
# Enterprise Computing – Cloud Service Benchmarking

Prof. Stefan Tai

| | | |
|---|---|---|
| Introduction | Thu, Oct 20<br>Introduction | |
| Cloud Services | Thu, Oct 27 + Thu, Nov 3 + Wed, Nov 9<br><br>Web Engineering Fundamentals<br>Cloud Fundamentals<br>Microservices/DevOps | |
| Cloud Storage | Thu, Nov 17<br>Intro NoSQL | Thu, Nov 24<br>Systems 1 |
| | Thu, Dez 1<br>Systems 2 | |
| Cloud Service Benchmarking | Thu, Dez 8<br>Qualities, Benchmarking 1 | Thu, Dez 15<br>Benchmarking 2 |
| Assignment (6 weeks) | Thu, Dec 15<br>Kick-Off | Thu, Feb 9<br>End of Assignment |
| Wrap-Up | Thu, Feb 16<br>Q&A | |

# Cloud Service Benchmarking – Agenda

1. [Performance] Benchmarking of distributed systems and databases – The traditional view
2. [{Quality}] Benchmarking of cloud systems and services – New(er) thinking
3. Benchmarking in the cloud / using the cloud for benchmarking



ISEngineering
Information Systems Engineering

# Benchmarking, since the early 90s

…is about evaluating and comparing different systems or components according to specific characteristics such as performance, availability and security.

*Traditionally*, important criteria include:

[Jim Gray (Ed.), The Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, 1993.]

- Relevance: the benchmark result has to measure the performance of the typical operation within the problem domain
- Portability: it should be easy to implement on many different systems and architectures
- Scalability: it should be scalable to cover small and large systems
- Simplicity: the benchmarkl should be understandable to avoid lack of credibility

# Benchmarking, a 2009 perspective

Five key criteria of a 'good' benchmark:

[K. Huppler, The Art of Building a Good Benchmark. 1st TPCTC, 2009.]

- Relevance: the benchmark has to reflect something important
- Repeatable: the benchmark result can be reproduced by re-running the benchmark under similar conditions with the same result
- Fair & Portable: All systems compared can participate equally (portability, 'fair' design)
- Verifiable: There has to be confidence that documented results are real. This can, for example, be assured by reviewing results by external auditors
- Economical: The cost of running the benchmark should be feasible

# Benchmarking requirements, refined

1. General Requirements
   - Strong target audience
   - Relevant
   - Economical
   - Simple

2. Implementation Requirements
   - Fair and Portable
   - Repeatable
   - Realistic and Comprehensive
   - Configurable

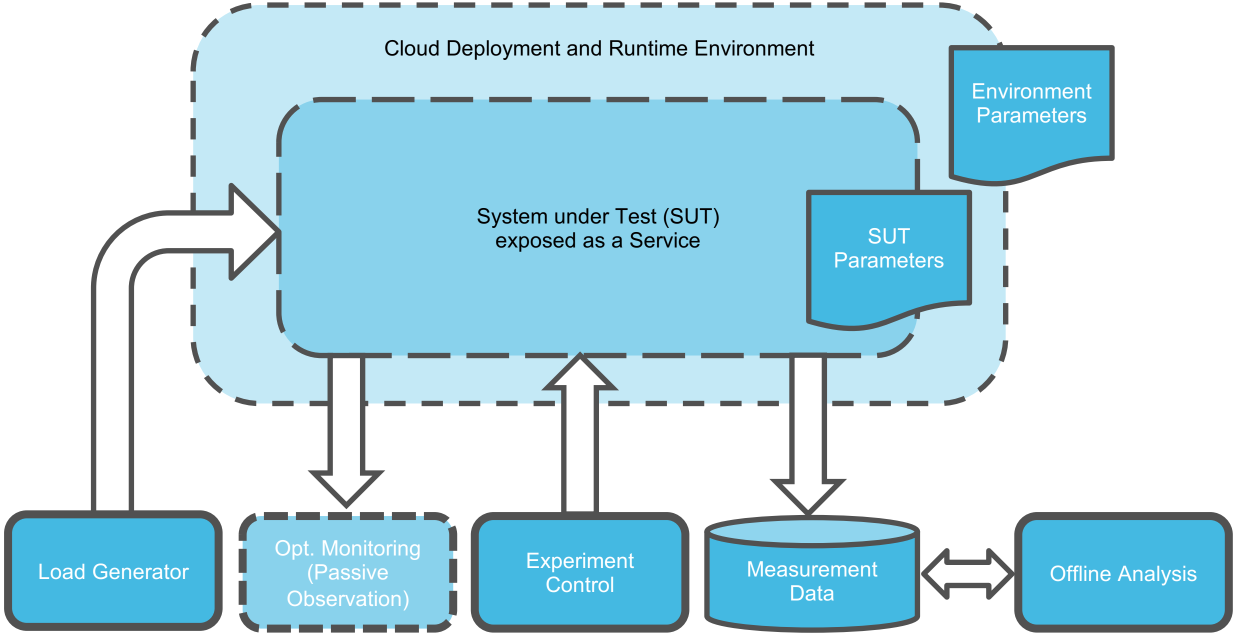
3. Workload Requirements
   - Representativeness
   - Scalable
   - Meaningful Metrics

Source: Folkerts et al. Benchmarking in the Cloud: What it Should, Can, and Cannot Be. TPCTC, 2012

# Some key questions to ask (still today)

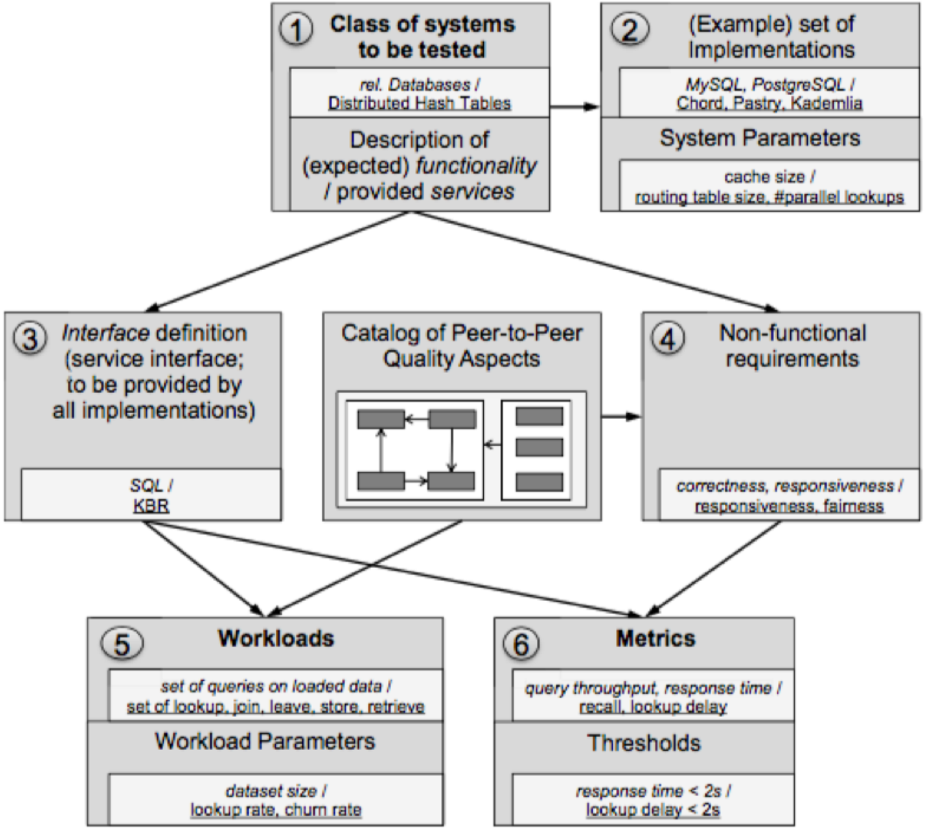
1. What quality are we interested in, what is "something important"? When is "something important"?
2. What is being benchmarked?
3. What are appropriate benchmarking experiments and conditions?
4. What makes results real and meaningful? How de we ensure this?
5. What is the cost of benchmarking?

# Cloud service benchmarking – Building blocks




Source: [BWT2017] D. Bermbach, E. Wittern, S. Tai. Cloud Service Benchmarking. Springer, 2017 (to appear)

# Benchmark Design (for databases and DHTs as examples)




Source: [Saller2013]

# Performance benchmarking

Probably the most advanced and standardized benchmarking methods and tools are in the area of performance benchmarking

*This tells us that performance is a critical quality, but it does not imply, of course, that other qualities are less important*

Key standardization organizations include SPEC (Standard Performance Evaluation Corporation, https://www.spec.org/) and the TPC (Transaction Processing Performance Council, http://www.tpc.org/)

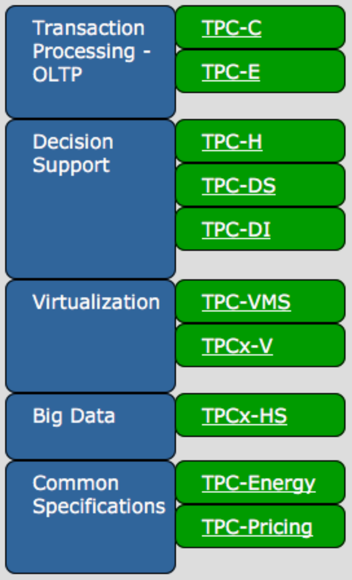
Known benchmarks include:

- SPEC CPU series for performance evaluation of CPUs
- SPEC HPC performance benchmarks
- Various (active and obsolete) TPC specifications (see next slides)

# TPC Benchmarks

…are the de-facto standard in the database area.

Current active benchmarks:




Obsolete benchmarks include:

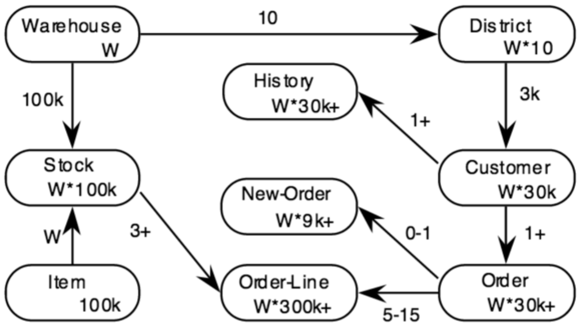
TPC-W, a transactional Web application benchmark (ca. 2002)

TPC-App, an application server and Web services benchmark (ca. 2005)

There is no active TPC benchmark for enterprise software systems today.

# Example: TPC-C

TPC-C measures the rate of common database transactions in an OLTP-type workload environment. TPC-C is measured in transactions per minute (tpmC).

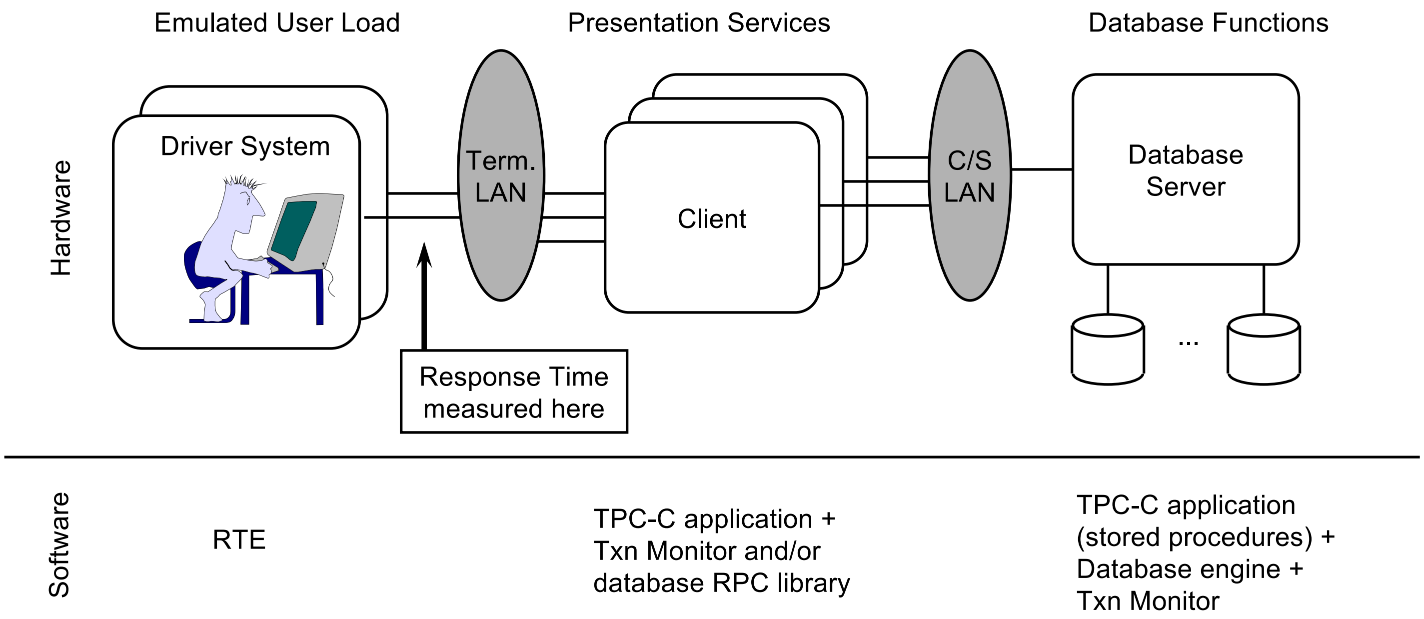



Source: www.tpc.org

“TPC-C simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. While the benchmark portrays the activity of a wholesale supplier, TPC-C is not limited to the activity of any particular business segment, but, rather represents any industry that must manage, sell, or distribute a product or service.“

# Conceptual TPC-C Configuration






Source: DeWitt, SIGMOD'97

# Sample TPC-C results

| Rank | Company | System | Performance (tpmC) | Price/tpmC | Watts/KtpmC | System Availability | Database | Operating System | TP Monitor | Date Submitted |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ORACLE | SPARC T5-8 Server | 8,552,523 | .55 USD | NR | 09/25/13 | Oracle 11g Release 2 Enterprise Edition with Oracle Partitioning | Oracle Solaris 11.1 | Oracle Tuxedo CFSR | 03/26/13 |
| 2 | IBM | IBM System x3650 M4 | 1,320,082 | .51 USD | NR | 02/25/13 | IBM DB2 ESE 9.7 | Red Hat Enterprise Linux 6.4 with KVM | Microsoft COM+ | 02/22/13 |
| 3 | SAP | Dell PowerEdge T620 | 112,890 | .19 USD | NR | 11/25/14 | SQL Anywhere 16 | Microsoft Windows 2012 Standard x64 | Microsoft COM+ | 11/25/14 |

'NR' in the Watts/Ktpmc column indicates that no energy data was reported for that benchmark.

Source: www.tpc.org

So, what does this tell us?

# Example: TPC-W

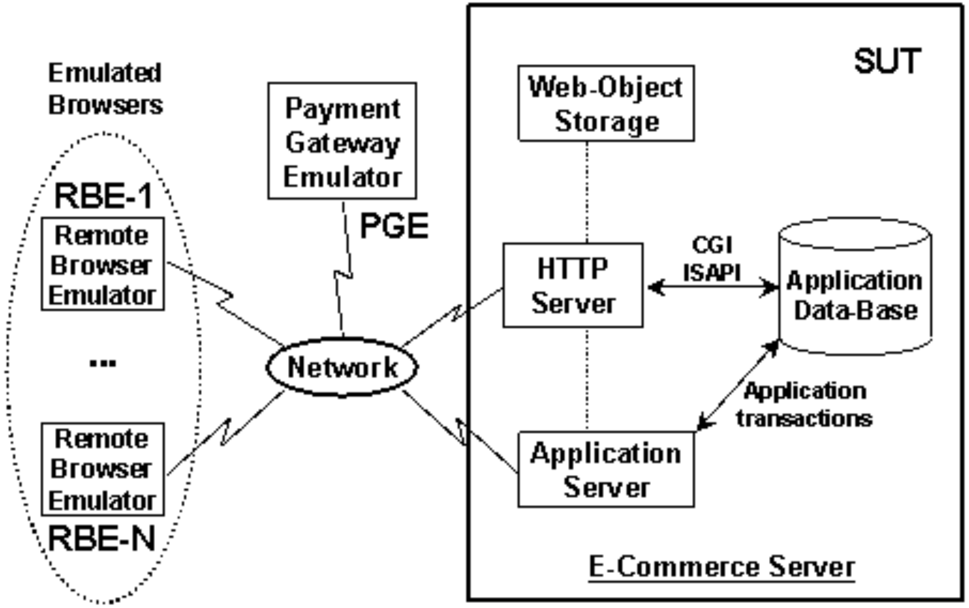
TPC-W is a transactional web benchmark. The workload is performed in a controlled internet commerce environment that simulates the activities of a business oriented transactional web server (an online bookstore), characterized by:

- Multiple on-line browser sessions
- Dynamic page generation with database access and update
- Consistent web objects
- The simultaneous execution of multiple transaction types
- On-line transaction execution modes
- Databases consisting of many tables with a wide variety of sizes, attributes, and relationships
- Transaction integrity (ACID properties)
- Contention on data access and update

The performance metric reported by TPC-W is the number of web interactions processed per second. Multiple web interactions are used to simulate the activity of a retail store, and each interaction is subject to a response time constraint.

TPC-W simulates three different profiles by varying the ratio of browse to buy: primarily shopping (WIPS), browsing (WIPSb) and web-based ordering (WIPSo). The primary metrics are the WIPS rate, the associated price per WIPS ($/WIPS), and the availability date of the priced configuration.

# Conceptual TPC-W architecture



# Example: TPC-W

TPC-W is a transactional web benchmark. The workload is performed in a controlled internet commerce environment that simulates the activities of a business oriented transactional web server (an online bookstore), characterized by:

- Multiple on-line browser sessions
- Dynamic page generation with database access and update
- Consistent web objects
- The simultaneous execution of multiple transaction types
- On-line transaction execution modes
- Databases consisting of many tables with a wide vari…
- Transaction integrity (ACID properties)
- Contention on data access …

**Is TPC-W appropriate for benchmarking enterprise software systems in the cloud?**

The performance r… number of web interactions processed per second. Multiple web interactions are … the activity of a retail store, and each interaction is subject to a response time constraint.

TPC-W simulates three different profiles by varying the ratio of browse to buy: primarily shopping (WIPS), browsing (WIPSb) and web-based ordering (WIPSo). The primary metrics are the WIPS rate, the associated price per WIPS ($/WIPS), and the availability date of the priced configuration.

# TPC-W is not appropriate for the cloud

The TPC-W benchmark is designed to test the complete application stack and does not make any assumptions on the technologies and software systems used. This introduces some problems:

1. Benchmark requires ACID → Cloud systems usually do not offer strong consistency (and web apps often require only lower levels of consistency)
2. No maximum WIPS reportable → Cloud systems should scale infinitely
3. $/WIPS does not work → No max. number of WIPS exist, no fixed load exists, instead different pricing plans exist
4. Outdated web interactions → no user-generated content, etc.

Source: [BKKL09] C. Binnig, D. Kossmann, T. Kraska, S. Loesing. How is the Weather tomorrow? Towards a Benchmark for the Cloud. DBTest'09, ACM, 2009

# Benchmarking the cloud – relevant questions

1. How well do different cloud services scale with an increasing workload? Can indeed a (virtually) infinite throughput be achieved?
2. How expensive are these offerings and how does their cost / performance ratio (i.e., bang for the buck) compare?
3. How predictable is the cost with regard to dynamic changes in the workload?

Source: [BKKL09]

# Different metrics required (examples)

Instead of measuring the average performance of a static system under maximal load (as in conventional systems benchmarking), new metrics should reflect the ability of the cloud services to adapt to a changing load with regard to performance and costs

Moreover, an additional metric should cover the robustness of the services against failures of single nodes as well as the outage of complete data centers.

Source: [BKKL09]

# Recap cloud (storage) systems and services

Recap the motivation/drivers:

- Reduced time-to-market
- CapEx => OpEx
- Better utilization of HW resources
- (Virtually) infinite scalability
- Improved flexibility & cost savings

How do we compare different cloud services / how can these be benchmarked?

# Some initial ideas (2009)

## Extend TPC-W scenario

- New metrics (e.g., vary WIPS)
- Emulated browsers (test drivers) to run in different locations

## 3 configurations (consistency levels)

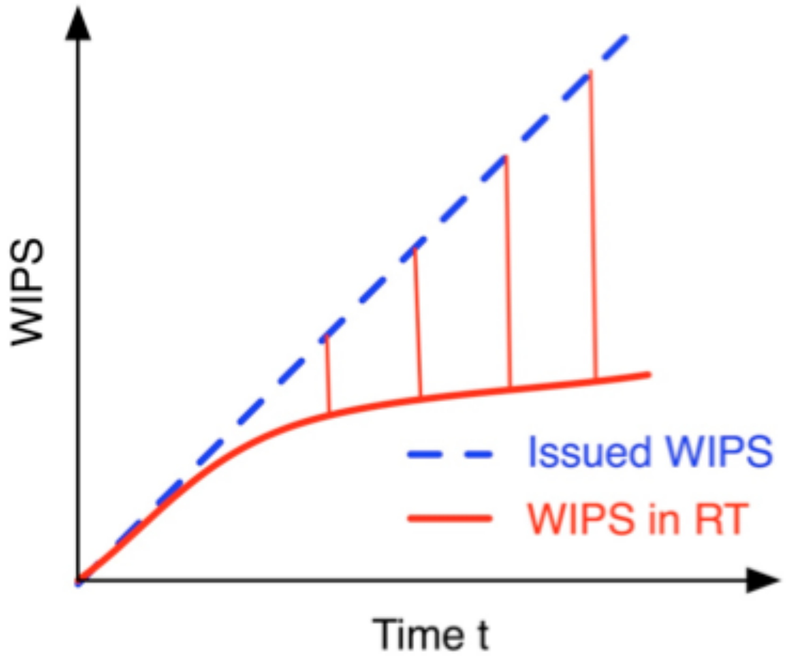
- Low: All WI use only BASE guarantees
- Medium: Mix between BASE and ACID
- High: All web-interactions require ACID

## 3 experiments (metrics)

- Scalability, Cost
- Peaks (Scale-Up and Down)
- Fault tolerance

Source: [BKKL09]

# Experiment 1a: Scalability




Benchmark scale-up

Increase issued WIPS against system over time

Measure WIPS in allowed response time (RT)

Metric:

**Correlation coefficient $R^2$** between perfect linear scaling and WIPS in RT

- 1 = perfect
- 0 = constant behavior (no scaling)

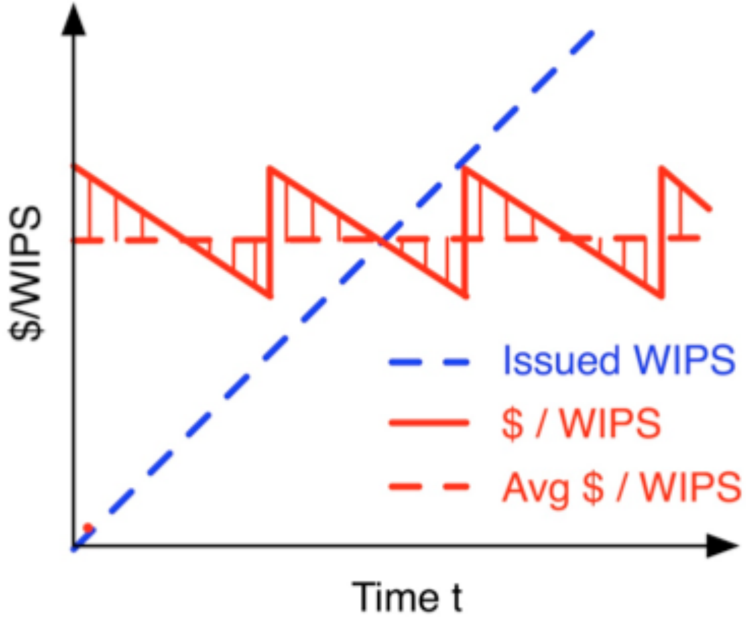
Stop experiment, if (WIPS in RT)/(Issued WIPS) < X or time > Y → report on time!

Source: [BKKL09]



ISEngineering Information Systems Engineering

# Experiment 1b: Cost




Increase issued WIPS against system over time
Metric:

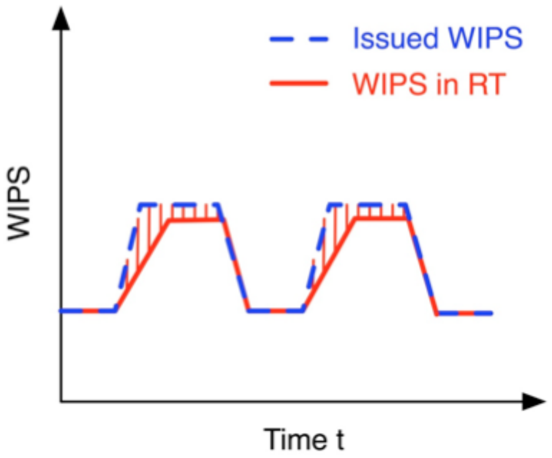
- Average Cost **$/WIPS**
- Standard deviation **S**
  - S = 0: perfect pay-per-use
  - S >> 0: traditional non- cloud scenario

Source: [BKKL09]



ISEngineering
Information Systems Engineering

# Experiment 2: Scale-Up/Down (Peaks)




Measure scale-up and scale-down

Metric:

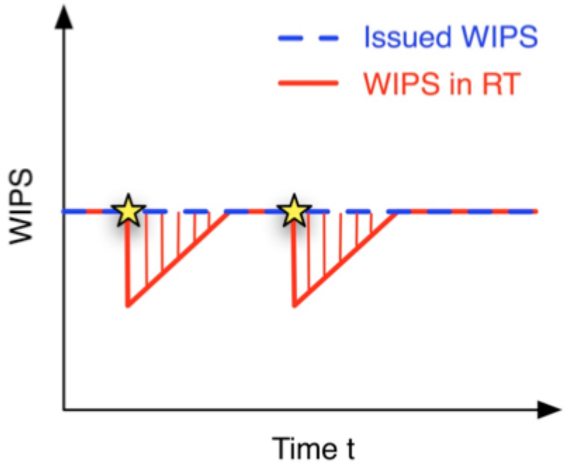
- Peak ratio =
  (WIPS in RT) / (Issued WIPS)
- Average cost + Cost deviation
- Issues:
  - Fix load increase vs. different scenarios
  - Alternative metric: Elevation factor (use different load increases until peak ratio < 1)

Source: [BKKL09]



ISEngineering
Information Systems Engineering

# Experiment 3: Fault tolerance




Measure failure behavior

Idea: Fail X percent (randomly) of the resources

Metric:

- Fault ratio: (WIPS in RT) /(issued WIPS)
- Cost + cost variance

Alternative metric: Maximum failure percentage until fault ratio < 1

Issues:

- Hard to measure
- Not always possible
- Might not be fair!

Source: [BKKL09]

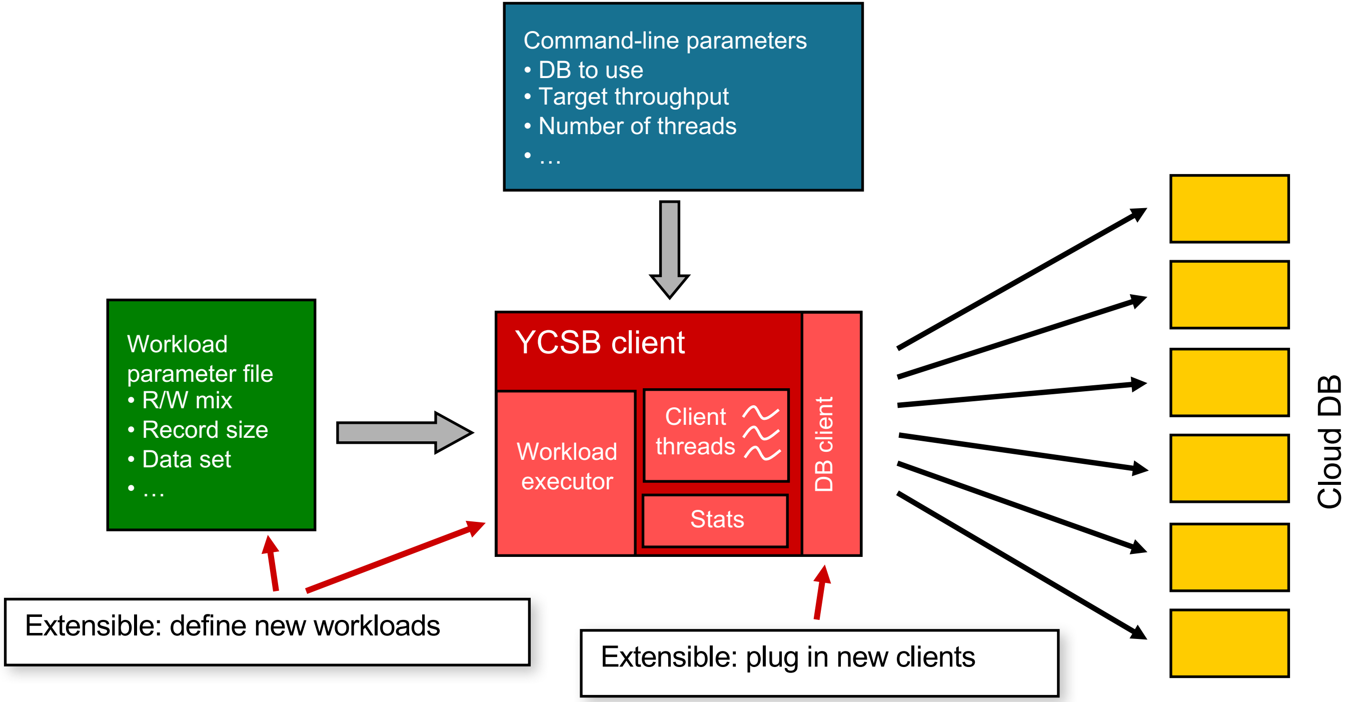
# Current Approaches to Cloud Benchmarking: YCSB

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-source framework and tool to support performance comparisons of cloud data serving systems (such as Cassandra and Hbase) for “non-traditional” (unlike TPC-C) workloads

References:

- http://wiki.github.com/brianfrankcooper/YCSB/
- B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears "Benchmarking cloud serving systems with YCSB.“, ACM SOCC, 2010

# YCSB client architecture




…to be continued in the lab

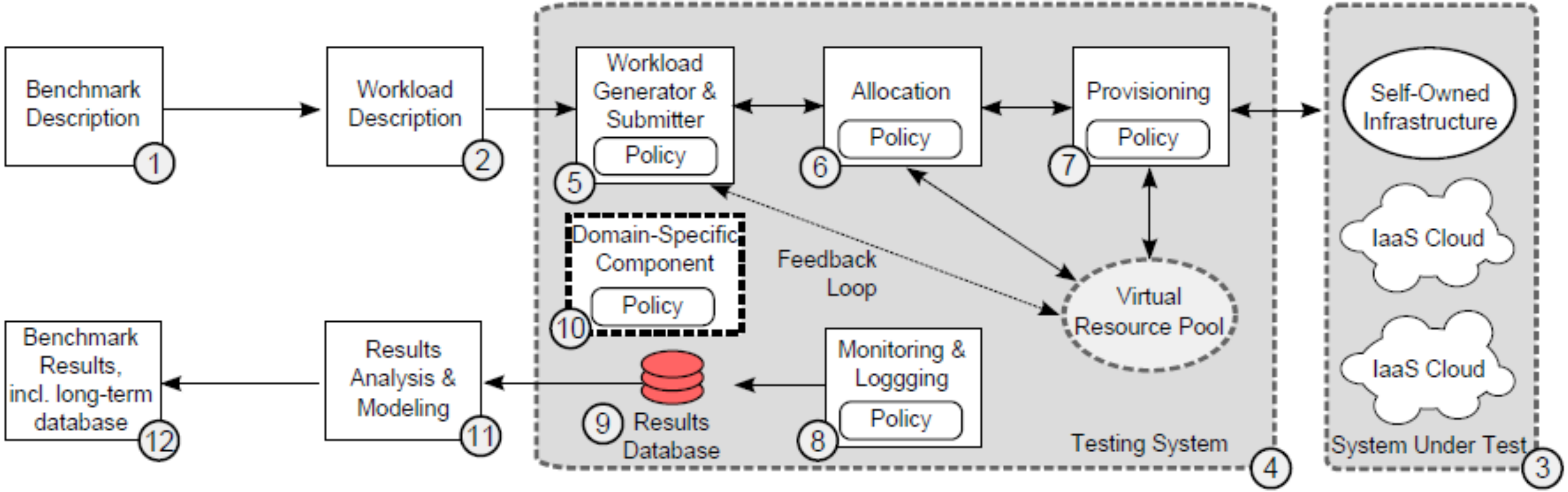
# Current Approaches to Cloud Benchmarking: A generic IaaS benchmarking architecture

A. Iosup et al. propose a generic architecture for IaaS cloud benchmarking, along with (yet open) challenges in cloud benchmark development.

Reference:

- A. Iosup et al. IaaS Cloud Benchmarking: Approaches, Challenges, and Experience. MTAGS, 2012
- A. Iosup et al. Towards Benchmarking IaaS and PaaS Clouds for Graph Analytics. WBDB 2014, Springer 2015

# Overview of the generic architecture




Source: A. Iosup

# Main challenges, grouped in 4 categories

## Methodological

- Experiment compression
- Beyond black-box testing through testing short-term dynamics and long-term evolution
- Impact of middleware

## System-Related

- Reliability, availability, and system-related properties
- Massive-scale, multi-site benchmarking
- Performance isolation, multi-tenancy models

## Workload-related

- Statistical workload models
- Benchmarking performance isolation under various multi-tenancy workloads

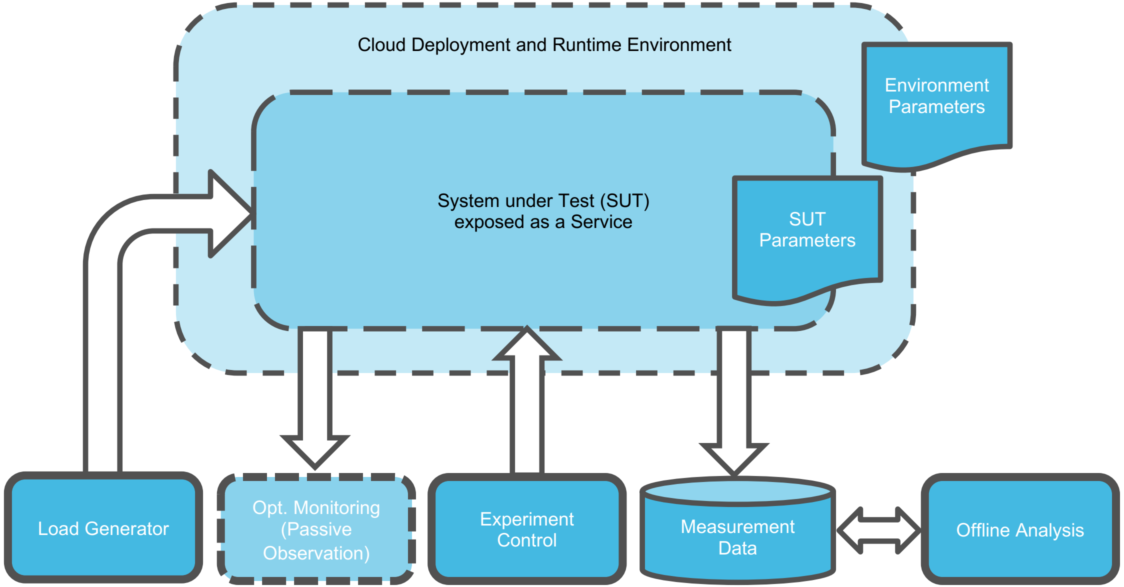
## Metric-Related

- Beyond traditional performance: variability, elasticity, etc.
- Closer integration with cost models

Source: A. Iosup

# Using the Cloud for Benchmarking

Recall the generic architecture introduced earlier:
- Different (including traditional) benchmarks may be executed in a cloud-based environment



# The Cloud as Experimentation Testbed

Challenge: It can be difficult to obtain hardware resources for experiments, particularly if they do not need to be run regularly

Solution: Allocate and de-allocate virtualized experiment resources *on-demand* (and pay-per-use) in a public compute cloud or storage cloud.

While keeping in mind:

Challenge: The effort involved in a complex experiment setup, requiring different skills for system, software, and networking configuration, can quickly overwhelm a single experimenter.

Solution: Fully *automate* the experiment setup process with deployment & configuration management tools, or build the setup on top of virtual appliances.

and:

Challenge: Working on a continuously evolving shared experiment setup and experiment plan requires diligent management and coordination.

Solution: Package experiment setups and processes as *executable experiment plans* which can be managed and shared with a *collaborative version control* system.

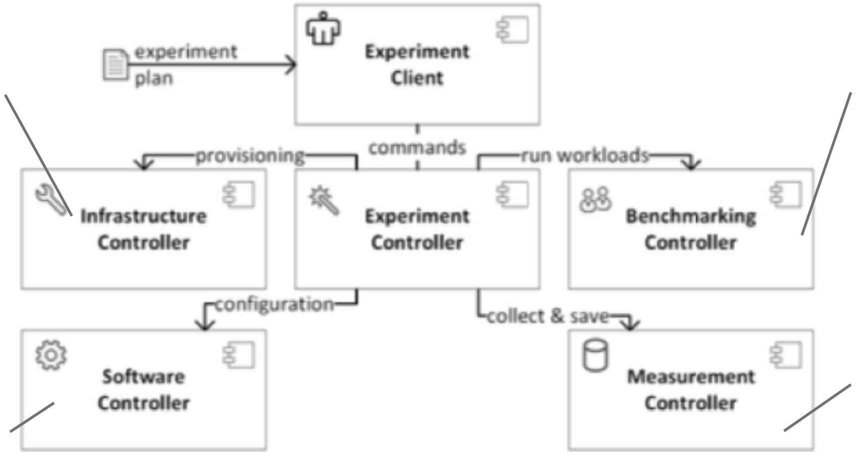
Source: M. Klems, 2016

# Experiment Automation System Design

Technische Universität Berlin

Provisions and configures infrastructure

- Deploys, installs, and configures SUT software
- Deploys and configures benchmarking clients (e.g. setting up workload files)
- Deploys and configures measurement collectors (e.g. monitoring agents)




Triggers benchmarking clients to execurte one or more workloads

Collects system monitoring data and client benchmarking results and saves measurements in a remote storage system

Source: M. Klems: Experiment-driven Evaluation of Cloud-based Distributed Sytems, Ph.D. Thesis, TU Berlin, 2016

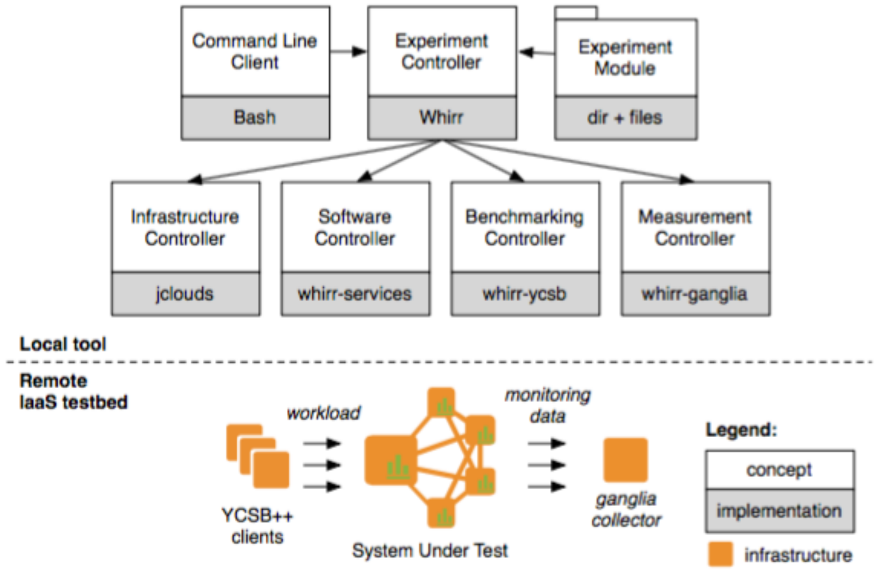


ISEngineering
Information Systems Engineering

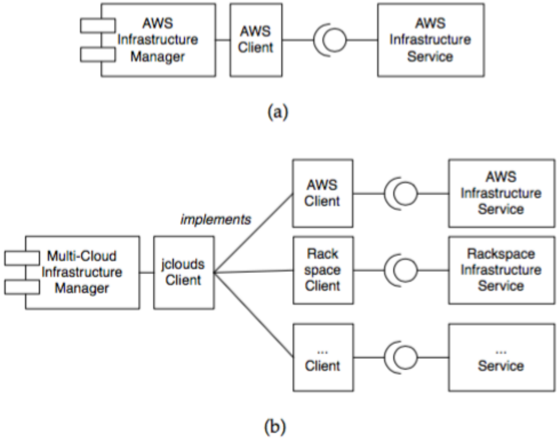
# Elastic Lab: an open-source experiment automation system [Klems]

https://github.com/markusklems/whirr-elasticlab

Builds on Apache Whirr [https://whirr.apache.org/], a tool for deploying cluster services on a compute cloud using the multi-provider software abstraction Apache "jclouds" [https://jclouds.apache.org/]



# Single-provider vs. Multi-provider infrastrcuture controllers [Klems]

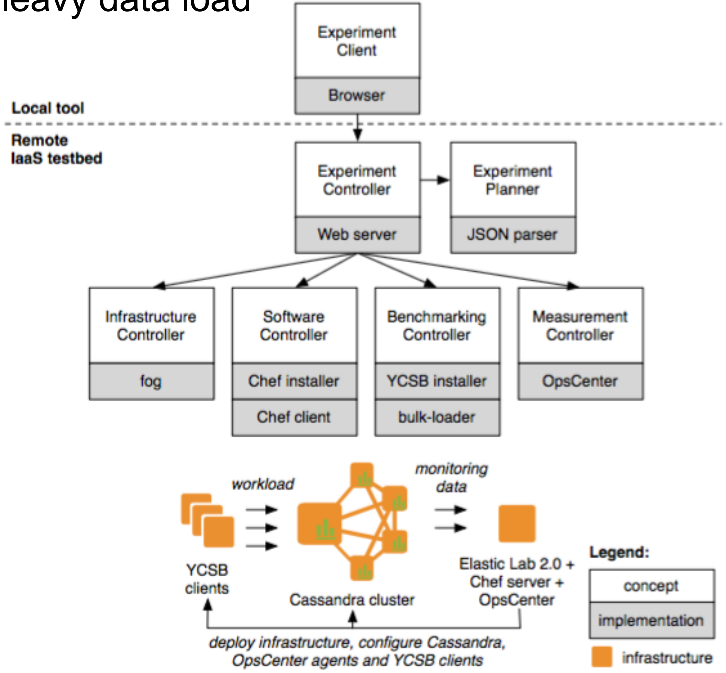



| Name | URI |
|---|---|
| boto | http://code.google.com/p/boto/ |
| deltacloud | http://incubator.apache.org/deltacloud/ |
| fog | http://github.com/geemus/fog |
| jclouds | http://jclouds.incubator.apache.org/ |
| JetS3t | http://jets3t.s3.amazonaws.com/ |
| libcloud | http://incubator.apache.org/libcloud/ |
| PyStratus | https://github.com/digitalreasoning/PyStratus/ |

*List of multi-cloud software libraries*

# Elastic Lab 2.0 [Klems]

- Better integration with config management solutions used in the IT industry, namely Chef and Puppet
- Different approach to automating system configuration and improved bulk-loading capabilities needed for testing databases under heavy data load
- Implemented in Ruby



# Quick summary, so far

Benchmarking is a well-established area of research and practice, especially in the (traditional) areas of database and (conventional) distributed systems

Traditionally, benchmarking has focused on performance as a quality

Cloud benchmarking is not a straightforward application of older benchmarking techniques: assumptions made in the past no longer hold for cloud systems and qualities other than performance (but still including performance) matter

Furthermore, clouds do not only offer elasticity on demand, they also offer resources for benchmarking on demand



ISEngineering
Information Systems Engineering

# Next

Three “unconventional” cloud benchmarking experiments conducted by the TUB-ISE research group:

1. Consistency benchmarking

2. Benchmarking security-performance trade-offs

3. Demystifying Web API usage