Tackling the Reproducibility Problem in Systems Research with Declarative Experiment Specifications

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

Figure 5: Per-OSD write performance. The horizontal line indicates the upper limit imposed by the physical disk. Replication has minimal impact on OSD throughput, although the number of OSDs is fixed; every replication reduces total effective throughput by a factor of 2 because replicated data must be written to n OSDs.

Figure 7: Write latency for varying write sizes and replicas. More than two replicas incur minimal additional cost for small writes because replicated updates occur concurrently. For large synchronous writes, multiple writes are performed sequentially. Claims that latency for writes over 128 KB by acquiring exclusive locks and asynchronously flushing the data.

Figure 6: Performance of EBDFS compared to general-purpose file systems. Although read performance suffers from coarse-threading and locking, EBDFS very nearly saturates the available disk bandwidth for writes larger than 72 KB, and significantly outperforms the others for read workloads because data is laid out on extents on disk that match the write sizes—even when they are very large. Performance was measured using a fresh file system. Experience with an earlier EBDFS design suggests that it will experience significantly lower fragmentation than each, but we have not yet evaluated the current implementation on an aged file system. In any case, we expect the performance of EBDFS after aging to be worse than the others.

6.1.2 Write Latency

Figure 7 shows the synchronous write latency (µ) for a single writer with varying write sizes (ε) and replication.

Cluster size

Throughput (MB/s)

Original

Reproduced

Figure 8: OSD write performance scales linearly with the size of the OSD cluster until the switch is saturated at 24 OSDs. CIRCUS and burst performance improves when more IOs issue requests to other OSDs.

Because the primary OSD simultaneously synchronizes updates to all replicas, small writes incur a minimal latency increase for more than two replicas. For larger writes, the cost of synchronization dominates: 1 MB write (not shown) takes 13 ms for one replica, and 3.5 times longer (35 ms) for three. Ceph clients partially mask this latency for synchronous writes over 128 KB by acquiring exclusive locks and then asynchronously flushing the data to disk. Alternatively, write-blocking applications can use CFS. With consistency thus relaxed, clients can buffer small writes and submit only large, synchronous writes to OSDs; the only latency seen by applications will be due to clients which fill their caches waiting for data to flush to disk.

6.1.3 Data Distribution and Scalability

Ceph's data performance scales nearly linearly in the number of OSDs. CIRCUS distributes data pseudorandomly such that OSD utilizations can be accurately modeled by a binomial or normal distribution—what one expects from a perfectly random process. Validation...
Need a piece of information that describes the relationship between these...
Outline

• Declarative Experiment Specification
• Case Study
• Discussion
Declarative Experiment Specification
Experiment Goal: Show that my algorithm/system/etc. is better than the state-of-the-art.
Validation Language Syntax

validation
  : 'for' condition ('and' condition)* 'expect' result ('and' result)*
    ;

condition
  : vars ('in' range | ('=' | '<' | '>' | '!=') value)
    ;

result
  : condition
    ;

vars
  : var (',' var)*
    ;

range
  : '[' range_num (',' range_num)* ']'
    ;

range_num
  : NUMBER '-' NUMBER | '*'
    ;

value
  : '*' | 'NUMBER (',' NUMBER)*'
    ;
Case Study
Ceph OSDI ‘06

• Select scalability experiment.
  – Distributed; makes use of all resources.
• Scaled-down version of original.
  – 1 client node instead of 20
• Experiment goal: system scales linearly.
  – This is the reproducibility criteria.
Validation Statement

for cluster_size = *
expect
  ceph >= (raw * .90) and
  not net_saturated

"independent_variables": [{
  "type": "cluster_size",
  "values": "2-28"
}, {
  "type": "method",
  "values": ["raw", "ceph"]
},{
  "type": "net_saturated",
  "values": ["true", "false"]
}],
"dependent_variable": {
  "type": "throughput",
  "scale": "mb/s"
}
The diagram shows the normalized per-OSD throughput as a function of OSD cluster size. The x-axis represents the OSD cluster size, while the y-axis shows the normalized throughput.

The table lists the original and reproduced components and their specifications:

<table>
<thead>
<tr>
<th>Component</th>
<th>Original</th>
<th>Reproduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>AMD 2212 @2.0GHz</td>
<td>Intel E5-2630 @2.3GHz</td>
</tr>
<tr>
<td>Disk drive</td>
<td>Seagate ST3250620NS</td>
<td>HP 6G 658071-B21</td>
</tr>
<tr>
<td>Disk BW</td>
<td>58 MB/s</td>
<td>120 MB/s (15 MB/s limit)</td>
</tr>
<tr>
<td>Linux</td>
<td>2.6.9</td>
<td>3.13.0</td>
</tr>
<tr>
<td>Ceph</td>
<td>commit from 2005</td>
<td>0.87.1</td>
</tr>
<tr>
<td>Storage</td>
<td>26 nodes</td>
<td>12 nodes</td>
</tr>
<tr>
<td>Clients</td>
<td>20 nodes</td>
<td>1 node</td>
</tr>
<tr>
<td>Network</td>
<td>Netgear GS748T</td>
<td>Same as original</td>
</tr>
<tr>
<td>Network BW</td>
<td>1400 MB/s</td>
<td>110 MB/s</td>
</tr>
</tbody>
</table>
Discussion
Discussion

• Falsifiability
• Usability
• Validation workflow
• Integration with existing infrastructure
• Early feedback
Falsifiability in Science

Falsifiability of a statement, hypothesis, or theory is an inherent possibility to prove it to be false.

• In other words, the ability to specify the conditions under which a statement is false
• Synonymous to Testability
• Example:
  – Statement: All swans are white
  – Falsifiable: Observe one black swan

source: en.wikipedia.org/wiki/Falsifiability
Falsifiability in Systems

Experiment Goal: Show that my algorithm/system/etc. is better than the state-of-the-art.

Means of Experiment

Raw data

Observations

Figure 4: Multiple instances of WFTT running in parallel

Figure 5 illustrates the time required to complete MySQL’s test-insert benchmark. Applying DTA and ISR on the server for the entire duration of the test increases execution time by 4.8x and 2.6x respectively, when compared to native execution. In contrast, partitioning slows down execution by 1.8x and 2.6x, when using DTA only for the non-authenticated part of the execution, and then switching to no instrumentation and ISR respectively. We observe that the overhead of applying DTA diminishes, as the unauthenticated partition runs only for a short period of time. In general, partitioned execution performs similarly to the mechanism applied on the authenticated partition.
Falsifiability in Systems

• To falsify a claim:
  – Describe the means of the experiments
  – Provide validation statements over the output data

• Conditional statement:
  – if means are properly recreated
  – then validation statements should hold

• Go from inert observations to falsifiable statements

From:
  
  We observe that our system outperforms the alternatives

To:
  
  Expect 25-30% performance improvement on hardware platform X, on workload Y, when configured like Z
Usability

• Creating ESFs represents little overhead
  – Researchers do this anyway

• Provides useful template
  template/checklist
  – Makes it easier to follow the scientific method
Validation Workflow

1. Obtain/recr eate means of experiment
2. Re-run and check validation clauses against output. Any validation failed?
   - no: Original work findings are corroborated
   - yes: Any significant differences between original and recreated means?
     - yes: Update means of experiment
     - no: Cannot validate original claims

Integration with Existing Infrastructure

- Unit Tests
- Integration Tests
- Validations

Push code and ESF

Pull

Test:
- Unit
- Integration
- Validations
Early Feedback

Creating an ESF helps to:

• Find meaningful/reproducible baselines
• Create a feedback loop in author’s mind
• Specify exactly what author means
• Make temporal context explicit
Challenges

• Bottlenecks “move” between resources
  – Include sanity checks as part of experiment
  – Example: corroborate that network/disk observes expected behavior

• Binary reproducibility
  – Example: GCC’s flags, $10^{806}$ combinations
  – Solution: provide image of complete software stack (e.g. linux containers)
Conclusion

The criterion of the scientific status of a theory is its falsifiability, or refutability, or testability.

- Karl Popper

ESFs:

• Describe high-level components of an experiment

• Specify falsifiable validation statements that back a claim

• Incorporate falsifiability to the field of experimental systems research

https://github.com/systemslab/esf
Thanks!
Geneiatakis et. al. CCS ‘12

![Bar Chart]

- Native
- Pin
- ISR
- DTA
- DTA/Pin
- DTA/ISR

Total Time (sec)
In this section, our goal is to evaluate the performance benefits that can be reaped, by utilizing virtual partitioning to apply otherwise expensive protection mechanisms on the most exposed part of applications. This allows us to strike a balance between the overhead imposed on the application and its exposure to attacks.
In this section, our goal is to evaluate the performance benefits that can be reaped, by utilizing virtual partitioning to apply otherwise expensive protection mechanisms on the most exposed part of applications. This allows us to strike a balance between the overhead imposed on the application and its exposure to attacks.
Schema
Schema

"independent_variables": [
  {
    "type": "method",
    "alias": ["technique"],
    "values": [
      "native", "pin", "isr", "dta",
      "dta_pin", "dta_isr"
    ]
  }
],
"dependent_variable": {
  "type": "runtime",
  "scale": "s"
},

<table>
<thead>
<tr>
<th>Method</th>
<th>Total Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td></td>
</tr>
<tr>
<td>Pin</td>
<td></td>
</tr>
<tr>
<td>ISR</td>
<td></td>
</tr>
<tr>
<td>DTA</td>
<td></td>
</tr>
<tr>
<td>DTA/Pin</td>
<td></td>
</tr>
<tr>
<td>DTA/ISR</td>
<td></td>
</tr>
</tbody>
</table>
Validations

"independent_variables": [
  {
    "type": "method",
    "alias": ["technique"],
    "values": [
      "native", "pin", "isr", "cta",
      "cta_pin", "cta_isr"
    ]
  }
],
"dependent_variable": {
  "type": "runtime",
  "scale": "s"
},

![Bar chart showing total time (sec) for different variables: Native, Pin, ISR, DTA, DTA/Pin, DTA/ISR.]}
Validations

```
expect
native < any
```

```
"independent_variables": [  
  {  
    "type": "method",  
    "alias": ["technique"],  
    "values": [  
      "native", "pin", "isr", "dta",  
      "dta_pin", "dta_isr"  
    ]  
  }  
],  
"dependent_variable": {  
  "type": "runtime",  
  "scale": "s"  
},
```
Validations

expect
native < any and

"independent_variables": [
  {
    "type": "method",
    "alias": ["technique"],
    "values": [
      "native", "pin", "isr", "dta",
      "dta_pin", "dta_isr"
    ]
  }
],
"dependent_variable": {
  "type": "runtime",
  "scale": "s"
};
Validations

**expect**
native < any and
dta_pin between pin and isr

```
"independent_variables": [
  {
    "type":  "method",
    "alias": ["technique"],
    "values": [
      "native", "pin", "isr", "dta",
      "dta_pin", "dta_isr"
    ]
  }
],
"dependent_variable": {
  "type":  "runtime",
  "scale": "s"
}
```

![Bar chart showing total time (sec)](chart.png)
Validations

expect
native < any and
dta_pin between pin and isr and

"independent_variables": [
  {
    "type": "method",
    "alias": ["technique"],
    "values": [
      "native", "pin", "isr", "dta",
      "dta_pin", "dta_isr"
    ]
  }
],
"dependent_variable": {
  "type": "runtime",
  "scale": "s"
}
Validations

```json
"independent_variables": [
    {
        "type": "method",
        "alias": ["technique"],
        "values": [
            "native", "pin", "isr", "dta",
            "dta_pin", "dta_isr"
        ]
    }
],
"dependent_variable": {
    "type": "runtime",
    "scale": "s"
},
"expect":
    native < any and
    dta_pin between pin and isr and
    dta_isr between isr and dta
```
Example 2
Example 2
Schema

"independent_variables": [
  {
    "type": "method",
    "values": [
      "native", "pin", "isr", "dta",
      "dta_pin", "dta_isr"
    ]
  }
],
"dependent_variable": {
  "type": "runtime",
  "scale": "s"
};
Schema

"independent_variables": [
  {
    "type": "method",
    "values": [
      "native", "pin", "isr", "dta",
      "dta_pin", "dta_isr"
    ]
  },
  {
    "type": "workload",
    "values": ["ftp", "samba", "ssh"]
  }
],
"dependent_variable": {
  "type": "runtime",
  "scale": "s"
},
Validations

for workload=* 
expect native < any and 
dta_pin between pin and isr and 
dta_isr between isr and dta

"independent_variables": [
  {
    "type": "method",
    "values": [
      "native", "pin", "isr", "dta",
      "dta_pin", "dta_isr"
    ]
  },
  {
    "type": "workload",
    "values": ["ftp", "samba", "ssh"]
  }
],
"dependent_variable": {
  "type": "runtime",
  "scale": "s"
}
Example 3
<table>
<thead>
<tr>
<th>Type</th>
<th>Cuckoo hashing (K keys/s)</th>
<th>Trie (K keys/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual insertion</td>
<td>10182</td>
<td>–</td>
</tr>
<tr>
<td>Bulk insertion</td>
<td>–</td>
<td>7603</td>
</tr>
<tr>
<td>Lookup</td>
<td>1840</td>
<td>208</td>
</tr>
</tbody>
</table>

Table 5: In-memory performance of index data structures in SILT on a single CPU core.
Experiment Goal

The high random read speed of flash drives means that the CPU budget available for each index operation is relatively limited. This microbenchmark demonstrates that SILT’s indexes meet their design goal of computation-efficient indexing.

<table>
<thead>
<tr>
<th>Type</th>
<th>Cuckoo hashing (K keys/s)</th>
<th>Trie (K keys/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual insertion</td>
<td>10182</td>
<td>–</td>
</tr>
<tr>
<td>Bulk insertion</td>
<td>–</td>
<td>7603</td>
</tr>
<tr>
<td>Lookup</td>
<td>1840</td>
<td>208</td>
</tr>
</tbody>
</table>

Table 5: In-memory performance of index data structures in SILT on a single CPU core.
Schema

```
{
    "type": "method",
    "values": ["raw", "cuckoo", "trie"]
},
{
    "type": "workload",
    "values": [
        "individual", "bulk", "lookup"
    ]
},
"dependent_variable": {
    "type": "throughput",
    "scale": "bytes/second"
}
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Cuckoo hashing (K keys/s)</th>
<th>Trie (K keys/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual insertion</td>
<td>10182</td>
<td>–</td>
</tr>
<tr>
<td>Bulk insertion</td>
<td>–</td>
<td>7603</td>
</tr>
<tr>
<td>Lookup</td>
<td>1840</td>
<td>208</td>
</tr>
</tbody>
</table>

**Table 5:** In-memory performance of index data structures in SILT on a single CPU core.
Validations

```python
for workload=*
expect
cuckoo > raw and trie > raw
for
lookup
expect
cuckoo > trie
and
for
individual and bulk
expect
cuckoo > trie
```

```json
{
  "type": "method",
  "values": ["raw", "cuckoo", "trie"]
},
{
  "type": "workload",
  "values": [
    "individual", "bulk", "lookup"
  ]
},
"dependent_variable": {
  "type": "throughput",
  "scale": "bytes/second"
}
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Cuckoo hashing (K keys/s)</th>
<th>Trie (K keys/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual insertion</td>
<td>10182</td>
<td>–</td>
</tr>
<tr>
<td>Bulk insertion</td>
<td>–</td>
<td>7603</td>
</tr>
<tr>
<td>Lookup</td>
<td>1840</td>
<td>208</td>
</tr>
</tbody>
</table>

Table 5: In-memory performance of index data structures in SILT on a single CPU core.