TODAY’S AGENDA

Autonomous DBMS History
Self-Driving DBMSs
Learned Components
MOTIVATION

Personnel is $\sim 50\%$ of the TOC of a DBMS.

Average DBA Salary (2017): $\$89,050$

The scale and complexity of DBMS installations have surpassed humans.

Source: [https://www.bls.gov/oes/current/oes151141.htm](https://www.bls.gov/oes/current/oes151141.htm)

Source: [https://www.highbeam.com/doc/1P3-1149052351.html](https://www.highbeam.com/doc/1P3-1149052351.html)
SELF-ADAPTIVE DATABASES (1970s-1990s)

Index Selection
Partitioning / Sharding Keys
Data Placement
SELF-ADAPTIVE DATABASES (1970s-1990s)

Admin

Tuning Algorithm

SELECT * FROM A JOIN B
ON A.ID = B.ID
WHERE A.VAL > 123
AND B.NAME LIKE 'XY%'

A.ID
A.VAL
B.ID
B.NAME

+100
+200
+50
SELF-ADAPTIVE DATABASES (1970s-1990s)

Admin

SELECT * FROM A JOIN B
ON A.ID = B.ID
WHERE A.VAL > 123
AND B.NAME LIKE 'XY%'

A.ID  B.ID
A.VAL  B.NAME

Tuning Algorithm

+100  +200  +50
SELF-ADAPTIVE DATABASES (1970s - 1990s)

**SQL Example**

```sql
SELECT * FROM A JOIN B ON A.ID = B.ID WHERE A.VAL > 123 AND B.NAME LIKE 'XY%'
```

**Algorithm**

- Admin
- Tuning Algorithm

SIGMOD 1976
SELF-TUNING DATABASES (1990s-2000s)

Admin

Tuning Algorithm

SELECT * FROM A JOIN B
ON A.ID = B.ID
WHERE A.VAL > 123
AND B.NAME LIKE 'XY%'

A.ID
A.VAL
B.ID
B.NAME

Optimizer
Cost Model

Microsoft
AutoAdmin
ABSTRACT

In this paper, we discuss advances in self-tuning database systems especially in the last decade, leading to new opportunities in the database community. We propose that self-tuning databases can be achieved by optimizing database designs in response to their environments. This approach has the potential to significantly improve the efficiency and performance of database systems.

1. HISTORY OF AUTOADMIN PROJECT

The AutoAdmin project was started in 1998 by J. roller and colleagues at the University of U. D. The project was initially focused on developing a tool that could automatically optimize database designs. The tool was designed to analyze the performance of database designs and automatically adjust their configurations to improve performance. The AutoAdmin project was successful in providing a tool that could automatically optimize database designs.

2. AN INTRODUCTION TO PHYSICAL DATABASE DESIGN

2.1 Importance of Physical Design

Physical design refers to the process of designing the structure and layout of a database. Physical design is concerned with the selection and assignment of physical characteristics to database objects. These characteristics include attributes such as data type, storage structure, and physical organization. Physical design is important because it affects the performance and efficiency of a database system.

2.2 State of the Art in 1997

In 1997, the state of the art in physical design was relatively immature. However, there was a growing interest in developing automated tools to assist in the process of physical design. Some of the early tools were experimental in nature and were not widely used in practice. Nonetheless, the field was advancing, and there was a need for more automated tools to improve the efficiency and performance of database systems.

3. SELECT WHERE AND TUNING DATABASES (1990s - 2000s)

In the 1990s and early 2000s, there was a significant amount of research and development in the area of self-tuning databases. This research focused on developing tools and techniques that could automatically optimize database designs in response to their environments. The AutoAdmin project was a key contributor to this field, and its success demonstrated the potential for self-tuning databases.

Copyright 2007 VLDB Endowment. This paper is an author draft of the paper "Self-Tuning Database Systems: A Decade of Progress" from the paper "Self-Tuning Database Systems: A Decade of Progress". It is authored by Arun Choudhary, Microsoft Research.
SELF-TUNING DATABASES (1990s-2000s)

Number of Configuration Knobs Per Release

Number of Knobs

Source: Dana Van Aken
CLOUD-MANAGED DATABASES (2010S)

Initial Placement
Tenant Migration
CLOUD-MANAGED DATABASES (2010S)
CLOUD-MANAGED DATABASES (2010S)
OBSERVATION

People have been working on autonomous database systems for 45 years.

Why is this previous work insufficient?
PREVIOUS WORK

Problem #1: Human Judgements
→ User has to make final decision on whether to apply recommendations.

Problem #2: Reactionary Measures
→ Can only solve previous problems. Cannot anticipate upcoming usage trends / issues.

Problem #3: No Transfer Learning
→ Tunes each DBMS instance in isolation. Cannot apply knowledge learned about one DBMS to another.
**OBSERVATION**

Just like there are different levels of autonomy in cars, there are different levels for databases.

→ SAE (J3016) Automation Levels

We need to reason about the autonomous systems to understand their capabilities and limitations.

→ This will help us reason about how much a human needs to be involved in its administration.
AUTONOMOUS DBMS TAXONOMY

System only does what humans tell it to do.  \rightarrow \textbf{Level \#0: Manual}
AUTONOMOUS DBMS TAXONOMY

Recommendation tools that suggest improvements. Human makes final decisions.

Level #0: Manual
Level #1: Assistant
AUTONOMOUS DBMS TAXONOMY

Level #0: Manual
Level #1: Assistant
Level #2: Mixed Management

DBMS and humans work together to manage the system. Human guides the process.
AUTONOMOUS DBMS TAXONOMY

Level #0: Manual
Level #1: Assistant
Level #2: Mixed Management
Level #3: Local Optimizations

Subsystems can adapt without human guidance. No higher-level coordination.
AUTONOMOUS DBMS TAXONOMY

Level #0: Manual
Level #1: Assistant
Level #2: Mixed Management
Level #3: Local Optimizations
Level #4: Direct Optimizations

Human only provides high-level direction + hints.
System can identify when it needs to ask humans for help.
AUTONOMOUS DBMS TAXONOMY

Level #0: Manual
Level #1: Assistant
Level #2: Mixed Management
Level #3: Local Optimizations
Level #4: Direct Optimizations
Level #5: Self-Driving
SELF-DRIVING DATABASE

A DBMS that can deploy, configure, and tune itself automatically without any human intervention.

→ Select actions to improve some objective function (e.g., throughput, latency, cost).
→ Choose when to apply an action.
→ Learn from these actions and refine future decision making processes.
ARCHITECTURE OVERVIEW

- SQL Statements
- Internal Metrics
- Workload Forecasts
- Component Models

Deploy & Observe

Modeling

Search & Planning

Actions

Where to Deploy?  When to Deploy?  How to Deploy?  Why?
SELF-DRIVING ENGINEERING

Environment Observations
→ How the DBMS collects training data.

Action Meta-Data
→ How the DBMS implements and exposes methods for controlling and modifying the system's configuration.

Action Engineering
→ How the DBMS deploys actions either for training or optimization.
ENVIRONMENT OBSERVATIONS

Logical Workload History
→ SQL queries with their execution context.
→ Need to compress to reduce storage size.

Runtime Metrics
→ Internal measurements about the DBMS's runtime behavior.

Database Contents
→ Succinct representation/encoding of the database tables.
SUB-COMPONENT METRICS

If the DBMS has sub-components that are tunable, then it must expose separate metrics for those components.

Bad Example: RocksDB
SUB-COMPONENT METRICS

RocksDB Column Family Knobs

```java
rockbdb_override_cf_options="
  cf_link_pk={prefix_extractor=capped:20}
```
# SUB-COMPONENT METRICS

## RocksDB Column Family Knobs

```java
rocksdb_override_cf_options=
    cf_link_pk={prefix_extractor=capped:20}
```

## Column Family Metrics

```sql
mysql> SELECT * FROM INFORMATION_SCHEMA.ROCKSDB_CFSTAT;
```

<table>
<thead>
<tr>
<th>CF_NAME</th>
<th>METRIC_NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>COMPACTION_PENDING</td>
<td>1</td>
</tr>
<tr>
<td>default</td>
<td>CUR_SIZE_ACTIVE_MEM_TABLE</td>
<td>21672</td>
</tr>
<tr>
<td>default</td>
<td>CUR_SIZE_ALL_MEM_TABLES</td>
<td>21672</td>
</tr>
<tr>
<td>default</td>
<td>MEM_TABLE_FLUSH_PENDING</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>NON_BLOCK_CACHE_SST_MEM_USAGE</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>NUM_ENTRIES_ACTIVE_MEM_TABLE</td>
<td>18</td>
</tr>
<tr>
<td>default</td>
<td>NUM_ENTRIES_IMM_MEM_TABLES</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>NUM_IMMUTABLE_MEM_TABLE</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>NUM_LIVE_VERSIONS</td>
<td>2</td>
</tr>
</tbody>
</table>

**Missing:**
- Reads
- Writes
SUB-COMPONENT METRICS

RocksDB Column Family Knobs

```
rocksdb_override_cf_options=
  cf_link_pk={prefix_extractor=capped:20}
```

Global Metrics

```
mysql> SHOW GLOBAL STATUS;
+---------------------------+-----------------------+
| METRIC_NAME               | VALUE                 |
+---------------------------+-----------------------+
| ABORTED_CLIENTS           | 0                     |
| ROCKSDB_BLOCK_CACHE_BYTES_READ | 295700537             |
| ROCKSDB_BLOCK_CACHE_BYTES_WRITE | 709562185             |
| ROCKSDB_BLOCK_CACHE_DATA_HIT | 64184                 |
| ROCKSDB_BLOCK_CACHE_DATA_MISS | 1001083               |
| ROCKSDB_BYTES_READ        | 5573794               |
| ROCKSDB_BYTES_WRITTEN     | 5817440               |
| ROCKSDB_FLUSH_WRITE_BYTES | 2906847               |
| UPTIME_SINCE_FLUSH_STATUS | 5996                  |
+---------------------------+-----------------------+
```
ACTION META-DATA

Configuration Knobs
→ Untunable flags
→ Value ranges

Dependencies
→ No hidden dependencies
→ Dynamic actions (i.e., an action creates new actions).
UNTUNABLE KNOBS

Anything that requires a human value judgement should be marked as off-limits to autonomous components.

→ File Paths
→ Network Addresses
→ Durability / Isolation Levels
KNOB HINTS

The autonomous components need hints about how to change a knob.
→ Min/max ranges.
→ Separate knobs to enable/disable a feature.
→ Non-uniform deltas.
KNOB HINTS

The autonomous components need hints about how to change a knob.

→ Min/max ranges.
→ Separate knobs to enable/disable a feature.
→ Non-uniform deltas.
ACTION ENGINEERING

No Downtime
Notifications
Replicated Training
The DBMS must be able to deploy any action without incurring downtime.

→ Restart vs. Unavailability

Without this, the system has to include the downtime in its cost model estimations.

→ Bad Example: MySQL Log File Size
NOTIFICATIONS

Provide a notification to indicate when an action starts and when it completes.
→ Need to know whether degradation is due to deployment or bad decision.

Harder for changes that can be used before the action completes.
ML models need lots of training data. But getting this data is expensive in a DBMS.
→ We don't want to slow down a production DBMS.
→ Building a simulator for the DBMS is too hard.

Ongoing Research: How to use the DBMS's replicas to explore configurations and train its models.
REPLICATED TRAINING

Master

Replica

Replica

Self-Driving Manager

Actions

Actions
REPLICATED TRAINING

Master

Replica

Self-Driving Manager

Replica

Actions

Actions

Actions

Actions
REPLICATED TRAINING

Master ➔ Replica

Replica ➔ Replica

Self-Driving Manager

Wait Time
Revert Actions

Actions
"""
REPLICATED TRAINING

- **Master**
  - Reads
  - Writes

- **Replica**
  - Writes

- **App Server**
  - Reads
  - Writes

- **Self-Driving Manager**
REPLICATED TRAINING

Master

App Server

???

SQL Statements

Physical Log

???

Reads

Writes

Replica

Replica

Self-Driving Manager
REPLICATED TRAINING

Master

SQL Statements
Physical Log

Replica

Component Models

Replica

Component Models

App Server

???

Reads
Writes

Self-Driving Manager
REPLICATED TRAINING

- Master
- Replica
- Replica

- Reads
- SQL Statements
- Physical Log

- Reads
- Writes

App Server

Component Models

Self-Driving Manager
REPLICATED TRAINING

Master

App Server

Actions

Component Models

Self-Driving Manager

Replica

Component Models

Replica

Component Models
REPLICATED TRAINING

Master

Replica

Component Models

Component Models

Actions

App Server

Self-Driving Manager
ORACLE SELF-DRIVING DBMS

September 2017

No Human Labor – Half the Cost
No Human Error – 100x More Reliable

Self-Driving Database Management Systems

Andrew Pavlo, Gustavo Angulo, Jey Aslam, Haldun Can, Jialu Lin, Lin Ma, Paulkumar Manon, Tel C. Myers, Matthew Perillo, Ian Qin, Siddharth Srikant, Arvind Tenneti, Sige You, Donald Yau, Jia Yu, Yang Zheng, Peng Zheng, Yuan Zhu

Carnegie Mellon University, National University of Singapore

ABSTRACT

In the last two decades, both database and cloud vendors have built advanced tools to assist database administrators (DBAs) in various aspects of system tuning and physical design. Over the last few years, however, it has become clear that the need for DBA to make manual tuning decisions has increased. This phenomenon is driven by several factors: the scale of applications deployed on modern cloud and data center platforms, the increase in the number of database workloads, and the complexity and performance variability of modern cloud and data center clusters. These factors have resulted in the need for tools that can automate the tuning process and improve database performance.

As the need for automation increases, it is becoming clear that a full-fledged self-driving database system (DBMS) is required. Oracle’s Autonomous Database (ADB) is an example of such a system that has the ability to automatically detect and resolve database performance issues. The self-driving nature of this system allows developers to focus on writing high-quality applications without the burden of managing database infrastructure. This system demonstrates the self-driving capabilities of a DBMS and provides insights into the future of database management systems.

1. INTRODUCTION

The idea of using a DBMS to ensure the efficiency of database management is not the avant-garde mind of the modern world. Oracle has been around for over 30 years, offering tools to help manage, query, and analyze data. More recently, Oracle has introduced autonomous database, a cloud database service that can manage its own infrastructure and automate many of the manual tasks associated with running a database.

The Oracle Autonomous Database (ADB) is a fully managed cloud database service that automatically performs routine maintenance and scale operations. Oracle has made a number of investments in this area, including the acquisition of two companies that make autonomous database solutions.

The ADB is designed to provide a simple and scalable infrastructure solution for database applications. It provides automatic scaling, backup, and recovery, as well as automated database patching and monitoring.

This paper presents several key findings from this self-driving approach to database management. The self-driving nature of this DBMS allows for automation of routine maintenance and scale operations. The performance of the ADB is compared to other cloud databases, demonstrating the benefits of self-driving database management.

2. PROBLEM OVERVIEW

The first challenge is to self-drive DBMSs to be understood as an application framework. The more basic level is to characterize the DBMS as a component of a larger system. If the DBMS is designed to be understood as an application framework, then it can make decisions about how to optimize the database. For example, if it is a CPU, then the DBMS should make decisions about which thread to run on the CPU. If it is a SQL, then the DBMS should make decisions about which query to run on the SQL.
ORACLE SELF-DRIVING DBMS

- Automatic Patching
- Automatic Indexing
- Automatic Recovery
- Automatic Scaling
- Automatic Query Tuning
ORACLE SELF-DRIVING DBMS

- Automatic Patching
- Automatic Indexing
- Automatic Recovery
- Automatic Scaling
- Automatic Query Tuning

Reactionary Measures
No Transfer Learning
ORACLE SELF-DRIVING DBMS

Automatic Patching
Automatic Indexing
Automatic Recovery
Automatic Scaling
Automatic Query Tuning

Reactionary Measures
No Transfer Learning

September 2017
OBSERVATION

There are many places in the DBMS that use human-engineered components to make decisions about the behavior of the system.

→ Optimizer Cost Models
→ Compression Algorithms
→ Data Structures
→ Scheduling Policies

What if the DBMS could "learn" these policies based on the data.
LEARNED COMPONENTS

Replace DBMS components with ML models trained at runtime.
PARTING THOUGHTS

True autonomous DBMSs are achievable in the next decade.

You should think about how each new feature can be controlled by a machine.
NEXT CLASS

SAP HANA Guest Lecture