DBOS

DBMS oriented Operating System

OUTLINE

INTRODUCTION

- Basic architecture has not changed

- But architecture and capabilities of the machines on which it runs have changed very drastically

1970´s

1990´s

Destination

Run on

- Large clusters of computers or data centers

not

- Single node operating Systems or a Desktop operating System

Large applications running on very large clusters of computers.

Mostly work on RPC (Remote Procedure Call) and heavy on network communications

So a lot of state to manage and that all state itself is a big data problem

WHY ?

ARCHITECTURE

- Level 4

USER APPLICATIONS

Large distributed applications such as machine learning, web search, and so on.
The programming model is similar to the serverless model we see in many commercial Cloud.
Whole application broken down into a series of smaller subtasks, each running on demand for a very short period of time

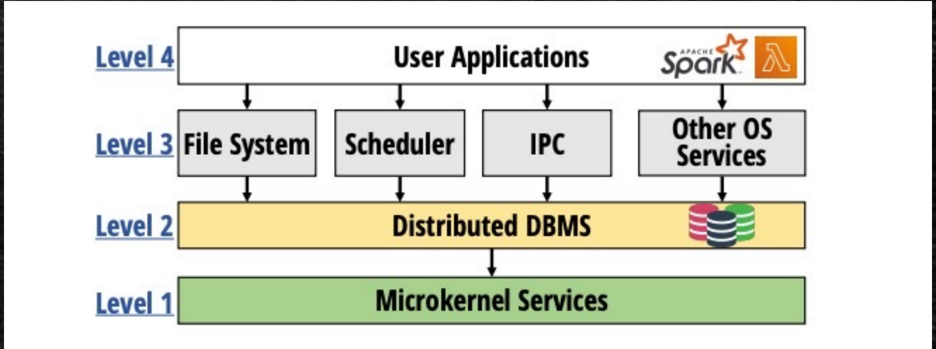


Figure 1: Proposed DBOS stack. Level 1 is the bottom layer.

ARCHITECTURE

- Level 4

  Much easier to get visibility into the operational status of these applications, things like monitoring, debugging .

  Can be done very easily because simply query the state of application from the database.

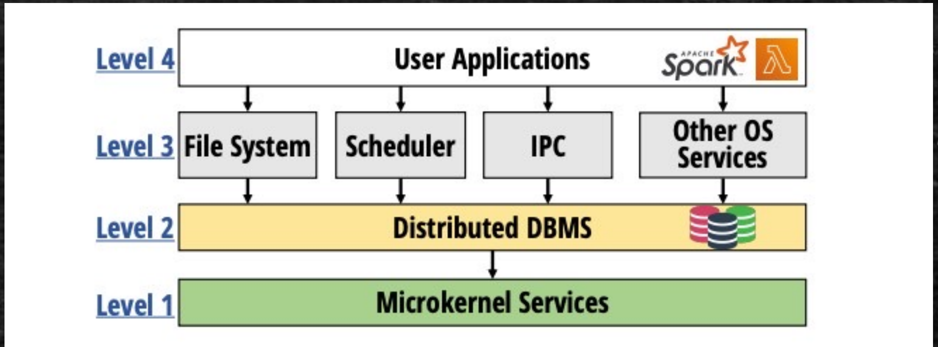


Figure 1: Proposed DBOS stack. Level 1 is the bottom layer.

ARCHITECTURE

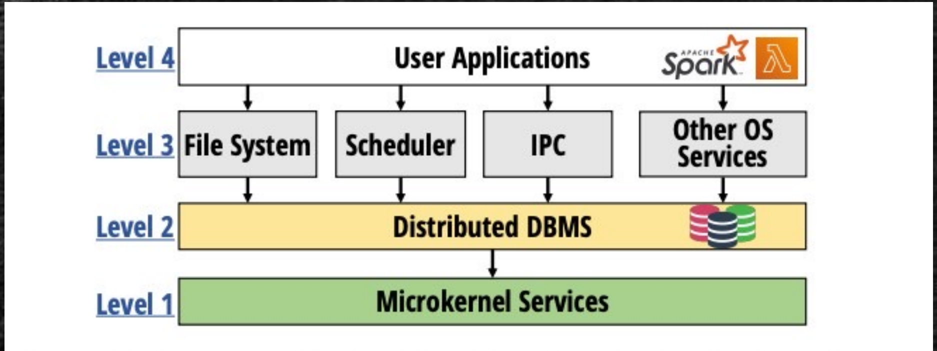


Figure 1: Proposed DBOS stack. Level 1 is the bottom layer.

- Level 3

  Basic operating system functionality things, like scheduling tasks or file systems or into inter-process communication

Dieses Foto https://cs.stanford.edu/~matei/papers/2022/vldb_dbos.pdf

ARCHITECTURE

- Level 2

  Heart of Architecture which is a distributed database system

  should be: multi-node main memory transactional database

  good SQL Query Engine

=> the basis of how a lot of the querying and state management is going to be done in this system

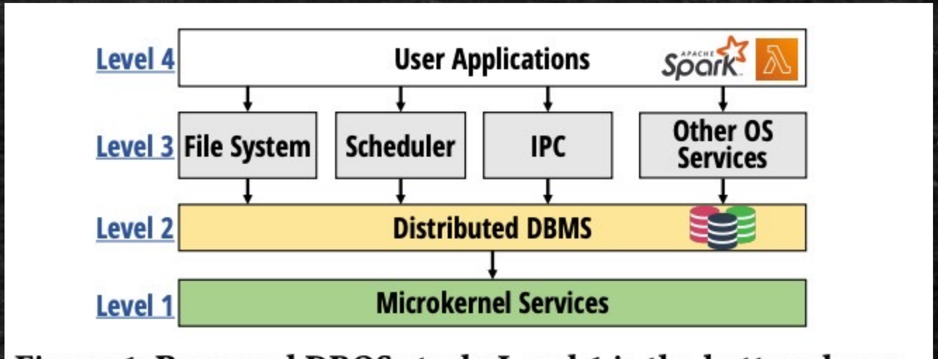


Figure 1: Proposed DBOS stack. Level 1 is the bottom layer.

Dieses Foto https://cs.stanford.edu/~matei/papers/2022/vldb_dbos.pdf

ARCHITECTURE

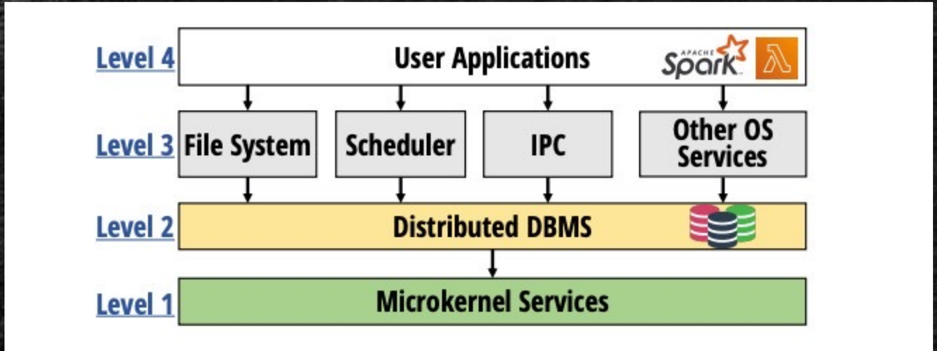


Figure 1: Proposed DBOS stack. Level 1 is the bottom layer.

- Level 1

  a set of microkernel services which abstracts the hardware and provides the very basic things that the distributed database system needs in order to run

=> things like the raw, device drivers, interrupts and so on.

TASKS

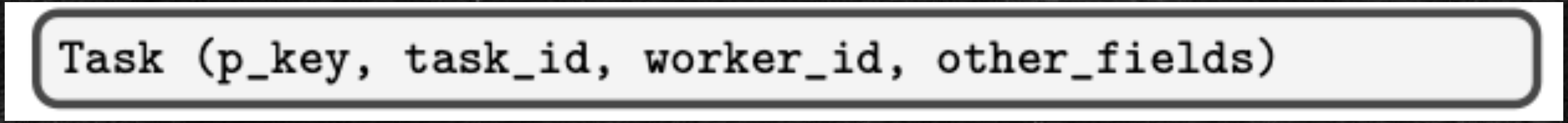
- Doesn't a full-fledged database need a lot of high-level operating system abstractions in order to run?

- not really. Because at most commercial database systems, actually bypass a lot of core high level abstractions like the file system . Directly used the raw disk and do their own management of data on the disk. So in fact it is very possible or a full-fledged modern database to run with very little support from the operating system

TASKS

- Schedule tasks

The fields of the Task table are:

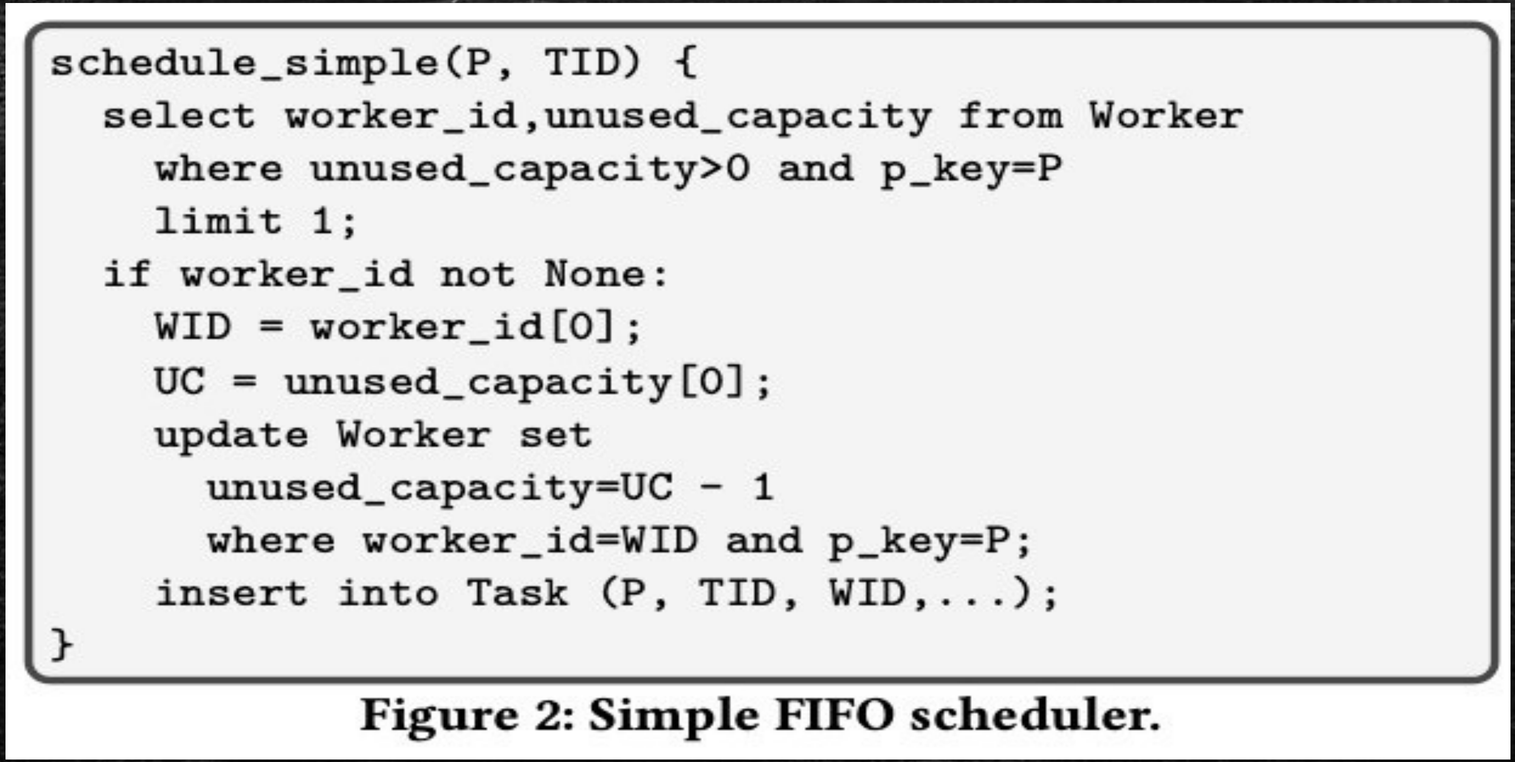


```
Task (p_key, task_id, worker_id, other_fields)
```

The next question is how can write a scheduler for our tasks that selects which task to schedule and when?

TASKS

Pseudocode for a FIFO task scheduler



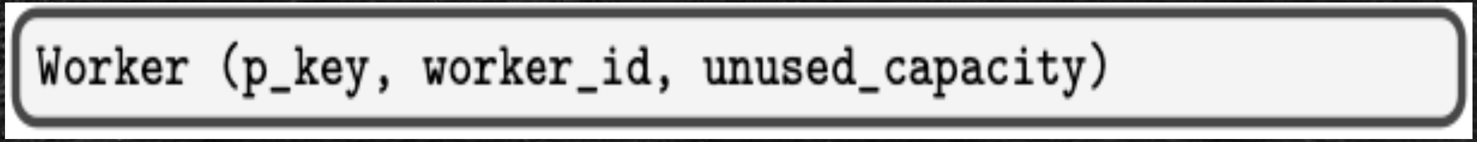
```
schedule_simple(P, TID) {
  select worker_id,unused_capacity from Worker
    where unused_capacity>0 and p_key=P
    limit 1;
  if worker_id not None:
    WID = worker_id[0];
    UC = unused_capacity[0];
    update Worker set
      unused_capacity=UC - 1
      where worker_id=WID and p_key=P;
  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

Figure 2: Simple FIFO scheduler.

Dieses Foto https://cs.stanford.edu/~matei/papers/2022/vldb_dbos.pdf

TASKS

Our schedulers also require a Worker table:



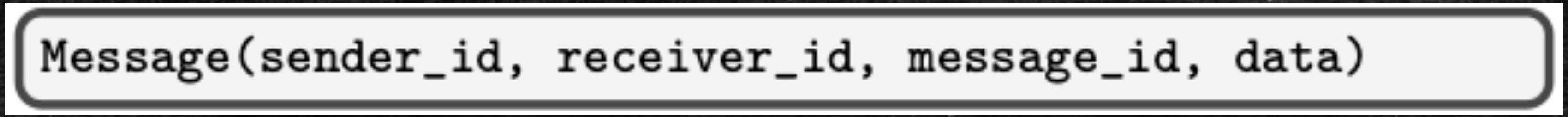
```
Worker (p_key, worker_id, unused_capacity)
```

But how does inter-process communication work?

MESSAGES

- That is also done by a table. So have a message table where can put messages and consumers can read messages from this table

A Message table is required:

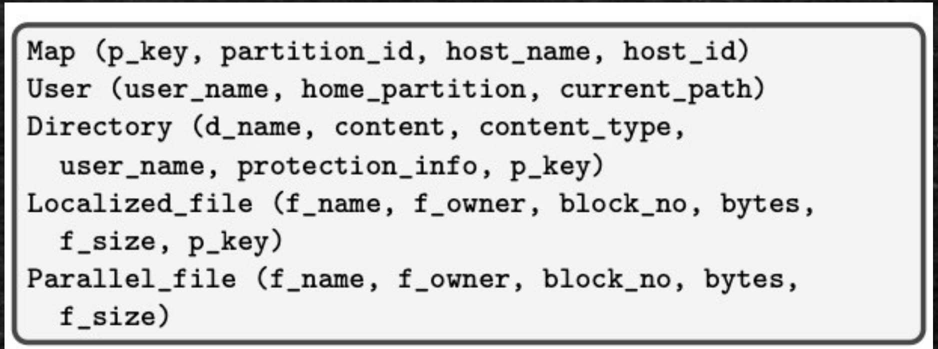


```
Message(sender_id, receiver_id, message_id, data)
```

=> and note that all of this already comes with transaction guarantees. because you have a modern transaction database underneath all of this

MESSAGES

The basic schema of how try to build a traditional file system in this operating system



```
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 9: Filesystem tables for data and metadata.

So, see that this has a very different flavor of how to build system logic on top of the primitives, instead of writing code for everything.

This very database and schema oriented way of doing things where write SQL Queries, inserts and updates to query our state, manage all state and update all the state.

MESSAGES

PROTYPE

I. DBOS - straw (very simple basic prototype)
II. DBOS - wood
III. DBOS - brick

PROTYPE

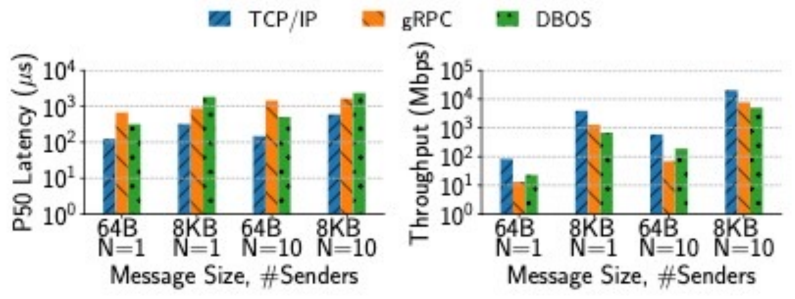
DBOS – straw

Base Layer is Linux on top of the hardware instead of a micro kernel

Data Base Layer is VoltDB

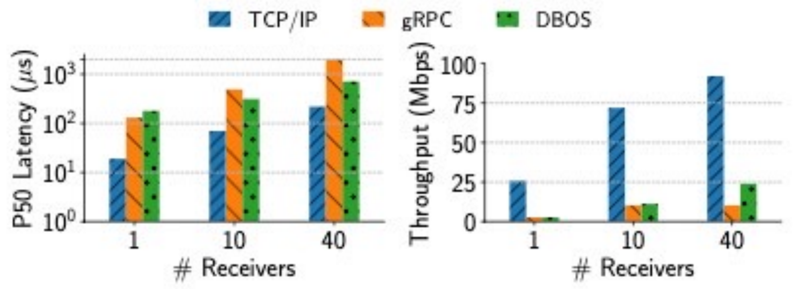
Against a more traditional operating system like Linux with a well-known RPC mechanism like gRPC on top of it.

RESULT



(a) Median latency (b) Throughput

Figure 7: Performance of ping^{20}-pong^{20} benchmark.



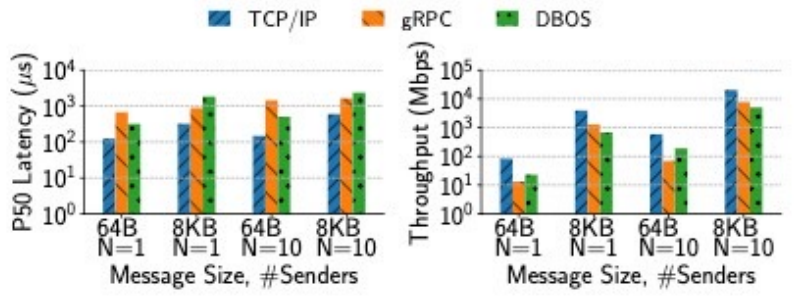
(a) Median latency (b) Throughput

Figure 8: Performance of multicasting benchmark.

DBOS outperforms gRPC by up to 2.7 times for small messages, while it underperforms by 48% for 8KB messages.

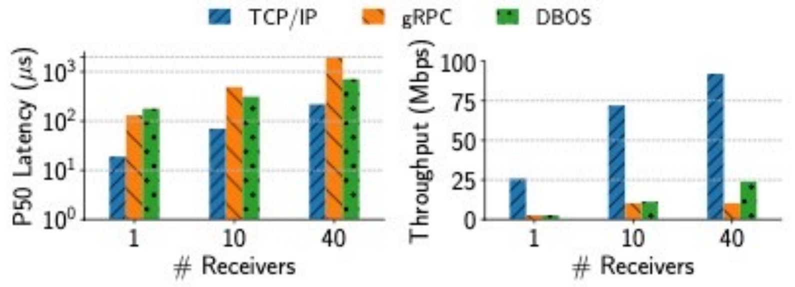
The difference between TCP/IP and DBOS has narrowed somewhat, but is still considerable.

RESULT



(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.

The small performance difference between DBOS and gRPC is impressive when consider that the DBOS messaging scheme is implemented in a few lines of SQL code running on an unmodified DBMS, whereas gRPC is a specialised communication framework developed by many engineers over many years.

But that's still really encouraging because this is a very unoptimised prototype.
So, this still shows that if optimise it and squeeze more performance out of it, this could be very very competitive with a more traditionally optimised operating system

PROTOTYPE

CONCLUSION

- A new operating system architecture that uses a modern database as the basic building block
- The claim is that this will drastically simplify the very complex state management that most modern operating systems have to manage with their large distributed applications on top of it
- Like this very different way of writing system code with SQL selects and inserts and updates that takes advantage of modern database features like transactional updates and so on to simplify the creation of all these applications