An Analytic Database Cloud for Software as a Service

Jan Schaffner

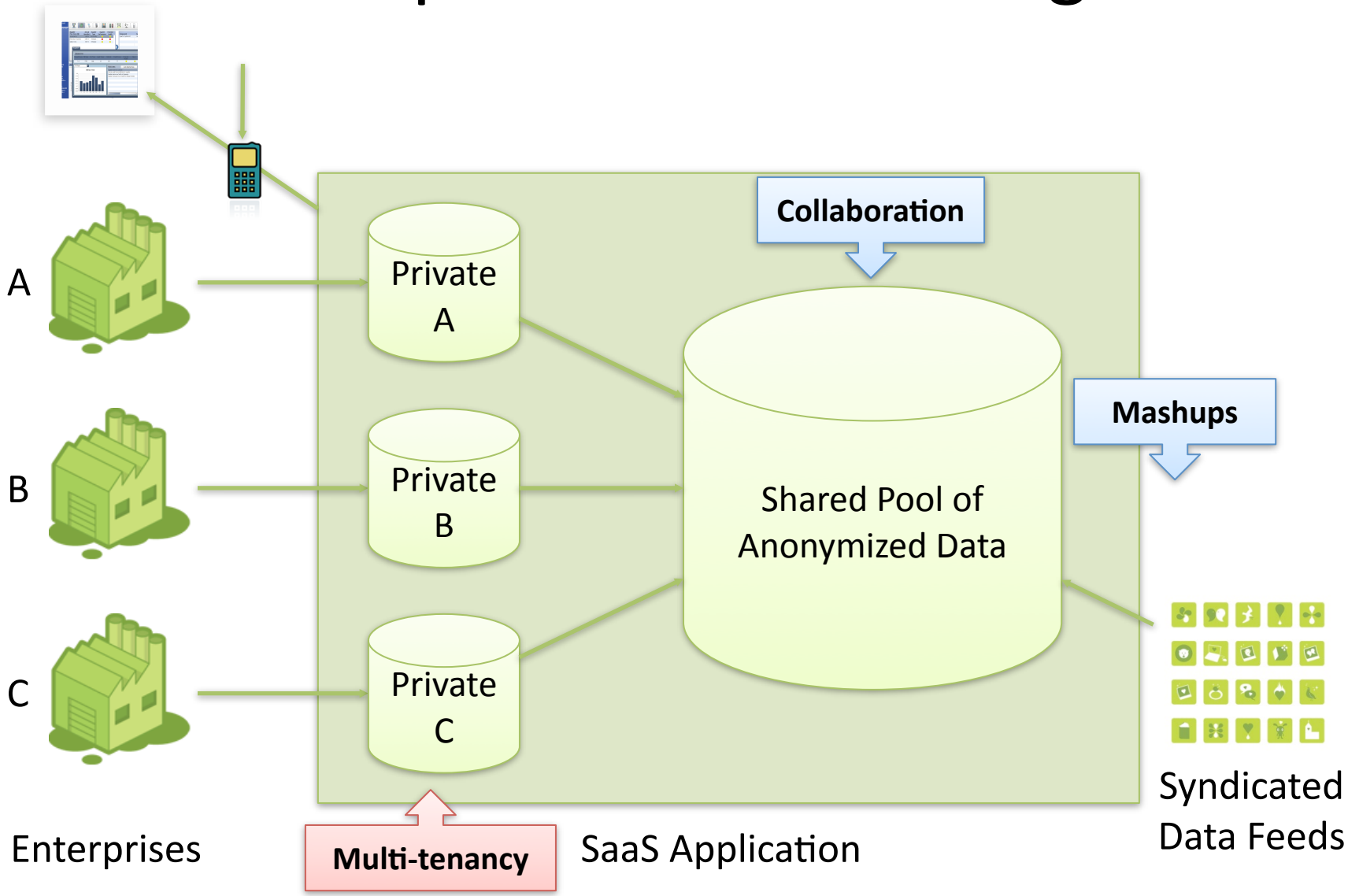
(joint work with Dean Jacobs, Benjamin Eckart, and Christian Schwarz)

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Outline

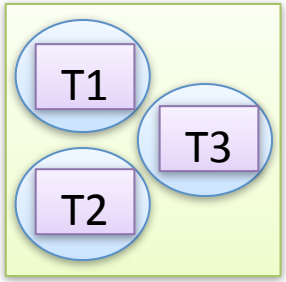
- **Context of This Research**
- The Rock Clustering Framework
- Experimental Results

Example Application: Enterprise Benchmarking



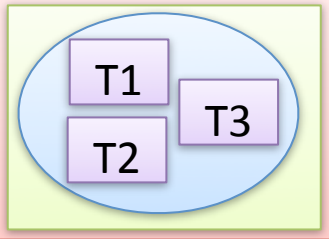
Multi-Tenant Data Management

- Shared machine – database process per tenant



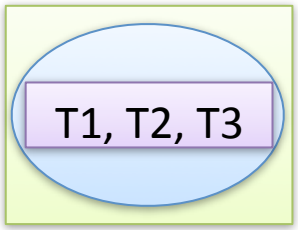
  - RightNow (2007) had 3000 tenants in 200 databases
  - 3000 vs 200 Amazon VMs cost $2,628,000 vs $175,200 / year
  - Plus the cost to administer the databases!

- Shared process – schema per tenant



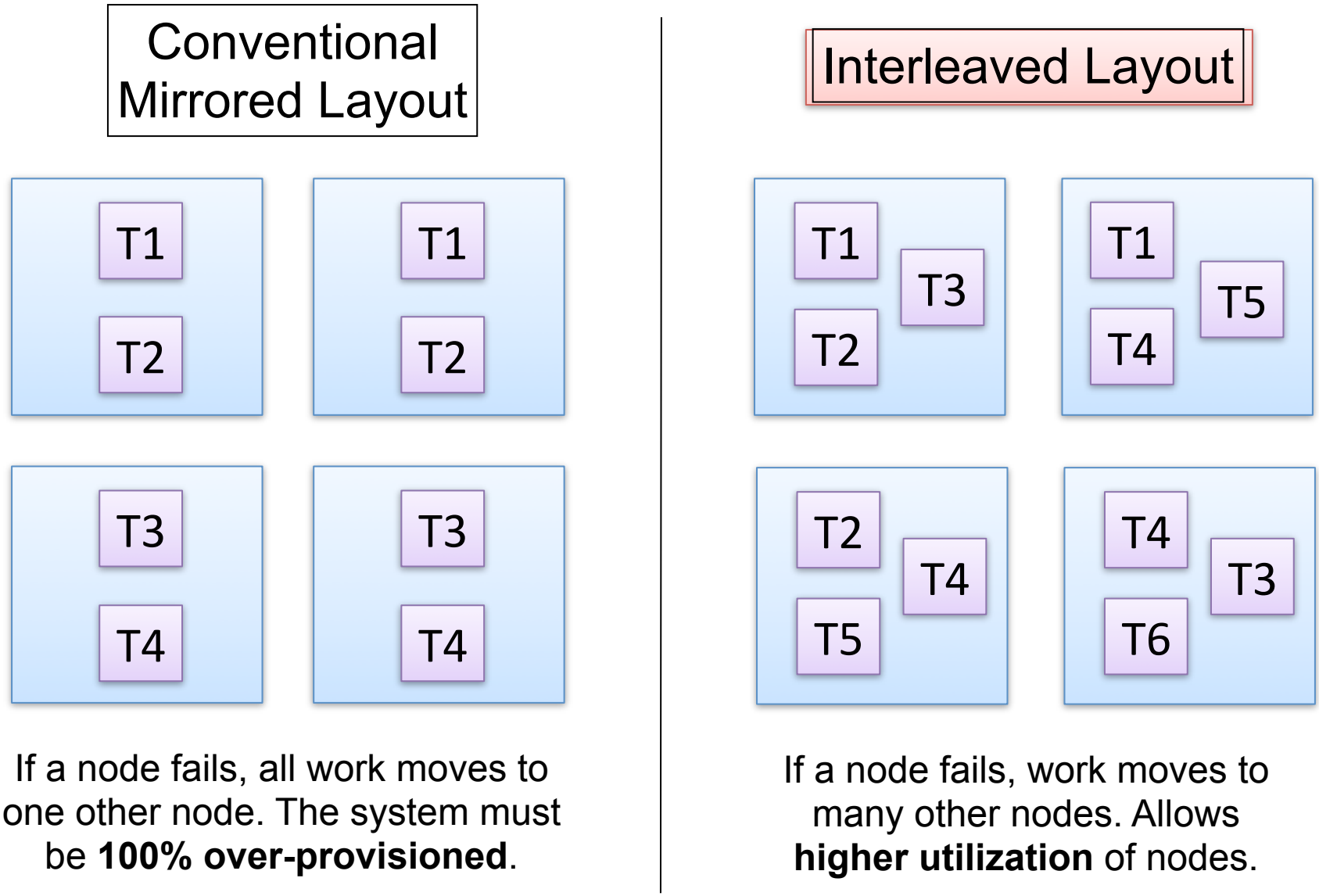
  - Must support large numbers of tables
  - Must support on-line schema extension and evolution

- Shared tables



  - Hard for individual tenants to extend the schema
  - Hard to backup/restore/migrate individual tenants
  - Hard to isolate tenants from each other
  - Table scans can be very inefficient

Tenant Placement



Related Work

Parallel Databases	Our Research	Cloud Databases
Tandem, Teradata, Bubba, Gamma	Rock	Big Table, Dynamo, SimpleDB, PNUTS…
Fixed set of servers	Dynamically sized cluster	Dynamically sized cluster
Fully decluster large relations	No large relations	Distribute large data sets
Replicate large relations by breaking into fragments		Replicate large data sets BigTable: hidden in GFS
Heuristics for small relations Balancing but not interleaving	Heuristics for small relations Balancing and interleaving	
Big bang reorganization	Incremental reorganization	Incremental reorganization
Minimize response time for a single query	Maximize utilization for multiple queries	

Outline

- Context of Our Research
- **The Rock Clustering Framework**
- Experimental Results

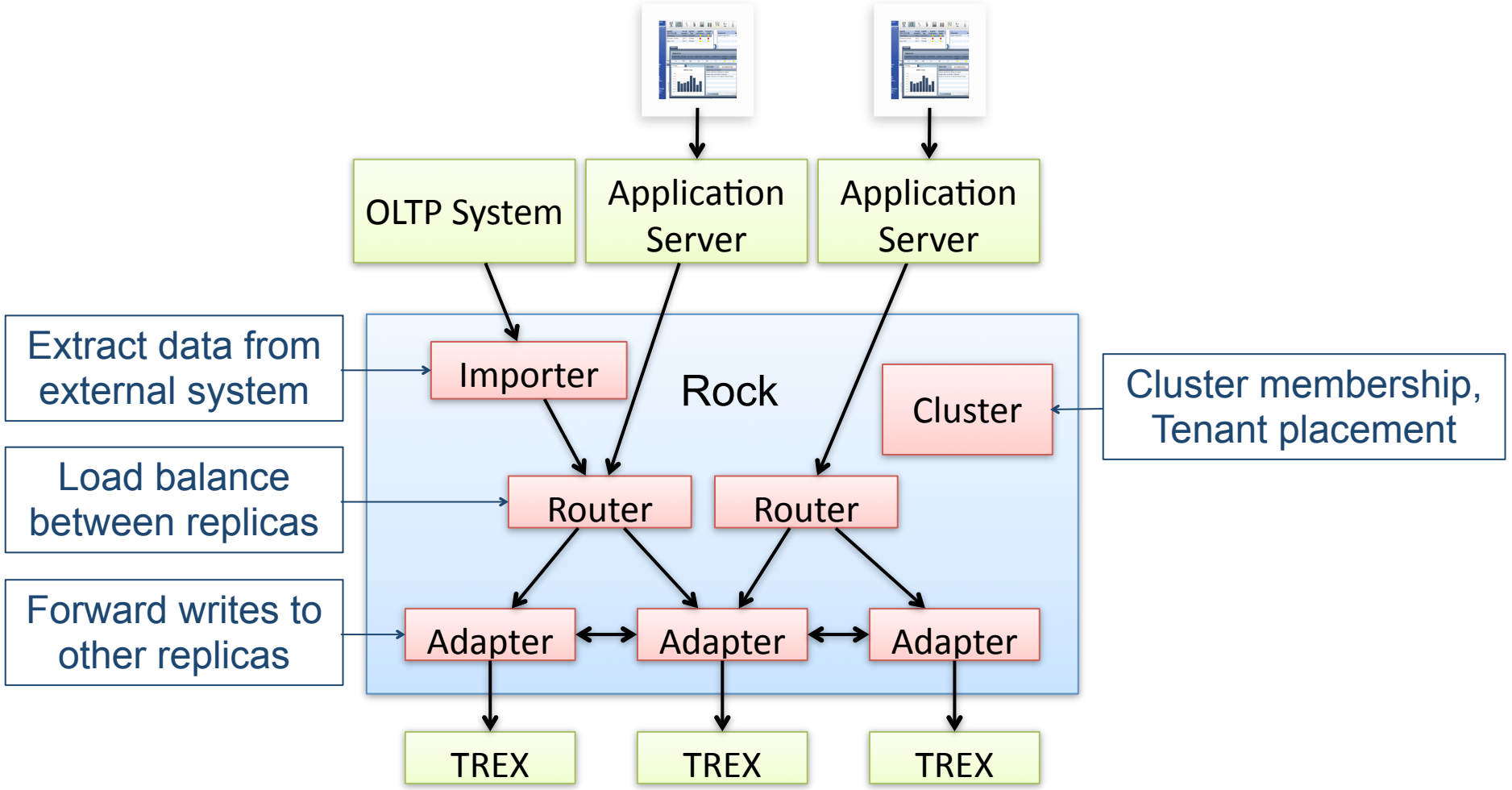
Rock Overview

- Based on SAP's in-memory column database TREX
- Adds a clustering infrastructure that supports
  - schema-per-tenant multi-tenancy
  - replication for scalability and high-availability
- Runs on the Amazon cloud
- Focused on analytic SaaS applications where the data is extracted from an external OLTP system
  - Example: Salesforce to BOBJ BI On-demand

Why In-Memory?

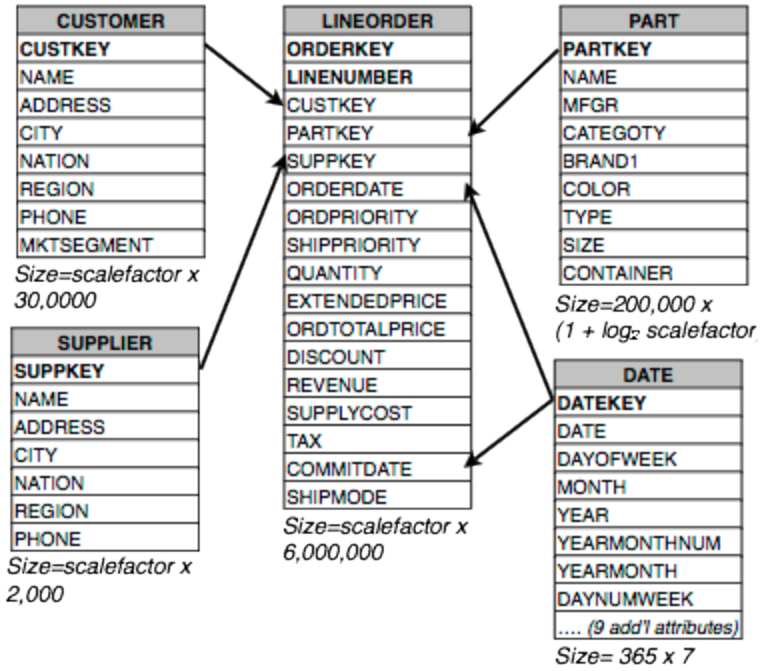
- Will ultimately win on performance (disk is tape)
- Avoid Vertica-style “projections” to reassemble row fragments from columns
  - Joins are cheap once the data is in memory
- Supports schema extension and evolution
  - Example: The SaaS ERP vendor WorkDay
    - Keeps all data in three tables in the database
    - When a tenant comes on-line, the data is read into memory and objects are constructed
    - Claim the resulting flexibility is a fundamental advance over static old-school ERP systems
  - On-line data reorganization is a fact of life (and a requirement for column databases anyway)

Rock Architecture



Benchmark

- A modified version of SSB, which is a modified version of TPC-H
- One instance of the SSB schema per tenant
- Added periodic batch writes from one source per tenant
- Queries grouped into *flights* which drill down into an issue
- Require snapshot isolation within a flight (implemented in TREX)
- Added multiple users with think times



Test Run Configurations

- A test run has a fixed set of tenants
- Each tenant has
  - a given amount of data
  - a base factor for number of simultaneous users
- The data is distributed across a set of servers according to a **tenant placement algorithm** under study
- Measure **throughput**: The number of users per tenant is scaled up until the response time at the 99th percentile exceeds one second

Jan's Thesis

- Ground rules: over time
  - tenants join and leave the system
  - the amount of data varies for each tenant
  - the request rate varies for each tenant (and is hard to quantify)
- Develop a tenant placement algorithm that
  - adds and removes tenants on-the-fly
  - migrates tenants as resource usage permits
  - optimizes both balancing and interleaving
  - minimizes the number of servers required
- Simulate execution of the algorithm over a long period of time and then test the resulting layouts

Outline

- Context of Our Research
- The Rock Clustering Framework
- **Experimental Results**

Handcrafted Best Case

- Perfect placement

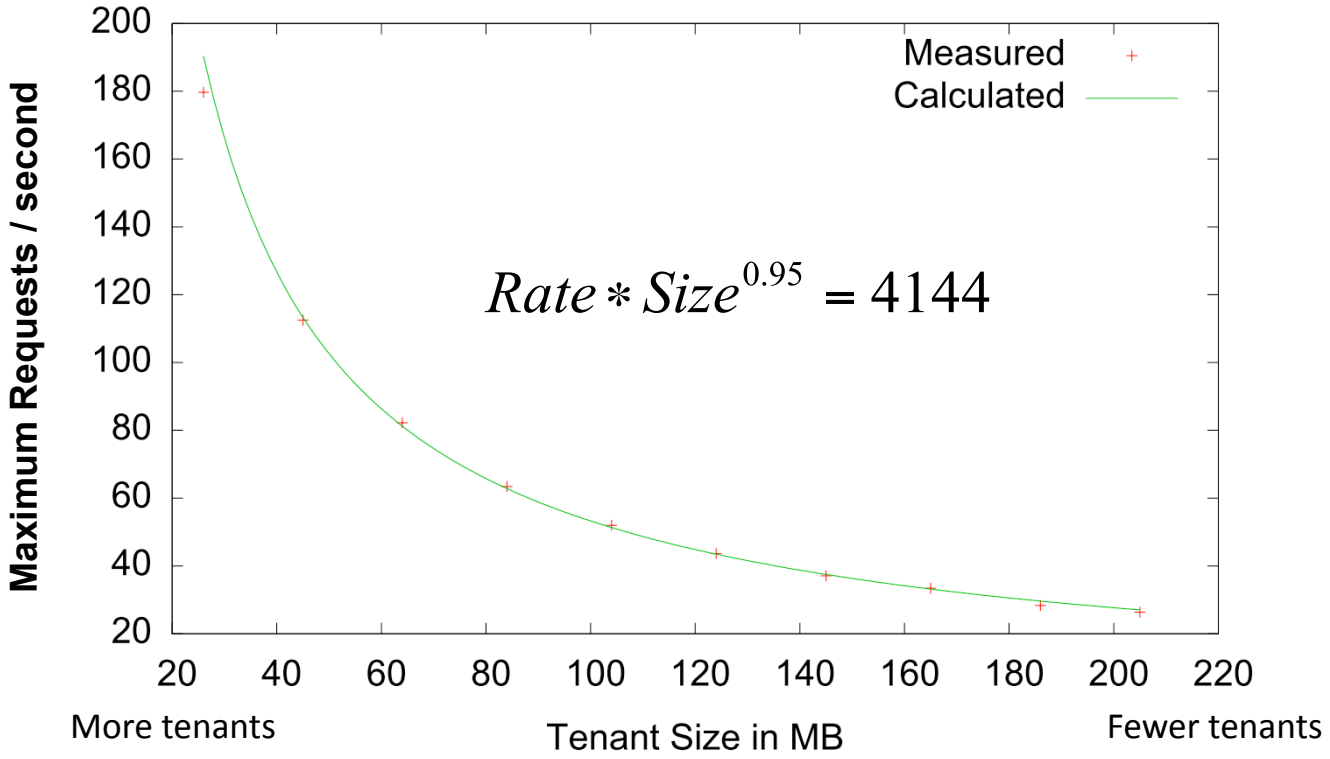
- 100 tenants on 10 servers with 10 tenants/server
- Perfect balancing: same load on every tenant
  - 6M rows (204 MB compressed) of data per tenant
  - The same (increasing) number of users per tenant
  - No writes

	Mirrored	Interleaved	Improvement
No failures	4218 users	4506 users	7%
Periodic single failures	2265 users	4250 users	88%

Throughput

System Capacity

- Fixed amount of data split equally among all tenants



- Capacity ≈ bytes scanned per second
  - A small overhead for processing requests
- In-memory databases behave very linearly!

Workload

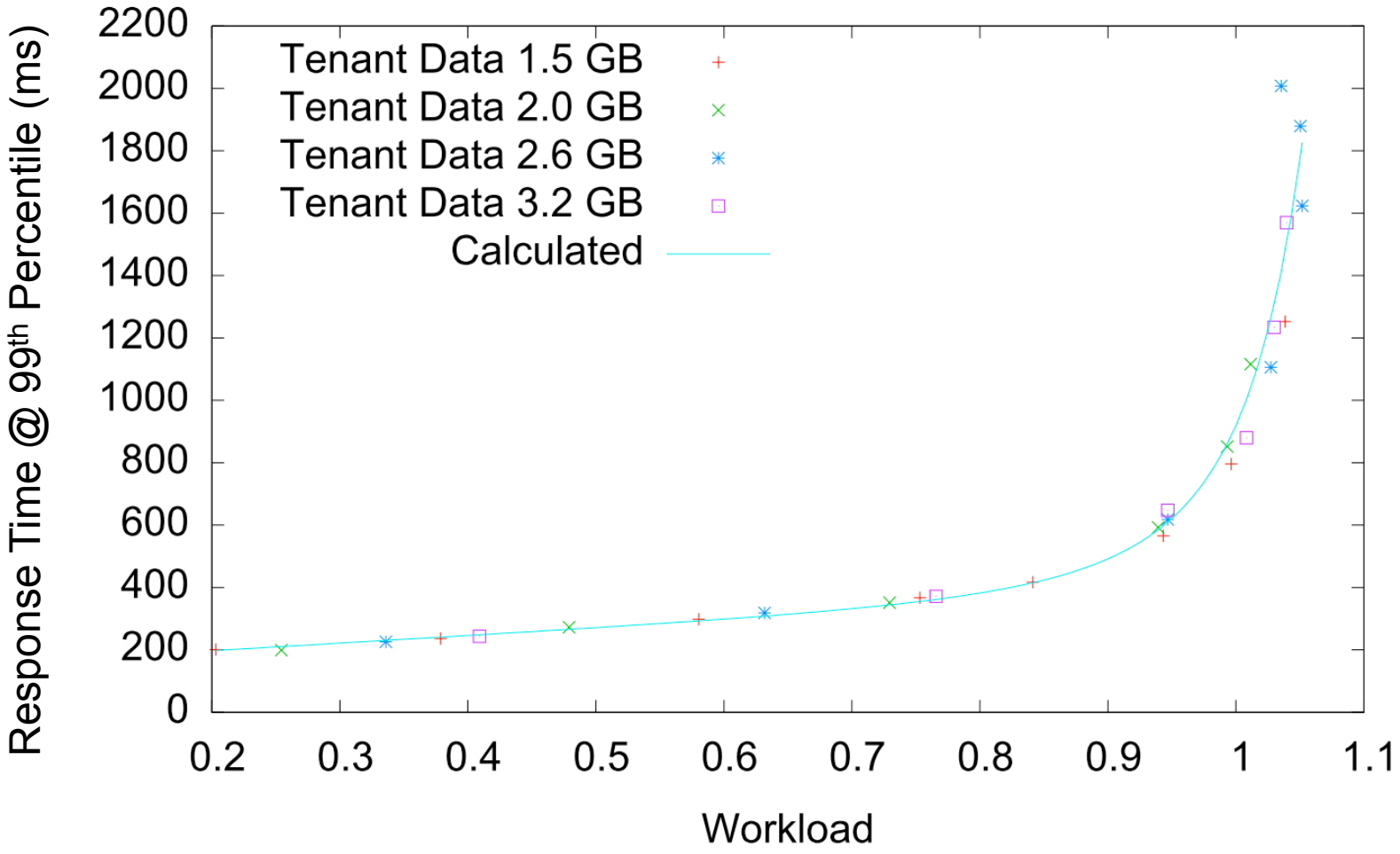
- Tenants generally have different rates and sizes
- For a given set T of tenants define

$$Workload = \sum_{t \in T} \frac{Rate_t * Size_t^{0.95}}{4144}$$

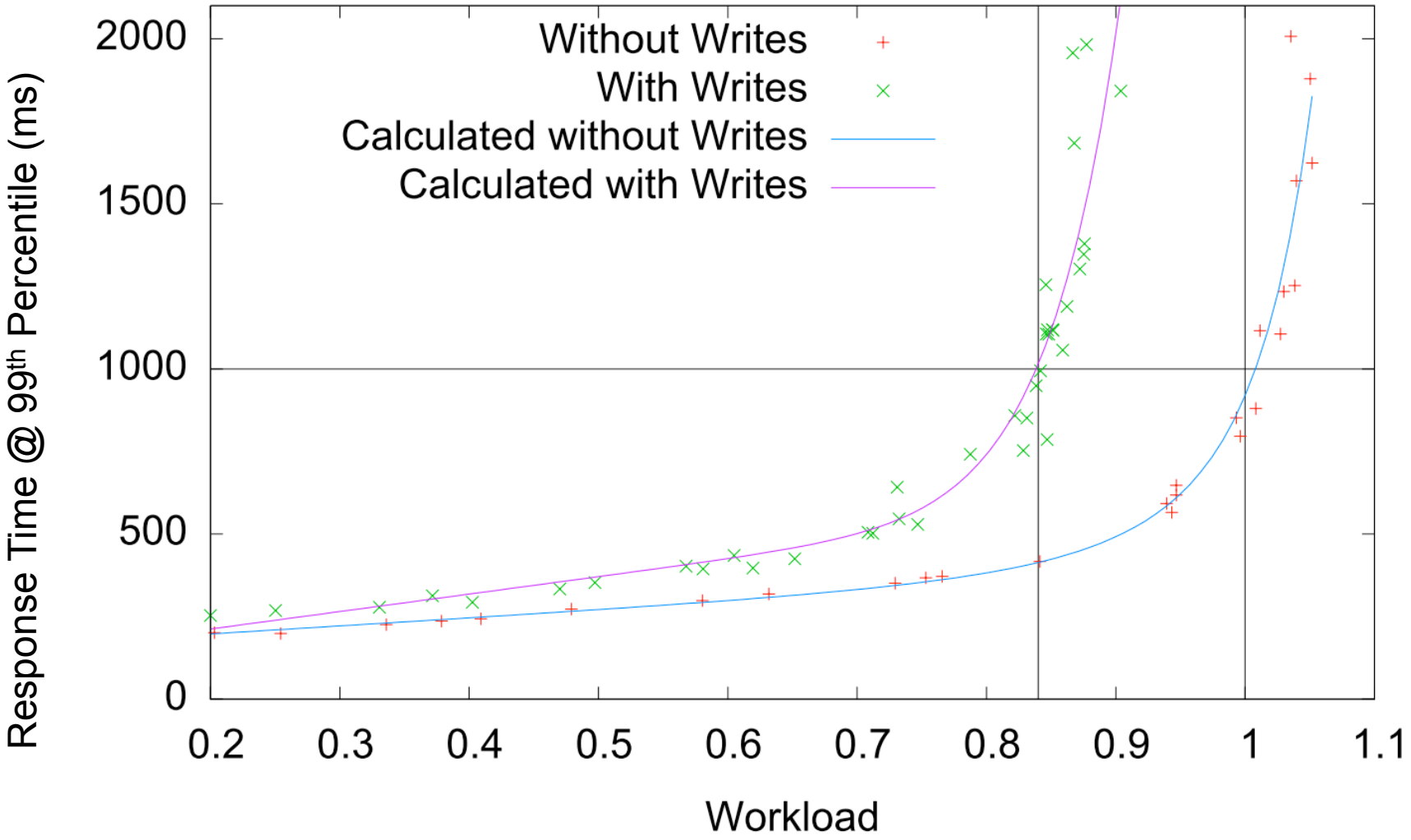
- When Workload = 1, the system is running at it's throughput level; if the level goes higher then response time goals will be violated

Response Time

- Different amounts of data and assorted tenant sizes
- Vary Workload by scaling the rates for tenants

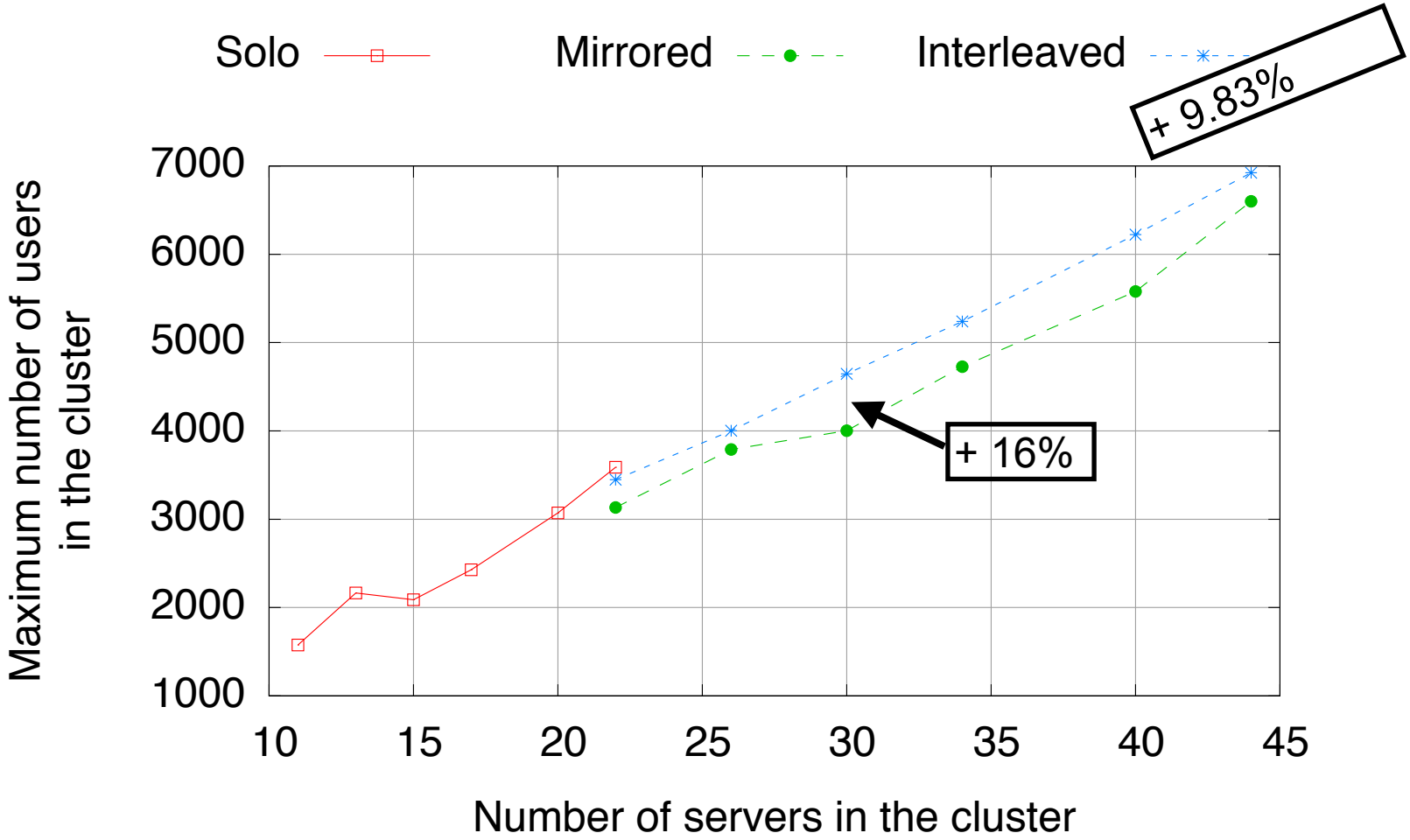


Impact of Writes



Simple Greedy Heuristic

- Initial study before we characterized capacity
- Bugs in load balancer



Future Work

- Incorporate capacity characterization in tenant placement algorithm
- Study impact of **on-line reorganization**
  - Migration of tenants
  - Schema evolution
  - Merge of delta into columns

Seminar Topics

- Build a simulation environment
  - Challenges: Balance both overlap and bytes scanned
  - Should be able to run both greedy heuristics and fancy machine learning algorithms
- Build an application on top of the Rock framework
  - Pick your favorite scenario
  - Do something „presentable“
- Build an on-line visualization of the Rock cluster state
  - Show active EC2 nodes and tenant layout
  - Visualize current query workload, tenant overlap, failures, migrations, merges, ...