“Cut Me Some Slack”: Latency-Aware Live Migration for Databases

Sean Barker†‡, Yun Chi‡, Hyun Jin Moon‡, Hakan Hacigumus‡, and Prashant Shenoy†

University of Massachusetts Amherst†
NEC Laboratories America‡
Cloud Database Systems

- **Multitenant database servers**
  - Multiple resident DBs per physical server
  - Independent workloads

- **Service Level Agreement (SLA)**
  - Minimum agreed performance
  - E.g., latency, throughput
  - Penalty for violations

- **Tenant packing (tenants per server)**
  - Maximize utilization (more packing)
  - Minimize SLA violations (less packing)
  - Tradeoff!
Dynamic Workloads

- Tenants and workloads are **dynamic**
  - Tenants added or deleted
  - Time of day variations
  - Flash crowds

- Potential for **overload**

![Diagram of resource usage and SLA violations](image)
Database Migration

- Respond to potential overload
  - E.g., new tenant, increasing load
- **Migrate** tenants to other servers
  - Alleviate source hotspot, increase target utilization
Problem 1: Downtime

- Tenant downtime during migration

Solution: Live migration
- Virtual machine level [Clark-NSDI 05]
- DB level [Elmore-SIGMOD 11, Das-PVLDB 11]
Problem 2: Interference

- Migration itself is a load!
  - Source server may be heavily loaded already
  - Migration may make situation temporarily worse

- Goal: Migrate with minimal tenant interference
Outline

- Motivation
- Study of migration interference
- Interference-aware live migration
- Prototype evaluation
- Conclusions
Case Study: Migration Interference

- Does migration interference matter?
- Scenario: heavily loaded source
  - Tenant 1 servicing workload
  - Tenant 2 migrating to target

- Impact on non-migrating tenant
  - ‘Unfair’ interference

- Migration workload measure
  - Migration rate (MB/sec)

- Interference measure
  - Transaction latency (ms)
Impact on Neighboring Tenant

- Limited capacity to be used for migration!
**Migration Slack**

- **Migration ‘slack’**
  - Resources usable towards migration
  - Migration rate as resource currency

- Slack determined by SLA
  - Migration speed vs. latency
  - ‘Acceptable’ latency, variance
Slack-Aware Live Migration

- Consume exactly all slack for migration
  - Too fast → poor performance or server overload
  - Too slow → migration takes too long
- But workloads are **dynamic**
  - So slack is dynamic too!

- Approach: **adaptive dynamic throttle**
  - Heavy workload → slow down migration
  - Lighter workload → speed up migration
- **Proportional-integral-derivative (PID) controller**
  - Well-known feedback loop in control systems
  - Current error $\rightarrow$ PID $\rightarrow$ action $\rightarrow$ new error $\rightarrow$ PID $\rightarrow$ ...

![PID Controller Diagram](image)
**Proportional-integral-derivative (PID) controller**
  - Well-known feedback loop in control systems
  - Current error → PID → action → new error → PID → ...

![Diagram showing a PID controller with inputs and outputs](image)
Migration Throttle Control (2)

- **Process variable**: What should we try to control?
  - Measure of interference, SLA-based: recent query latency

- **Setpoint**: Where do we want latency to be?
  - Goals: low latency **and** high slack utilization

- **Modestly elevated target latency**
  - But still acceptable to SLA
### Migration Throttle Control (3)

A block diagram of a PID controller is shown in the figure. A PID controller is a well-known algorithm in control theory for driving a system actuator such that the system output matches the setpoint.

Given the ability to dynamically vary the throttle, we employ an automated technique based on experiments demonstrating the importance and effects of the PID controller. We apply a PID controller to the problem of database migration in Slacker to manage migration speed.

Here, we discuss the migration of tenants and workloads using a PID controller. The controller is in charge of determining the current transaction latency as an indicator of available throttling speed adjustment; either speeding up or slowing down the migration speed as necessary. In other words, using the illustration in Figure vi, we observe the transaction latency while staying within the green circle, which is achieved by keeping the controller input low enough to maintain reasonable performance. We present a suitable setpoint setting. More specifically, the setpoint indicates an effort to stabilize the controller input, since individual transaction latencies are averaged to come up with a setpoint value. Of course, the effort to prevent the current level of performance from dropping too low will aggressively migrate as fast as possible so long as the migration remains under the available slack, which is achieved by setting the PID controller output variable to be as automated as possible. That is, we do not want the control to require an operator such as a database administrator to manually control the migration speed for controlling the migration in Slacker.

The controller at each timestep consists of the current average transaction latency over a small sliding window of time. We present the size of the sliding window provides some useful smoothing effects of the latency over time. The controller at each timestep consists of the current average transaction latency as a feedback to stabilize the controller input, since individual transaction latencies are averaged to come up with a setpoint value.

Thus, in order to ensure that we are using most slack and try to stay in the acceptable region along the red curve, we configure our PID controller to maintain that latency over time by adjusting the throttling speed as necessary.

So long as the migration remains under the available slack, we use current transaction latency as an indicator of available throttling directly, a nontrivial problem in itself. Instead of targeting available slack as a setpoint of zero, we could be to target available slack or a setpoint of zero without exceeding it.

The intuition behind this is as follows. As we increase the current level of performance from an acceptable level, the effort to prevent the current level of performance from dropping too low will aggressively migrate as fast as possible so long as the migration remains under the available slack, which is achieved by setting the PID controller output variable to be as automated as possible.

That is, we do not want the control to require an operator such as a database administrator to manually control the migration speed for controlling the migration in Slacker.
### Prototype Implementation

- **Slacker**: our framework for database migrations
  - Middleware system above database itself

- **Live migration via hot backup**
  - Copy/delta/handover
  - MySQL and xtrabackup
  - Other DBs possible

- **Latency-aware migration**
  - Throttled migration rate
  - PID-controlled throttle

---

**Figure 1**: The Slacker architecture.

- **Server 1**
  - migration controller
  - `mysqld`

- **Server 2**
  - migration controller
  - `mysqld`

- **Server 3**
  - `mysqld`

---

**Diagram Notes**

- Migration controller on each server monitors all tenants located on the server.
- Tenant is simply a directory containing all data and a corresponding MySQL process.
- Slacker is transparent to tenants'
- Tenant no longer resides on the original server.
- This issue can be resolved cleanly by issuing an vRP packet advertising
- Server XYZ.

---

**Table**

<table>
<thead>
<tr>
<th>Prototype Implementation</th>
<th>Live migration via hot backup</th>
<th>Latency-aware migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slacker</td>
<td>Live migration via hot backup</td>
<td>Latency-aware migration</td>
</tr>
<tr>
<td></td>
<td>Live migration via hot backup</td>
<td>Latency-aware migration</td>
</tr>
<tr>
<td></td>
<td>Live migration via hot backup</td>
<td>Latency-aware migration</td>
</tr>
</tbody>
</table>

---

**References**

- [Slack within the Slacker framework](#)
- [Evaluation of our end-to-end system](#)
- [Prototype Implementation](#)
Yahoo Cloud Serving Benchmark (YCSB)
  - Multi-op transactions, read/write mix, I/O intensive

Static (fixed) throttle vs. dynamic (PID) throttle
Evaluation: Controlling Latency

- Accuracy of setpoint latency

- Successfully achieves setpoint during migration
Evaluation: Dynamic vs. Static Throttle

- Migration performance at given latency

Same latency, faster migration
Evaluation: Dynamic vs. Static Throttle

- Migration performance at given latency

![Graph showing comparison between fixed throttle and Slacker in terms of txn latency vs. average throttle speed. The graph indicates that Slacker achieves the same speed but with lower latency.](image-url)
Evaluation: Dynamic vs. Static Throttle

- Migration performance at given latency

```
Outperforms fixed throttle at same speed or latency
```

```
Steady throttle near max slack
```

```
Outperforms fixed throttle at same speed or latency
```
Evaluation: Slack Sensitivity

- Responsiveness to ‘micro’-level workload changes

Exploits normal latency variation to speed migration
Evaluation: Dynamic Workload

- Responsiveness to ‘macro’-level workload changes
  - 40% workload increase after 60s

- Slacker quickly adjusts to dynamic workloads
Conclusions

- Managing interference from database migrations
  - Impact on neighboring tenants

- **Slacker**: Latency-aware live migration
  - Database-independent middleware
  - PID controller to manage migration speed
  - Reacts in real-time to maintain SLA-acceptable performance

- Ongoing work
  - Automatic setpoint management
  - Other migration questions
    - e.g. Which tenant? Which server?
Questions?

Sean Barker
sbarker@cs.umass.edu
PID Controller Parameters

- **P, I, D** parameters determine **speed and degree** of controller response

**Bad:** Too cautious

**Bad:** Too optimistic

**Good:** Near-optimal

- Currently manually tuned
  - Many other well-known approaches possible
Evaluation: Multitenant Workload

- Average latency across five tenant workloads

Maintains latency in multi-tenant environment