The Role of Container Technology in Reproducible Computer Systems Research

Ivo Jimenez, Carlos Maltzahn (UCSC)
Adam Moody, Kathryn Mohror (LLNL)
Jay Lofstead (Sandia)
Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau (UW)
Why Care About Reproducibility

• Theoretical, Experimental, Computational, Data-intensive research.
• Reproducibility well established for the first two, but impracticably hard for the last two.
• Negative impact on science, engineering, and education.
Status Quo of Reproducibility

The UI is capable of graphing the ratio \( R \) (Fig. 2) in parallel with the analysis of the query stream. A high ratio indicates that WFTT generates good recommendations.

We also asked participants if they would rate the feedback feature "bad". At this point, the candidate-index set can dynamically grow/shrink and be repartitioned over time based on the calculations of index interactions associated with each statement.

This brings the bid into a completely different mode where it can operate autonomously without any user intervention.

Figure 3: Evolution of the candidate set with respect to partitioning (by calculating index interactions at each step).

Each set corresponds to phases 1, 2, and 3, respectively.

We will see again how the algorithm generates a configuration at each step, however, in this scenario the partitioning of the candidate set will evolve for each of the three phases of the workflow (Fig. 4). We will show that this feature actually improves the quality of the recommendations.

Scenario #3: We complete the picture and show the effect that feedback has on the performance of WFTT by demonstrating one of the key contributions of our work, a principled feedback mechanism that is tightly integrated with the logic of the online algorithm: WFTT.

By inspecting the recommended set of indices at any point in time, the DBA can decide whether to up- or downvote indices according to key criteria (or not vote at all). For the first time, we show that the feedback component can interactively send feedback to the recommendation engine. We can evaluate three instances of WFTT concurrently, with distinct feedback (good, bad, and no-feedback) and show the differences in performance for each (Fig. 4).

The audience will see, in the case of good feedback, the performance of WFTT increases in relation to the performance of the "no-feedback" instance, being the performance of the feedback instance the best. For the first time, we show the effect of feedback on the performance of WFTT, the effect of feedback on the performance of WFTT.

This recovery mechanism is another important feature of the WFTT algorithm.

Scenario #4: The last scenario executes the Buffer workload (up to 30 rows on Kaisen). This is a complex workload consisting of approximately 1000 statements (queries and updates) that refer to several databases (TPC-D, TPC-H, TPC-E, TPC-H and NIEP).

We will show two WFTT scenarios: one with a stable and fixed candidate set partitioning; another whose candidate set is allowed to be automatically maintained. Similarly for scenario #1, we will show the SFT vs. WFTT ratio in real-time as the workflow is processed (Fig. 5).

Figure 4: Multiple instances of WFTT running in parallel. The vote for the "good" and "bad" instances is done at step 1, causing the divergence in their behavior with respect to the "no-feedback" instance.

Figure 5: Two instances of WFTT running the Online Index Selection Benchmark. One with a fixed and stable candidate set (FIXED), another with an automatically maintained candidate set (AUTO).

5. REFERENCES

Status Quo of Reproducibility
Status Quo of Reproducibility

![Algorithm Comparison](image)

Figure 4: Multiple instances of WFIT running in parallel.
Status Quo of Reproducibility
Status Quo of Reproducibility

![Diagram showing code at the center with layers of data, OS, libs, and hardware]

![Graph comparing Algorithm OPT, GOOD, BAD, and NO-FEEDBACK over Queries]

*Figure 4: Multiple instances of WFIT running in parallel*
You can download our code from the URL supplied. Good luck downloading the only postdoc who can get it to run, though

#overlyhonestmethods
Sharing Code Is Not Enough

The diagram illustrates the components of software development: code, data, OS, hardware, and libraries (libs). The algorithm comparison graph shows different performance scenarios: OPT, GOOD, NO-FEEDBACK, and BAD. The graph is labeled as "Figure 4: Multiple instances of WFT running in parallel."
Results Rely on Complete Context

![Diagram showing the relationship between libraries (libs), operating system (OS), code, and hardware.]
Potential Solution: Containers
Potential Solution: Containers

• Can containers reproduce any experiment?
  – Taxonomize CS experiments.
  – Determine challenging ones.

• What is container technology missing?
  – Answer empirically by reproducing an already-published experiment.

• Delineate missing components.
  – Based on learned lessons, define characteristics that enhance reproducibility capabilities.
Figure 4: Multiple instances of WFIT running in parallel
Effects of Containerizing Experiments

Before containerization:
- code
- data
- hardware
- OS
- libs

After containerization:
- code
- data
- hardware
- OS
- libs

container
Does it work for any experiment?

☑ Analyze output data.
☑ Evaluate analytic models.
☑ Handle small amounts of data.
× Depend on special hardware.
× Observe performance metrics.
Does it work for any experiment?

✔ Analyze output data.
✔ Evaluate analytic models.
✔ Handle small amounts of data.
❌ Depend on special hardware.
❌ Observe performance metrics.

Diagram:
- libs
- data
- code
- OS
- container
Experiments in systems research

- Runtime.
- Throughput.
- Latency.
- Caching effects.
- Performance model.
- etc.
Experiments in systems research

- Runtime.
- Throughput.
- Latency.
- Caching effects.
- Performance model.
- etc.

Sample of 100 papers from 10 distinct venues spanning 5 years: ~80% have one or more experiments measuring runtime.
Experiments in systems research

• Runtime.
• Throughput.
• Latency.
• Caching effects.
• Performance model.
• etc.
Experiments in systems research

- Runtime.
- Throughput.
- Latency.
- Caching effects.
- Performance model.
- etc.
Experiments in systems research

- Runtime.
- Throughput.
- Latency.
- Caching effects.
- Performance model.
- etc.
Ceph OSDI ‘06

• Select scalability experiment.
  – Distributed; makes use of all resources.
• Scaled-down version of original.
  – 1 client instead of 20
• Implement experiment in containers.
  – Docker 1.3 and LXC 1.0.6
• Experiment goal: system scales linearly.
  – This is the reproducibility criteria.
Ceph OSDI ‘06
Ceph OSDI ‘06

Throughput (MB/s)

OSD cluster size

Original
Ceph OSDI ‘06

Throughput (MB/s)

Original

Reproduced

OSD cluster size

1 2 3 4 5 6 7 8 9 10 11 12 13
Ceph OSDI ‘06

Throughput (MB/s)

OSD cluster size

Original

Reproduced

Non-scalable behavior
Repeatability Problems

1. High variability in old disk drives.
   – Causes cluster to be unbalanced.

2. Paper assumes uniform behavior.
   – Original author (Sage Weil) had to filter disks out in order to get stable behavior.
Repeatability Problems

1. High variability in old disk drives.
   – Causes cluster to be unbalanced.

2. Paper assumes uniform behavior.
   – Original author (Sage Weil) had to filter disks out in order to get stable behavior.

**Solution**: throttle I/O to get uniform raw-disk performance. 30 MB/s as the lowest common denominator.
Ceph OSDI ’06 (throttled I/O)
Ceph OSDI ’06 (throttled I/O)

Throughput (MB/s)

OSD cluster size
Lessons

1. Resource management feature of containers makes it easier to control sources of noise.
   – I/O and network bandwidth, CPU allocation, amount of available memory, etc.

2. Stuff that is not in the original paper but it’s important for reproducibility cannot be captured in container images.
   – Details about the context matter.
Container Execution Engine

Linux Kernel

Host
Container Execution Engine

<table>
<thead>
<tr>
<th>Container Execution Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux Kernel</td>
</tr>
<tr>
<td>Host</td>
</tr>
</tbody>
</table>
Container Execution Engine

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Execution Engine</td>
</tr>
<tr>
<td>Linux Kernel</td>
</tr>
<tr>
<td>Host</td>
</tr>
</tbody>
</table>
Container Execution Engine (LXC)
cgroups

CPU  MEMORY  NETWORK  STORAGE I/O
cgroups
Container “Virtual Machine”

host’s raw performance + cgroups configuration = “virtual machine”
Experiment Execution Engine

<table>
<thead>
<tr>
<th></th>
<th>LXC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cgrouops</td>
</tr>
<tr>
<td></td>
<td>Linux Kernel</td>
</tr>
<tr>
<td></td>
<td>Host</td>
</tr>
</tbody>
</table>
Experiment Execution Engine

- LXC
- cgroups
- namespace
- Linux Kernel
- Host
# Experiment Execution Engine

<table>
<thead>
<tr>
<th>Monitor</th>
<th>LXC</th>
<th>cgroups</th>
<th>namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux Kernel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Experiment Execution Engine

<table>
<thead>
<tr>
<th>Monitor</th>
<th>LXC</th>
<th>cgroups</th>
<th>namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux Kernel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Experiment**
- **Host**
- **Linux Kernel**
- **Monitor**
- **LXC**
- **cgroups**
- **namespace**
Experiment Execution Engine

- Experiment
  - Monitor
    - LXC
    - cgroups
    - namespace
  - Linux Kernel
  - Host

- Profile Repository
Experiment Execution Engine

Experiment

Monitor

LXC

cgroups
namespace

Linux Kernel

Host

Profile Repository
Experiment Profile

1. Container image
2. Platform profile
3. Container configuration
4. Execution profile
Platform Profile

- Host characteristics
  - Hardware specs
  - Age of system
  - OS/BIOS conf.
  - etc.

- Baseline behavior
  - Microbenchmarks
  - Raw performance characterization
Container Configuration

• cgroups configuration

Experiment Container

CPU Memory Network Block IO
Execution Profile

• Container metrics
  – Usage statistics (overall)
Execution Profile

- Container metrics
  - Usage statistics (overall)
  - Over time

![CPU by image](image1)

![CPU by container](image2)

![RSS by image](image3)

![RSS by container](image4)
Experiment Profile

1. Container image
2. Platform profile
3. Container configuration
4. Execution profile
Experiment Profile

1. Container image
2. Platform profile
3. Container configuration
4. Execution profile

Having all this information at hand is invaluable when validating an experimental result and is usually not found in an academic article.
Mapping Between Hosts

Reproduce on host B (original ran on host A):

1. Obtain the platform profile of A.
2. Obtain the container configuration on A.
3. Obtain the platform profile of B.
4. Using 1-3, generate configuration for B.

Example: emulate memory/io/network on B so that characteristics of A are reflected.
Mapping Between Hosts

System A

CPU  Memory  Network  Block IO
Mapping Between Hosts

System A

Experiment Container
Mapping Between Hosts

System A

Experiment Container

System B
Mapping Between Hosts

System A

- CPU
- Memory
- Network
- Block IO

Experiment Container

System B

- CPU
- Memory
- Network
- Block IO
Does it work?
Does it work?
Mapping doesn’t always work

• Experiments that rely on unmanaged operations and resources.
  – Asynchronous I/O, memory bandwidth, L3 cache, etc.

• Enhancing isolation guarantees of container execution engine results in supporting more of these cases.
  – E.g. if cgroups now isolate asynchronous I/O for every distinct group.
Open Question

• Given strong isolation guarantees, can we automatically check for repeatability by looking at low-level metrics?
Open Question

• Given strong isolation guarantees, can we *automatically* check for repeatability by looking at low-level metrics?
Open Question

• Given strong isolation guarantees, can we **automatically** check for repeatability by looking at low-level metrics?
Open Question

• Given strong isolation guarantees, can we **automatically** check for repeatability by looking at low-level metrics?
### Open Question

- Given strong isolation guarantees, can we **automatically** check for repeatability by looking at low-level metrics?

---

**System A**

<table>
<thead>
<tr>
<th>CPU by container</th>
<th>CPU by image</th>
<th>RSS by image</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**System B**

<table>
<thead>
<tr>
<th>CPU by container</th>
<th>CPU by image</th>
<th>RSS by image</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Open Question

• Given strong isolation guarantees, can we automatically check for repeatability by looking at low-level metrics?

System A = System B
On-going and Future Work

• Taking more already-published experiments to test robustness of our approach.
• Integrate our profiling mechanism into container orchestration tools.
Thanks!