Vamsidhar Thummala

Joint work with Shivnath Babu, Songyun Duan, Nedyalkov Borisov, and Herodotous Herodotou
Duke University
20th May 2009

AUTOMATED EXPERIMENT-DRIVEN MANAGEMENT OF (DATABASE) SYSTEMS

Claim:

- "Current" techniques for managing systems have limitations
  - Not adequate for end-to-end systems management
- Closing the loop
  - Experiment-driven management of systems

An example scenario

- A "CEO Query" does not meet the SLO
- Reason: Violates the response time objective
- Admin's observation: High disk activity
- Admin's dilemma: 
  - What corrective action should I take?
  - How to validate the impact of my action?

- Hardware-level changes
  - Add more DRAM
- OS-level changes
  - Increase memory/CPU cycles (VMM)
  - Increase swap space
- DB-level changes
  - Partition the data
  - Update database statistics
  - Change physical database design – indexes, schema, views
  - Tune the query/Manually change query plan
  - Change configuration parameters like buffer pool sizes, I/O daemons, and max connections

How to find the corrective action?

- Get more insight into the problem
  - Use domain knowledge
    - Admin's experience
  - Use *apriori* models if available
    - Fast prediction
    - Systems are complex
    - Hard to capture the behavior of the system *apriori*
  - Rely on "Empirical Analysis"
    - More accurate prediction
    - Time-consuming
    - Sometimes the only choice!

How Admins do Empirical Analysis

- Conduct an experiment run with a prospective setting (trial)
  - Pay some extra cost, get new information in return
- Learn from observations (error)
- Repeat until satisfactory solution is found

- Automating the above process is what we call

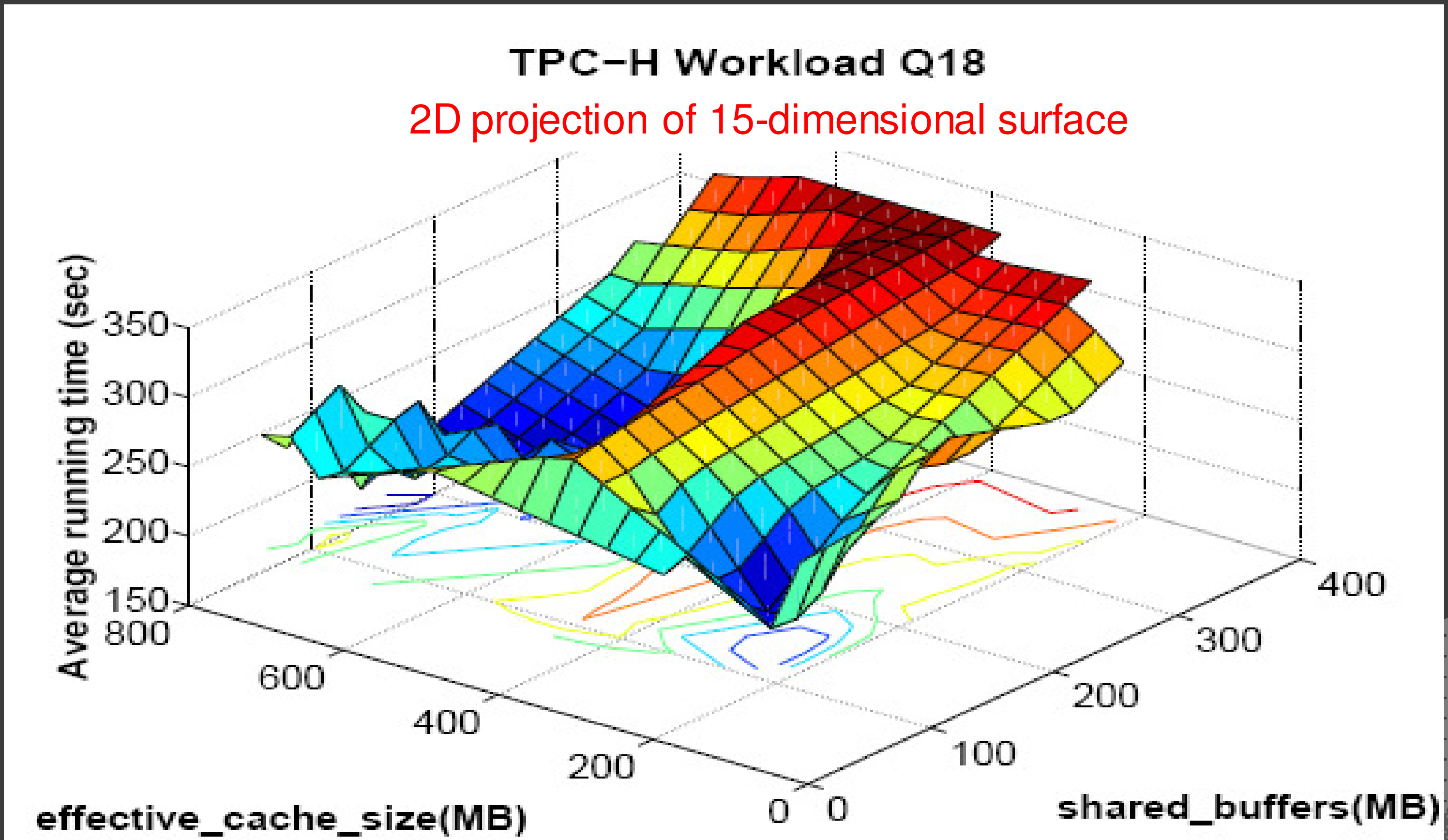
Experiment-driven Management

An example where experiment-driven management can be used

- Configuration parameter tuning
  - Database parameters (PostgreSQL-specific)
    - Memory distribution
      - `shared_buffers, work_mem`
    - I/O optimization
      - `fsync, checkpoint_segments, checkpoint_timeout`
    - Parallelism
      - `max_connections`
    - Optimizer's cost model
      - `effective_cache_size, random_page_cost, default_statistics_target, enable_indexscan`

Configuration parameter tuning

```
TPC-H Q18: Large Volume Customer Query
Data size: 4GB, Memory: 1GB
```



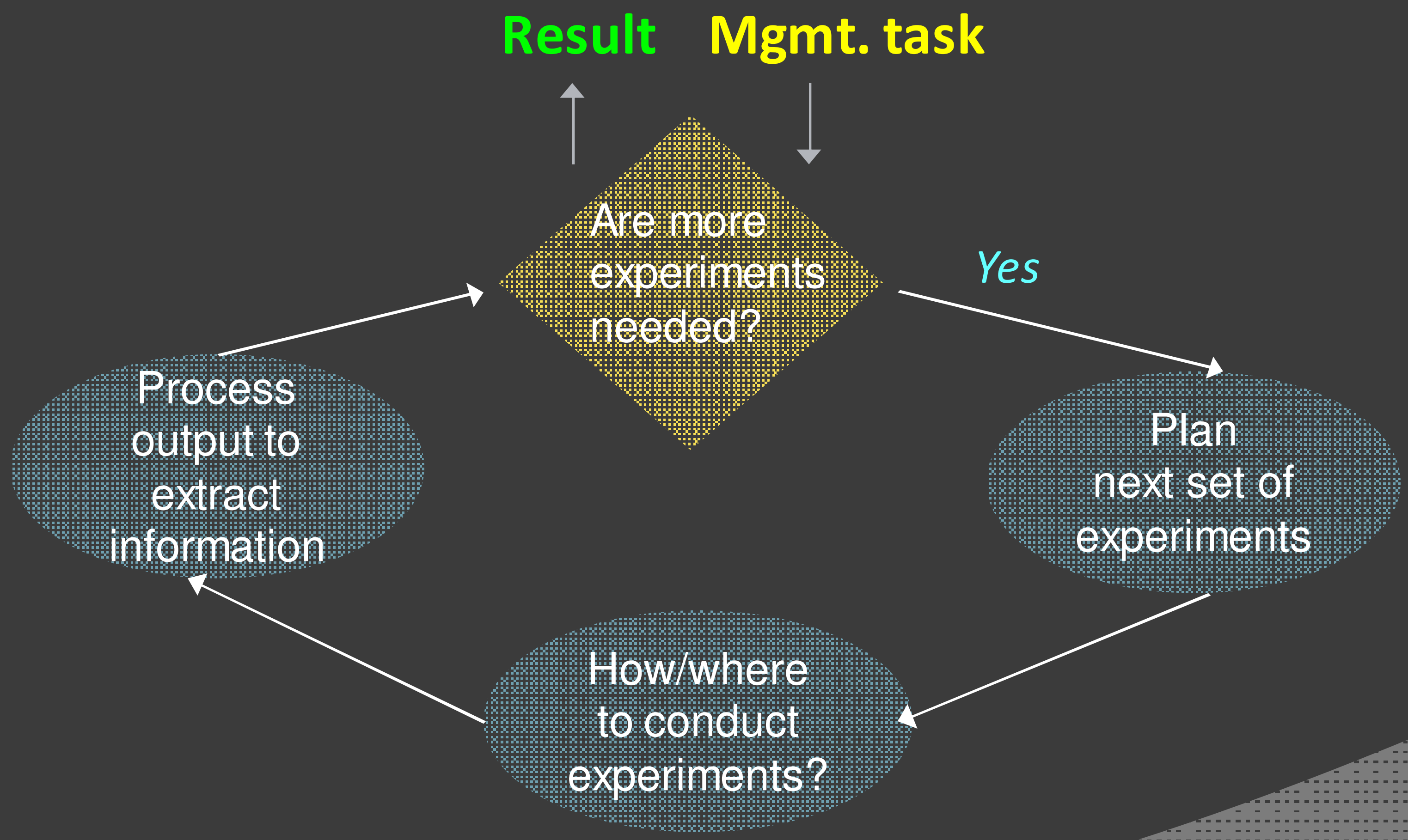
OS cache (prescriptive)

DB cache (dedicated)

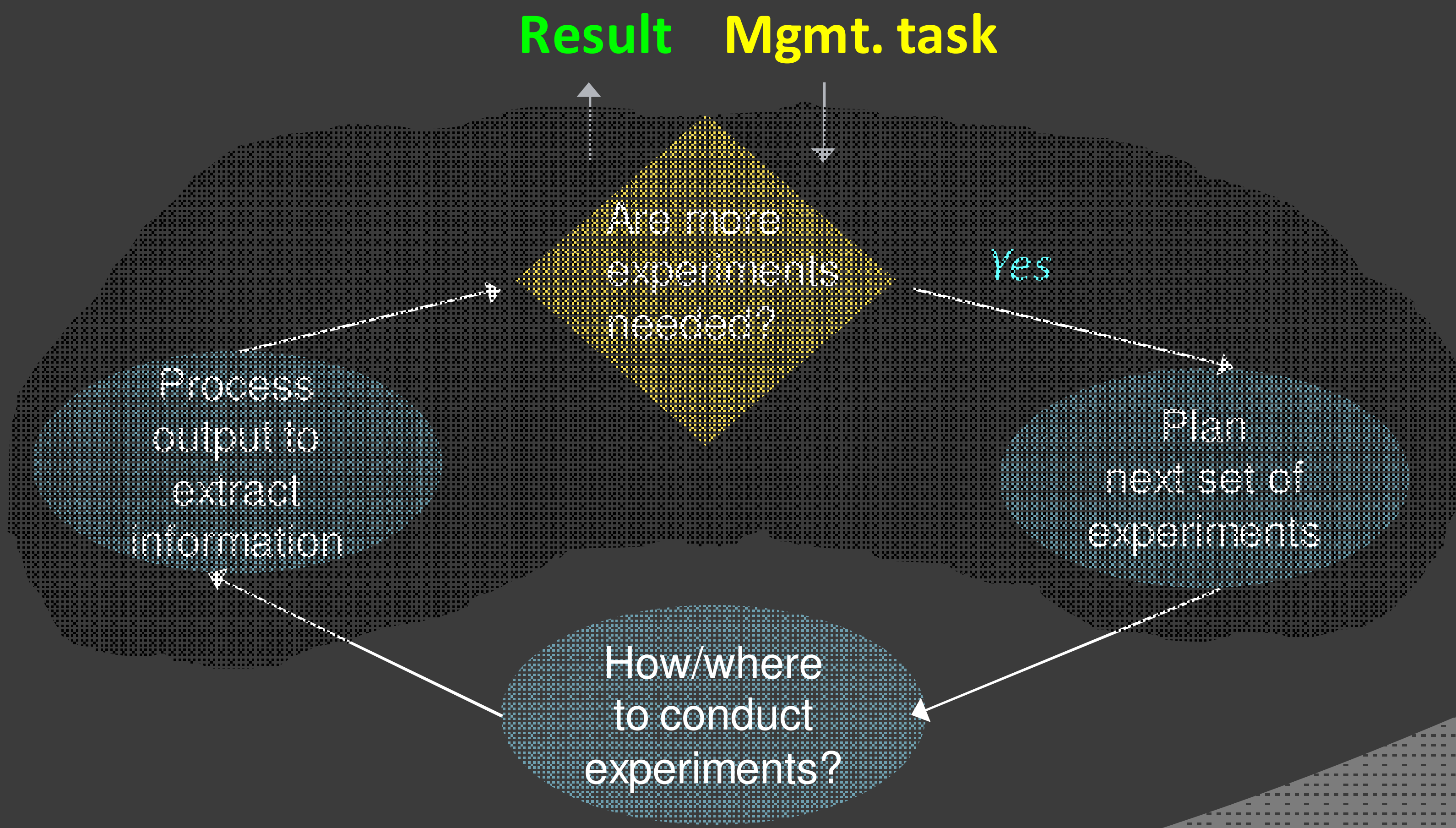
More examples where experiment-driven management can be used

- Configuration parameter tuning
- Problem diagnosis (troubleshooting), finding fixes, and validating the fixes
- Benchmarking
- Capacity planning
- Speculative execution
- Canary in server farm (James Hamilton, Amazon Web Services)

Workflow for Experiment-driven Management



Outline



Result
Mgmt. task
Are more experiments needed?
Yes
Process output to extract information
Plan next set of experiments
How/where to conduct experiments?

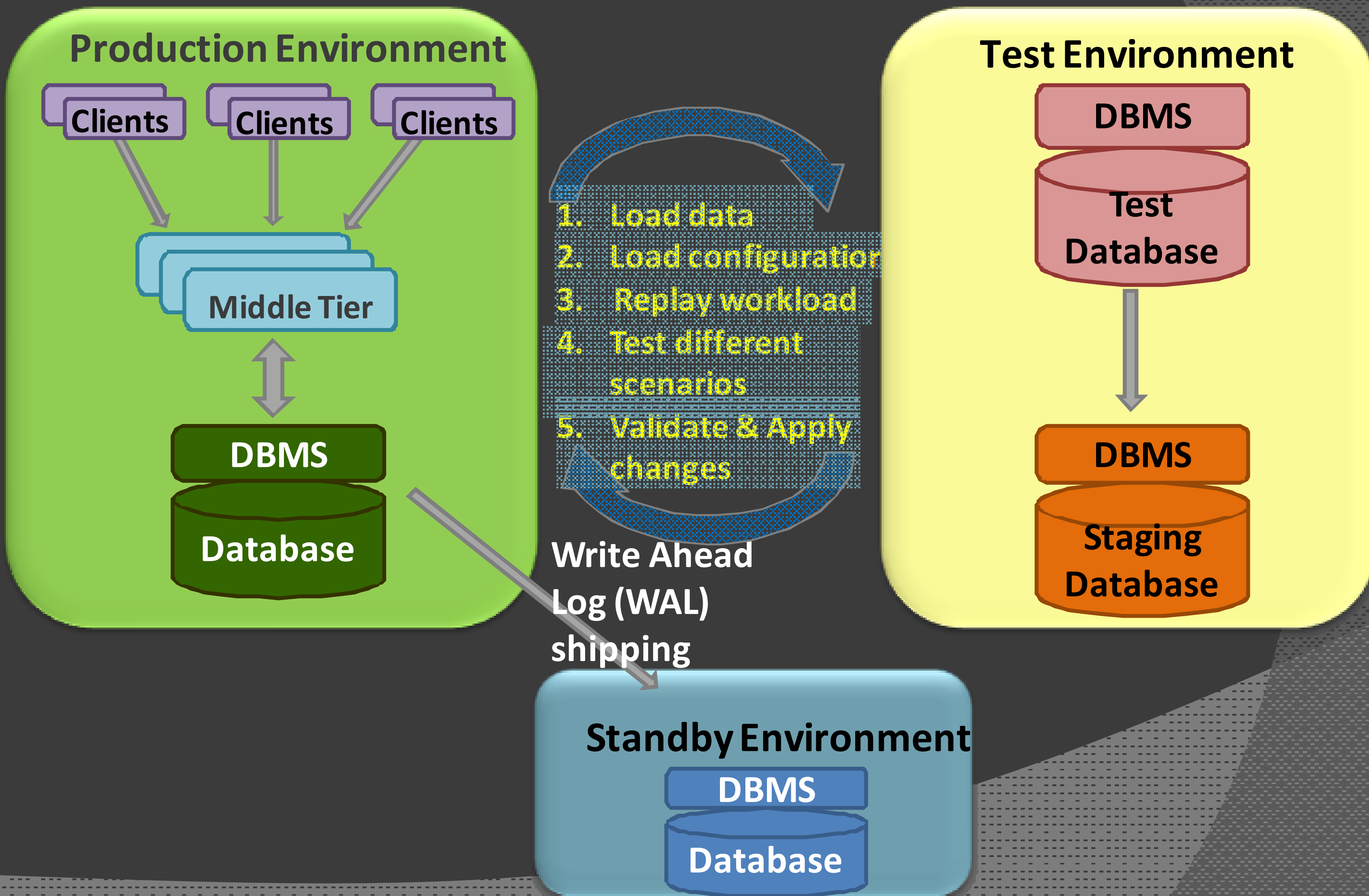
Challenges in setting up an experiment

- What is the right abstraction for an experiment?
- Ensuring representative workloads
  - Can be tuning task specific
    - Detecting deadlocks vs. performance tuning
- Ensuring representative data
  - Full copy vs. sampled data?

Where to conduct experiments in a 24X7 production environment?

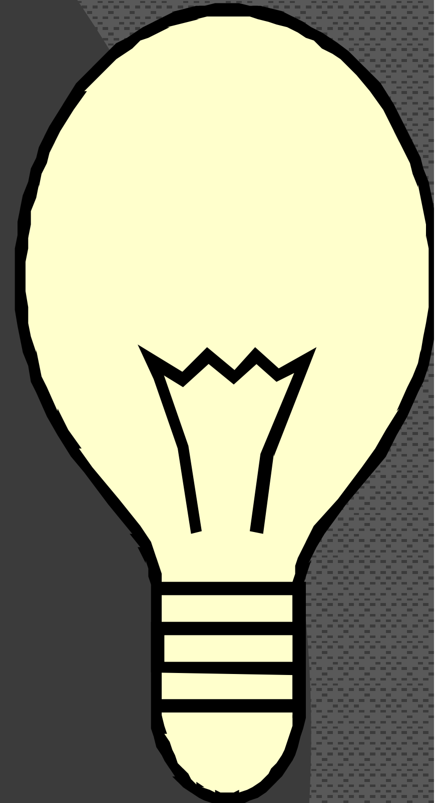
- Production system itself [USENIX'09, ACDC'09]
  - May impact user-facing workload
- Test system
  - Hard to replicate exact production settings
  - Manual set-up
- How and where to conduct experiments?
  - Without impacting user-facing workload
  - As close to production runs as possible

What do DB Administrators do today?



An idea

- How to conduct experiments?
  - Exploit underutilized resources
- Where to conduct experiments?
  - Production system, Standby system, Test system



❖Need mechanisms and policies to utilize idle resources efficiently
- ➢Mechanisms: Next slide
- ➢Policies: If CPU, memory, & disk utilization is below 10% for past 10 minutes, then resource X can be used for experiments

Mechanisms

Production Environment

Client

“Enterprises that have 99.999% availability have standby databases that are 99.999% idle”, Oracle DBA’s handbook

Test Environment

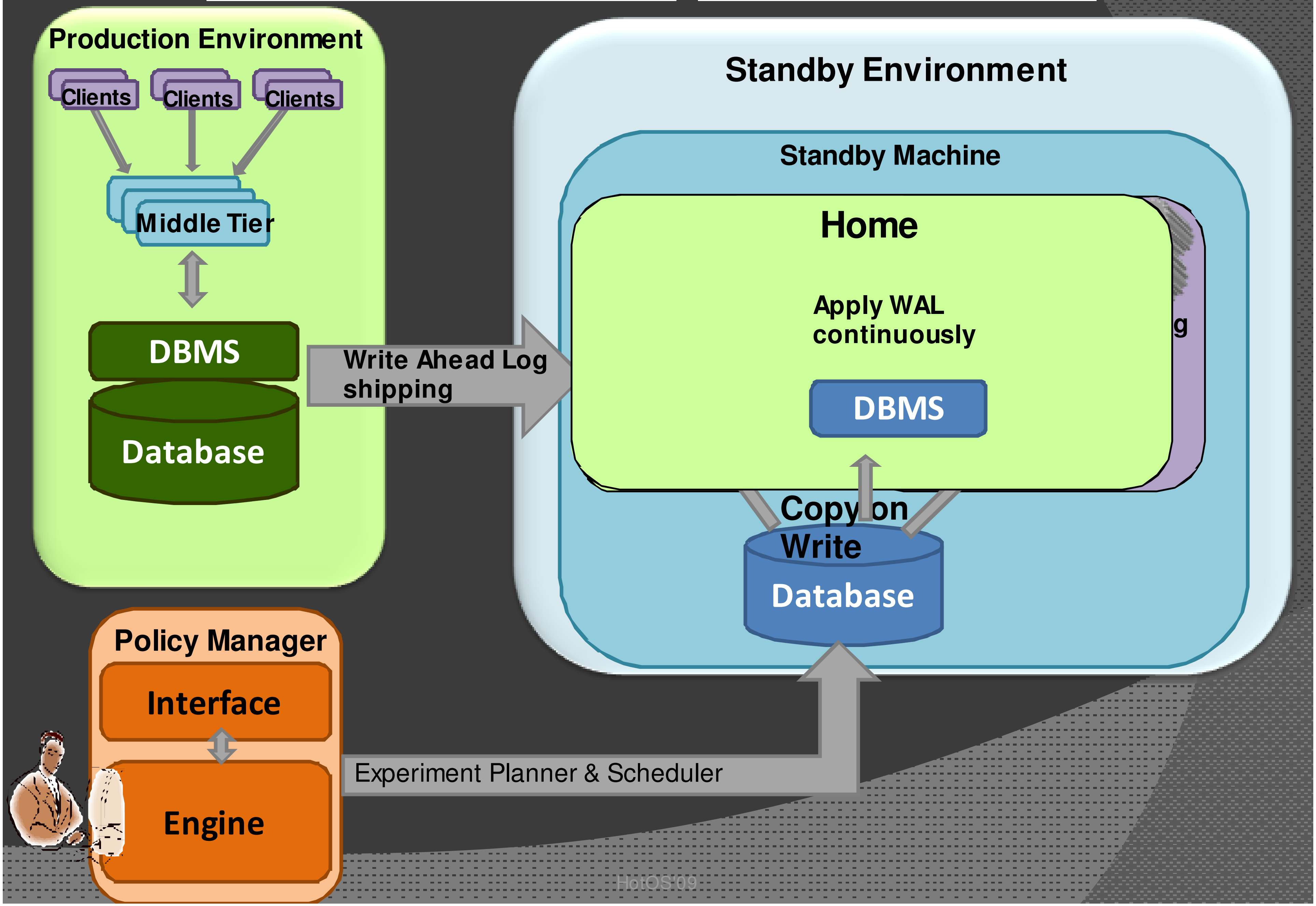
shipping

Standby Environment

DBMS

Database

Mechanisms: Workbench



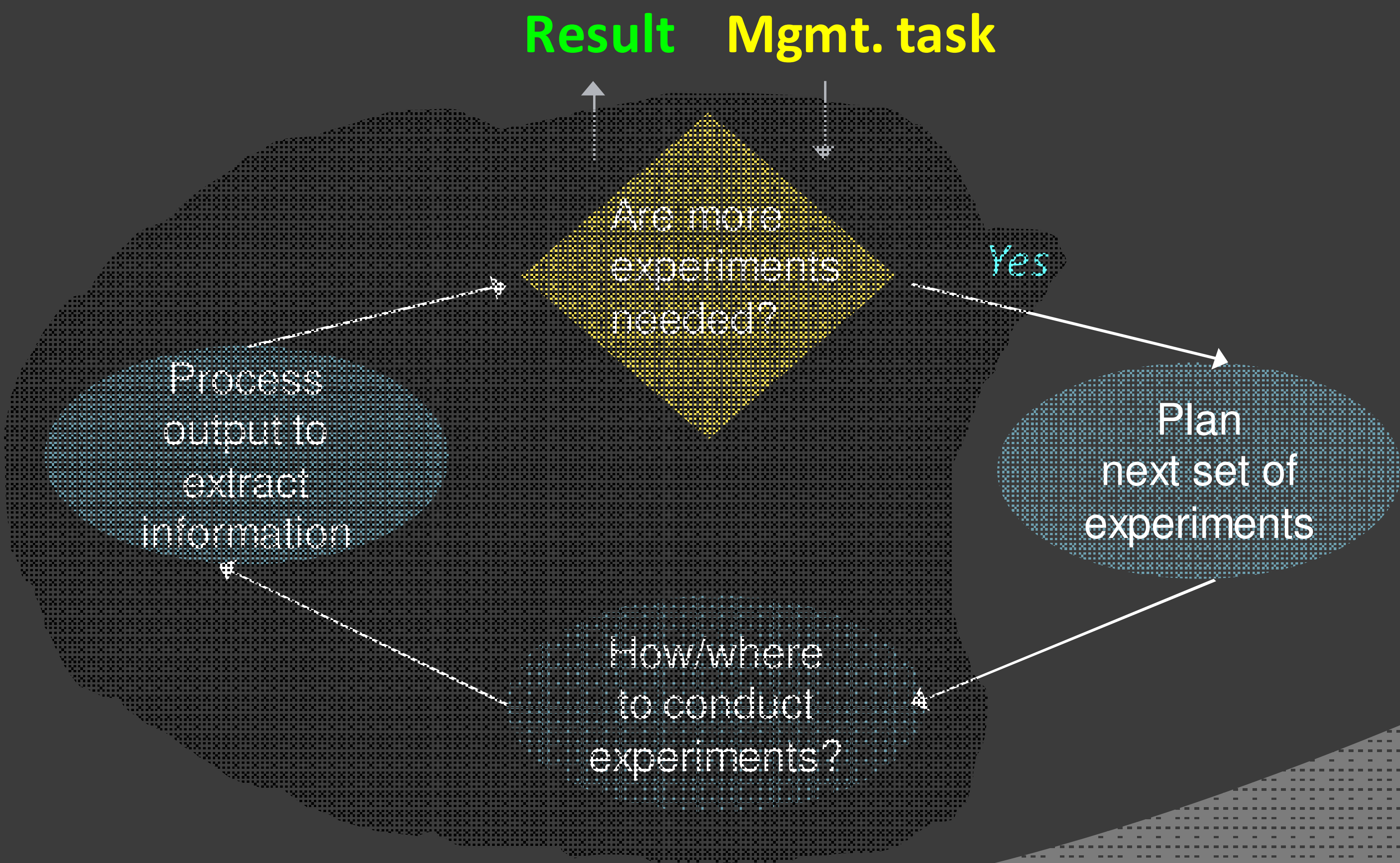
Workbench features

- Implemented using Solaris OS
  - Zones to isolate resources between home & garage containers
  - ZFS to create fast snapshots
  - Dtrace for resource monitoring

Overhead of workbench

Operation by workbench	Time (sec)	Description
Create Container	610	Create a new garage (one time process)
Clone Container	17	Clone a garage from already existing one
Boot Container	19	Boot garage from halt state
Halt Container	2	Stop garage and release resources
Reboot Container	2	Reboot the garage
Snapshot-R DB (5GB, 20GB)	7, 11	Create read-only snapshot of the database
Snapshot-RW DB (5GB, 20GB)	29, 62	Create read-write snapshot of database

Outline



Result
Mgmt. task
Are more experiments needed?
Yes
Plan next set of experiments
How/where to conduct experiments?
Process output to extract information

Which experiments to run?

- Gridding
- Random Sampling
- Simulated Annealing
- Space-filling Sampling
  - Latin Hypercube Sampling
  - *k*-Furthest First Sampling
- Design of Experiments (Statistics)
  - Plackett-Burman
  - Fractional Factorial
- Can we do better than above?

Our approach

- Adaptive Sampling

1 **Bootstrapping:**
Conduct initial set of experiments

2 **Sequential Sampling:**
Select NEXT experiment based on previous samples

Stopping Criteria:
Based on budget

Main idea:
1. Compute the utility of the experiment
2. Conduct experiment where utility is maximized
3. We used Gaussian Process for computing the utility

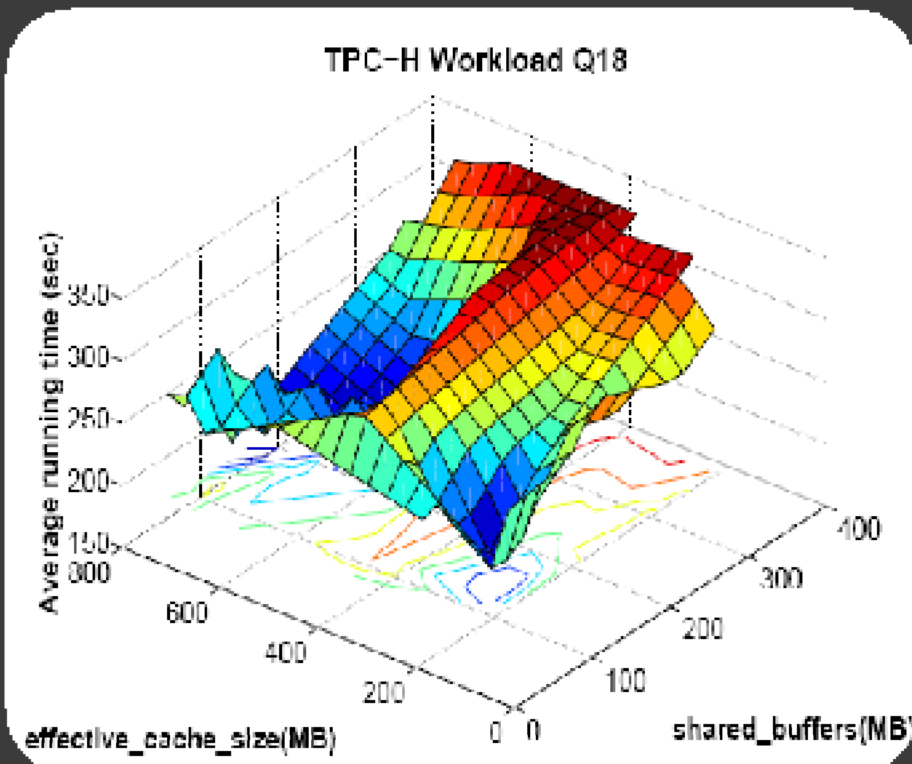
Results

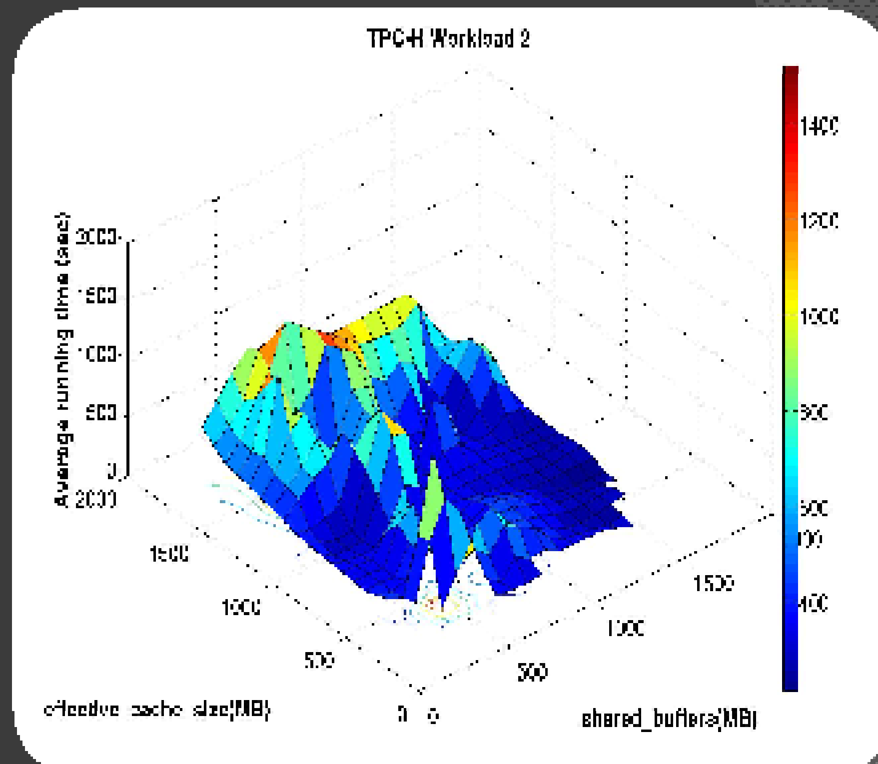
- Empirical Setting
  - PostgreSQL v8.2: Tuning up to 30 parameters
  - 3 Sun Solaris machines with 3 GB RAM, 1.8 GHz processor
  - Workloads
    - TPC-H benchmark
      - SF = 1 (1GB, total database size = 5GB)
      - SF = 10 (10GB, total database size = 20GB)
    - TPC-W benchmark
  - Synthetic response surfaces

Results on Real Response Surfaces

Simple Workload: W1-SF1
TPC-H Q18, Large Volume Customer Query

Complex Workload: W2-SF1
Random mix of 100 TPC-H Queries

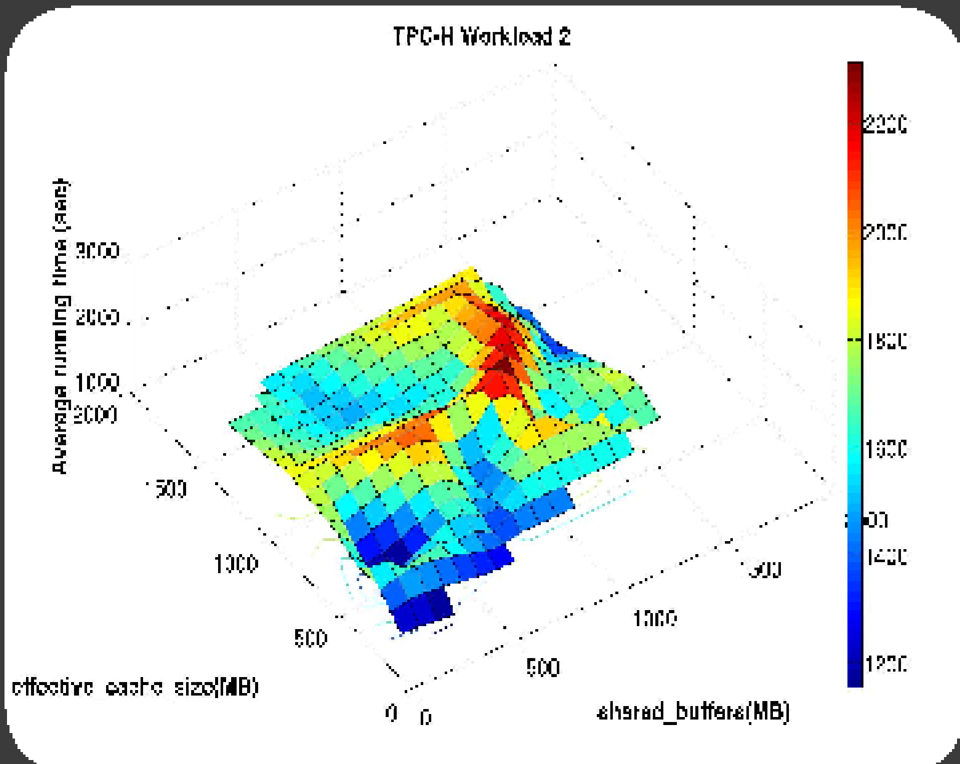


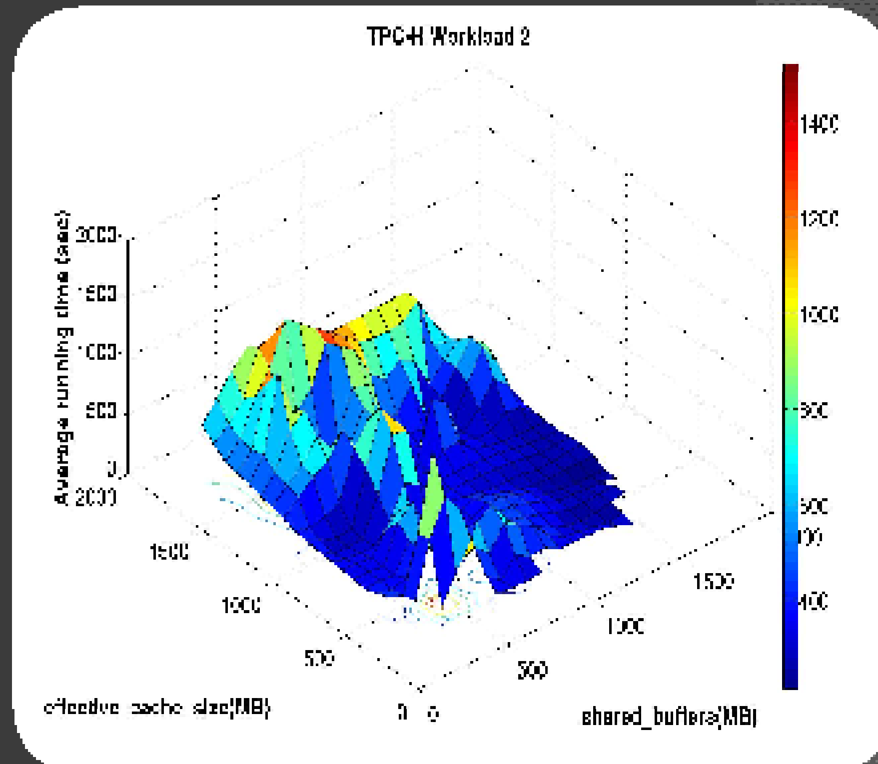


Results on Real Response Surfaces

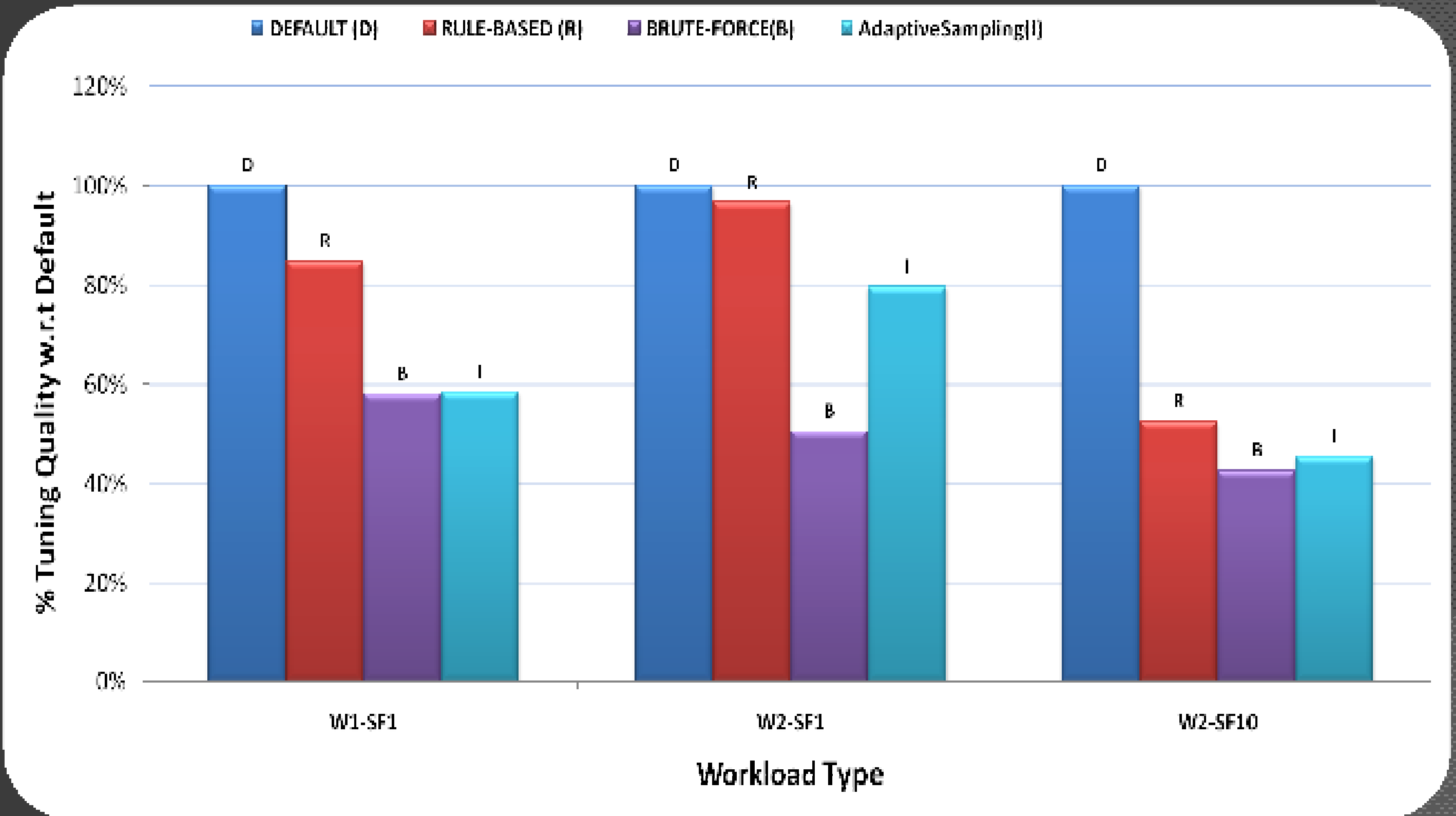
Complex Workload: W2-SF10
Random mix of 100 TPC-H Queries

Complex Workload: W2-SF1
Random mix of 100 TPC-H Queries





Comparison of Tuning Quality



Comparison of Tuning time

Cutoff time for each query : 90 minutes

	BruteForce	AdaptiveSampling
W1-SF1	8 hours	1.4 hours
W2-SF1	21.7 days	4.6 days
W2-SF10	68 days	14.8 days

- We further reduced the time using techniques
  - Workload compression
  - Database specific information

Conclusion

- Experiment-driven management is an essential part of system administration
  - Our premise: Experiments should be supported as *first-class* citizens in systems
  - Compliments existing approaches
- Experiments in the cloud – the time has come!

Q & A

- Thanks!