p_key, worker_id, unused_capacity)
```

The inter-process communication is also done by a table.

```
Message(sender_id, receiver_id, message_id, data)
```

So hier is message table, which could put messages into it and Users can read messages out of this table which is already coming with transactional guarantees. because This is transactional modern database underneath all this and could keep mapping of entire operating system design in terms of these tables and schemas.

```
Map (p_key, partition_id, host_name, host_id)
User (user_name, home_partition, current_path)
Directory (d_name, content, content_type,
  user_name, protection_info, p_key)
Localized_file (f_name, f_owner, block_no, bytes,
  f_size, p_key)
Parallel_file (f_name, f_owner, block_no, bytes,
  f_size)
```

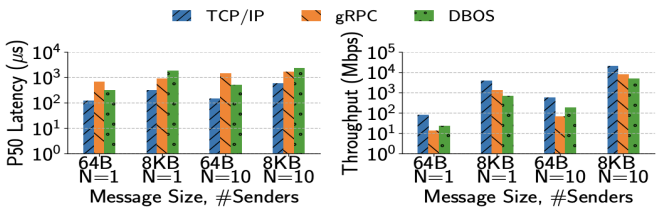
Figure 3: Filesystem tables for data and metadata.

For example this is the basic schema for how would tried to build a traditional file system in this operating system.

So instead of writing code for everything, this has a very different flavor of how to build system logic on top of the primitives. It is possible to write SQL queries, inserts, and updates to query the state, manage the state, and update the entire state in this very database- and schema-oriented way. That is an aspect of both the operating system and the application.

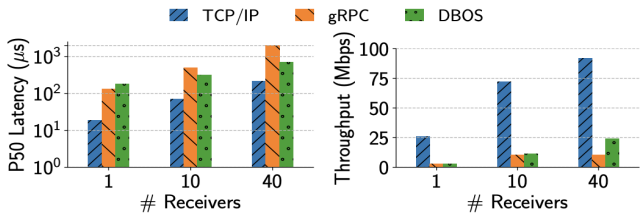
4 EVALUATION

So as to evaluate the viability of this concept, is created a very rudimentary prototype. The initial prototype, which is called "Straw", will be followed by systems termed "Wood" and "Brick" in the future.



(a) Median latency **(b) Throughput**

Figure 4: Performance of ping20-pong20 benchmark.



(a) Median latency **(b) Throughput**

Figure 5: Performance of multicasting benchmark.

For Straw Prototype is picked Linux as the base layer on top of the hardware instead of a micro kernel and the database layer is VoltDB. A variety of benchmarks is also conducted to compare the performance of their prototype to more established operating systems like Linux with well-known RPC mechanisms like gRPC.

For small messages, DBOS outperforms gRPC up to 2.7 times, while for large messages (8 KB), it underperforms [2]. Although the difference between TCP/IP and DBOS has decreased slightly, it is still wide.

When considered that the DBOS messaging strategy is implemented in a few lines of SQL code running on an unmodified DBMS, and gRPC is a specialized communication framework created by many developers over many years, the minor performance difference between DBOS and gRPC is astounding.

However, considering that this is a very crude prototype and this is still hopeful.Consequently, still shows that if could be improve it and get more performance out of it, might be highly competitive with a more conventionally designed operating system.

5 CONCLUSION

Finally, this study proposed a new operating system design that consists primarily of modern databases.Furthermore, it is claimed that this would significantly simplify the extremely difficult state management that most contemporary operating systems, along with the sizable distributed applications they support, must handle and how with sql queries, inserts, and updates. We have this very new approach of creating system code.It streamlines the development of all these apps by utilizing contemporary database features like transactional updates and others.

REFERENCES

[1] Michael J. Cafarella, David J. DeWitt, Vijay Gadepally, Jeremy Kepner, Christos Kozyrakis, Tim Kraska, Michael Stonebraker, and Matei Zaharia. 2020. DBOS: A Proposal for a Data-Centric Operating System. *CoRR* abs/2007.11112 (2020).

[2] 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, and Matei Zaharia. 2021. DBOS: A DBMS-oriented Operating System. *Proc. VLDB Endow.* (2021).

[3] Lalith Suresh, João Loff, Faria Kalim, Nina Narodytska, Leonid Ryzhyk, Sahan Gamage, Brian Oki, Zeeshan Lokhandwala, Mukesh Hira, and Mooly Sagiv. 2019. Automating Cluster Management with Weave. *CoRR* abs/1909.03130 (2019).